

Реферат

Отчёт 34 страницы, 8 источников, 1 приложение.

Ключевые слова: информационная безопасность, межсетевой экран, веб-приложение, фильтрация, HTTP-запрос.

Цель проведения научно-исследовательской работы является исследование видов веб-приложений и способов их фильтрации. В процессе были изучены источники в научных публикациях на заданную тему. Были рассмотрены существующие виды веб-приложений, способы фильтрации веб-приложений в WAF, а также была реализована программа, реализующая простой способ фильтрации HTTPS-трафика.

Содержание

Реферат	4
Термины и определения.....	6
Перечень сокращений и обозначений	7
Введение.....	8
1. Виды веб-приложений и атаки на них	9
1.1 Классификация веб-приложений.....	9
1.2 Атаки на веб-приложения.....	14
2. Фильтрация веб-приложений.....	21
2.1 Фильтрация на основе регулярных выражений и сигнатур.....	22
2.2 Фильтрация на основе поведения.....	23
2.3 Практическая работа.....	25
Заключение.....	29
Список использованных источников.....	30
ПРИЛОЖЕНИЕ А. Листинг программы nirs.py	31

Термины и определения

В настоящем отчете о НИР применяют следующие термины с соответствующими определениями:

Межсетевой экран - программный или программно-аппаратный элемент компьютерной сети, осуществляющий контроль и фильтрацию проходящего через него сетевого трафика в соответствии с заданными правилами

Модель OSI – концептуальная модель, принимаемая для описания и объяснения работы компьютерных сетей. Выделяют семь уровней в данной модели: физический, канальный, сетевой, транспортный, сеансовый, представления, прикладной. На каждом из этих уровней реализованы свои протоколы, используемые для передачи данных на этом уровне

IP адрес – это числовой идентификатор, присваиваемый устройству (компьютеру, маршрутизатору и т. д.), подключенному к компьютерной сети, для идентификации и обеспечения связи в Интернете

TCP – это протокол передачи данных, работающий на транспортном уровне модели OSI. TCP предоставляет функции установления соединения, управления потоком, контроля ошибок и гарантированной доставки данных между узлами в компьютерной сети. Для установки соединения протокол использует трёхэтапный процесс установления соединения. Также данный протокол обеспечивает гарантированную доставку данных

Перечень сокращений и обозначений

В настоящем отчете о НИР применяют следующие сокращения и обозначения:

CDN – Content Delivery Network

CSRF – Cross-Site Request Forgery

CSS – Cascading Style Sheets

cURL – client Uniform Resource Locator

DNS – Domain Name System

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

IP – Internet Protocol

MPA – Multi Page Application

OSI – Open Systems Interconnection

PHP – Hypertext Preprocessor

PWA – Progressive Web Application

RWA – Rich Web Application

SPA – Single Page Application

SQL – Structured Query Language

SSL – Secure Socket Layer

TCP – Transmission control protocol

TLS – Transport Layer Security

URL – Uniform Resource Locator

WAF – Web Application Firewall

XSS – Cross-Site Scripting

МЭ – Межсетевой экран

Введение

Программа называется веб-приложением, если она работает на удаленном веб-сервере через интернет при этом формирует запросы и отвечает на них по протоколу HTTP. Интерфейс веб-приложений реализуется с использованием языков HTML, CSS и Javascript, а серверная часть – с использованием PHP, Python, Java, Ruby и т.д.

За последние десятилетия Интернет проник во все сферы жизни людей, от развлечений до коммерции. На фоне этого огромную роль начали играть веб-приложения. Веб-приложения регулярно обслуживают миллионы людей в процессе поиска информации, покупок, просмотра контента и т. п. Веб-приложения стали значительно сложнее, чем они были на заре Интернета. Они стали асинхронными, интерактивными и динамичными благодаря появлению новых технологий. Но одновременно с этим для своей работы они требуют только установленный браузер на клиентской стороне. При этом они могут работать как на смартфоне, так и персональном компьютере и практически независимы от характеристик оборудования.

Таким образом, важную роль обрела безопасность веб-приложений из-за их повсеместного распространения и оперирования большими объемами важных данных. В рамках данной научно-исследовательской работы будет описана классификация веб-приложений, а также описаны некоторые алгоритмы их фильтрации. Для этого необходимо проанализировать соответствующую литературу по теме и практически рассмотреть некоторые вопросы безопасности веб-приложений.

1. Виды веб-приложений и атаки на них

Веб-приложение – клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети.

Клиентская часть реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него. Серверная часть получает запрос от клиента, выполняет вычисления, после этого формирует веб-страницу и отправляет её клиенту по сети с использованием протокола HTTP.

В данном разделе будет дана классификация веб-приложений по способу их реализации.

1.1 Классификация веб-приложений

Веб-страницы хранятся на веб-сервере в виде файлов. Пользователь может просматривать эти страницы различными способами. Например, он может ввести полный адрес файла, содержащего веб-страницу, или перемещаться по веб-сайту, используя гиперссылки. Полный адрес (называемый URL) нужной веб-страницы связан с гиперссылкой, так, что при нажатии на гиперссылку, выбирается адрес, указанный под ней, и новая веб-страница передается с сервера клиенту.

Веб-сервер – это специальное программное обеспечение, которое отправляет запросы клиенту (веб-браузеру), предоставляя ресурсы (например, документы HTML). Когда пользователи вводят URL-адрес в веб-браузер, они запрашивают определенный документ с веб-сервера, веб-сервер сопоставляет URL-адрес с файлом на сервере и воспроизводит запрошенный документ клиенту. HTTP протокол используется для связи веб-сервера и клиента независимо от платформы.

Рассмотрим сначала два основных вида веб-приложений: статические и динамические. [1]

Веб-приложение считается **статическим**, если оно не меняет своего поведения в ответ на внешние действия. Статическая веб-страница остается

неизменной на протяжении всего срока ее работы, до тех пор, пока кто-либо вручную не изменит ее содержимое. Каждый раз, когда любой пользователь отправляет HTTP-запрос на веб-сервер, веб-сервер возвращает пользователю одно и то же содержимое в виде HTTP-ответа. Примерами статичных веб-страниц являются некоторые домашние страницы, страницы с указанием контактных данных и т.д., т. е. которые не меняются длительное время.

Процесс получения статической веб-страницы проиллюстрирован на рисунке 1. Когда клиент (веб-браузер) отправляет HTTP-запрос на получение веб-страницы, клиент отправляет этот запрос веб-серверу. Веб-сервер находит веб-страницу (т.е. файл на диске с расширением .html или .htm) и отправляет его обратно пользователю в HTTP-ответе. Другими словами, задача сервера в этом случае – просто найти файл на диске и отправить его содержимое обратно браузеру. Сервер не выполняет никакой дополнительной работы при этом.

