МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по «производственной» практике

Тема: Динамическая автоматизация сканирования периметра для обнаружения уязвимостей с помощью OpenSource инструментов.

Студент гр. 8306 _____ Занин Д.С.

Руководитель _____ Юшкевич.И.А.

Санкт-Петербург

2023

ЗАДАНИЕ

НА «ПРОИЗВОДСТВЕННУЮ» ПРАКТИКУ

Студент Занин Д.С.

Группа 8306

Тема практики: Динамическая автоматизация сканирования периметра для обнаружения уязвимостей с помощью Open Source инструментов.

Задание на практику:

Создание графического интерфейса для программного обеспечения предназначенного для сканирования opensource инструментов. Корректировка очередности вызова инструментов и самих инструментов, обработки результатов выполнения и дальнейшее перенаправления данных для оптимизации работы используемых утилит. Корректировка программного комплекса позволяющего проанализировать домены и/или IP адреса на наличие уязвимостей.

Все задачи были выполнены в соответствии с поставленным планом и календарным графиком.

Сроки прохождения практики: 01.12.2023 – 19.06.2023

Дата сдачи отчета: 20.12.2023

Дата защиты отчета: 27.12.2023

Студент		Занин Д.С.
Руководитель		Юшкевич И.А.

АННОТАЦИЯ

Сканирование сетевого периметра на наличие уязвимостей должно помочь организации найти прорехи в сетевом периметре и не дать злоумышленнику ими воспользоваться. В данной работе предоставляется результат автоматизации по данной этой теме. Демонстрируется разработануый графический интерфейс. Представлена работа тех инструменты, которые рассматривались в обзоре литературы [5] и использовались в данной работе для решения поставленной проблемы, а также прочих инструментов применяемых для создания программного обеспечения.

SUMMARY

Scanning the network perimeter for vulnerabilities is something that should help an organization find gaps in the network perimeter and prevent an attacker from taking advantage of them. This paper provides the result of automation on this topic. The developed graphical interface is demonstrated. The work of those tools that were considered in the literature review [5] and used in this work to solve the problem, as well as other tools used to create software, is presented.

Оглавление

Введение

В современном цифровом мире, где безопасность в сети играет ключевую роль, создание эффективных инструментов для обнаружения уязвимостей на веб-ресурсах является неотъемлемой частью стратегии обеспечения информационной безопасности. Данная работа посвящена разработке графического интерфейса терминального приложения, цель которого – проведение сканирования периметра сети с использованием OpenSource инструментов. Основное внимание уделено созданию интуитивно понятного и функционального графического интерфейса, обеспечивающего удобство использования приложения для технических специалистов и администраторов сети, а также обычных пользователей.

Актуальность - OpenSource инструменты являются терминальными, что создает ряд следующих проблем: доступность и простота использования, повышение точности и уменьшение ошибок, визуализация информации, интеграция с современными технологиями. Создание графического интерфейса упростит разработчикам поиск уязвимостей на их проектах, а также увеличит скорость поиска данных уязвимостей, так как множество инструментов будут собраны в одно приложение и не будет необходимости использовать каждый из них через терминал. Интуитивный интерфейс уменьшает вероятность ошибок, связанных с неправильным вводом команд и параметров. Это особенно важно в области безопасности, где точность важна для успешного обнаружения уязвимостей и предотвращения атак. GUI позволяет визуализировать данные, полученные в результате сканирования сети. Графическое представление уязвимостей и сетевой активности делает анализ более наглядным, что упрощает принятие решений по устранению обнаруженных проблем. Современные пользователи ожидают удобства в использовании и интеграции с другими инструментами. GUI облегчает интеграцию с современными системами управления, интерактивными аналитическеми панелями и платформами мониторинга безопасности.

Проблема — Добавить динамический графический интерфейс в программу автоматизации сканирования периметра для обнаружения уязвимостей.

Цель работы – Основная задача состоит в написании программного продукта, который позволяет автоматизировать сканирование сетевого периметра с использование известных инструментов.

Объект исследования – исследование возможности добавления графического интерфейса в объединенную работу различных OpenSource инструментов для сканирования периметра.

Предмет исследования – Автоматизация, оценка и исследование возможности доступных инструментов для тестирования на проникновение.

Задачи данной работы:

Реализовать графический интерфейс для упрощения взаимодействия с различными терминальными OpenSource инструментами, собранными в одном приложении и предназначенными для сканирования цели на наличие уязвимостей в периметре её сети.

