Использование сканера уязвимостей ZAP для тестирования веб-приложений

О.Р. Лапонина, С.А. Малаховский

Аннотация **– В статье рассмотрены инструменты для тестирования безопасности и поиска различных типов уязвимостей в веб-приложениях. Рассмотрены основные функциональные возможности и основные компоненты инструментальных средств тестирования уязвимостей W3AF, Burp Suite и OWASP ZAP. Также рассмотрены вопросы развертывания окружения для тестирования безопасности приложений и реализованы различные варианты применения сканера уязвимостей OWASP ZAP. Тестовый поиск уязвимостей проводился в приложении OWASP Mutillidae. Mutillidae реализует все типы уязвимостей, перечисленные в OWASP Top 10 2007, 2010 и 2013. В приложении представлен программный код для работы со сканером уязвимостей через Java API.**

Ключевые слова **– сканеры уязвимостей, тестирование веб-приложений, OWASP ZAP, среда тестирования.**

I. Введение. Инструменты тестирования безопасности и поиска уязвимостей

Тестирование безопасности невозможно без использования специализированных инструментов и руководств. Таких инструментальных средств и руководств существует достаточно большое количество.

Одним из таких инструментов являются сканеры безопасности, которые позволяют проводить автоматическое исследование информационной безопасности. Они представляют собой программные и/или аппаратные комплексы, позволяющие выполнять мониторинг и диагностику сетевых устройств и ресурсов, а также сканирование сети, компьютеров и сетевых приложений на предмет обнаружения возможных уязвимостей [1], [2].

Примерами могут служить:

- *XSpider, OpenVAS, NMAP, Metasploit* — сетевые сканеры для тестирования сетевой инфраструктуры.
- *Acunetix Web Vulnerability Scanner, Nikto, OWASP Live CD* — набор инструментов для тестирования безопасности веб-приложений.

Статья получена 11 июля 2017.

О.Р. Лапонина – МГУ имени М.В. Ломоносова (email: laponina@oit.cmc.msu.ru).

С.А. Малаховский – МГУ имени М.В. Ломоносова (email: malakhovskysa@mail.ru).

Для ручного тестирования могут быть использованы анализаторы сетевого трафика — снифферы, такие как *WinDump, WireShark* и другие. Для тестирования веб-приложений с помощью отправки различных HTTP-запросов можно использовать *FireBug, Burp Suite, OWASP ZAP*.

Особый интерес представляет тест-фреймворк *Selenium*, который изначально предназначен для написания тестов с использованием графического интерфейса. С помощью простого и понятного интерфейса он позволяет создавать макросы для манипуляции браузером и моделирования действий пользователя. С его помощью возможно создавать скрипты, описывающие действия атакующего, для поиска или использования уязвимостей в веб-приложениях, начиная от простого перебора паролей к форме аутентификации до проверки полей ввода на наличие SQL-инъекции или XSS.

В настоящее время существует большое число организаций, занимающихся сетевой безопасностью. Одной из наиболее авторитетных организаций является *Open Web Application Security Project (OWASP)* — онлайн сообщество, которое занимается созданием свободно распространяемых статей, методологий, инструментов и технологий в области безопасности веб-приложений. Данная организация была создана в сентябре 2001 года.

В рамках сообщества OWASP ведется работа над более чем 130 проектами, каждый из которых определяется набором связанных задач, определенным планом развития и командой разработчиков. Все проекты направлены на создание инструментов и документации об информационной безопасности в следующих категориях:

- защита от атак и уязвимостей в веб-приложениях;
- обнаружение атак и недостатков в уже существующих системах;
- проектирование и реализация программного обеспечения с учетом требований информационной безопасности на протяжении всего жизненного цикла ПО.

Приведем наиболее популярные проекты OWASP.

OWASP Top Ten [6]. Цель проекта — увеличение информированности о существующих и наиболее часто используемых уязвимостях в веб-приложениях. На данный проект ссылается множество стандартов, инструментов и организаций, включая MITRE, PCI DSS, DISA,

FTC, и множество других. Проект существует с 2003 года. На данный момент доступна версия *release candidate* 2017 года.

OWASP Development Guide. Руководство по разработке, освещающее вопросы безопасности приложений с практическими советами и примерами кода на J2EE, ASP.NET и PHP.

OWASP Testing Guide [7]. Это руководство включает практическую основу для тестирования проникновений, а также описывает технологии тестирования наиболее распространенных проблем с безопасностью в веб-приложениях и веб-сервисах.

OWASP Code Review Guide [8]. Руководство по обзору и рефакторингу кода — является одним из ключевых продуктов, позволяющее бороться с проблемами в безопасности программного обеспечения.