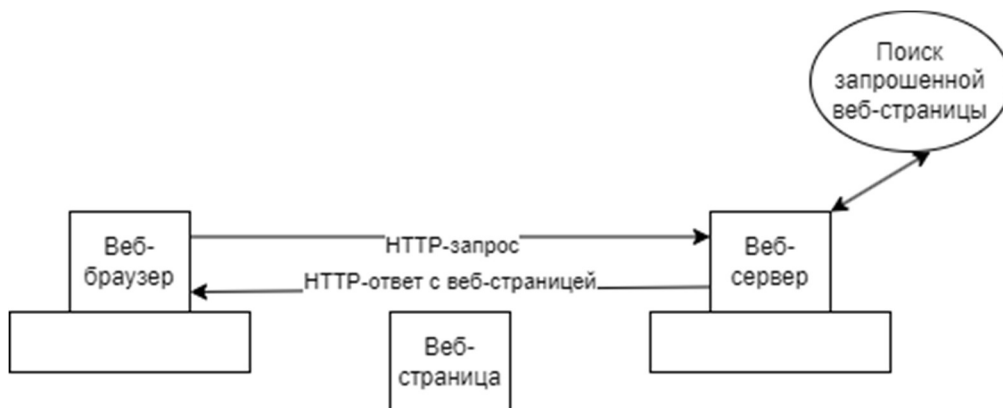


Рисунок 1. Схема работы статического веб-приложения

Веб-приложение является **динамическим**, если оно изменяет свое поведение в ответ на внешние действия. Т. е., если в ответ на HTTP-запрос пользователя веб-страница каждый раз выдает разные выходные данные, то веб-приложение считается динамическим.

Например, если необходимо создать страницу с курсом валюты, то для этого будет использоваться динамическая веб-страница. Динамическая веб-страница не создается заранее, в отличие от статической. Вместо этого она подготавливается «на лету». Всякий раз, когда пользователь отправляет HTTP-запрос на динамическую веб-страницу, сервер ищет название динамической страницы,

которая на самом деле является программой. Сервер выполняет программу локально. Программа выдает выходные данные во время выполнения, которые снова представлены в формате HTML. Этот вывод HTML отправляется обратно в браузер как часть HTTP-ответа. Схема работы показана на рисунке 2.

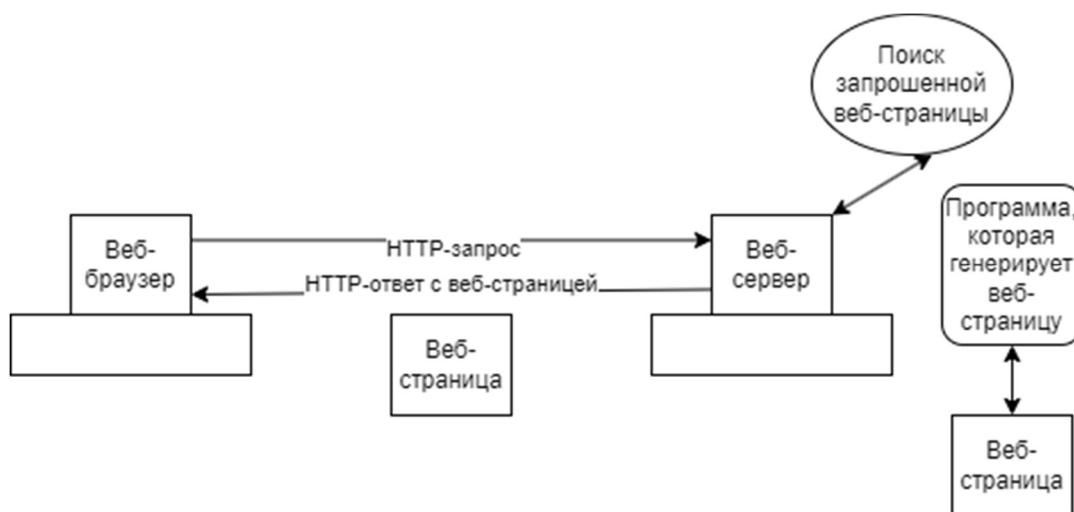


Рисунок 2. Схема работы динамического веб-приложения

Опишем другие виды веб-приложений в зависимости от способа их реализации и используемых технологий и концепций.

Начнем с **одностраничных веб-приложений**. В SPA все приложение запускается как единственная веб-страница.

В классическом веб-приложении каждый запрос на новое представление (HTML-страницу) приводит к обращению к серверу. Когда на стороне клиента требуются новые данные, запрос отправляется на сторону сервера, где он обрабатывается. После объединения данных и представления происходит возврат ответа в браузер. Браузер получает новую HTML-страницу, и после обновления пользовательского интерфейса пользователь видит новое представление, содержащее запрошенные данные.

В SPA процесс создания представлений и управления переносится с сервера на пользователя. Общий дизайн SPA практически не отличается от традиционного. Основные изменения заключаются в следующем: полное обновление браузера не выполняется, логика представления находится в клиенте, а серверные обращения могут быть связаны только с данными. SPA напоминают обычные

приложения, с той лишь разницей, что исполняются в рамках браузера, а не в собственном процессе операционной системы. [2]

Одностраничные приложения чаще всего используют в сервисах, где пользователь проводит на одной странице много времени или совершает с ней какие-то действия. Это может быть почтовый сервис, социальная сеть, сервис для просмотра видеоконтента или для просмотра фотографий.

Противоположность SPA – это **многостраничные приложения или МРА**. Многостраничные приложения работают по традиционной схеме. Это означает, что при каждом незначительном изменении данных или загрузке новой информации страница обновляется. МРА существуют с первых дней существования Интернета и широко используются по сей день. Они особенно распространены на веб-сайтах с большим количеством контента, таких как новостные сайты или сайты электронной коммерции, где каждая страница представляет отдельный фрагмент. Поскольку МРА отправляют отдельные запросы для каждой страницы, к ним проще применять такие меры безопасности, как аутентификация и авторизация. По сравнению с МРА, SPA могут быть более уязвимы для угроз безопасности, поскольку большая часть содержимого загружается динамически и может не подвергаться таким же проверкам безопасности, как у традиционных веб-страниц.

Насыщенное веб-приложение или RWA – это веб-приложение, обладающее многими характеристиками десктопного прикладного программного обеспечения. RWA тесно связано с одностраничным приложением и может предоставлять пользователю интерактивные функции, такие как перетаскивание, фоновое меню и т.д. Впервые это понятие было введено в 2002 году компанией Macromedia для описания продукта Macromedia Flash MX (который позже стал Adobe Flash). На протяжении 2000-х годов этот термин был обобщен для описания браузерных приложений, разработанных с использованием других конкурирующих технологий плагинов для браузера, включая Java-апплеты и Microsoft Silverlight.