Практическая ценность работы: Результат работы позволит использовать бесплатные, но в тоже время эффективные и гибкие Open Source инструменты для динамической автоматизации сканирования периметра с целью обнаружения уязвимостей используя простой в управлении графический интерфейс, которым смогут воспользоваться и обычные пользователи программного обеспечения, а не только специально обученные специалисты.

Результат работы в осеннем семестре

1. Исследование OpenSource инструментов, которыми можно заменить Amass.

В прошлой версии программы использовался Amass в качестве сканера доменного имени цели на наличие под-доменов для составления карты сети, но, в связи с вышедшим обновлением данного инструмента, изменился вывод работы программы, что повлекло за собой ошибки в работе всего проекта. В связи с этим было принято заменить данный инструмент на один из аналогов:

1. Subfinder
2. Assetfinder
3. DNSenum
4. Aquatone
5. Recon-ng

Из данных инструментов не подходят для данного проекта Recon-ng так как порог входа довольно велик. В связи с этим в ходе данной работы были рассмотрены Subfinder, DNSenum и Aquatone.

1.1. DNSenum

Проблемы, которые могут возникнуть:

- Ограниченная функциональность: DNSenum предназначен в основном для перечисления DNS и может не иметь некоторых более продвинутых функций, доступных в других инструментах разведки.
- Зависимость от сторонних сервисов: Некоторые функции могут зависеть от доступности и отклика сторонних DNS-сервисов и Whois-серверов, что может ограничивать его эффективность.
- Обновления и поддержка: Если инструмент не обновляется авторами или сообществом, он может стать устаревшим и менее эффективным против современных мер безопасности.

1.2. Aquatone

Возможные сложности:

- Требования к ресурсам: Захват скриншотов веб-сайтов может быть

ресурсоемким процессом, особенно если сканируется большое количество хостов.

- Блокировка по IP: Интенсивное сканирование с одного IP-адреса может привести к блокировке этого IP со стороны некоторых веб-сайтов или сетевых инфраструктур.
- Сложность настройки: Aquatone может потребовать дополнительной настройки, например, для работы с прокси-серверами или для тонкой настройки времени ожидания запросов.

1.3. Subfinder

Имеет ряд отличительных преимуществ:

- Широкий охват: Subfinder агрегирует данные из множества источников, что позволяет находить большое количество поддоменов.
- Производительность: Он оптимизирован для быстрой работы и способен обрабатывать большие объемы данных без значительного ущерба для скорости.
- Простота использования: Subfinder имеет простой и понятный интерфейс командной строки, который упрощает его использование для специалистов в области безопасности.
- Интеграция с другими инструментами: Subfinder может быть легко интегрирован с другими инструментами и фреймворками для дальнейшего анализа и автоматизации процессов.
- Активное сообщество: Subfinder поддерживается сообществом и часто обновляется, что обеспечивает добавление новых функций и поддержку современных технологий.

1.4. Результат исследования.

После подробного изучения аналогов использованного ранее инструмента Amass, было принято решение использовать Subfinder, так как он дастаточно быстр. Подробнее об этом смотреть в пункте 2. Остальные два инструмента имели ряд недостатков, которые мешают их использованию в данном проекте, а именно Зависимость от сторонних сервисов, сложность настройки и блокировка по IP.

2. Внедрение OpenSource инструмента Subfinder

В отличие от Subfinder, Amass, несмотря на свою эффективность, имеет ограниченные источники данных, что снижает охват собираемой информации.

Внедрение Subfinder в проект оправдано стремлением к повышению эффективности и расширению возможностей сканирования периметра сети. Этот шаг обеспечивает проекту надежный и актуальный инструмент для обнаружения уязвимостей в сетевом окружении, согласованный с принципами открытого исходного кода и активной поддержкой сообщества.

Subfinder предоставляет поддержку многопоточности, что позволяет выполнять параллельное сканирование множества доменов. Это значительно увеличивает скорость сбора информации о доменных именах в сравнении с более линейным подходом Amass. Эффект повышения скорости работы алгоритма является значимым шагом в направлении оптимизации процессов обнаружения уязвимостей. Этот переход обеспечивает не только улучшенные характеристики скорости, но также более эффективное использование ресурсов, что содействует общей эффективности проекта.