OWASP Application Security Verification Standard [5]. Стандарт для проведения проверок уровня безопасности приложений.

OWASP ZAP Project: The Zed Attack Proxy (ZAP). Инструмент тестирования проникновений, служащий для нахождения уязвимостей в веб-приложениях. Он разработан для использования людьми с различным опытом в сфере безопасности.

Webgoat: учебное, ненадежное веб-приложение, созданное OWASP как руководство по написанию безопасного кода. Вместе с приложением поставляется учебник и набор различных курсов, рассказывающих как использовать информацию об уязвимостях для написания безопасного кода.

Все инструменты и руководства OWASP свободно распространяемы и используются многими крупными организациями и частными лицами.

В данной статье:

- Проанализированы существующие инструментальные средства тестирования веб-приложений.
- Реализованы различные варианты использования инструментального средства анализа уязвимостей ZAP для тестирования безопасности приложений.

II. Инструменты тестирования W3AF, Burp Suite и OWASP ZAP

Следует проводить тестирование не только самого веб-приложения, но и всего окружения, включая сервер приложений, базу данных, операционную систему и сетевые сервисы.

Существует множество различных инструментов позволяющих производить тестирования безопасности с помощью поиска уязвимостей, например W3AF, Burp Suite, OWASP ZAP.

W3AF (*Web Application Attack and Audit Framework*) — свободно распространяемый фреймворк с открытым исходным кодом, предназначенный для поиска и эксплуатации уязвимостей в веб-приложениях. Он состоит из ядра и плагинов. Ядро управляет главным процессом и предоставляет сервисы для работы плагинов, с помощью которых можно находить и использовать уязвимости в веб-приложениях. Плагины могут взаимодействовать друг с другом с помощью специализированной базы знаний. Все плагины можно разделить на 9 категорий.

- Плагины для поиска точек входа в приложение, которые формируют карту запросов тестируемого приложения. Примером плагина является модуль WebSpider.
- Аудит-плагины, использующие точки входа в приложения для обнаружения различных типов уязвимостей, таких как XSS, SQL-инъекция и т.п.
- *Grep*-плагины, которые обрабатывают все HTTP-запросы и ответы с целью поиска необходимых данных, например внутренних IP-адресов, адресов электронной почты, а также потенциально опасный JavaScript-код.
- *BruteForce*-плагины, позволяющие производить подбор паролей для различных механизмов аутентификации, например *HTTP Basic* или формы для входа.
- Атакующие плагины, использующиеся для эксплуатации найденных уязвимостей.
- *Mangle*-плагины, предназначенные для изменения на лету отправляемых HTTP-запросов и ответов.
- Плагины уклонения (*evasion*), использующиеся для обхода различных IDS.
- Плагины вывода, позволяющие формировать отчеты о результатах работы, используя общую базу знаний.
- Плагины авторизации, управляющие пользовательскими сессиями, а именно, выполняющие вход в веб-приложении, поддерживающие сессии в активном состоянии и осуществляющие корректный выход по окончанию работы.

К достоинствам W3AF относится то, что являясь мощным и бесплатным фреймворком для тестирования безопасности веб-приложений, его функциональность можно расширять с помощью различных плагинов. Он позволяет проводить как ручной, так и автоматический поиск уязвимостей. Доступно использование как графического интерфейса (UI), так и интерфейса командной строки (CLI). Работа с W3AF из других приложений возможна как через CLI, так и через REST API. Однако он обладает не очень большой базой уязвимостей, в основном, нацеленной на обнаружение XSS и SQL-injection. К минусам также можно отнести необходимость разработки плагинов на Python.

Одним из самых популярных сканеров безопасности является ***Burp Suite*** — это

интегрированная платформа для исследования безопасности веб-приложений, предусматривающая функционирование, как в ручном, так и в автоматических режимах. Принцип работы основан на перехвате и обработке всех HTTP-запросов и ответов. Также возможность установки SSL-сертификата для работы через HTTPS-соединение.

Сам сканер состоит из набора связанных утилит, определяющих его функциональность.

Существует прокси модуль, работающий по протоколам HTTP и HTTPS. Он позволяет перехватывать, просматривать и обрабатывать трафик, идущий от браузера к серверу и обратно. Перечислим основные утилиты.

