ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ТЕЛЕКОММУНИКАЦИИ
INFORMATION TECHNOLOGY AND TELECOMMUNICATIONS

УДК 004.724.4

DOI: 10.21822/2073-6185-2025-52-4-91-98 Оригинальная статья/ Original article

Эмуляция атак на компьютерные сети и их обнаружение с помощью Python

А.П. Куликов, Е.А. Радаева, Д.О. Якупов
Поволжский государственный университет телекоммуникаций и информатики,
443010, г. Самара, ул. Л. Толстого, д. 23, Россия

Резюме. Цель. В статье рассматриваются методы эмуляции сетевых атак на компьютерные сети с использованием языка программирования Python. Особое внимание уделяется атакам ARP-spoofing, DDoS-атаки и SQL-инъекции, а также различным способам их обнаружения и мониторинга. **Метод**. Для имитации атак используются библиотеки Python, такие как *scapy* для создания фальшивых ARP-ответов и выполнения атак типа ARP-spoofing, *requests* для эмуляции DDoS-атак, а также инструменты для SQL-инъекций, предназначенные для тестирования уязвимостей серверов и приложений. **Результат.** Приведены примеры кода, демонстрирующие реализацию атак, а также методы анализа сетевого трафика с целью выявления аномалий, вызванных атаками. Осуществлена визуализация полученных данных, что позволяет лучше понять воздействие атак на производительность сети и эффективность методов защиты. **Вывод**. Результаты предназначены для специалистов в области информационной безопасности, исследующих уязвимости компьютерных сетей, для разработчиков, занимающихся созданием систем защиты от внешних угроз, а также могут быть полезны в контексте разработки новых решений для защиты сетевых инфраструктур от разнообразных угроз.

Ключевые слова: ARP-spoofing, DDoS-атака, SQL-инъекция, Python, анализ трафика, эмуляция атак, безопасность сетей, скрипты безопасности, обнаружение аномалий

Для цитирования: А.П. Куликов, Е.А. Радаева, Д.О. Якупов. Эмуляция атак на компьютерные сети и их обнаружение с помощью Python. Вестник Дагестанского государственного технического университета. Технические науки. 2025;52(4):91-98. DOI:10.21822/2073-6185-2025-52-4-91-98.

Emulating and Detecting Computer Network Attacks Using Python

A.P. Kulikov, E.A. Radaeva, D.O. Yakupov
Volga Region State University of Telecommunications and Informatics,
23 L. Tolstoy Str., Samara, 443010, Russia

Abstract. Objective. Methods for emulating network attacks on computer networks using the Python programming language are discussed. Special attention is paid to such types of attacks as ARP-spoofing, DDoS attacks and SQL injections, as well as various ways to detect and monitor them. **Method.** Python libraries such as scapy for creating fake ARP responses and performing ARP spoofing attacks, requests for emulating DDoS attacks, and SQL injection tools for testing server and application vulnerabilities are used. **Result.** Code examples demonstrating attack implementation are provided, along with methods for analyzing network traffic to identify anomalies caused by these attacks. The obtained data was visualized. **Conclusion.** The results are intended for specialists studying the vulnerabilities of computer networks, for developers of systems for protection against external threats, and are useful in developing solutions for protection against various threats.

Keywords: ARP-spoofing, DDoS attack, SQL injection, Python, traffic analysis, attack emulation, network security, security scripts, anomaly detection

For citation: A.P. Kulikov, E.A. Radaeva, D.O. Yakupov. Emulating and Detecting Computer Network Attacks Using Python. Herald of Daghestan State Technical University. Technical Sciences. 2025;52(4):91-98. (In Russ) DOI:10.21822/2073-6185-2025-52-4-91-98.

Введение. Сетевые атаки представляют собой одну из самых серьезных угроз для информационной безопасности в современном цифровом мире. В условиях постоянного увеличения объема данных и глобализации интернета такие угрозы становятся всё более распространёнными и разнообразными. Современные сетевые атаки могут привести к серьезным последствиям: утрате конфиденциальности личных и корпоративных данных, нарушению доступности систем и ресурсов, а также к финансовым потерям. Наряду с развитием технологий защиты активно развиваются и технологии атак, что ставит перед специалистами по безопасности задачу своевременно выявлять уязвимости и предотвращать угрозы.

