Санкт-Петербургский политехнический университет Петра Великого

Институт кибербезопасности и защиты информации

**КУРСОВАЯ РАБОТА**

**Исследование методов анализа безопасности веб-сайтов**

по дисциплине «Технологии реверс-инжиниринга программного обеспечения»

Выполнил

студент гр. 4851003/00802 <*подпись*> Голуб В.В.

Руководитель

ст. преподаватель <*подпись*>

Санкт-Петербург

2022

Оглавление

[1. ВВЕДЕНИЕ 3](#_Toc128334795)

[2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4](#_Toc128334796)

[3. РЕЗУЛЬТАТ РАБОТЫ 5](#_Toc128334797)

[3.1. Угрозы. 5](#_Toc128334798)

[3.1.1. A1:2021 Ошибки контроля доступа. 5](#_Toc128334803)

[3.1.2. A2:2021 Криптографические сбои. 5](#_Toc128334804)

[3.1.3. A3:2021 Внедрение кода(Инъекции). 5](#_Toc128334805)

[3.1.4. A4:2021 Небезопасный дизайн. 6](#_Toc128334806)

[3.1.5. A5:2021 Неправильные настройки безопасности 6](#_Toc128334807)

[3.1.6. А6:2021 Устаревшие компоненты 6](#_Toc128334808)

[3.1.7. А7:2021 Ошибки идентификации и аутентификации. 6](#_Toc128334809)

[3.1.8. А8:2021 Cбои программного обеспечения и целостности данных 7](#_Toc128334810)

[3.1.9. А9:2021 Регистрация безопасности и мониторинг сбоев 7](#_Toc128334811)

[3.1.10. A10:2021 Подделка запросов на стороне сервера(Атака SSRF) 7](#_Toc128334812)

[3.2. Виды атак. 8](#_Toc128334813)

[3.2.1. Подделка серверных запросов(SSRF) 11](#_Toc128334814)

[3.2.2. Http request smuggling. 14](#_Toc128334815)

[3.3. Исследование сканеров веб-уязвимостей 18](#_Toc128334816)

[3.3.1. OWASP ZAP 18](#_Toc128334817)

[Таблица 1 – Уязвимости найденные OWASP ZAP 22](#_Toc128334818)

[3.3.2. Rapid7 insightAppSec 23](#_Toc128334819)

[Таблица 2 – Уязвимости найденные Rapid7 30](#_Toc128334820)

[3.3.3. Burp Suite Professional 31](#_Toc128334821)

[Таблица 3 – Уязвимости найденные Burp Suite 37](#_Toc128334822)

[*3.3.4.* *Сравнение веб-сканеров* 37](#_Toc128334823)

[3.4. Разработка плагина для веб-сканера Burp Suite 38](#_Toc128334824)

[СПИСОК ИСТОЧНИКОВ 39](#_Toc128334825)

[ПРИЛОЖЕНИЕ А 41](#_Toc128334826)

# ВВЕДЕНИЕ

**Цели курсовой работы:**

1. Анализ статистики OWASP
2. Анализ существующих сканеров веб-уязвимостей
3. Написание своего плагина для сканера

В настоящее время отрасль веб-разработки развивается невероятно быстро. Даже небольшие частные компании на сегодняшний день не могут обойтись без своего веб-приложения или веб-сайта, не говоря уже о больших корпорациях или государственных компаниях. Но с появлением новых веб-технологий, фреймворков, веб-приложений, веб-сайтов растет количество уязвимостей и злоумышленников в данной сфере. Из этого можно сделать вывод, что тема веб-безопасности актуальна и будет оставаться таковой очень долгое время.

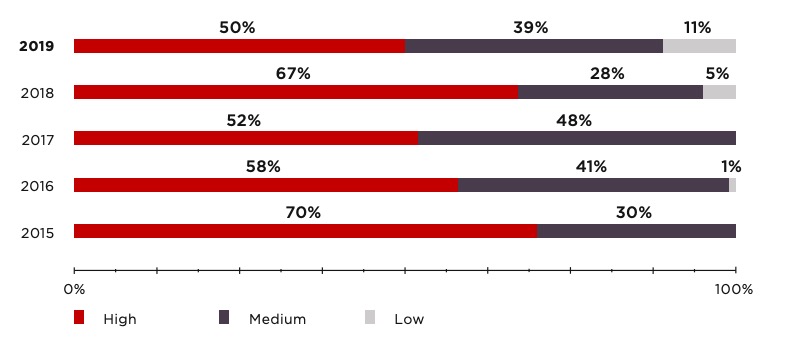


Рисунок 1 – Статистика Positive Technologies. Защищенность веб-приложений.

# ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Веб-приложения — это клиент-серверные приложения, основанные на протоколе HTTP, которые обычно управляют данными содержимого HTML, JavaScript, JSON и XML. При разработке веб-приложений разработчики больше внимания уделяют обычно на обеспечение требуемой функциональности. А вопросам безопасности и качества программного кода уделяется недостаточно внимания. В результате подавляющее большинство веб-приложений содержит уязвимости различной степени критичности.

Уязвимости веб-приложений связаны с системным недостатком или слабостью веб-приложения. Они существуют уже много лет, в основном из-за отсутствия проверки старого кода, неправильной настройки веб-серверов и недостатков дизайна веб-приложений, и их можно использовать для нарушения безопасности приложения. Эти уязвимости отличаются от других распространенных типов уязвимостей. Они возникают из-за того, что веб-приложениям необходимо взаимодействовать с миллионами пользователей в нескольких сетях, и хакеры легко используют этот уровень доступности.

Существуют решения разработанные специально для обеспечения безопасности именно веб-приложений, и поэтому важно не ограничиваться традиционными механическими способами обнаружения веб-уязвимостей, когда дело доходит до выявления брешей в безопасности приложений больших организации.

Такие решения называются веб-сканерами. Они автоматизируют процесс отслеживания уязвимостей. А при должной настройке могут сразу и эксплуатировать найденную уязвимость.

# РЕЗУЛЬТАТ РАБОТЫ

Open Web Application Security Project(OWASP) - Открытый проект по обеспечению безопасности веб-приложений. Раз в четыре года происходит составление рейтинга 10 самых опасных уязвимостей веб-приложений. Рейтинг составлен путем сбора информации, от сотен организаций и более 100 000 приложений и API.

## Угрозы.

Ниже представлен список десяти наиболее опасных и распространённых уязвимостей по результатам исследования OWASP:



### A1:2021 Ошибки контроля доступа.

Наличие подобных уязвимостей в веб-приложении может стать причиной несанкционированного доступа к конфиденциальной информации, модификации или удаления данных, а также несанкционированного доступа к личному кабинету или функциональности приложения.