2.1. Исследование скорости работы

Для краткого сравнения эффективности замены Amass на Subfinder были проведены контрольные замеры скорости сканирования. Полученные данные выявили заметное ускорение процесса сбора информации о доменах, снижение времени выполнения алгоритма и повышение общей эффективности.

Amass в пасивном режиме справился с задачей сканирования за 3 минуты 58 секунд.

Subfinder справился с поставленной задачей сканирования за 6 - 7 секунд.

$$\frac{3 * 60 + 58}{7} = 34$$

Прирост в скорости 3 мин 51 секунда, что быстрее в 34 раза.

Таким образом процесс сбора поддоменов целевого домена значительно был ускорен и успешно внедрён в процесс динамического автоматизированного сканирования цели.

3. Исследование библиотек и инструментов для создания графического интерфейса

Для создания графического интерфейса (GUI) с целью объединения OpenSource инструментов в единый рабочий процесс, были исследованы следующие инструменты и библиотеки. Вот некоторые из них:

1. Qt: является мощным фреймворком для создания кроссплатформенных приложений с графическим интерфейсом. Он поддерживает множество языков программирования, включая C++ и Python.
2. Electron: позволяет создавать настольные приложения с использованием веб-технологий, таких как HTML, CSS и JavaScript. Это может быть полезно, если необходимо интегрировать веб-интерфейсы для OpenSource инструментов.
3. Библиотеки PyQt или Tkinter: использует Python. PyQt (Qt для Python) и Tkinter являются отличными инструментами для создания графического интерфейса.
4. Vue.js или React: Если проект ориентирован на веб-технологии. Эти фреймворки предоставляют множество инструментов для построения интерактивных веб-приложений.
5. Создание веб-интерфейса с использованием фреймворков вроде Flask (Python), Express (Node.js) или Django (Python).

Выбор конкретного инструмента зависит от поставлнной задачи, предпочтений, языка программирования, платформы и требований проекта.

React, Qt, PyQt являются слишком сложными, и имеют ограничения в плане внешнего вида и функциональности. Tkinter имеет ограничения в плане внешнего вида и функциональности. Использование данных инструментов затруднено и является избыточным для данной работы.

3.1. Electron

Electron - это кроссплатформенный фреймворк, который позволяет разработчикам создавать настольные приложения с использованием веб-технологий, таких как HTML, CSS и JavaScript. Он объединяет Chromium (для отображения веб-содержимого) и Node.js (для работы с backend-частью и файловой

системой), позволяя разработчикам использовать один и тот же код для создания приложений для Windows, macOS и Linux.

Минусы:

1. Большой размер приложения: Поскольку Electron включает в себя копии Chromium и Node.js, базовый размер дистрибутива приложения может быть значительным даже для простых приложений. Это может стать проблемой для пользователей с ограниченным дисковым пространством или при распространении приложения через сети с ограниченной пропускной способностью.

2. Потребление ресурсов: Electron-приложения часто критикуют за высокое потребление оперативной памяти и процессорного времени, особенно если сравнивать с нативными приложениями. Это связано с тем, что каждое Electron-приложение запускает свою собственную копию Chromium и Node.js.

3. Безопасность: Будучи основанным на веб-технологиях, Electron наследует потенциальные уязвимости веб-браузеров. Разработчикам приложений необходимо внимательно следить за обновлениями безопасности и своевременно обновлять свои приложения.

4. Производительность: В силу архитектуры Electron и его зависимости от веб-технологий, приложения, созданные на Electron, могут работать медленнее, чем их нативные аналоги, особенно в части графической производительности и анимаций.

5. Зависимость от веб-технологий: Разработчики должны иметь опыт работы с веб-технологиями, чтобы эффективно использовать Electron. Это может быть недостатком для команд, специализирующихся на разработке нативных приложений.

6. Обновления: Electron-приложения не обновляются так же автоматически, как нативные приложения или веб-приложения. Разработчики должны внедрять механизмы обновлений в свои приложения и управлять ими.

3.2. Vue.js

Несмотря на множество преимуществ, таких как легкость в освоении, гибкая система и быстрая отзывчивость, у Vue.js есть и свои недостатки:

1. Меньше ресурсов и поддержка сообщества: По сравнению с такими гигантами, как React и Angular, у Vue.js меньше сообщество, хотя оно быстро растет. Это может сказаться на доступности учебных материалов, плагинов и сторонних библиотек.