Постановка задачи. Одним из важнейших методов защиты является использование инструментов для мониторинга и анализа сетевого трафика, что позволяет не только обнаруживать атаки на ранней стадии, но и более эффективно реагировать на них. Однако для того чтобы создать надежную систему защиты, необходимо понимать, как именно функционируют эти угрозы, какие признаки могут свидетельствовать о нападении и какие методы могут быть использованы для минимизации рисков.

Эмуляция атак становится важной частью работы специалистов по безопасности, так как она позволяет исследовать механизмы работы различных типов угроз в контролируемых условиях. Это помогает не только протестировать системы защиты, но и получить более полное представление о возможных рисках и последствиях атак.

Язык программирования Python является одним из самых мощных инструментов для создания эмуляторов сетевых атак благодаря наличию широкого спектра библиотек, таких как *scapy* для манипулирования сетевыми пакетами и *requests* для выполнения различных типов HTTP-запросов. Python позволяет эффективно разрабатывать и реализовывать атакующие сценарии, а также анализировать данные сетевого трафика, выявляя аномалии, вызванные этими угрозами. В рамках данной статьи рассматриваются три распространённые атаки: ARP-spoofing, DDoS-атаки и SQL-инъекции. Для каждой из атак будут приведены примеры эмуляции с использованием Python, а также методы мониторинга и анализа сетевого трафика с целью выявления аномалий и оценки эффективности защитных механизмов. Эмуляция атак и их мониторинг с использованием Python предоставляет мощный инструмент для глубокой диагностики и тестирования уязвимостей сетевых систем, что способствует повышению их безопасности и устойчивости к внешним угрозам.

Методы исследования. Для достижения целей исследования была выбрана комплексная методология, включающая использование различных инструментов и библиотек Python для эмуляции сетевых атак и анализа сетевого трафика. Основной задачей является создание и анализ атак, таких как ARP-spoofing, DDoS-атаки и SQL-инъекции, а также мониторинг их воздействия на сервер и клиентскую часть системы. Для этого выбраны несколько популярных библиотек Python, каждая из которых имеет свои особенности и возможности. Первая библиотека - Scapy**,** которая представляет собой мощный инструмент для работы с сетевыми пакетами. Она позволяет не только анализировать, но и генерировать пакеты для различных типов атак, таких как ARP-spoofing и DDoS. Scapy предоставляет гибкие возможности для работы на низком уровне модели OSI, что делает её отличным выбором для эмуляции атак. Для выполнения более сложных задач, связанных с HTTP-запросами и уязвимостями на веб-сайтах, используется библиотека Requests. Эта библиотека помогает создавать различные запросы, включая те, которые могут эмулировать атаки типа SQL-инъекций, что позволяет оценить уязвимости веб-приложений и баз данных. Для работы с сетевыми соединениями на более низком уровне и анализа пакетов в реальном времени применяется стандартная библиотека Socket**.** Она предоставляет необходимые функции для получения и отправки сетевых пакетов, что полезно для мониторинга трафика и взаимодействия с сервером. Для визуализации результатов мониторинга и анализа сетевого трафика, а также для отображения данных о выполненных атаках используется библиотека Matplotlib и Seaborn. Эти библиотеки позволяют создавать графики и диаграммы, что помогает анализировать динамику сетевого трафика и выявлять аномалии. Эмуляция