### A2:2021 Криптографические сбои.

Криптографические ошибки, которые служат причиной раскрытием конфиденциальных данных. К примеру, приложение может надежно зашифровать данные кредитной карты с помощью автоматического шифрования базы данных. К сожалению, при доступе к этой информации она сразу же расшифровывается, что позволяет сбою SQL-инъекции извлекать информацию о кредитной карте в открытом виде, чем может воспользоваться злоумышленник.

Примеры раскрытия конфиденциальной информации:

* Токены сеанса
* Идентификаторы входа и пароли
* Онлайн-транзакции

### A3:2021 Внедрение кода(Инъекции).

Инъекционные недостатки как SQL, NoSQL, OS и LDAP, возникают, когда ненадежные данные отправляются интерпретатору как часть команды или запроса. Враждебные данные атакующего могут обмануть интерпретатора в выполнение непреднамеренных команд или доступ к данным без надлежащей авторизации.

### A4:2021 Небезопасный дизайн.

Небезопасный дизайн означает риски, связанные с недостатками дизайна и архитектуры, которые заложены с самого начала разработки программного обеспечения, если не будут приняты соответствующие меры безопасности. Одним из факторов, способствующих небезопасному проектированию, является отсутствие профилирования бизнес-рисков, присущего разрабатываемому программному обеспечению или системе.

### A5:2021 Неправильные настройки безопасности

Наиболее часто встречающаяся проблема – это неправильная конфигурация системы. Обычно это результат небезопасных конфигураций по умолчанию, неполных или специальных конфигураций, открытого облачного хранилища, неправильно сконфигурированных заголовков HTTP и подробных сообщений об ошибках, содержащих конфиденциальную информацию. Мало того, что все операционные системы, фреймворки, библиотеки и приложения должны быть надежно настроены, но они должны быть исправлены / обновлены своевременно.

### А6:2021 Устаревшие компоненты

Уязвимости интернет-приложений, заключающиеся в содержании ошибок и уязвимостей сторонних фреймворках, которые могут приводить к нежелытельным результатам, нарушениям контроля целостности, SQL-инъекции.

### А7:2021 Ошибки идентификации и аутентификации.

Говоря об этой категории, имеются ввиду проблемы, когда злоумышленник получает информацию сеансов идентификации, учетных данных пользователя. Если приложение допускает автоматизированные атаки, такие как подмена учетных данных — когда злоумышленник имеет доступ к спискам реальных пользователей и паролей — или предопределенным, более слабым и распространенным паролям, таким как «Password1» или «admin/admin», это может быть признаком недостатков аутентификации.

### А8:2021 Cбои программного обеспечения и целостности данных

Это новая категория, и она направлена на обеспечение целостности обновлений программного обеспечения, жизненно важных данных и процедур CI/CD без их проверки. Одним из примеров является использование приложениями расширений, модулей или репозиториев из сетей доставки контента (CDN) или из неавторизованных источников. Незащищенный процесс непрерывной интеграции/непрерывной доставки ( CI/CD ) может повысить риск вредоносного кода, компрометации системы или несанкционированного доступа.

### А9:2021 Регистрация безопасности и мониторинг сбоев

Отсутствие отслеживания при наличии подозрительных действий и происшествий может увеличить промежутки времени, которые остаются неотслеживаемыми, позволяя нарушениям безопасности оставаться незамеченными дольше, чем при лучшем ведении журналов. Этот раздел OWASP Top 10 2021 предназначен для помощи в выявлении, эскалации и устранении недавних нарушений. Обнаружение нарушения безопасности маловероятно без записи и мониторинга.

### A10:2021 Подделка запросов на стороне сервера(Атака SSRF)

Результаты для этой категории показывают охват тестированием выше среднего, достаточно низкий уровень заболеваемости и рейтинги воздействия и эксплуатации выше среднего. SSRF развивается, когда запросы на стороне сервера выполняются без проверки URL-адреса, указанного пользователем. Это позволяет злоумышленнику заставить приложение передать поддельный запрос в нежелательное место, даже если оно защищено виртуальными частными сетями (VPN), брандмауэрами или списком контроля доступа к сети (ACL). Получение URL-адреса стало обычным явлением, поскольку новые онлайн-приложения предоставляют конечным пользователям удобные функции. Следовательно, распространенность SSRF увеличивается. Кроме того, интенсивность SSRF растет из-за облачных сервисов и сложности проектирования.

## Виды атак.

Атаки на web-приложения крайне распространены. Согласно исследованию Forrester, 39% всех атак разрабатывались для использования уязвимостей в веб-приложениях. Данный вид кибератак остается самым частым типом атак.

Ниже представлены основные способы осуществления внешних атак (по отчету Forrester за 2021 год добавить в ссылки).



Существует множество различных атак на веб-приложения, и с каждым годом их становится всё больше и больше. В данной главе рассмотрены некоторые виды сетевых атак. Проведено детальное рассмотрение каждой из атак и описаны способы защиты.