С прекращением использования интерфейсов плагинов для браузеров и переходом на стандартные технологии HTML5 RWA были заменены одностраничными приложениями и прогрессивными веб-приложениями.

Прогрессивное веб-приложение или PWA – это тип приложения, поставляемого через Интернет, созданного с использованием распространенных веб-технологий, таких как HTML, CSS, JavaScript и WebAssembly. Оно предназначено для работы на любой платформе с браузером, соответствующим стандартам. Прогрессивность PWA заключается в том, что они:

1. Выглядят и функционируют как нативные приложения, хотя созданы на основе веб-технологий.
2. Объединяют в себе лучшие черты (и одновременно – компенсируют недостатки) мобильных и веб-приложений.

В отличие от обычных веб-приложений, для запуска PWA не нужно открывать адрес страницы в браузере. Их можно установить на домашний экран и запускать, как обычные мобильные приложения. PWA обеспечивают более быструю загрузку и более плавную работу, чем обычные веб-приложения и в меньшей степени зависят от подключения к интернету. Помимо магазина приложений, PWA можно установить прямо с сайта. Кроме того, прогрессивные веб-приложения позволяют снизить расходы на разработку и поддержку мобильных приложений.

PWA – это скорее концепция, реализованная с помощью набора технологий, чем приложение с конкретной структурой. Таковыми технологиями, в частности, являются:

- Service Worker – прокси-сервер, который позволяет присылать push-уведомления и сохранять информацию в кэше. За счет этого приложение может работать без доступа к интернету и не требует обновления и перезагрузки в офлайн-режиме в отличие от мобильной версии сайта, поскольку оно сохраняет данные каждой сессии.
- HTTPS-запросы – работа со своими ресурсами обеспечивается с помощью HTTPS через браузер аналогично тому, как нативное приложение

обращается через файловую систему. Браузер выступает в роли виртуальной машины, которая запускает PWA.

- Web App Manifest – это JSON-файл, который сообщает браузеру и операционной системе, как нужно отображать и запускать веб-приложение на мобильном устройстве при установке на главный экран. В нем настраиваются иконка, название, цвета темы и поведение приложения.

Итоговая диаграмма со всеми типами веб-приложений представлена на рисунке 3 ниже.

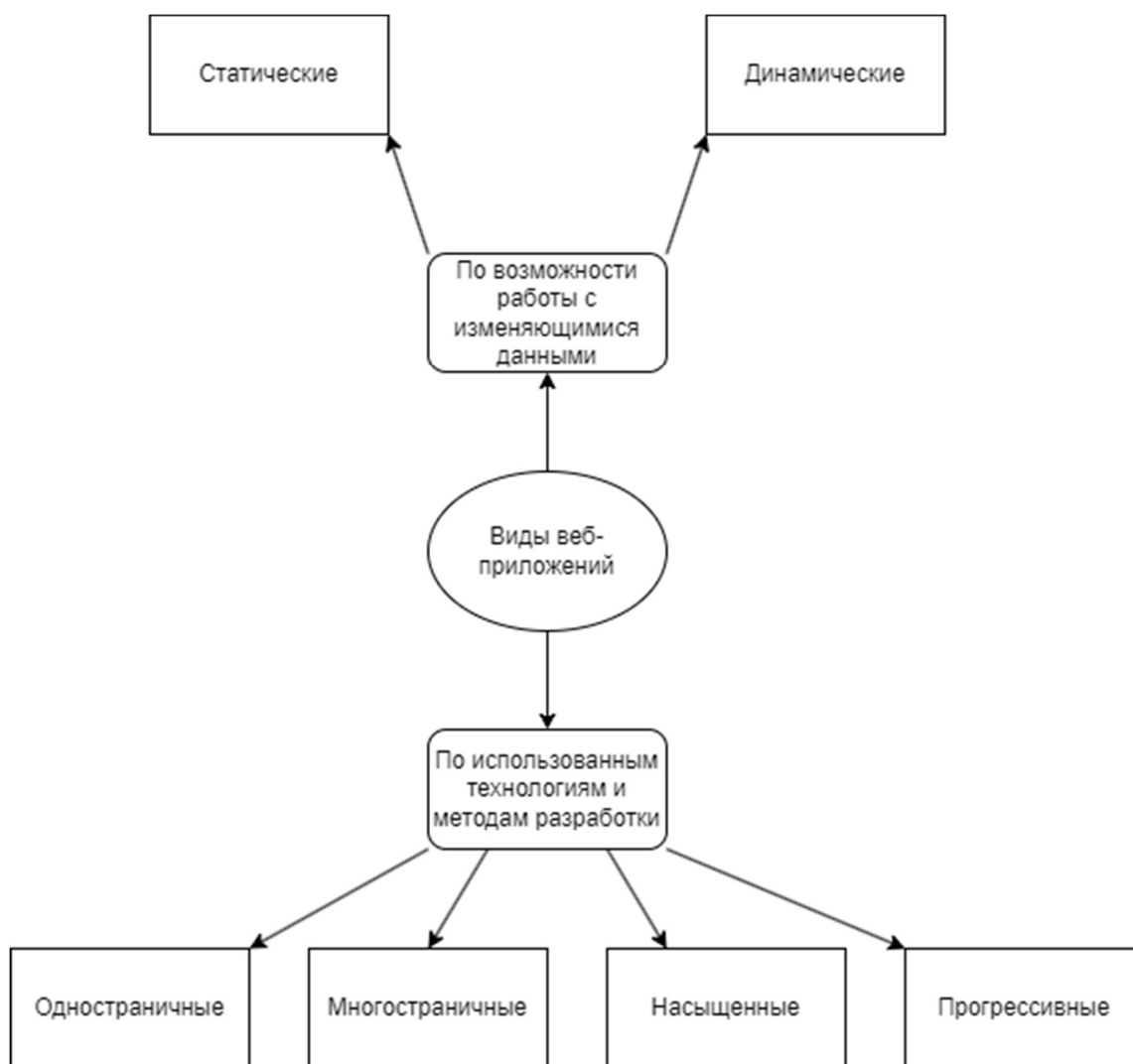


Рисунок 3. Виды веб-приложений

1.2 Атаки на веб-приложения

В первую очередь, скажем о разнице в защите статических и динамических веб-приложений. Динамический контент есть не что иное, как реакция системы на изменение ее внешних условий. Такая реакция может быть как

документированной, т. е. явно описанной или однозначно предполагаемой со стороны системы, так и являться результатом побочных явлений, происходящих в системе. Эти побочные явления могут носить непредсказуемый характер, и именно их принято называть уязвимостями.