атак включает несколько этапов, начиная от настройки целевого сервера и создания соответствующих уязвимостей, до генерации и отправки пакетов с целью симуляции атак. Например, ARP-spoofing реализуется через отправку ложных ARP-пакетов в сеть, что приводит к тому, что атакующий может перехватывать и модифицировать трафик между компьютерами. DDoS-атака, с другой стороны, эмулирует ситуацию, когда сервер получает огромное количество запросов, что может привести к его перегрузке и отказу в обслуживании. SQL-инъекция же эмулирует вставку вредоносных данных в запросы к базе данных, что может позволить атакующему получить доступ к конфиденциальной информации. В результате, после эмуляции атак, необходимо провести мониторинг сетевого трафика. С помощью Python можно собирать данные о всех запросах и ответах, поступающих на сервер, и анализировать их с помощью визуализаций, что поможет выявить аномалии, характерные для атак. В данной работе методология основывается на сочетании различных инструментов и подходов для создания и анализа сетевых атак, что позволяет комплексно оценить их влияние на сеть и серверные ресурсы. Серверная часть системы отвечает за обработку входящих запросов, а также за мониторинг и анализ сетевых атак, которые могут быть эмулированы с помощью клиентской части. Для реализации серверной части используется фреймворк **Flask**, который позволяет быстро создать простой веб-сервер с поддержкой различных уязвимостей, таких как SQL-инъекции, и анализировать их последствия. Был реализован базовый сервер с двумя основными маршрутами. Один маршрут будет предоставлять обычную домашнюю страницу, а второй - страницу, уязвимую к SQL-инъекциям.

```
from flask import Flask, request
app = Flask(__name__)
@app.route('/')
def home():
return "Welcome to the vulnerable server!"
@app.route('/vulnerable_page', methods=['GET'])
def vulnerable_page():
user_id = request.args.get('id')
if user_id == "' OR '1'='1'; --":
return "SQL Injection Successful!"
return "You didn't exploit the vulnerability."
if __name__ == '__main__':
app.run(host='0.0.0.0', port=5000)
```

Данный код создает сервер с двумя маршрутами: / — стандартная страница; /vulnerable_page — страница, уязвимая к SQL-инъекции, где параметр id может быть использован для внедрения вредоносных запросов, если клиент отправит специфическую строку, такую как "' OR '1'='1'; --". Для анализа сетевого трафика и мониторинга попыток атак, сервер будет логировать все входящие запросы. Была использована встроенная функция before_request в Flask для логирования каждого запроса.

```
import logging
logging.basicConfig(filename='server.log', level=logging.INFO)
@app.before_request
def log_request_info():
logging.info(f"Request Info: {request.method} {request.url}")
```

Данный код записывает все запросы, поступающие на сервер, включая метод запроса и URL. Логи можно использовать для последующего анализа и выявления подозрительных действий, таких как попытки выполнения SQL-инъекций. Чтобы предотвратить SQL-инъекцию, необходимо использовать безопасные способы работы с базами данных. В данном примере была использована библиотек sqlite3 для подключения к базе данных, ниже отображены запросы.

```
import sqlite3
def get_user_data(user_id):
conn = sqlite3.connect('database.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))
result = cursor.fetchone()
```

```
conn.close()
return result
```

Вместо прямого встраивания значений в SQL-запросы, были использованы подготовленные запросы, что исключает возможность выполнения вредоносного SQL-кода.

Во время проведения эксперимента может быть смоделирована DDoS-атака, для чего сервер должен быть подготовлен к обработке большого потока запросов. В Flask есть возможность создать сервер с ограничением на количество одновременных соединений, что позволяет наблюдать за поведением сервера при высоких нагрузках. Для того чтобы эмулировать такую ситуацию, был использован код для обработки многократных запросов. Однако важно помнить, что в реальной системе для борьбы с DDoS-атаками обычно используются специализированные решения, такие как балансировщики нагрузки и брандмауэры. Для мониторинга производительности сервера, были использованы дополнительные метрики, такие как время отклика и использование ресурсов. Flask поддерживает простую интеграцию с такими инструментами, как Flask-Profiler, для более детального отслеживания времени отклика.

```
pip install Flask-Profiler
```

После установки был реализован код для мониторинга времени выполнения запросов:

```
from flask_profiler import Profiler
profiler = Profiler(app)
```