* SQL-инъекция — это атака, направленная на веб-приложение, в ходе которой конструируется SQL-выражение из пользовательского ввода путем простой конкатенации (например, $query="SELECT \* FROM users WHERE id=".$\_REQUEST["id"]). В случае успеха атакующий может изменить логику выполнения SQL-запроса так, как это ему нужно. Чаще всего он выполняет простой fingerprinting СУБД, а также извлекает таблицы с наиболее «интересными» именами (например, «users»). После этого, в зависимости от привилегий, с которыми запущено уязвимое приложение, он может обратиться к защищенным частям бэк-энда веб-приложения (например, прочитать файлы на стороне хоста или выполнить произвольные команды). Есть пять основных классов SQL-инъекций, и все их поддерживает sqlmap:
  + UNION query SQL injection. Классический вариант внедрения SQL-кода, когда в уязвимый параметр передается выражение, начинающееся с «UNION ALL SELECT».
  + Error-based SQL injection. В случае этой атаки сканер заменяет или добавляет в уязвимый параметр синтаксически неправильное выражение, после чего парсит HTTP-ответ (заголовки и тело) в поиске ошибок DBMS, в которых содержалась бы заранее известная инъецированная последовательность символов и где-то «рядом» вывод на интересующий подзапрос.
  + Stacked queries SQL injection. Сканер проверяет, поддерживает ли вебприложение последовательные запросы, и, если они выполняются, добавляет в уязвимый параметр HTTP-запроса точку с запятой (;) и следом внедряемый SQLзапрос.
  + Boolean-based blind SQL injection. Реализация так называемой слепой инъекции: данные из БД в «чистом» виде уязвимым веб-приложением нигде не возвращаются.
  + Time-based blind SQL injection. Полностью слепая инъекция.
* XSS (Cross-Site Scripting — «межсайтовый скриптинг») — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «внедрение кода». Специфика подобных атак заключается в том, что вредоносный код может использовать авторизацию пользователя в веб-системе для получения к ней расширенного доступа или для получения данных пользователя. Вредоносный код может быть вставлен в страницу как через уязвимость в веб-сервере, так и через уязвимость на компьютере пользователя
* Cross-Site Request Forgery (CSRF) (Подделка межсайтовых запросов). Атака CSRF может происходить, когда злоумышленник находит воспроизводимую ссылку, которая выполняет определенное действие на сервере, когда жертва находится там. Злоумышленник может встроить такую ссылку, как ссылки на изображения или ссылки на спам по электронной почте, и заставить жертву открыть ее, заставив выполнять определенные действия без согласия пользователя. Пример: злоумышленник находит следующую ссылку, которая переводит средства учетной записи пользователя, когда пользователь правильно вошел в систему 243 Злоумышленник может встроить ссылку в вредоносные сайты или спам-сообщения и обманным путем заставить законного пользователя получить доступ к этим ссылкам, пока жертва остается в системе. Злонамеренное действие будет выполнено без согласия пользователя.
* CSRF работает, когда злоумышленник **получает доступ к браузеру жертвы** — обычно через вредоносную ссылку. Затем этот доступ используется для выполнения вредоносного запроса к любому приложению с активным сеансом, в котором пользователь аутентифицирован. Поскольку у злоумышленника есть доступ к браузеру, запросы приложений считаются законными, поскольку они отправляются вместе с файлом cookie сеанса, который позволяет выполнять запрос.  Чтобы CSRF имел эффект и имел смысл, **должны присутствовать несколько условий** :
  + Должно быть действие, которое злоумышленник хочет выполнить от имени жертвы, которое предоставит злоумышленнику некоторую выгоду (например, получить доступ и изменить пароль жертвы).
  + Приложение должно **использовать файлы cookie сеанса** для аутентификации запроса, сделанного пользователем.
  + Злоумышленник **знает параметры** запросов, которые он хочет выполнить.

Подробнее остановимся на двух следующих атаках:

* SSRF
* Http Smugling

## Подделка серверных запросов(SSRF)

Механизм атаки Server Side Request Forgery (SSRF) реализован путем внедрения запроса в веб-приложение, которое выполняется со стороны сервера. Злоумышленник может создать запрос для внутреннего ресурса, такого как локальный файл или внутренний URL-адрес, или для внешнего ресурса, такого как другой веб-сайт или служба.

Для проведения атаки злоумышленник обычно использует уязвимость в веб-приложении, например, возможность указать URL-адрес в параметре, который не проверен должным образом. Злоумышленник затем может использовать эту уязвимость, чтобы заставить сервер сделать запрос к желаемому целевому ресурсу.

Например, если веб-приложение позволяет пользователю указать URL-адрес для получения данных, злоумышленник может создать запрос, указывающий внутренний IP-адрес или локальный хост, что позволит ему получить доступ к конфиденциальной информации, хранящейся на сервере.

Алгоритм атаки с подделкой запросов на стороне сервера (SSRF) обычно состоит из следующих шагов:

* Выявление уязвимого веб-приложения.
* Создание вредоносного запроса.
* Внедрение запроса в веб-приложение
* Выполнение запроса с сервера
* Доступ к конфиденциальной информации или выполнение вредоносных действий

Атаки с подделкой запросов на стороне сервера (SSRF) могут быть реализованы различными способами, помимо использования URL-адресов. Некоторые другие методы включают в себя:

* IP-адреса: Злоумышленник может использовать внутренние IP-адреса для доступа к конфиденциальной информации, хранящейся на сервере.
* Пути к файлам: Злоумышленник может использовать пути к файлам для доступа к конфиденциальной информации, хранящейся на сервере, или для выполнения произвольных команд на сервере.
* Полезные нагрузки, связанные с протоколом: злоумышленник может использовать определенные полезные данные, связанные с протоколами, такими как SMTP или FTP, для проведения атаки SSRF.
* Манипулирование сценариями на стороне сервера: Злоумышленник может манипулировать сценариями на стороне сервера для проведения атаки SSRF, например, изменяя поведение сценария, чтобы сделать запрос к ресурсу по своему выбору.
* Обход механизмов безопасности: злоумышленник может обойти механизмы безопасности, такие как брандмауэры или системы обнаружения вторжений, для проведения атаки SSRF.

Существует несколько методов защиты от атак с подделкой запросов на стороне сервера (SSRF):

* Проверка входных данных: разработчики должны проверять все входящие запросы, чтобы убедиться, что они законны и не содержат вредоносных полезных данных. Это может включать в себя проверку правильности формата всех URL-адресов, отсутствия в них опасных символов или зарезервированных доменов и доступа только к утвержденным ресурсам.
* Ограничение доступа сервера к внешним ресурсам. Разработчики должны ограничить доступ сервера к внешним ресурсам, таким как другие веб-сайты или службы, чтобы свести к минимуму риск атак SSRF. Это можно сделать, используя фильтры для блокировки запросов к внутренним IP-адресам или зарезервированным доменам.
* Регулярное обновление веб-приложения с помощью последних исправлений безопасности и устранение уязвимостей может помочь предотвратить атаки SSRF.
* Использование веб-фреймворков, ориентированных на безопасность, таких как OWASP или Ruby on Rails, может помочь смягчить атаки SSRF за счет внедрения лучших практик безопасности и предоставления встроенных функций безопасности.
* Использование систем обнаружения вторжений, таких как брандмауэры или системы предотвращения вторжений, может помочь в выявлении и предотвращении SSRF-атак путем мониторинга сетевого трафика и оповещения администраторов при обнаружении подозрительной активности.
* Регулярные аудиты безопасности, такие как тестирование на проникновение или проверка кода, могут помочь выявить и устранить потенциальные уязвимости SSRF и обеспечить безопасность веб-приложения.

## Http request smuggling.

Контрабанда HTTP-запросов — это тип атаки, использующий способ, которым веб-серверы и прокси-серверы обрабатывают HTTP-запросы. Атака включает в себя отправку двух или более HTTP-запросов, предназначенных для того, чтобы ввести в заблуждение сервер или прокси-сервер и обойти элементы управления безопасностью.