2. Риск переусложнения: Из-за гибкости Vue.js легко начать переусложнять приложения, создавая слишком много настраиваемых директив и плагинов, что может затруднить поддержку и масштабирование проекта.

3. Изменения в версиях: Vue 3 принес множество изменений по сравнению с Vue 2, включая новый API для композиции. Это может вызвать проблемы при обновлении уже существующих проектов и требует дополнительного времени на переобучение разработчиков.

4. Интеграция с большими командами: Для команд, которые привыкли работать с более строгими фреймворками вроде Angular, переход на Vue может вызвать трудности из-за его гибкости и декларативного стиля программирования.

5. SEO: В отличие от серверно-рендерируемых приложений, одностраничные приложения (SPA) на Vue.js могут иметь проблемы с индексацией поисковыми системами, хотя это улучшается с развитием технологий и можно использовать SSR (Server-Side Rendering) или решения типа Nuxt.js для улучшения SEO.

6. Зависимость от экосистемы: По мере роста проекта может возникнуть зависимость от специфических Vue-плагинов и библиотек, что потенциально может привести к затруднениям, если эти инструменты перестанут поддерживаться.

3.3. Flask (Python):

- Минимализм: Flask является микрофреймворком, что значит, что он предоставляет лишь базовый функционал для веб-разработки. Для более продвинутых функций потребуется использование дополнительных расширений, что может усложнить проект.
- Масштабирование: Маленькие и простые проекты на Flask легко запустить.
- Асинхронная обработка: Flask не предназначен для асинхронной

обработки из коробки, что может быть недостатком для приложений, требующих высокой производительности и параллельной обработки запросов.

3.4. Express (Node.js):

- Производительность: Хотя Node.js известен своей способностью к обработке большого количества одновременных соединений, CPU-интенсивные задачи могут заблокировать основной поток, что снижает производительность.
- "Callback Hell": При использовании асинхронных callback-функций код может стать трудночитаемым. Хотя это проблема может быть решена с помощью Promises и async/await, она все еще остается важной для старых проектов.
- Структура и организация кода: Express предоставляет много свободы в структурировании приложения, что может привести к проблемам с поддержкой и масштабированием, особенно для новичков.

3.5. Django (Python):

- Монолитность: Django предлагает много встроенных функций, что делает его отличным выбором для быстрого старта проектов. Однако это может быть ограничивающим фактором, если требуется больше контроля или если проект требует нестандартных решений.
- Кривая обучения: Django имеет относительно крутую кривую обучения по сравнению с более легкими фреймворками вроде Flask, особенно для новичков в веб-разработке.
- Весомость: Django может быть перегружен для маленьких проектов, где многие из его функций не будут использоваться.
- ORM: ORM Django очень удобен, но может быть менее гибким в сравнении с более низкоуровневыми решениями доступа к данным. Это может ограничить производительность и контроль над запросами к базам данных для сложных приложений.

4. Создание графического интерфейса

Наиболее предпочтительным языком программирования ввиду личных предпочтений является python. Выбор был сделан в пользу Flask из-за простоты его использования и простоты самого web-приложения выполняющего лишь роль графического интерфейса, а также ввиду использования python.

4.1. HTML-страница графического интерфейса

Рассмотрим функции, которые используют XMLHttpRequest для взаимодействия с сервером:

1. **sendAll():** Эта функция вызывает три другие функции (**sendDomainName()**, **sendVulnerabilityTags()**, **sendSeverity()**), чтобы отправить данные о домене, выбранных тегах уязвимостей и уровне критичности на сервер.

- **Шаги:**

1. Вызывает **sendDomainName()** для отправки данных о домене.
2. Вызывает **sendVulnerabilityTags()** для отправки данных о выбранных тегах уязвимостей.
3. Вызывает **sendSeverity()** для отправки данных об уровне критичности.

2. **sendDomainName():** Эта функция собирает значение из текстового поля с идентификатором 'domain_name' и отправляет его на сервер методом POST.

- **Шаги:**

1. Получает значение из текстового поля с идентификатором 'domain_name'.
2. Создает объект XMLHttpRequest.
3. Открывает соединение с сервером методом POST и указывает конечную точку '/create_domain_file'.
4. Устанавливает заголовок 'Content-Type' в 'text/plain'.
5. Отправляет значение на сервер.

3. **sendVulnerabilityTags():** Эта функция собирает значения выбранных чекбоксов для тегов уязвимостей, формирует строку и отправляет её на сервер методом POST.