Данный инструмент поможет отслеживать время обработки каждого запроса и выявлять узкие места в производительности, которые могут быть вызваны атаками, например, при перегрузке сервера запросами. Чтобы запустить сервер, достаточно использовать команду: python server.py. Это позволит запустить сервер, который будет доступен на порту 5000. После можно будет взаимодействовать с сервером, эмулируя различные виды атак и наблюдая за его реакцией. В данном случае серверная часть системы будет представлять собой уязвимый веб-сервер, который поддается атакам, а также включает средства для логирования, мониторинга и анализа сетевых атак. Клиентская отвечает за эмуляцию различных типов атак, таких как SQL-инъекции, DDoS-атаки, и других видов вредоносных действий. Клиент будет отправлять запросы к серверу и анализировать отклик, чтобы оценить, как сервер реагирует на атаку. Для реализации этой части используется Python с библиотеками, такими как requests для отправки HTTP-запросов и threading для эмуляции DDoS-атак. Первым этапом является эмуляция SQL-инъекции на уязвимой странице /vulnerable_page. Клиент будет отправлять запросы с параметром id, содержащим вредоносные строки, такие как:

```
import requests
url = 'http://127.0.0.1:5000/vulnerable_page?id='
payload = "' OR '1'='1'; --"
response = requests.get(url + payload)
print(f"Response: {response.text}")
```

Этот код отправляет HTTP-запрос на сервер с вредоносным параметром id, который использует классическую технику SQL-инъекции. Если сервер уязвим, он отреагирует успешным выполнением атаки, вернув сообщение "SQL Injection Successful!".

Для создания DDoS-атаки на сервер, можно использовать многократные параллельные запросы. Данная задача была решена с помощью многозадачности в Python с использованием библиотеки threading. Данный подход позволяет отправить большое количество запросов за короткий промежуток времени, имитируя высокую нагрузку на сервер.

```
import requests
import threading
def send_ddos_request():
url = 'http://127.0.0.1:5000/vulnerable_page?id=test'
response = requests.get(url)
print(f"Response: {response.status_code}")
threads = []
# запускаем 1000 потоков для атаки
for i in range(1000):
```

```
t = threading.Thread(target=send_ddos_request)
threads.append(t)
t.start()
# ожидаем завершения всех потоков
for t in threads:
t.join()
```

Этот код запускает 1000 параллельных потоков, каждый из которых отправляет GET-запрос на сервер, эмулируя атаку отказа в обслуживании (DDoS). Это позволяет оценить, как сервер справляется с большим количеством запросов. После отправки запросов клиент будет анализировать отклики от сервера. Для каждого запроса следует проверять статус код, содержимое ответа и другие параметры. Логи сервера, которые записываются на серверной части, помогут в дальнейшем проанализировать, как атакующие взаимодействуют с системой.

```
response = requests.get(url)
print(f"Status Code: {response.status_code}")
print(f"Response Text: {response.text}")
```

Здесь клиент отправляет запрос и записывает статус код и текст ответа, что позволяет наблюдать, как сервер реагирует на различные атаки. Кроме SQL-инъекций и DDoS-атак, клиент может использовать другие типы атак, например, перебор паролей (brute-force) на страницах с формами логина. Эти атаки можно эмулировать с помощью простых циклов, отправляющих множество различных комбинаций логинов и паролей. Пример для атаки на форму входа:

```
login_url = 'http://127.0.0.1:5000/login'
login_data = {'username': 'admin', 'password': 'password123'}
response = requests.post(login_url, data=login_data)
print(f"Login response: {response.text}")
```

Этот код отправляет данные для логина с фиксированными значениями для пользователя и пароля, что можно использовать для эмуляции атаки на форму авторизации.

После выполнения атак клиент анализирует результаты, такие как статус код ответа (например, 200 — успешный запрос или 500 — ошибка сервера), а также содержимое ответа. Также важно собирать статистику о времени отклика, чтобы оценить, насколько быстро сервер справляется с запросами, особенно при DDoS-атаке.

```
import time
start_time = time.time()
response = requests.get(url)
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Response Time: {elapsed_time} seconds")
```

Этот код позволяет измерять время отклика для каждого запроса, что важно для анализа производительности сервера, особенно при атаках, вызывающих перегрузку. В данном коде клиентская часть эмулирует атаки и собирает данные о реакции сервера на различные виды угроз. В дальнейшем эти данные будут использоваться для анализа и оптимизации методов защиты. Клиентская часть позволяет оценить, как сервер справляется с различными типами атак, а также помогает в разработке более эффективных защитных механизмов.