Один из распространенных сценариев контрабанды HTTP-запросов — это когда злоумышленник отправляет два конфликтующих запроса на целевой сервер с намерением заставить сервер обработать один запрос не так, как предполагалось. Например, злоумышленник может отправить два запроса с конфликтующими заголовками «Content-Length», в результате чего сервер неправильно интерпретирует длину тела запроса и обрабатывает его неожиданным образом. Это может привести к различным уязвимостям системы безопасности, таким как атаки с использованием межсайтовых сценариев (XSS), атаки с подделкой межсайтовых запросов (CSRF) или кража конфиденциальной информации.

Другой сценарий контрабанды HTTP-запросов — когда злоумышленник отправляет два или более запросов на целевой сервер или прокси-сервер с намерением обойти элементы управления безопасностью, такие как брандмауэры или системы обнаружения вторжений (IDS). Злоумышленник может формировать запросы определенным образом, что позволяет обрабатывать второй запрос таким образом, чтобы обойти элементы управления безопасностью, что позволяет злоумышленнику выполнять вредоносные действия, такие как кража конфиденциальной информации или компрометация целевого сервера.

Техническая реализация атаки контрабанды HTTP-запросов может различаться в зависимости от конкретной уязвимости. Однако обычно это включает в себя создание HTTP-запросов особым образом, чтобы использовать уязвимость в способе обработки HTTP-запросов сервером или прокси-сервером. Например, злоумышленник может использовать такие методы, как разделение HTTP-запросов или мультиплексирование HTTP-запросов, для отправки конфликтующих запросов на целевой сервер или прокси-сервер.

Атаки HTTP Request Smuggling происходят, когда злоумышленник отправляет два или более HTTP-запросов на целевой сервер или прокси-сервер с намерением использовать уязвимость в способе обработки этих запросов сервером или прокси-сервером. Злоумышленник формирует запросы особым образом, что позволяет обрабатывать второй запрос таким образом, чтобы обойти меры безопасности, что позволяет злоумышленнику выполнять вредоносные действия, такие как кража конфиденциальной информации или компрометация целевого сервера.

Существует два распространенных сценария атак контрабанды HTTP-запросов:

* Конфликтующие заголовки Content-Length: в этом сценарии злоумышленник отправляет два конфликтующих HTTP-запроса с разными заголовками Content-Length, в результате чего сервер неправильно интерпретирует длину тела запроса и обрабатывает его неожиданным образом. Это может привести к различным уязвимостям системы безопасности, таким как атаки с использованием межсайтовых сценариев (XSS), атаки с подделкой межсайтовых запросов (CSRF) или кража конфиденциальной информации.
* Обход средств защиты: в этом сценарии злоумышленник отправляет два или более HTTP-запроса на целевой сервер или прокси-сервер с намерением обойти средства защиты, такие как брандмауэры или системы обнаружения вторжений (IDS). Злоумышленник может формировать запросы определенным образом, что позволяет обрабатывать второй запрос таким образом, чтобы обойти элементы управления безопасностью, что позволяет злоумышленнику выполнять вредоносные действия, такие как кража конфиденциальной информации или компрометация целевого сервера.

Конкретные детали того, как происходят атаки HTTP Request Smuggling, могут различаться в зависимости от целевой уязвимости, но основная концепция заключается в том, что злоумышленник использует уязвимость в способе обработки сервером или прокси-сервером HTTP-запросов для выполнения вредоносных действий.

Важно отметить, что атаки контрабанды HTTP-запросов бывает трудно обнаружить и предотвратить, поскольку они часто связаны с использованием тонких различий в том, как веб-серверы и прокси-серверы обрабатывают HTTP-запросы. Чтобы снизить риск атак с контрабандой HTTP-запросов, организациям следует внедрить надлежащие меры безопасности и передовые методы, такие как проверка пользовательского ввода, обеспечение строгой безопасности транспорта (STS) и правильная настройка веб-серверов и прокси-серверов для обработки HTTP-запросов.

Для дополнительной защиты от атак HTTP Request Smuggling организации могут реализовать следующие меры безопасности:

* Проверка ввода: проверяйте все вводимые пользователем данные, чтобы убедиться, что они соответствуют определенным требованиям и не содержат вредоносного контента. Это может помочь предотвратить внедрение злоумышленниками вредоносных запросов в приложение.
* Проверка заголовка Content-Length: проверьте заголовок Content-Length во всех HTTP-запросах, чтобы убедиться, что он точно отражает длину тела запроса. Это может помочь предотвратить использование злоумышленниками уязвимостей, связанных с конфликтующими заголовками «Content-Length».
* Брандмауэр веб-приложений (WAF): внедрите WAF для защиты веб-приложения от различных угроз безопасности, включая атаки HTTP Request Smuggling. WAF может анализировать входящие HTTP-запросы и блокировать любые запросы, которые считаются вредоносными.
* Нормализация HTTP-запросов: нормализуйте HTTP-запросы до стандартного формата, чтобы гарантировать, что все запросы обрабатываются согласованным образом. Это может помочь предотвратить использование злоумышленниками различий в способах обработки HTTP-запросов разными веб-серверами и прокси-серверами.
* Сегментация сети. Сегментируйте сеть, чтобы ограничить масштаб любых потенциальных нарушений безопасности. Это может помочь снизить риск того, что злоумышленники смогут получить доступ к конфиденциальной информации или скомпрометировать целевой сервер.
* Регулярные проверки безопасности. Проводите регулярные проверки безопасности для выявления и устранения любых уязвимостей в веб-приложении. Это может помочь предотвратить использование злоумышленниками известных или неизвестных уязвимостей в приложении.

В заключение, контрабанда HTTP-запросов — это тип атаки, использующий способ, которым веб-серверы и прокси-серверы обрабатывают HTTP-запросы. Организации могут снизить риск контрабанды HTTP-запросов, внедрив надлежащие меры безопасности и передовые методы, такие как проверка входных данных, проверка заголовка Content-Length, WAF, нормализация HTTP-запросов, сегментация сети и регулярные проверки безопасности.

## Исследование сканеров веб-уязвимостей

Для тестирования выбранных сканеров веб-уязвимостей было взято уязвимое веб-приложение — это Damn Vulnerable Web Application (DVWA).

Это веб-приложение PHP/MySQL, которое черезмерно уязвимо. Его основные цели — помочь специалистам по безопасности проверить свои навыки и инструменты в правовой среде, помочь веб-разработчикам лучше понять процессы защиты веб-приложений и помочь начинающим специалистам в обучении.

А также для тестирования использовался удаленный веб-сайт WebScanTest. Это веб-сайт, созданный специально для тестирования автоматических анализаторов веб-уязвимостей.

Для тестирования использовались стандартные настройки сканеров без дополнительных опций и плагинов.

### OWASP ZAP

OWASP ZAP – Это веб-сканер с открытым исходным кодом, разработанный командой OWASP.