Именно динамический контент несет в себе потенциальную угрозу. Сайт, построенный целиком на статическом контенте, состоящий только из статических HTML-страниц, будет неуязвим к атакам на скрипты хотя бы потому, что скриптов там нет. По определению, статическая система никак не реагирует на изменение внешних условий, поэтому можно предположить, что и недокументированная реакция отсутствует. Не следует думать, что статический сайт будет неуязвим абсолютно при всех типах нападений. Например, будет возможна атака через другие сервисы – через уязвимости на сторонних веб-сайтах, физически расположенных на том же сервере, однако являющихся частью другой системы. Кроме того, будут возможны атаки на сам веб-сервер.

DoS-атака – атака, которая старается довести атакуемый сервис до отказа. Целью может быть достижение того, чтобы пользователи не могли взаимодействовать с сайтом, что может нанести ущерб, как материальный так и репутационный его владельцу.

Как правило, чтобы уронить веб-сервис, надо нагрузить его таким количеством запросов, чтобы он просто физически не мог их все обработать. Сделать это злоумышленникам проще всего с нескольких машин, отправляя пачки запросов одновременно. Такая DoS-атака называется распределённой или DDoS-атакой. Мерами защиты от DoS-атаки являются:

- Использование CDN позволяет распространять контент веб-сайта по нескольким серверам, что затрудняет его атаку.
- Использование WAF с защитой от DDoS-атак может отличать реальный трафик от трафика ботов.
- Настройка инфраструктуры с использованием ограничения скорости, дабы установить ограниченное количество запросов в секунду на основе

таких факторов, как диапазон IP-адресов, геолокация или информация в заголовках.

Как было сказано ранее, основой всех уязвимостей в Web-приложениях является динамический контент, то есть различная реакция скриптов на внешние условия. Если под внешними условиями функционирования понять данные, посылаемые клиентом или браузером по протоколу HTTP, то в первую очередь следует рассмотреть уязвимости в скриптах.

PHP-инъекция – один из способов взлома веб-сайтов, работающих на PHP, заключающийся в выполнении постороннего кода на серверной стороне. Уязвимость возникает как следствие недостаточной проверки переменных, используемых внутри таких функций, как `include()`, `require()` и т. п. В результате пользователь может сконструировать специальный запрос, чтобы заставить PHP подключить и выполнить вредоносный PHP-файл.

Рассмотрим приложение, которое передает параметры с помощью запроса GET в функцию `include()` PHP. Например, веб-сайт может иметь URL-адрес, подобный этому:

`http://testsite.com/index.php?page=contact.php`

Значение параметра страницы передается непосредственно в функцию `include()` без проверки. Если вводимые данные не будут проверены должным образом, злоумышленник может выполнить код на веб-сервере, например, следующим образом:

`http://testsite.com/?page=http://evilsite.com/evilcode.php`

Затем скрипт будет запущен на веб-сервере, что позволит реализовать удаленное выполнение кода.

Рассмотрим следующий PHP-код:

```
$myvar = "varname";
```

```
$x = $_GET['arg'];
```

```
eval("$myvar = $x;");
```

Проблема с этим кодом заключается в том, что он использует значение параметра URL `arg` без проверки, непосредственно в функции `eval()`. Предположим, что злоумышленник вводит в параметр `arg` следующие входные данные:

```
http://testsite.com/index.php?arg=1 ; phpinfo()
```

Это приведет к выполнению команды `phpinfo()` на сервере, что позволит злоумышленнику просмотреть конфигурацию системы.

Подобно внедрению кода PHP, **SQL-инъекция** вставляет скрипт SQL - язык, используемый большинством баз данных для выполнения операций запроса - в поле ввода текста. Скрипт отправляется приложению, которое выполняет его непосредственно в своей базе данных. Существует 5 основных типов SQL инъекций:

- Классическая (In-Band или Union-based). SQL-инъекция на основе объединения предполагает использование оператора UNION, который объединяет результаты нескольких операторов SELECT для извлечения данных из нескольких таблиц в единый набор результатов.
- Error-based. Позволяет получать информацию о базе, таблицах и данных на основе выводимых ошибок базы данных. Злоумышленник намеренно вводит некорректные SQL-запросы, вызывающие ошибки в базе данных и с помощью этого получает информацию о структуре базы данных.
- Boolean-based. При внедрении SQL-кода на основе логических значений злоумышленники используют утверждения true/false, чтобы узнать структуру и содержимое базы данных. Наблюдая за тем, как приложение ведет себя в ответ на эти запросы, злоумышленник может постепенно определить необходимые ему данные.
- Time-based. Внедрение SQL-кода, основанное на времени, предполагает использование временных задержек для определения структуры и содержимого базы данных. Внедряя SQL-код, который заставляет базу данных работать определенный промежуток времени перед ответом, злоумышленники могут определить, был ли их запрос истинным или ложным, на основе времени ответа.

- Out-of-Band. Это продвинутый метод, который предполагает использование уязвимости SQL-инъекции для передачи данных по другим каналам, таким как запросы DNS или HTTP. Этот метод полезен, когда приложение не отображает результаты запроса или когда злоумышленнику требуется более скрытный подход. [3]

Диаграмма с видами SQL-инъекций представлена на рисунке 4.

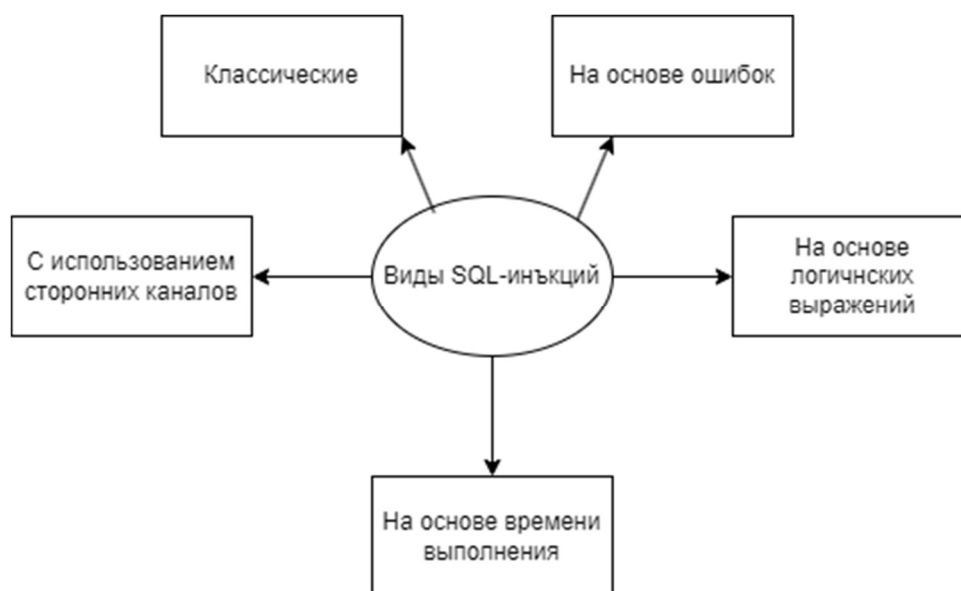


Рисунок 4. Виды SQL-инъекций

XSS – это метод атаки, который заставляет веб-сайт отображать вредоносный код, который затем выполняется в веб-браузере пользователя. XSS-код, как правило, написан на языке HTML/JavaScript, при этом код не выполняется на сервере. Сервер является всего лишь хостом, в то время как атака выполняется в веб-браузере. Злоумышленник использует надежный веб-сайт только в качестве канала для проведения атаки. Предполагаемой жертвой является пользователь, а не сервер. Как только злоумышленник успешно провел атаку, он может совершить множество действий, таких как угон учетной записи, запись нажатий клавиш, взлом внутренней сети и т. д.