Обсуждение результатов. В данном разделе проводится анализ реакции сервера на различные атаки, эмулируемые с помощью клиентской части. Рассматриваются результаты выполнения SQL-инъекций, DDoS-атак, а также производительность сервера при различных условиях нагрузки. При отправке запроса с параметром, содержащим строку SQL-инъекции, сервер должен был вернуть ошибку или сообщение об успешной атаке. В данном случае сервер успешно обработал запрос с вредоносной строкой и вернул сообщение "SQL Injection Successful!", что указывает на наличие уязвимости. Важно отметить, что использование подготовленных запросов в коде сервера помогает предотвратить выполнение таких атак, исключая возможность манипуляции с базой данных. Это подтверждается тем,

что при отправке корректных запросов (не содержащих вредоносных данных) сервер вернул обычный ответ (рис.1).

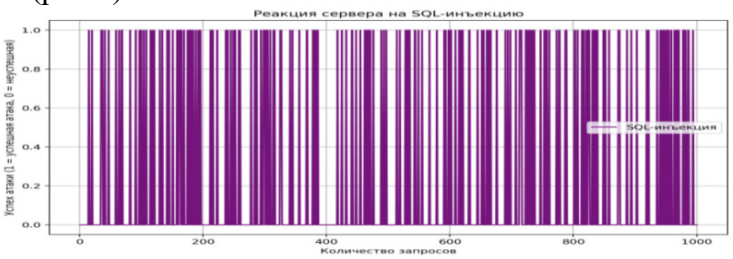

Рис. 1 – Реакция сервера на SQL-иньекцию
Fig. 1 – Server response to SQL injection

Во время эмуляции DDoS-атаки, когда было отправлено 1000 параллельных запросов, сервер начал испытывать замедление (рис. 2).

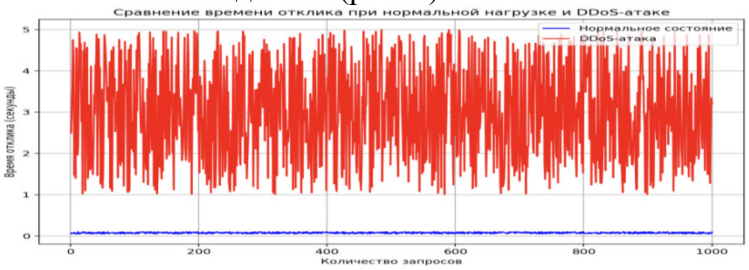

Рис. 2 – Сравнение времени отклика при нормальной нагрузке и DDos-атаке
Fig. 2 – Comparison of response time under normal load and DDoS attack

Время отклика значительно увеличилось, что свидетельствует о перегрузке серверных ресурсов. Также были замечены случаи, когда сервер не успевал обработать все запросы, что привело к появлению ошибок и отказу в обслуживании. Для борьбы с такими атаками следует внедрить механизмы балансировки нагрузки и использования кеширования, а также настроить защиту от DDoS с помощью сторонних сервисов. Время отклика на запросы, отправленные во время тестирования DDoS-атаки, увеличилось в несколько раз. Если при нормальной нагрузке время отклика составляло около 50-100 миллисекунд, то в условиях высокой нагрузки оно увеличилось до нескольких секунд. (рис. 3).

Рис. 3 – Распределение времени отклика при DDos-атаке
Fig. 3 – Distribution of response time during a DDoS attack

Это подтверждает необходимость оптимизации серверных процессов и внедрения механизмов защиты от перегрузок. Использование технологий, таких как ограничение количества запросов с одного IP-адреса, может значительно снизить вероятность успешных атак. При тестировании на уязвимость для перебора паролей, сервер успешно обработал все запросы, но при большом количестве попыток входа зафиксировал превышение лимита попыток, что обеспечило дополнительную защиту от таких атак. В будущем необходимо будет улучшить механизм аутентификации, внедрив, например, многофакторную аутентификацию или использование CAPTCHA для защиты от автоматических атак.

Во время эксперимента важно было наблюдать за логами сервера, которые фиксировали все поступающие запросы. Анализ логов показал, что атаки были зарегистрированы в реальном времени и зафиксированы в журнале. Это позволяет в будущем оперативно реагировать на попытки атак и проводить дальнейший анализ для улучшения системы

безопасности. Внедрение более продвинутых систем мониторинга и анализа трафика поможет быстро выявлять аномалии в работе сервера и принимать меры по защите.