Graphical user interface, text

Description automatically generated

Рисунок 3 – Интерфейс веб-сканера OWASP ZAP

Graphical user interface, text, application

Description automatically generated

Рисунок 4 – Найденные уязвимости OWASP ZAP

|  |  |  |
| --- | --- | --- |
| **Тип** | **Описание** | **Критичность** |
| Missing Anti-clickjacking Header | Ответ не включает ни Content-Security-Policy с директивой «frame-ancestors», ни X-Frame-Options для защиты от атак «ClickJacking». | Средняя |
| Cookie No HttpOnly Flag | Файл cookie был установлен без флага HttpOnly, что означает, что доступ к файлу cookie возможен с помощью JavaScript. Если на этой странице можно запустить вредоносный скрипт, то файл cookie будет доступен и может быть передан на другой сайт. Если это файл cookie сеанса, то возможен перехват сеанса. | Низкая |
| Cookie without SameSite Attribute | Файл cookie был установлен без атрибута SameSite, что означает, что файл cookie может быть отправлен в результате «межсайтового» запроса. Атрибут SameSite — это эффективная мера противодействия подделке межсайтовых запросов, включению межсайтовых сценариев и атакам по времени. | Низкая |
| Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) | Веб-сервер/сервер приложений пропускает информацию через один или несколько заголовков HTTP-ответа «X-Powered-By». Доступ к такой информации может помочь злоумышленникам идентифицировать другие платформы/компоненты, от которых зависит ваше веб-приложение, и уязвимости, которым могут быть подвержены такие компоненты. | Низкая |
| Timestamp Disclosure - Unix | Отметка времени была раскрыта приложением/веб-сервером — Unix | Низкая |
| X-Content-Type-Options Header Missing | Заголовок Anti-MIME-Sniffing X-Content-Type-Options не был установлен на «nosniff». Это позволяет более старым версиям Internet Explorer и Chrome выполнять анализ MIME в теле ответа, что может привести к тому, что тело ответа будет интерпретировано и отображено как тип контента, отличный от объявленного типа контента. Текущая (начало 2014 г.) и устаревшая версии Firefox будут использовать объявленный тип контента (если он установлен), а не выполнять анализ MIME. | Низкая |
| Information Disclosure - Suspicious Comments | Похоже, что ответ содержит подозрительные комментарии, которые могут помочь злоумышленнику. Примечание. Совпадения, сделанные в блоках сценария или файлах, относятся ко всему содержимому, а не только к комментариям. | Незначительная |