Существуют 3 основных вида XSS-атак:

- Хранимые (постоянные). Такие уязвимости возникают, когда злоумышленнику удастся получить доступ к серверу и сохранить на нем вредоносный код. Такой сохраненный скрипт будет активироваться каждый раз, когда пользователь заходит на зараженную страницу. Часто хранимые XSS-уязвимости

можно найти на различных форумах, а также в соцсетях, имиджбордах, блогах. Для их внедрения на ресурс злоумышленники могут использовать обычный текст (например, в виде комментариев), рисунки, гифки и другой размещаемый контент.

- Отраженные (непостоянные). Это наиболее распространенный тип XSS-атак. Их смысл в том, что злоумышленник прячет вредоносный код в заранее подготовленной ссылке, которую потом передает пользователям-жертвам в почтовой рассылке или размещает на веб-странице. Когда пользователь переходит по зараженной ссылке, серверные скрипты исполняют ее код без обработки и возвращают его в виде ответа. Браузер пользователя исполняет зараженный скрипт, а злоумышленник получает cookies жертвы.

- Основанные на DOM-модели. Суть этой уязвимости — в использовании злоумышленником Document Object Model, объектной модели документа. Это программный интерфейс, дающий программам и сценариям доступ к контенту HTML-документов и возможность этим содержимым (а также структурой и оформлением) управлять. XSS-уязвимости на основе DOM-модели могут быть как отраженными, так и хранимыми. Их главная особенность — сама страница сайта или веб-приложения не меняется, но меняется ее отображение в браузере пользователя из-за вредоносных модификаций DOM. [4]

Виды XSS-атак представлены на рисунке 5.



Рисунок 5. Виды XSS-атак

CSRF – это атака, которая вынуждает конечного пользователя выполнять нежелательные действия в веб-приложении, в котором он в данный момент аутентифицирован. Если жертва является обычным пользователем, успешная CSRF-атака может вынудить пользователя выполнить действия, такие как перевод средств, изменение адреса электронной почты и т. д. Если под удар попала учетная запись администратора, CSRF может скомпрометировать все веб-приложение. CSRF - это атака, которая обманом заставляет жертву отправить вредоносный запрос. Она наследует идентификационные данные и привилегии жертвы для выполнения нежелательной функции от ее имени. Для большинства сайтов запросы браузера автоматически включают любые учетные данные, связанные с сайтом, такие как файл cookie сеанса пользователя, IP-адрес, учетные данные домена Windows и так далее. Таким образом, если пользователь в данный момент прошел аутентификацию на сайте, у сайта не будет возможности отличить поддельный запрос, отправленный жертвой, от легитимного запроса. [5]

Данная атака в чем-то похожа на классическую XSS, в которой злоумышленнику необходимо было вынудить жертву перейти по некоторой ссылке на уязвимую страницу. Здесь же необходимо вынудить пользователя перейти на специально подготовленную злоумышленником страницу, на которую был добавлен некоторый код. При выполнении данного кода браузер жертвы совершает некий запрос к другому серверу (допустим под видом загрузки изображения), и тем самым выполняет некие, нужные злоумышленнику, действия. Опасность CSRF в том, что данное поведение браузеров и всего HTTP протокола считается нормальным.

2. Фильтрация веб-приложений

Классические механизмы фильтрации трафика работают на базовых уровнях модели OSI при помощи анализа характеристик отдельных пакетов данных. То есть если IP-адреса в их заголовках либо номера сетевых портов оказываются в «черном списке», то такие пакеты отклоняются. В настройках межсетевого экрана часто блокируются все порты, затем открываются при запросе соединения приложением. Данный метод маскирует при сканировании портов и защищает от грубых попыток несанкционированного доступа, но неэффективен при атаках изнутри. Например, нельзя гарантировать того, что открытые порты используют только легитимные приложения. По этой причине усовершенствованные МЭ создают список правил для конкретных приложений или их компонентов и сверяют их контрольные суммы. [6]

Современный подход к защите веб-приложений основан на использовании специализированных решений – WAF. Обычно они работают с протоколами HTTP/HTTPS, но делают они это на более сложном уровне. Помимо традиционных методов, таких как репутационный анализ IP-адресов и распознавание атак по сигнатурам, они используют и некоторые уникальные подходы. Архитектура WAF позволяет анализировать целиком каждый сеанс связи и выполнять более точный поведенческий анализ. Рассмотрим некоторые способы фильтрации WAF, показанные на рисунке 6, более подробно.



Рисунок 6. Способы фильтрации в WAF

2.1 Фильтрация на основе регулярных выражений и сигнатур

Регулярное выражение – это формула для сопоставления строк, которые соответствуют определенному шаблону. Регулярные выражения могут состоять из обычных символов и специальных символов. Обычные символы включают заглавные и строчные буквы и цифры. В простейшем случае регулярное выражение выглядит как стандартная текстовая строка.

Большинство существующих WAF основаны на правилах, основанных на регулярных выражениях. Для их создания разработчиком WAF изучаются некоторые известные наборы атак и, следовательно, определяются ключевые синтаксические конструкции, наличие которых может быть признаком осуществления атаки. На основе полученных результатов пишутся регулярные выражения, которые могут находить такие конструкции. Например, анализируя HTTP-заголовки, WAF может блокировать ответ и тем самым предотвращать утечку информации с сервера или, в качестве альтернативы, выдавать предупреждение.

Однако у такого подхода есть ряд недостатков. Диапазон применимости регулярного выражения ограничен одним запросом, а зачастую даже конкретным параметром запроса, что, очевидно, снижает эффективность таких запросов. Во-вторых, синтаксис регулярных выражений допускает замену эквивалентных конструкций и использование различных символов, что приводит к ошибкам при создании таких правил.