- Модуль *Spider* позволяет в автоматическом режиме исследовать архитектуру веб-приложения. Автоматический сканер уязвимостей доступен только в Professional версии.
- *Intruder* — утилита для атаки методом прямого перебора, позволяющая автоматизировать подбор пароля, перебор идентификаторов и т.п.
- *Sequencer* — утилита для анализа случайных чисел, генерируемых в приложении, определение алгоритмов для генерации.
- *Decoder* — утилита для автоматического и ручного преобразования данных веб-приложения.
- *Comparer* — утилита для сравнения данных и определения их различия.
- *Extender* — утилита для работы с расширениями. Можно использовать как собственной разработки, так и из магазина приложений BApp Store.

Burp Suite является очень популярным инструментом. Из анализа статистики bu*g*-репортов видно, что в подавляющем большинстве случаев для обнаружения проблем с безопасностью используется данный инструмент. Он обладает одной из самых больших баз уязвимостей. Однако бесплатная версия хоть и является полноценным инструментом, но значительно ограничена по функциональности. Например, в ней отсутствует сканер уязвимостей и установлено ограничение на количество отсылаемых запросов. Есть ограничения и по использованию дополнений из BApp Store.

OWASP ZAP — это активный сканер безопасности с богатым функционалом и открытым исходным кодом. Он обладает понятным пользовательским интерфейсом, имеет низкий порог входа для начала использования и не требователен к опыту в тестировании безопасности приложений. Поскольку он разрабатывается в OWASP, то в нем поддерживаются все уязвимости, описанные в многочисленных руководствах OWASP по тестированию.

Он предоставляет аналогичную функциональность, что и сканеры безопасности, рассмотренные выше. Например, модули паук и активной атаки, позволяющие исследовать приложения на уязвимости. Модуль прокси для исследования и модификации запросов и ответов, при взаимодействии браузера с удаленным веб-сервером. При этом возможны режимы пассивной атаки, когда трафик только просматривается, что позволяет найти ограниченное число хорошо известных уязвимостей, и активный режим, когда запросы модифицируются, и анализируется ответ, приходящий от сервера. Добавление различных правил для этих режимов позволяет тонко настраивать ZAP для снижения ложно-положительных срабатываний. Также есть механизм для управления авторизацией и сессиями пользователя. Расширение функциональности ZAP возможно с использованием ZAP Marketplace, которое содержит значительное количество расширений.

OWASP ZAP имеет следующие особенности:

- является одним из самых популярных инструментов для тестирования безопасности;
- содержит большую базу уязвимостей для автоматического сканирования;
- полнофункциональная версия доступна под бесплатной лицензией;
- имеет открытый исходный код;
- обладает кроссплатформенностью;
- предоставляет API с широкими возможностями, позволяющий использовать его на разных языках программирования.

Отдельно необходимо отметить интеграцию ZAP со средой *Continuous Integration* (*CI*), например, с помощью *Jenkins ZAPRoxy Plugin*, что позволяет проводить исследования безопасности как только код передается в систему контроля версий. Предусмотрены несколько вариантов использования:

- Регрессионные тесты, основанные на прокси-модуле ZAP. Например, можно запускать *Selenium* тесты, для исследования сценариев поведения пользователей веб-приложения.
- Использовать модуль активной атаки для части кода, не проанализированном регрессионными тестами.
- Запускать активный поиск уязвимостей для имитации атаки на веб-приложение.

Таким образом, OWASP ZAP является мощным инструментом тестирования безопасности веб-приложений.

III. Среды и приложения для проведения тестирования

В настоящее время существует большое количество приложений, созданных специально для демонстрации типичных уязвимостей. Их целью является предоставление площадки для проверки

работы различных сканеров безопасности, а также обучению нахождению и использованию различных типов уязвимостей. Условно их можно разделить на онлайн и офлайн версии, которые хотя и требуют дополнительных действий по инсталляции, но предоставляют большую свободу по поиску уязвимостей и применению различных инструментов тестирования безопасности. Также существует большое количество руководств, описывающих наиболее часто встречающиеся уязвимости. Примерами таких руководств являются OWASP TOP 10 или OWASP Testing Guide.

Web Security Dojo — свободно распространяемый контейнер с обучающей средой для тестирования инструментов проникновения веб-приложений, основанный на дистрибутиве *Linux Ubuntu v.16.04 LTS* и оптимизированный для работы на виртуальных машинах. Включает в себя большой спектр как тестовых приложений, так и инструментов для тестирования. Например, DVWA, WebGoat, Insecure Web App, Burp Suite (бесплатная версия), W3AF, Arachnid, Metasploit, Zed Attack Proxy, WebScarab и множество других.

Значительным преимуществом является возможность разворачивания тестового окружения на виртуальной машине, при этом отсутствует необходимость в подключении к локальной сети или интернет. Данное тестовое окружение широко используется в различных университетах.