### Таблица 1 – Уязвимости найденные OWASP ZAP

### Rapid7 insightAppSec



Рисунок 5 – Интерфейс Rapid7

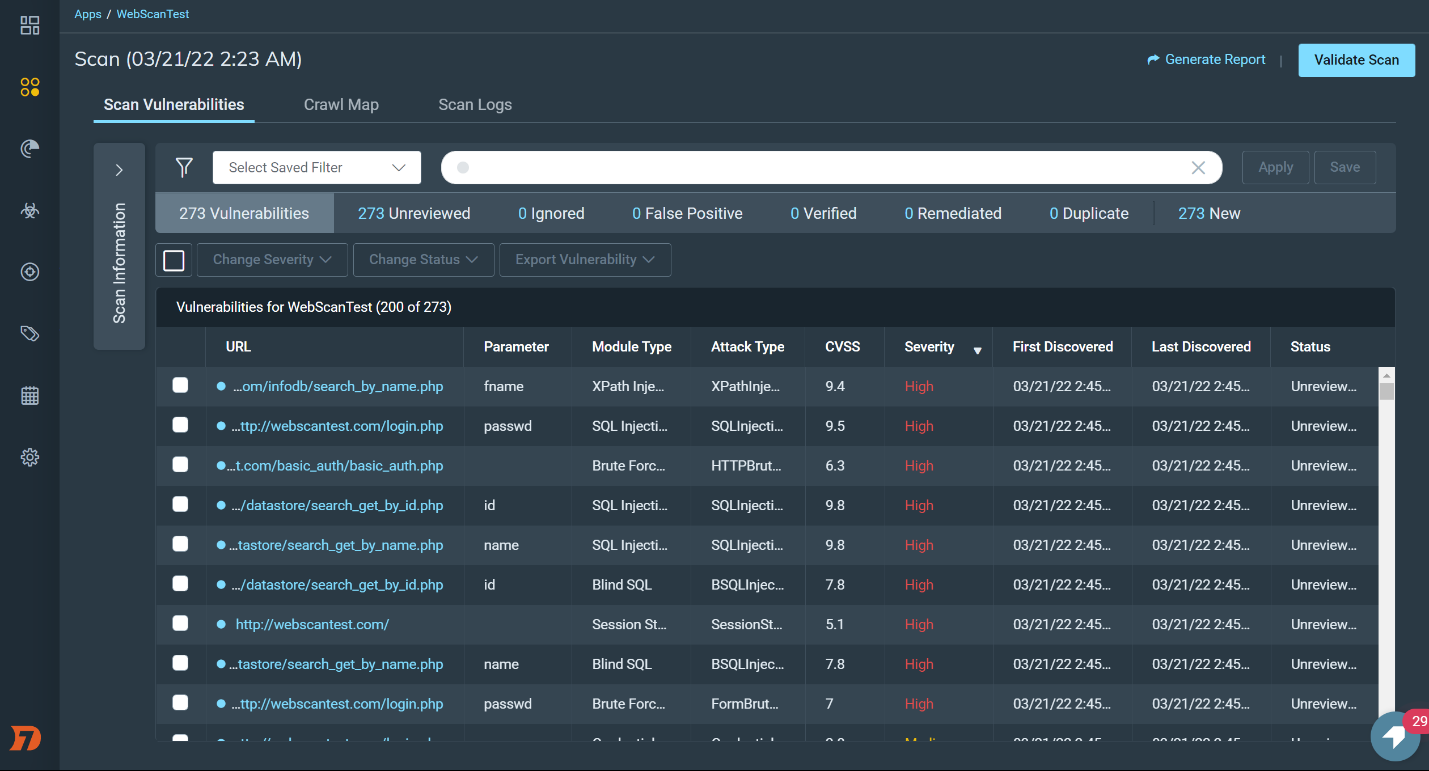


Рисунок 6 – Найденные уязвимости Rapid7

|  |  |  |
| --- | --- | --- |
| **Тип** | **Описание** | **Критичность** |
| XPath Injection | Символ синтаксиса XPATH, представленный в параметре URL, вызывает ошибку в запросе приложения. Это указывает на то, что может быть осуществлена ​​атака с внедрением XPath. Эти типы атак манипулируют запросами из введенных пользователем данных для запроса или навигации по XML-документам.  Если атака с внедрением XPath может быть использована для чтения произвольных данных, то будет раскрыта сохраненная информация пользователей, такая как учетные данные для аутентификации, адрес электронной почты или финансовая информация.  Таким образом, успешная инъекционная атака может поставить под угрозу хост приложения и другие хосты в его локальной сети, даже если они защищены от Интернета брандмауэром. | Высокая |
| SQL Injection Auth Bypass | Было обнаружено, что методы SQL-инъекций могут использоваться для обмана аутентификации приложения, при этом злоумышленнику не требуются действительные учетные данные.  Уязвимости SQL-инъекций на страницах входа в систему делают приложение неавторизованным и, вполне вероятно, на уровне администратора, тем самым серьезно нарушая безопасность приложения.  Для приложения обнаружена допустимая комбинация имени пользователя и пароля.  Пароли по умолчанию и простые пароли делают приложение неавторизованным. Если пользователь выбирает очень ненадежный пароль, информация об учетной записи этого пользователя может быть скомпрометирована, и учетная запись может быть использована для попытки взлома приложения. | Высокая |
| Brute Force (HTTP Auth) | Для приложения обнаружена допустимая комбинация имени пользователя и пароля.  Пароли по умолчанию и простые пароли делают приложение неавторизованным. Если пользователь выбирает очень ненадежный пароль, информация об учетной записи этого пользователя может быть скомпрометирована, и учетная запись может быть использована для попытки взлома приложения.  Токены сеанса с низкой энтропией («случайность») часто подвержены атакам с прогнозированием. Небезопасные токены могут быть связаны с неадекватным генератором псевдослучайных чисел, значениями на основе времени, статическими значениями или значениями, основанными на атрибутах пользователя (имя пользователя или идентификатор пользователя). Это означает, что злоумышленник сможет угадать действительный токен сеанса после наблюдения за приложением в течение короткого периода времени и сбора создаваемых им токенов сеанса. Если злоумышленник определяет действительный токен сеанса для другого пользователя, то можно просматривать, изменять или удалять данные произвольных пользователей без необходимости угадывать имя пользователя или пароль жертвы. Следовательно, способность выводить действительные токены сеанса позволяет злоумышленнику обходить страницы входа в систему и устранять необходимость взлома учетных записей.  Кроме того, статические токены могут позволить злоумышленнику нацеливаться на пользователей, даже если жертва в данный момент не вошла в приложение. Это увеличивает пул жертв, на которых может нацелиться злоумышленник.  Токены сеанса должны создаваться с помощью сильного генератора случайных чисел и собираться из большого пула чисел. Например, функции rand() в операционной системе обычно может быть достаточно, если она может выдавать 32-битные значения со статистически однородным распределением. Плохие маркеры сеанса являются добавочными, полагаются на идентификатор учетной записи пользователя, используют только временные метки или содержат другую строго детерминированную информацию. Другие методы защиты безопасности токена сеанса заключаются в том, чтобы всегда передавать их через SSL, автоматически истекать срок действия токена по истечении определенного периода времени и явно указывать срок действия токена всякий раз, когда пользователь выходит из приложения. | Высокая |
| Blind SQL | Эти методы внедрения SQL анализируют реакцию приложения на значения параметров, которые предназначены для интерпретации и выполнения базой данных. Эти запросы содержат аргументы, на которые не влияют входные фильтры проверки. Приложение отправляет исходную полезную нагрузку в базу данных, где база данных интерпретирует полезную нагрузку как допустимый SQL-запрос. Это означает, что произвольные команды SQL могут быть выполнены через это значение параметра. Эти тесты не генерируют ошибок базы данных, и ошибки базы данных не должны отображаться в ответе HTML.  Уязвимости, обнаруженные этим модулем, указывают на проблемы с процедурами проверки ввода и созданием запросов SQL. Их следует решать с помощью фундаментальных подходов, используемых для противодействия распространенным атакам SQL-инъекций. | Высокая |

### Таблица 2 – Уязвимости найденные Rapid7

### Burp Suite Professional

Graphical user interface, application, website

Description automatically generated

Рисунок 7 – Интерфейс Burp Suite

A screenshot of a computer

Description automatically generated

Рисунок 8 – Найденные уязвимости Burp Suite

|  |  |  |
| --- | --- | --- |
| **Тип** | **Описание** | **Критичность** |
| |  |  | | --- | --- | |  | External service interaction (DNS) | | Взаимодействие с внешней службой возникает, когда можно заставить приложение взаимодействовать с произвольной внешней службой, такой как веб-сервер или почтовый сервер. Способность запускать произвольные взаимодействия с внешними службами сама по себе не является уязвимостью, а в некоторых случаях даже может быть предполагаемым поведением приложения. Однако во многих случаях это может указывать на уязвимость с серьезными последствиями.  В тех случаях, когда взаимодействие на основе DNS может быть инициировано, обычно можно инициировать взаимодействие с использованием других типов служб, и о них сообщается как об отдельных проблемах. Если полезная нагрузка, указывающая конкретный тип службы (например, URL-адрес), инициирует только взаимодействие на основе DNS, то это однозначно указывает на то, что приложение пыталось подключиться с использованием этой другой службы, но ему не позволили сделать это выходные фильтры, установленные на месте. сетевой слой. Возможность отправлять запросы в другие системы может позволить использовать уязвимый сервер в качестве прокси-сервера для атаки. Отправляя подходящие полезные нагрузки, злоумышленник может заставить сервер приложений атаковать другие системы, с которыми он может взаимодействовать. Это могут быть общедоступные сторонние системы, внутренние системы в той же организации или службы, доступные на локальном адаптере обратной связи самого сервера приложений. В зависимости от сетевой архитектуры это может привести к тому, что внутренние службы будут очень уязвимы, которые иначе недоступны для внешних злоумышленников. | Высокая |
| |  |  | | --- | --- | |  | Cookie without HttpOnly flag set | | Если для файла cookie установлен атрибут HttpOnly, то значение файла cookie не может быть прочитано или установлено клиентским JavaScript. Эта мера делает некоторые атаки на стороне клиента, такие как межсайтовый скриптинг, немного более сложными для использования, поскольку не позволяет им тривиально захватить значение файла cookie с помощью внедренного скрипта. | Высокая |
| |  |  | | --- | --- | |  | Input returned in response (reflected) | | Отражение ввода возникает, когда данные копируются из запроса и отображаются в немедленном ответе приложения.  Входные данные, возвращаемые в ответах приложений, сами по себе не являются уязвимостью. Однако это обязательное условие для многих уязвимостей на стороне клиента, включая межсайтовые сценарии, открытое перенаправление, подделку содержимого и внедрение заголовка ответа. Кроме того, некоторые уязвимости на стороне сервера, такие как SQL-инъекция, часто легче идентифицировать и использовать, когда входные данные возвращаются в ответах. В приложениях, в которых получение входных данных происходит редко, а среда устойчива к автоматическому тестированию (например, из-за брандмауэра веб-приложений), возможно, стоит подвергнуть его экземпляры целенаправленному ручному тестированию. | Высокая |
| |  |  | | --- | --- | |  | Frameable response (potential Clickjacking | | Если странице не удается установить соответствующий HTTP-заголовок X-Frame-Options или Content-Security-Policy, возможно, страница, контролируемая злоумышленником, загрузит ее в iframe. Это может привести к атаке кликджекинга, при которой страница злоумышленника перекрывает интерфейс целевого приложения с другим интерфейсом, предоставленным злоумышленником. Побуждая пользователей-жертв выполнять такие действия, как щелчки мышью и нажатия клавиш, злоумышленник может заставить их непреднамеренно выполнять действия в приложении, на которое нацелена атака. Этот метод позволяет злоумышленнику обойти защиту от подделки межсайтовых запросов и может привести к несанкционированным действиям.  Обратите внимание, что некоторые приложения пытаются предотвратить эти атаки изнутри самой HTML-страницы, используя код «блокировки кадров». Однако этот тип защиты обычно неэффективен, и опытный злоумышленник обычно может его обойти.  Вы должны определить, могут ли какие-либо функции, доступные на страницах с фреймами, использоваться пользователями приложения для выполнения каких-либо конфиденциальных действий в приложении. | Незначительная |