- **Шаги:**

1. Создает массив идентификаторов чекбоксов тегов.
2. Для каждого идентификатора чекбокса проверяет, выбран ли он.
3. Если выбран, добавляет значение с заглавной буквой в массив.
4. Создает объект XMLHttpRequest.
5. Открывает соединение с сервером методом POST и указывает конечную точку '/create_tags_file'.
6. Устанавливает заголовок 'Content-Type' в 'text/plain'.
7. Отправляет сформированную строку на сервер.

4. **sendSeverity():** Эта функция аналогична **sendVulnerabilityTags()**, но работает с чекбоксами для уровня критичности.

- **Шаги:**

1. Создает массив идентификаторов чекбоксов уровня критичности.
2. Для каждого идентификатора чекбокса проверяет, выбран ли он.
3. Если выбран, добавляет значение с заглавной буквой в массив.
4. Создает объект XMLHttpRequest.
5. Открывает соединение с сервером методом POST и указывает конечную точку '/create_severity_file'.
6. Устанавливает заголовок 'Content-Type' в 'text/plain'.
7. Отправляет сформированную строку на сервер.

5. **startScan():** Эта функция отправляет запрос на сервер для запуска сканирования без передачи каких-либо данных.

- **Шаги:**

1. Создает объект XMLHttpRequest.
2. Открывает соединение с сервером методом POST и указывает конечную точку '/start_scan_command'.
3. Отправляет пустое тело запроса на сервер.

4.2. Использование фреймворка Flask для создания сервера

В ходе данного проекта будет реализован лишь минимальный графический интерфейс для взаимодействия с сервером, позволяя передавать параметры,

сохранять их в файлах, и запускать процесс сканирования.

1. **Импорты:**

- **from flask import Flask, request, send_from_directory**: Импорт необходимых модулей Flask.
- **import os**: Импорт модуля для взаимодействия с операционной системой.

2. **Инициализация Flask:**

- **app = Flask(__name__, static_folder='static')**: Создание объекта Flask. **static_folder='static'** указывает, что статические файлы (например, CSS, JS) будут храниться в папке 'static'.

3. **Маршруты:**

- **@app.route('/', methods=['GET'])**: Маршрут для обработки GET-запросов к корневому URL.
- **send_from_directory(app.static_folder, 'index.html')**: Отправка файла 'index.html' из папки 'static'.
- **@app.route('/create_domain_file', methods=['POST'])**: Маршрут для обработки POST-запросов по адресу '/create_domain_file'.
- **text = request.data.decode('utf-8')**: Извлечение данных из тела запроса и декодирование в текст.
- **with open('./automated_scanning/domain_name.txt', 'w') as file: ...**: Запись полученного текста в файл 'domain_name.txt'.
- **@app.route('/create_tags_file', methods=['POST'])**: Маршрут для обработки POST-запросов по адресу '/create_tags_file'.
- **tags = request.data.decode('utf-8')**: Извлечение данных из тела запроса и декодирование в текст.
- **with open('./automated_scanning/tags_vulnerabilities.txt', 'w') as file: ...**: Запись полученного текста в файл 'tags_vulnerabilities.txt'.
- **@app.route('/create_severity_file', methods=['POST'])**: Маршрут для обработки POST-запросов по адресу '/create_severity_file'.
- **severity = request.data.decode('utf-8')**: Извлечение данных из тела

запроса и декодирование в текст.

- **with open('./automated_scanning/severity_vulnerabilities.txt', 'w') as file: ...**: Запись полученного текста в файл 'severity_vulnerabilities.txt'.
- **@app.route('/start_scan_command', methods=['POST'])**: Маршрут для обработки POST-запросов по адресу '/start_scan_command'.
- **os.system('start_scan_target.bat')**: Запуск команды для запуска сканирования. Процесс может быть запущен с использованием пакета **os**.

4. **Обработка ошибок:**

- **try: ... except Exception as e: ...**: Блок для обработки исключений при запуске сканирования. В случае ошибки возвращается код состояния 500 (внутренняя ошибка сервера) с описанием ошибки.

5. **Запуск приложения:**

- **if __name__ == '__main__': ... app.run()**: Условие для запуска приложения, если файл запускается напрямую, а не импортируется. Метод **app.run()** запускает веб-сервер Flask.