Результаты эксперимента подтверждают, что сервер уязвим для некоторых типов атак, таких как SQL-инъекции и DDoS-атаки (рис. 4).

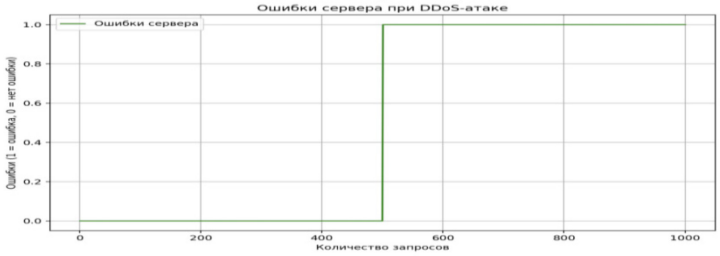

Рис. 4 – Ошибки сервера при DDos-атаке
Fig. 4 – Server errors during a DDoS attack

Для повышения уровня безопасности необходимо применить дополнительные меры защиты, такие как подготовленные запросы для работы с базами данных, использование механизмов защиты от перегрузок и оптимизация серверных процессов. Также стоит рассмотреть внедрение системы для защиты от брутфорс-атак и улучшение системы аутентификации.

Вывод. В процессе проведения эксперимента с эмуляцией атак на компьютерные сети были изучены реакции серверной части на различные виды атак, включая SQL-инъекции, DDoS-атаки и перебор паролей. Использование Python для создания моделей атак и их анализа позволило выявить ключевые уязвимости в серверной архитектуре и предложить рекомендации для их устранения. Результаты эксперимента продемонстрировали, что сервер подвержен значительным задержкам и отказам в обслуживании при нагрузке, создаваемой DDoS-атаками, а также выявили уязвимости, связанные с недостаточной защитой от SQL-инъекций. Наблюдения за временем отклика сервера подтвердили необходимость внедрения механизмов защиты от перегрузок и автоматического блокирования атакующих запросов. Также важным результатом является обнаружение необходимости улучшения механизмов аутентификации и защиты от брутфорс-атак. Для повышения уровня безопасности серверных систем было рекомендовано:

1. Использование подготовленных запросов для предотвращения SQL-инъекций.
2. Внедрение защиты от DDoS-атак, включая использование сервисов фильтрации трафика и балансировки нагрузки.
3. Оптимизация серверной инфраструктуры для более быстрой обработки запросов.
4. Внедрение многофакторной аутентификации и защита от автоматических атак через CAPTCHA.

Проведенные эксперименты и анализ полученных данных подтвердили важность комплексного подхода к защите серверных систем, где необходимо сочетать механизмы защиты от различных типов атак, улучшать серверную инфраструктуру и внедрять современные методы мониторинга и анализа трафика.

Библиографический список:

1. "ARP-spoofing". Википедия. – Режим доступа: https://ru.wikipedia.org/wiki/ARP-spoofing
2. "DDoS Attacks: A Primer". Cloudflare. Режим доступа: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/
3. "Understanding SQL Injection Attacks". Acunetix. – Режим доступа: https://www.acunetix.com/websitesecurity/sql-injection/
4. "Scapy: Packet Manipulation Tool". Официальная документация Scapy. – Режим доступа: https://scapy.readthedocs.io/en/latest/
5. "Requests: HTTP for Humans". Официальная документация Requests. – Режим доступа: https://docs.python-requests.org/en/master/
6. "Python Network Programming". Автор: John Goerzen. – Режим доступа: https://www.oreilly.com/library/view/python-network-programming/9781788621755/