### Таблица 3 – Уязвимости найденные Burp Suite

* + 1. *Сравнение веб-сканеров*

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **OWASP ZAP** | **RAPID7** | **BURP** |
| Open source | + | - | - |
| Plugins | + | - | + |
| Больше всего найденных уязвимостей | - | + | - |
| Работа с локальным сервером | + | - | + |
| Детальный отчет | +- | + | +- |
| Гибкая настройка плагина | - | - | + |
| Облачные вычисления | - | + | - |

## Разработка плагина для веб-сканера Burp Suite

Для разработки собственного плагина среды Burp понадобится Jython Standalone Edition, так как само приложение написано на Java.

В проекте будет использоваться модуль *exceptions\_fix.py* из открытого репозитория *burp-exceptions*. Это поможет транслировать исключения Java в вид привычный для Python исключений.

java.lang.RuntimeException: org.python.core.PyException

  at burp.fl.a(Unknown Source)

  at burp.edd.a(Unknown Source)

  at burp.e2g.a(Unknown Source)

  at burp.e2g.g(Unknown Source)

  at burp.i1c.stateChanged(Unknown Source)

  at javax.swing.JTabbedPane.fireStateChanged(JTabbedPane.java:416)

  at javax.swing.JTabbedPane$ModelListener.stateChanged(JTabbedPane.java:270)

Пример - Java исключение

\*\*\* PYTHON EXCEPTION

Traceback (most recent call last):

  File "/Users/mb/Desktop/burp extension/exceptions\_fix.py", line 8, in decorated\_function

    return original\_function(\*args, \*\*kwargs)

  File "/Users/mb/Desktop/burp extension/CustomEditorTab.py", line 78, in setMessage

    self.\_txtInput.setEsditable(self.\_editable)

AttributeError: &apos;burp.ul&apos; object has no attribute &apos;setEsditable&apos;

Пример – Python исключение

Первый класс, который нужно будет создать:

**class** **BurpExtender**(IBurpExtender, IHttpListener):

IBurpExtender - главный интерфейс, его должны наследовать все расширения для Burp. В классе BurpExtender используются интерфейсы IBurpExtender и IHttpListener, что позволяет ему регистрироваться как расширение и обрабатывать HTTP-сообщения.

Реализованы функции, для проверки возможных вариантов реализации SSRF атак.

* def checkForPrivateIP(self, host):
* def checkForReservedIP(self, host):
* def checkForIternalHostname(self, host):

Реализованные функции вызываются из функции проверки атаки SSRF:

def checkForSSRF(self, requestInfo):

Механизм работы скрипта:

Скрипт реализует интерфейсы IBurpExtender и IHttpListener из Burp Suite API для расширения функциональности инструмента.

Когда расширение загружается, вызывается метод «registerExtenderCallbacks», который устанавливает имя расширения как «Сканер уязвимостей SSRF» и регистрирует прослушиватель HTTP для перехвата входящих HTTP-запросов.

Метод «processHttpMessage» вызывается для каждого перехваченного HTTP-запроса, где скрипт анализирует запрос с помощью метода «self.\_helpers.analyzeRequest».

Затем скрипт проверяет, уязвим ли запрос для SSRF, вызывая метод «checkForSSRF», который выполняет несколько проверок на хосте URL-адреса запроса. Если хост — «127.0.0.1» или «localhost», частный IP-адрес, зарезервированный IP-адрес или внутреннее имя хоста, сценарий считает запрос уязвимым для SSRF.

Если скрипт определяет, что запрос уязвим для SSRF, он добавляет проблему сканирования на вкладку «Сканер» Burp Suite, указывая, что запрос уязвим для SSRF, и включая информацию об уязвимости, такую как ее серьезность, достоверность и описание.

# СПИСОК ИСТОЧНИКОВ

1. https://owasp.org/www-project-top-ten/ - OWASP TOP 10
2. https://www.ptsecurity.com/upload/corporate/ww-en/analytics/web-vulnerabilities-2020-eng.pdf - Positive technologies
3. https://developer.mozilla.org/ru/docs/Web/HTTP/CORS - CORS
4. https://portswigger.net/burp/extender/writing-your-first-burp-suite-extension - Burp extension
5. https://2sharp.pro/infosec/92-owasp-top-10-2021.html#hmenu-1 - OWASP TOP 10 description
6. https://portswigger.net/web-security/cors - CORS
7. https://www.whitesourcesoftware.com/resources/blog/the-top-web-vulnerability-scanners/ - Top web scanners
8. https://portswigger.net/web-security/cors/lab-basic-origin-reflection-attack - Lab 1 - Origin Reflect
9. https://portswigger.net/web-security/cors/lab-null-origin-whitelisted-attack - Lab 2 - Null Origin
10. https://portswigger.net/web-security/cors/lab-breaking-https-attack - Lab 3 - Untrusted protocol