Похожий контейнер есть и у OWASP – ***OWASP Web Testing Framework***, который содержит набор простых в использовании инструментов безопасности и документацию, создаваемую OWASP. Он предназначен для поддержки продуктов OWASP и представляет собой образ виртуальной машины, который можно запускать на VMWare, VirtualBox и Parallels. Также возможен запуск как live-cd.

Основные цели данного проекта:

- Предоставить витрину для многочисленных проектов OWASP.
- Предоставить лучшие свободно распространяемые инструменты в доступном окружении.
- Гарантировать, что предлагаемые инструменты не требуют значительных усилий в использовании.
- Расширять и улучшать поставляемую вместе с дистрибутивом документацию.
- Актуализировать многочисленные инструменты для тестирования безопасности согласно руководству по тестированию OWASP.

Приложений, реализующих уязвимости существует большое количество. Все они распространяются бесплатно и имеют открытый исходный код. Основная цель их применения — обучение поиску и исправлению ошибок и уязвимостей в веб-приложениях. В рамках данной статьи рассматриваются OWASP Mutillidae и OWASP WebGoat. Они являются полноценными веб-приложениями, построенными по клиент-серверной архитектуре и использующие базы данных.

Mutillidae реализует все типы уязвимостей, перечисленные в OWASP Top 10 2007, 2010 и 2013. Для каждой из уязвимостей есть несколько вариантов реализации. В состав приложения входит подробная документация с описанием уязвимостей, их поиска и борьбы с ними, а также исходный код с примерами, содержащими рассматриваемые уязвимости. Mutillidae написан на PHP и использует Mysql. Для его запуска необходим какой-либо веб-сервер, например Apache.

В данном приложении нет определенных заданий по поиску уязвимостей, и на одной и той же странице могут быть приведены примеры разных типов уязвимостей. Например, на странице просмотра документов *document-viewer.php* реализуются XSS, HTML-инъекции, инъекции в журналы приложения, компрометация HTTP-параметров и др.

WebGoat — уязвимое приложение, активно развиваемое OWASP для обучения в области информационной безопасности. Оно состоит из набора уроков, посвященных разным типам уязвимостей. В каждом уроке определена цель, которую необходимо достичь, выполняя определенные действия, а также приведено подробное описание, как это сделать. Задания разнообразны, например, найти SQL-инъекцию и получить скрытую информацию или внедрить защиту от уязвимости в исходный код приложения. Приложение WebGoat написано на Java, использует внутреннюю базу данных MongoDB, в качестве сервера приложений используется встроенный Apache Tomcat 7, то есть для запуска приложения не требуется дополнительного веб-сервера, а необходима только установленная JVM.

Разворачивание инфраструктуры и установка этих приложений может отнять много времени. Было бы удобно использовать какие-либо инструменты для автоматизации установки этих приложений на тестовый стенд, например, с помощью инструментов для автоматизации сборки приложений (Ant, Maven, Phing).

В качестве альтернативного варианта можно использовать Docker — платформу для управления приложениями в среде виртуализации. С помощью него достаточно легко создать образ какого-либо приложения со всей его инфраструктурой в виде контейнера, который может быть перенесён на любой стенд.

Docker состоит из сервера контейнеров и клиентских расширений, позволяющих с помощью интерфейса командной строки управлять запуском и работой с контейнерами. Также доступен REST

API, делающий возможным работу с контейнером из пользовательских приложений.

Сервер контейнеров предоставляет запускаемому внутри него образу операционной системы с необходимыми приложениями собственную файловую систему, разделяемые ресурсы и виртуальные сетевые интерфейсы, обеспечивая полную изоляцию запускаемых на узле контейнеров на уровне процессов и файловой системы.

Для большинства известных приложений уже существуют Docker-образы, но в случае их отсутствия подготовить такой образ не сложно собственными силами. При этом нет необходимости разворачивать всю инфраструктуру с нуля. Такой образ собирается как конструктор, например если используется, база данных Mysql, то ее достаточно просто подключить к собираемому образу через Docker-хранилище. Используемые в рамках данной работы Docker-образы перечислены в таблице 1.

Таблица 1. Используемые Docker-образы

Приложение	Название образа
ZAP	owasp/zap2docker-stable
WebGoat	webgoat/webgoat-8.0
Mutillidae	citizenstig/nowasp

Основной образа является dockerfile, который описывает на основе какого существующего образа и с помощью каких команд его собирать. Подготавливать такой образ можно в ручном режиме. В случае, когда приложение находится в активной фазе разработки, и его необходимо постоянно пересобирать и устанавливать на тестовые стенды, можно использовать плагины для систем автоматизации сборок, например com.spotify:docker-maven-plugin. Необходимо только указать с какими параметрами и на каком этапе сборки его запускать.