Например, для предотвращения SQL-инъекции, WAF обнаруживает и сопоставляет ключевые слова SQL, специальные символы, операторы и символы комментариев:

- Ключевые слова SQL: union, select, from, as, asc, desc, order by, dba_user, case, и тому подобное
- Специальные символы: ';; ()
- Математические операторы: ±, *, /, % и |
- Операторы: =, >, <, >=, <=, !=, +=, и -=
- Символы комментариев: – или /**/ [7]

Похожим способом работает фильтрация на основе сигнатур. Правила WAF, основанные на сигнатурах, сопоставляют входящий трафик с базой данных известных сигнатур атак. Эти сигнатуры обычно представляют собой текстовые строки или регулярные выражения, представляющие вредоносные шаблоны. При обнаружении совпадения WAF блокирует запрос.

Правила, основанные на сигнатурах, очень эффективны при блокировании известных атак. Они также относительно просты в понимании и обслуживании. Однако правила, основанные на сигнатурах, могут не обнаружить новые атаки.

2.2 Фильтрация на основе поведения

Поведенческий анализ строится на машинном обучении. Это позволяет обнаружить аномалии в поведении веб-приложений. Для атак на веб-приложения злоумышленники активно используют уязвимости нулевого дня, что делает бесполезными сигнатурные методы анализа. Вместо этого можно анализировать сетевой трафик и системные журналы для создания модели нормального функционирования приложения, и на основе этой модели выявлять аномальное поведение системы.

В WAF используются две основные модели безопасности: негативная модель безопасности и позитивная модель безопасности. WAF, использующий негативную модель безопасности пропускает весь трафик, но если он соответствует определенным правилам, то трафик блокируется. Если трафик не соответствует параметрам, отрицательная модель безопасности пропускает его. Негативная модель безопасности обычно реализуется в WAF, основанных на сигнатурах. WAF, использующий позитивную модель безопасности, наоборот пропускает трафик, если он соответствует политикам. Если трафик не соответствует политике, он блокируется. Негативная и позитивная модели безопасности имеют свои преимущества и недостатки. Например, негативная модель безопасности может оказаться недостаточной, поскольку хакеры могут разрабатывать атаки в обход существующих политик безопасности. И наоборот, позитивная модель безопасности может потребовать тщательного планирования, а ее реализация основана на глубоком знании приложения и может быть негибкой.

Эволюция методов искусственного интеллекта привлекла внимание к применению машинного обучения для обнаружения аномалий с помощью WAF. Современные методы включают искусственные нейронные сети, которые обучаются на различных параметрах запросов. [8]

HTTP-запросы, включают в частности:

- Стартовая строка: она состоит из трех элементов. Первый – это HTTP-метод (GET, PUSH, POST, ...). Второй соответствует цели запроса и включает URL-адрес или полный адрес протокола, порта и домена. Наконец, третий – это версия HTTP, которая определяет структуру сообщений и версию ожидаемого ответа.

- Заголовки: в запросах может передаваться большое число различных заголовков, но все их можно разделить на три категории:

1. Общего назначения, которые применяются ко всему сообщению целиком.

2. Заголовки запроса уточняют некоторую информацию о запросе, сообщая дополнительный контекст или ограничивая его некоторыми логическими условиями.

3. Заголовки представления, которые описывают формат данных сообщения и используемую кодировку. Добавляются к запросу только в тех случаях, когда с ним передается некоторое тело.

- Тело: не у каждого HTTP-метода предполагается наличие тела. Так, например, методам вроде GET, HEAD, DELETE, OPTIONS обычно не требуется тело. Некоторые виды запросов могут отправлять данные на сервер в теле запроса: самый распространенный из таких методов – POST.

HTTP-ответы состоят из:

- Строка статуса: состоит из трех элементов. Первый – это версия протокола HTTP, описывающая действие, которое необходимо выполнить. Второй соответствует коду состояния, который указывает на успех или неудачу запроса и, наконец, третий – это текст состояния, который представляет собой краткое текстовое описание того, что означает код состояния.

- Заголовки: они имеют ту же структуру, что и в HTTP-запросах.
- Тело: этот блок отображается не для всех ответов.

Набор данных для обучения WAF включает обычные и аномальные HTTP-запросы, представляющие собой распространённые веб-атаки. Впоследствии, WAF будет работать и блокировать или разрешать трафик на основе результатов своего обучения.

2.3 Практическая работа

В качестве практического задания для НИР была написана программа `nirs.py`, чей листинг приведен в приложении А, а также на Github (https://github.com/AntonGrigorev/NIRS_10_Semestr). Программа получает HTTPS-трафик и, на основании правил, решает – разрешен трафик или нет.

HTTPS – расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. HTTPS не является отдельным протоколом. Это обычный HTTP, работающий через зашифрованные транспортные механизмы SSL и TLS.

Первоначально, необходимо было создать простой сервер для принятия HTTPS-запросов по порту 4443. Для создания HTTPS-сервера использовалась библиотека OpenSSL. OpenSSL – это криптографический инструментальный, реализующий сетевые протоколы SSL и TLS и соответствующие им стандарты криптографии. Сгенерированные в OpenSSL ключи могут использоваться для шифрования различных данных, в том числе и в HTTPS протоколе, где используется асимметричное шифрование. Пример использования OpenSSL для создания самоподписанного сертификата:

1. `openssl genpkey -algorithm RSA -out private.key`
2. `openssl req -new -x509 -key private.key -out certificate.crt -days 365`

Самоподписанный SSL-сертификат – это сертификат открытого ключа, изданный и подписанный тем же лицом, которое он идентифицирует. Такие сертификаты не являются безопасными, их подтверждением должен заниматься специальный центр сертификации. Но их допустимо использовать в целях отладки и разработки веб-приложений.

Алгоритм для обработки запросов представлен на рисунке 7.