4.3. Визуальный результат

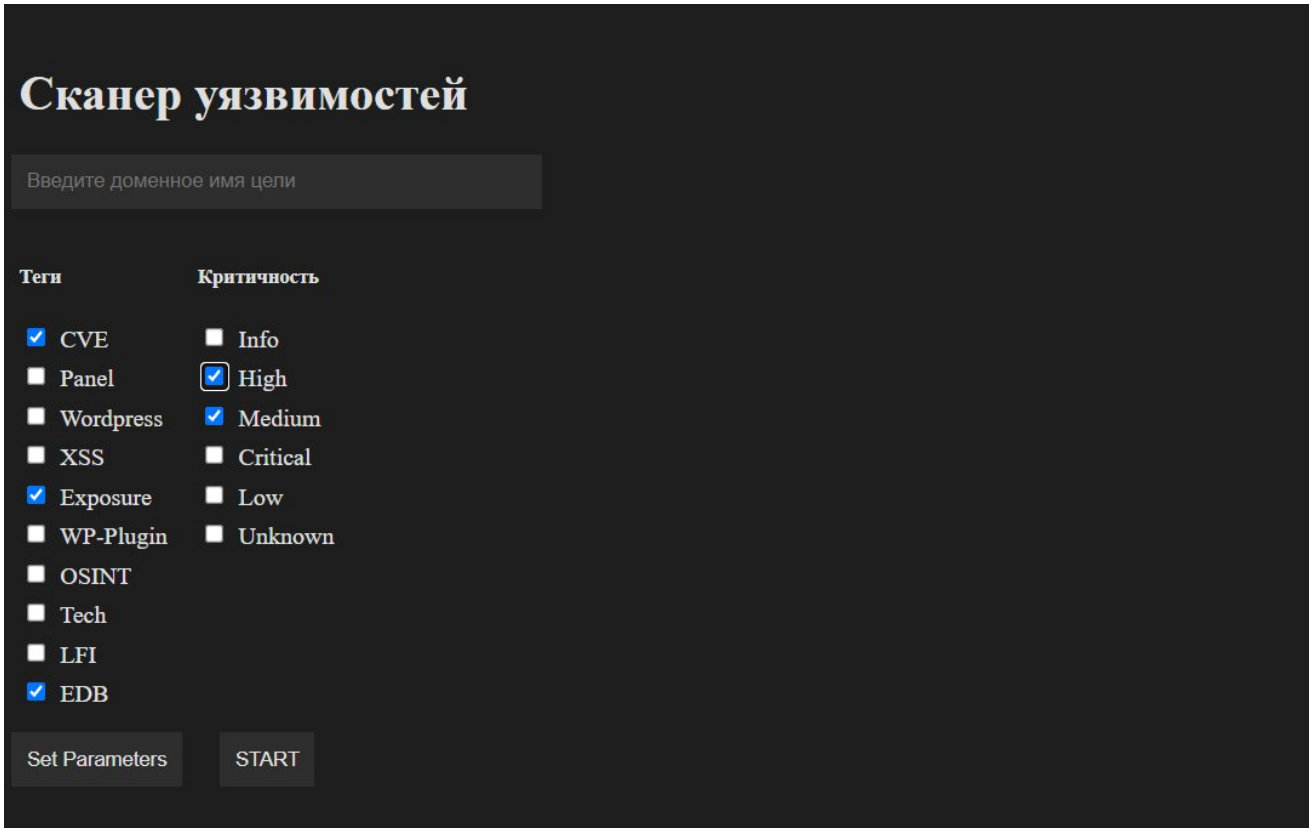

Рис. 1. Графический интерфейс

Рассмотрим смысл использованных тегов в графическом интерфейсе:

1. CVE: Common Vulnerabilities and Exposures - стандартизированный индекс для общедоступной информации о безопасности (уязвимостях).

2. Panel: Административная панель или пользовательский интерфейс для управления приложением или службой, часто является целью для атак, направленных на получение несанкционированного доступа.

3. WordPress: Система управления содержимым (CMS), которая часто целится в связи с её популярностью и обилием плагинов, которые могут содержать уязвимости.

4. XSS: Cross-Site Scripting - тип атаки, при которой злоумышленник встраивает вредоносные скрипты в контент веб-страниц, которые затем выполняются в браузере жертвы.

5. Exposure: Экспозиция, обнаружение или "выставление на показ" чувствительной информации, конфигураций или сервисов, которые должны быть скрытыми от публичного доступа.