# ПРИЛОЖЕНИЕ А

ssrf-scanner.py

from burp import IBurpExtender

from burp import IScannerCheck

from burp import IScanIssue

import array

from urlparse import urlparse, unquote

import re

class BurpExtender(IBurpExtender, IScannerCheck):

def registerExtenderCallbacks(self, callbacks):

self.\_callbacks = callbacks

self.\_helpers = callbacks.getHelpers()

callbacks.setExtensionName("SSRF Scanner")

callbacks.registerScannerCheck(self)

def doPassiveScan(self, baseRequestResponse):

try:

requestInfo = self.\_helpers.analyzeRequest(baseRequestResponse)

headers = requestInfo.getHeaders()

requestBody = baseRequestResponse.getRequest()[requestInfo.getBodyOffset():]

url = self.\_helpers.analyzeRequest(baseRequestResponse).getUrl()

parsedUrl = urlparse(str(url))

# Check if the URL scheme is http or https

if not parsedUrl.scheme in ["http", "https"]:

return None

# Extract the parameters from the request

parameters = requestInfo.getParameters()

for parameter in parameters:

paramValue = parameter.getValue()

# Check if the parameter value is a URL

if re.match("^http[s]?", paramValue):

# Decode the URL

paramValue = unquote(paramValue)

# Create a new URL with the parameter value

newUrl = urlparse(paramValue)

if not newUrl.scheme or not newUrl.hostname:

print("some prob")

continue

# Check if the host of the new URL is the same as the host of the original URL

if newUrl.hostname != parsedUrl.hostname:

# Send a request to the new URL and check if the response is successful

checkRequest = self.\_helpers.buildHttpRequest(requestInfo.getUrl())

malUrlhostname = "localhost/admin"

malUrlport = 443

checkRequest = checkRequest.replace(requestInfo.getUrl().toString(), "localhost/admin")

checkResponse = self.\_callbacks.makeHttpRequest(malUrlhostname, malUrlport, False,

checkRequest)

if checkResponse.getStatusCode() < 400:

issue = ScanIssue(

baseRequestResponse.getHttpService(),

self.\_helpers.analyzeRequest(baseRequestResponse).getUrl(),

[self.\_callbacks.applyMarkers(baseRequestResponse, None,

[parameter.getNameStart(), parameter.getNameEnd(),

parameter.getValueStart(), parameter.getValueEnd()])],

"SSRF Vulnerability",

"The parameter " + parameter.getName() + " in " + requestInfo.getUrl().getPath() + " appears to be vulnerable to SSRF. The application was able to successfully make a request to an external URL.",

"High"

)

print("Potential SSRF vulnerabilities found in " + url.toString())

return [issue]

else: print("No potential SSRF vulnerabilities found in " + url.toString())

except Exception as e:

print(e)

return None

def doActiveScan(self, baseRequestResponse, insertionPoint):

try:

requestInfo = self.\_helpers.analyzeRequest(baseRequestResponse)

headers = requestInfo.getHeaders()

requestBody = baseRequestResponse.getRequest()[requestInfo.getBodyOffset():]

url = self.\_helpers.analyzeRequest(baseRequestResponse).getUrl()

parsedUrl = urlparse(str(url))

# Check if the request is a GET or POST request

if not requestInfo.getMethod() in ["GET", "POST"]:

return None

# Check if the URL scheme is http or https

if not parsedUrl.scheme in ["http", "https"]:

return None

# Extract the parameters from the request

parameters = requestInfo.getParameters()

for parameter in parameters:

paramValue = parameter.getValue()

# Check if the parameter value is a URL

if re.match("^http[s]?", paramValue):

#Decode the URL and extract the hostname

paramValue = unquote(paramValue)

newUrl = urlparse(paramValue)

#print(newUrl)

#print(newUrl.hostname)

#print(newUrl.port)

if not newUrl.scheme or not newUrl.hostname:

print("some prob")

continue

# Check if the host of the new URL is the same as the host of the original URL

if newUrl.hostname != parsedUrl.hostname:

# Send a request to the new URL and check if the response is successful

checkRequest = self.\_helpers.buildHttpRequest(requestInfo.getUrl())

malUrlhostname = "localhost/admin"

malUrlport = 443

checkRequest = checkRequest.replace(requestInfo.getUrl().toString(), "localhost/admin")

checkResponse = self.\_callbacks.makeHttpRequest(malUrlhostname, malUrlport, False,

checkRequest)

if checkResponse.getStatusCode() < 400:

# Issue description

description = "The parameter " + parameter.getName() + " in " + requestInfo.getUrl().getPath() + " appears to be vulnerable to SSRF. The application was able to successfully make a request to an external URL."

# Issue severity

severity = "High"

# Issue name

issueName = "SSRF Vulnerability"

# Issue background

issueBackground = "Server-Side Request Forgery (SSRF) is a vulnerability that occurs when an application allows an attacker to send requests from the server-side of the application to a destination specified by the attacker."

# Issue detail

issueDetail = "The parameter " + parameter.getName() + " in " + requestInfo.getUrl().getPath() + " appears to be vulnerable to SSRF. The application was able to successfully make a request to an external URL."

# Create a ScanIssue object

scanIssue = ScanIssue(

baseRequestResponse.getHttpService(),

self.\_helpers.analyzeRequest(baseRequestResponse).getUrl(),

[self.\_callbacks.applyMarkers(baseRequestResponse, None,

[parameter.getNameStart(), parameter.getNameEnd(),

parameter.getValueStart(), parameter.getValueEnd()])],

issueName,

issueDetail,

severity

)

print("Potential SSRF vulnerabilities found in " + url.toString())

self.\_callbacks.addScanIssue(scanIssue)

else:print("No potential SSRF vulnerabilities found in " + url.toString())

return None

except Exception as e:

print(e)

return None

def consolidateDuplicateIssues(self, existingIssue, newIssue):

if existingIssue.getUrl() == newIssue.getUrl() and existingIssue.getIssueName() == newIssue.getIssueName() and existingIssue.getIssueDetail() == newIssue.getIssueDetail():

return -1

else:

return 0

class ScanIssue(IScanIssue):

def \_\_init\_\_(self, httpService, url, httpMessages, name, detail, severity):

self.\_httpService = httpService

self.\_url = url

self.\_httpMessages = httpMessages

self.\_name = name

self.\_detail = detail

self.\_severity = severity

def getUrl(self):

return self.\_url

def getIssueName(self):

return self.\_name

def getIssueType(self):

return 0

def getSeverity(self):

return self.\_severity

def getConfidence(self):

return "Certain"

def getIssueBackground(self):

return None

def getRemediationBackground(self):

return None

def getIssueDetail(self):

return self.\_detail

def getRemediationDetail(self):

return None

def getHttpMessages(self):

return self.\_httpMessages

def getHttpService(self):

return self.\_httpService