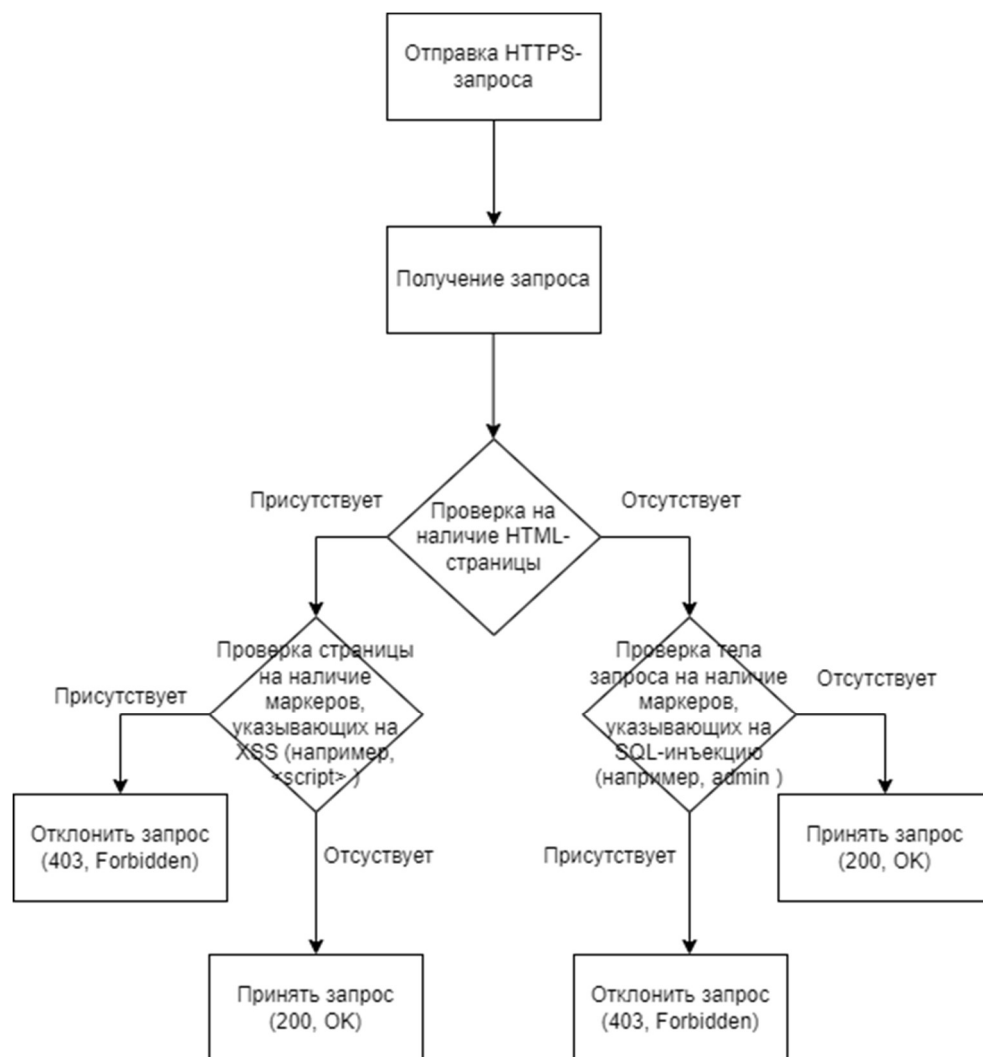


Рисунок 7. Алгоритм фильтрации

При появлении подозрительных запросов, информация о них заносится в лог-файл в следующем формате [Время, Статус, Адрес]:

2024-09-15 22:30:16,937 Status: BLOCK, Src Adr: 10.0.2.15:4443

Для отправки запросов локально использовалась утилита cURL. cURL – кроссплатформенная служебная программа командной строки, позволяющая взаимодействовать с множеством различных серверов по множеству различных протоколов с синтаксисом URL. Программа cURL может автоматизировать передачу файлов или последовательность таких операций. Например, это хорошее средство для моделирования действий пользователя в веб-браузере.

С помощью cURL был сформирован скрипт с 15 запросами с различными типами данных для отправки на сервер. Для отправки запросов использовались команды следующего вида:

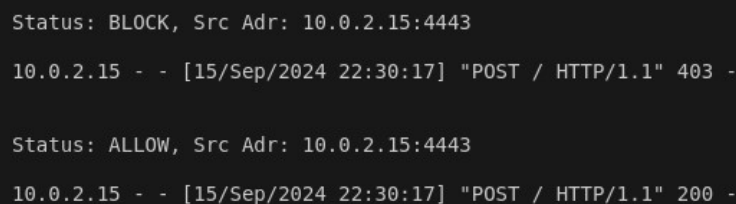
```
curl -x post http://10.0.2.15:4443 -d "param1=value1 param2=value2"
```

```
curl -k -F "web=@/home/user/tests/malic.html" https://10.0.2.15:4443
```

Запросы разделены на 3 категории:

1. Простые POST-запросы с параметрами, отправленными в теле;
2. Простые HTML-страницы (небольшие файлы с полями ввода);
3. Сложные HTML-страницы (крупные файлы с кнопками и дизайном);

Результат работы программы приведен на рисунке 8.



```
Status: BLOCK, Src Addr: 10.0.2.15:4443
10.0.2.15 - - [15/Sep/2024 22:30:17] "POST / HTTP/1.1" 403 -

Status: ALLOW, Src Addr: 10.0.2.15:4443
10.0.2.15 - - [15/Sep/2024 22:30:17] "POST / HTTP/1.1" 200 -
```

Рисунок 8. Отправка запросов на сервер

Время обработки различных запросов было измерено и показано на графике на рисунке 9. Исходя из него можно сделать вывод, что в среднем быстрее всего фильтрацию проходят небольшие POST-запросы, у которых проверяется их тело. Наибольшее время ушло на парсинг крупных HTML-страниц со множеством элементов.

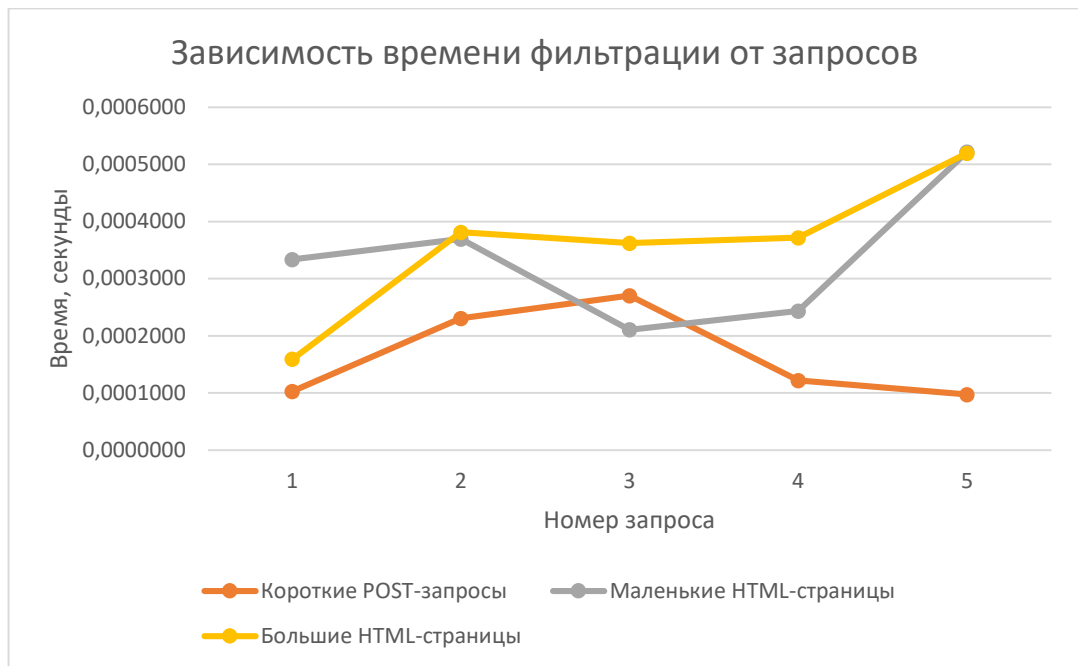


Рисунок 9. Время обработки различных запросов

Заключение

Веб-приложения в настоящее время получили широкое распространение во всех сферах жизнедеятельности человека и, в связи с этим, возникла необходимость и в их защите от злоумышленников. Для этого активно используются WAF. При этом способы фильтрации в них, помимо, например, обычного анализа IP-адресов и портов, включает и более сложные техники.