6. WP-Plugin: Плагин для WordPress, который может содержать уязвимости. Nuclei может сканировать плагины на наличие известных проблем безопасности.

7. OSINT: Open Source Intelligence - сбор и анализ информации из открытых источников для разведывательных или безопасностных целей.

8. Tech: Технологии или технические компоненты, такие как серверы, базы данных, фреймворки, которые могут иметь уязвимости, поддерживаемые в базе данных Nuclei или других инструментах для сканирования.

9. LFI: Local File Inclusion - тип уязвимости, при котором атакующий может включать файлы, находящиеся на сервере, в вывод страницы, что может привести к раскрытию чувствительной информации или выполнению вредоносного кода.

10. EDB: Exploit Database - база данных, содержащая информацию об уязвимостях и эксплойтах, которые могут быть использованы для их эксплуатации.

11. Nuclei использует шаблоны для сканирования различных целей на

наличие этих и многих других уязвимостей, используя информацию из различных источников, включая CVE и Exploit Database.

В контексте информационной безопасности и управления уязвимостями, термины "info", "high", "medium", "critical", "low", и "unknown" обычно используются для классификации уровня серьезности уязвимостей или оповещений безопасности. Вот их общий смысл:

1. Critical (критический): Уязвимости с этим уровнем серьезности представляют непосредственную и серьезную угрозу безопасности системы и могут позволить злоумышленнику выполнить произвольный код, получить полный контроль над системой, украсть конфиденциальные данные или выполнить другие действия с высоким уровнем воздействия без значительных препятствий.

2. High (высокий): Уязвимости с высоким уровнем серьезности также представляют серьезную угрозу безопасности, но могут потребовать от злоумышленника выполнения более сложных действий или наличия специфических условий для эксплуатации.

3. Medium (средний): Уязвимости со средним уровнем серьезности имеют ограниченный потенциал воздействия и могут потребовать от злоумышленника выполнения специализированных или сложных действий для их эксплуатации.

4. Low (низкий): Уязвимости с низким уровнем серьезности считаются относительно безопасными или маловероятными для эксплуатации. Они могут иметь минимальное воздействие на безопасность системы и являются наименьшим приоритетом для устранения.

5. Unknown (неизвестный): Этот уровень означает, что уровень серьезности уязвимости неизвестен. Может быть, что недостаточно информации для оценки воздействия уязвимости или она не была должным образом проанализирована.

6. Info (информационный): Это не указывает на наличие уязвимости, а скорее предоставляет информацию, которая может быть полезна для понимания состояния безопасности системы. Например, это может включать подробности о конфигурации системы или предупреждения о незначительных недостатках,

которые не влияют на безопасность.

Эти уровни используются в различных системах управления уязвимостями, включая инструменты сканирования и базы данных уязвимостей, для помощи администраторам и специалистам по безопасности в приоритизации устранения уязвимостей и решения проблем безопасности в соответствии с их потенциальным воздействием на организацию.

План работ на весенний семестр

- Доработка графического интерфейса и увеличение количества используемых OpenSource инструментов.
- Реализация сохранения полученных результатов в отдельном файле с понятным оформлением и статистикой
- Реализация кроссплатформенного графического интерфейса

Заключение

При прохождении практики, было написано программное обеспечение по теме практики, с использованием Open Source инструменты для решения поставленной проблемы. Цель практики достигнута и результаты этой учебной практики впоследствии будут использованы при написании магистерской диссертации.

Список использованных источников

1. Быстрый и настраиваемый сканер уязвимостей на основе простого DSL на основе YAML, Nuclei Templates // DSL URL: https://github.com/projectdiscovery/nuclei-templates (дата обращения: 30.06.23).

2. Docker образ kali-rolling // DSL URL: https://hub.docker.com/r/kalilinux/kali-rolling (дата обращения: 30.06.23).

3. Ссылка на репозиторий GitHub с созданным приложением, для сканирования сетевого периметра на наличие уязвимостей // DSL URL: https://github.com/projectdiscovery/nuclei-templates (дата обращения: 30.06.23).

4. Уязвимые тестовые веб-сайты для сканера веб-уязвимостей Acunetix. // DSL URL: docker-compose run docker_scan

5. Отчёт обзора литературы по теме данной работы, Занина Д.С. // DSL URL: https://disk.yandex.ru/d/QZ_aqxSoY6q_qg

Приложение А

Ссылка на полный код проекта:

https://github.com/JackoPrograms/automated_scanning.git