7. "Black Hat Python: Python Programming for Hackers and Pentesters". Автор: Justin Seitz. – Режим доступа: [https://www.no starch.com/black-hat-python] (https://www.no starch.com/black-hat-python)
8. "Wireshark User Guide". Wireshark. – Режим доступа: https://www.wireshark.org/docs/wsug_html_chunked/
9. "Metasploit Unleashed". Offensive Security. – Режим доступа: https://www.offensive-security.com/metasploit-unleashed/
10. "OWASP Top Ten Project". OWASP. – Режим доступа: https://owasp.org/www-project-top-ten/
11. "Hacking: The Art of Exploitation". Автор: Jon Erickson. – Режим доступа: https://www.nostarch.com/hacking2.htm
12. "The Web Application Hacker's Handbook". Автор: Dafydd Stuttard, Marcus Pinto. – Режим доступа: https://www.wiley.com/en-us/The+Web+Application+Hacker%27s+Handbook%3A+Discovering+and+Exploiting+Security+Flaws%2C+2nd+Edition-p-9781118026476
13. "Practical Packet Analysis". Автор: Chris Sanders. Режим доступа: https://www.oreilly.com/library/view/practical-packet-analysis/9781593273037/
14. "The Hacker Playbook 2: Practical Guide To Penetration Testing". Автор: Peter Kim. – Режим доступа: https://www.hackerplaybook.com/
15. "Gray Hat Python: Python Programming for Hackers and Reverse Engineers". Автор: Justin Seitz. – Режим доступа: [https://www.no starch.com/gray-hat-python] (https://www.no starch.com/gray-hat-python)

References:

1. "ARP Spoofing." Wikipedia. – Available at: https://ru.wikipedia.org/wiki/ARP-spoofing
2. "DDoS Attacks: A Primer." Cloudflare. Available at: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/
3. "Understanding SQL Injection Attacks." Acunetix. –/www.acunetix.com/websitesecurity/sql-injection/
4. "Scapy: Packet Manipulation Tool." Official Scapy Documentation. https://scapy.readthedocs.io/en/latest/
5. "Requests: HTTP for Humans." Official Requests Documentation. – Available at: https://docs.python-requests.org/en/master/
6. "Python Network Programming". By John Goerzen. https://www.oreilly.com/library/view/python-network-programming/9781788621755/
7. "Black Hat Python: Python Programming for Hackers and Pentesters." By Justin Seitz. – [https://www.no starch.com/black-hat-python] (https://www.no starch.com/black-hat-python)
8. "Wireshark User Guide." Wireshark. –https://www.wireshark.org/docs/wsug_html_chunked/
9. "Metasploit Unleashed." Offensive Security. –https://www.offensive-security.com/metasploit-unleashed/
10. "OWASP Top Ten Project." OWASP. – Access mode: https://owasp.org/www-project-top-ten/
11. "Hacking: The Art of Exploitation" by Jon Erickson. –https://www.nostarch.com/hacking2.htm
12. "The Web Application Hacker's Handbook" by Dafydd Stuttard, Marcus Pinto. https://www.wiley.com/en-us/The+Web+Application+Hacker%27s+Handbook%3A+Discovering+and+Exploiting+Security+Flaws%2C+2nd+Edition-p-9781118026476
13. "Practical Packet Analysis" by Chris Sanders. https://www.oreilly.com/library/view/practical-packet-analysis/9781593273037/
14. "The Hacker Playbook 2: Practical Guide to Penetration Testing." By Peter Kim. – Available at: https://www.hackerplaybook.com/
15. "Gray Hat Python: Python Programming for Hackers and Reverse Engineers." By Justin Seitz.: [https://www.no starch.com/gray-hat-python] (https://www.no starch.com/gray-hat-python)

Сведения об авторах:
Куликов Алексей Петрович, студент, кафедра «Программная инженерия»; a.k.010@yandex.ru
Радаева Елизавета Андреевна, студент, кафедра «Программная инженерия»; radaeva.liza@list.ru
Якупов Денис Олегович, старший преподаватель, кафедра «Программная инженерия»; d.yakupov@psuti.ru
Information about the authors:
Alexey P. Kulikov, Student, Department of Software Engineering; a.k.010@yandex.ru
Elizaveta A. Radaeva, Student, Department of Software Engineering; radaeva.liza@list.ru
Denis O. Yakupov, Senior Lecturer, Department of Software Engineering, d.yakupov@psuti.ru
Конфликт интересов/Conflict of interest.
Авторы заявляют об отсутствии конфликта интересов/The authors declare no conflict of interest.
Поступила в редакцию/ Received 19.03.2025.
Одобрена после рецензирования/Reviced 17.05.2025.
Принята в печать/Accepted for publication 30.09.2025.