В данной работе была дана классификация веб-приложений по способу их реализации, в общем случае были описаны статические и динамические веб-приложения, а в частности – SPA, MPA и т. д. Также были рассмотрены основные виды атак на веб-приложения, такие как PHP- и SQL-инъекция, XSS и CSRF.

Помимо этого, были рассмотрены способы фильтрации веб-приложений с использованием WAF на основе регулярных выражений, сигнатур и машинного обучения. Также в рамках НИР была реализована программа, фильтрующая HTTPS-запросы по простым правилам. Ее листинг доступен на Github:

https://github.com/AntonGrigorev/NIRS_10_Semestr

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Achyut Godbole, Atul Kahate; Web Technologies: TCP/IP, Web/ Java Programming, and Cloud Computing. //URL: <https://pdfdrive.to/dl/web-technologies-tcpip-architecture-and-java-programming-0> (дата обращения: 15.07.24)
2. Emmit A. Scott, Jr.; SPA Design and Architecture: Understanding single-page web applications //URL: <https://www.programmer-books.com/wp-content/uploads/2019/10/SPA-Design-and-Architecture.pdf> (дата обращения: 15.07.24)
3. STACKHAWK: What is SQL Injection? // URL: <https://www.stackhawk.com/blog/what-is-sql-injection/#:~:text=There%20are%20three%20main%20types,and%20Union%2Dbased%20SQLi.%20Error%2Dbased%20SQLi> (дата обращения: 21.07.24)
4. Seth Fogie, Jeremiah Grossman, Robert Hansen, Anton Rager, Petko D. Petkov; XSS Attacks - Exploits and Defense. // URL: https://www.academia.edu/36765412/XSS_Attacks_Exploits_and_Defense_pdf (дата обращения: 21.07.24)
5. OWASP, Cross Site Request Forgery (CSRF); // URL: [https://owasp.org/www-community/attacks/csrf#:~:text=Cross%2DSite%20Request%20Forgery%20\(CSRF,which%20they're%20currently%20authenticated](https://owasp.org/www-community/attacks/csrf#:~:text=Cross%2DSite%20Request%20Forgery%20(CSRF,which%20they're%20currently%20authenticated). (дата обращения: 28.07.24)
6. И. П. Леонькова, А. И. Ларин; Анализ методов обеспечения безопасности веб-приложений. // URL: <https://moluch.ru/archive/250/57416/> (дата обращения: 21.07.2024).
7. Huawei Cloud, How Does WAF Detect SQL Injection, XSS, and PHP Injection Attacks?; // URL: https://support.huaweicloud.com/intl/en-us/waf_faq/waf_01_0457.html (дата обращения: 28.07.24)
8. Simon Applebaum. Tarek Gaber, Ali Ahmed; Signature-based and Machine-Learning-based Web Application Firewalls: A Short Survey // URL: https://www.researchgate.net/publication/353249413_Signature-based_and_Machine-Learning-based_Web_Application_Firewalls_A_Short_Survey (дата обращения: 28.07.24)

ПРИЛОЖЕНИЕ А. Листинг программы nirs.py

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import logging
import ssl
import time

class S(BaseHTTPRequestHandler):
    def _set_response(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

    def _deny_response(self):
        self.send_response(403)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

    def do_GET(self):
        logging.info("GET request,\nPath: %s\nHeaders:\n%s\n", str(self.path), str(self.headers))
        self._set_response()

    def do_POST(self):
        start_time = time.time()
        content_length = int(self.headers['Content-Length'])
        post_data = self.rfile.read(content_length)
        if(str(self.headers['Content-Type']).find('multipart/form-data') != -1):
            injection = ['<script>', 'data:']
            for obj in injection:
                if post_data.decode('UTF-8').find(obj) != -1:
                    log = "Status: BLOCK, Src Adr: " + str(self.headers['Host'])
                    print(log)
                    logging.warning(log)
                    end_time = time.time()
                    elapsed_time = str(round(end_time - start_time, 7))
                    f = open("timefile.txt", "a")
```

```

        f.write(elapsed_time + '\n')
        f.close()
        self._deny_response()
        print('\n')
        break
    else:
        log = "Status: ALLOW, Src Adr: " + str(self.headers['Host'])
        print(log + '\n')
        logging.info(log)
        end_time = time.time()
        elapsed_time = str(round(end_time - start_time, 7))
        f = open("timefile.txt", "a")
        f.write(elapsed_time + '\n')
        f.close()
        self._set_response()
        print('\n')
        break
    else:
        sql = ['admin', 'database']
        for obj in sql:
            if post_data.decode('UTF-8').find(obj) != -1:
                log = "Status: BLOCK, Src Adr: " + str(self.headers['Host'])
                print(log + '\n')
                logging.warning(log)
                end_time = time.time()
                elapsed_time = str(round(end_time - start_time, 7))
                f = open("timefile.txt", "a")
                f.write(elapsed_time + '\n')
                f.close()
                self._deny_response()
                print('\n')
                break
            else:
                log = "Status: ALLOW, Src Adr: " + str(self.headers['Host'])
                print(log + '\n')

```

```

        logging.info(log)
        end_time = time.time()
        elapsed_time = str(round(end_time - start_time, 7))
        f = open("timefile.txt", "a")
        f.write(elapsed_time + '\n')
        f.close()
        self._set_response()
        print('\n')
        break

def run(server_class=HTTPServer, handler_class=S, port=4443):
    logging.basicConfig(level=logging.INFO)
    server_address = ("", port)
    httpd = server_class(server_address, handler_class)
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    context.load_cert_chain('/home/user/conf/cert.pem', '/home/user/conf/key.pem')
    httpd.socket = context.wrap_socket(httpd.socket, server_side=True)
    #logging.info('Starting httpd...\n')
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass
    httpd.server_close()
    #logging.info('Stopping httpd...\n')

if __name__ == '__main__':
    from sys import argv
    logging.basicConfig(filename='py_log.log', filemode='w', format="%%(asctime)s %(message)s")
    with open('timefile.txt', 'r+') as f:
        f.truncate(0)
    if len(argv) == 2:
        run(port=int(argv[1]))
    else:

```

run()