IV. Поиск уязвимостей в ручном режиме

В рамках данной статьи проводился тестовый поиск уязвимостей в приложении OWASP Mutillidae. В нем присутствует достаточно большое число специально реализованных уязвимостей. Так, например, на странице входа в приложение /login.php существует возможность обхода аутентификации с помощью SQL-инъекции, используя поля ввода логина и пароля пользователя, также можно выполнить XSS, обойти проверку правильности ввода с помощью JavaScript и др.

Например, если ввести в поле *username* символ одинарной кавычки, то на экране отобразится сообщение об ошибке, в котором будет присутствовать кроме служебной информации еще и SQL-запрос, проанализировав который легко составить вредоносную строку:

```
1' OR '1'='1'; --,
```

Такой ввод в поле логина или пароля которой, приведет ко входу в приложение.

Таким образом, на странице входа в приложение обнаружено две уязвимости: небезопасная конфигурация приложения, заключающаяся в выводе избыточной служебной информации неавторизированному пользователю и SQL-инъекция, приводящая к успешной аутентификации в приложении.

Эти уязвимости можно найти, используя различные инструменты поиска уязвимостей, например, OWASP ZAP.

Для тестирования безопасности можно использовать следующие модули:

- Модуль паук, позволяющий искать в тестируемом приложении доступные адреса URL, которые будут служить точками входа для дальнейших атак.
- Модуль активной атаки, который позволяет осуществлять поиск вероятных уязвимостей с помощью существующей базы данных для последующего анализа.
- Модуль перебора значений (fuzzer), который может изменять в GET и POST-запросах значения выбранных параметров на заранее определенные. Областью его использования может быть атака на подбор пароля по словарю или поиск уязвимостей типа SQL-инъекции по заранее заданным сигнатурам.
- Прокси-модуль для перехвата запросов и ответов к веб-приложению. Он позволяет искать и эксплуатировать определенные уязвимости в ручном режиме. Например, в приложении OWASP WebGoat, в котором надо выполнять задания по поиску уязвимостей, с помощью него можно изменять значения в параметрах запросов, обходя различные проверки на стороне клиента.

V. Варианты автоматизации поиска уязвимостей при тестировании приложений

OWASP ZAP предоставляет мощный API для использования в собственных приложениях, например, при проведении тестирования. С его помощью можно управлять модулями, относящимися к автоматизированным инструментам — паук и активная атака, а также их настройками и получать различные отчеты о работе.

Существуют API для Java и Python, которые являются оберткой для REST, поэтому возможно работа со сканером ZAP на любой платформе, где доступны библиотеки для работы с HTTP-протоколом.

Для вызова через предоставляемый API необходимо сформировать запрос в виде URL, удовлетворяющего следующему формату (табл. 2):

```
http://zapurl/<format>/<component>/
<operation>/<operation
name>[/?<parameters>]
```

Таблица 2. Описание формата REST-запроса

Запрос	Описание
zapurl	адрес, по которому доступен сканер безопасности OWASP ZAP
format	формат ответов, доступны варианты JSON, XML или HTML
component	название модуля, функции которого вызываются.
operation	тип операции, может быть *view* — получить отчет или *action* — запуск модуля сканирования.
operation name	название операции, например *scan*.
parameters	параметры операции

Примеры REST-запроса:

Запуск сканирования доступных URL:

```
http://zap/XML/spider/action/scan/?
url=localhost:8080
```

Просмотр статуса сканирования:

```
http://zap/JSON/spider/view/status/
```

Для работы со сканером было разработано приложение на Java, часть исходного кода которого приведена в Приложении. Для работы с ZAP использовалось zap-clientapi, подключить которое можно с помощью maven-зависимости:

```
<groupId>org.zaproxy</groupId>
<artifactId>zap-
clientapi</artifactId>
<version>1.2.0</version>
```

В ходе выполнения программа запускает сканирование страницы входа в приложение OWASP Mutillidae. После окончания сканирования запрашивается отчет, который приведен на рис. 1. Как из него видно, удалось обнаружить уязвимости, которые были найдены в ходе выполнения ручного запуска OWASP ZAP.

Таким образом, возможно проводить регрессионное тестирование безопасности приложения при его сборке, например написав unit-тест. Другим вариантом может служить использование плагина для системы сборки, например zap-maven-plugin, который взаимодействует с ZAP также с помощью доступного API. Также доступен плагин для сервера автоматизации Jenkins, позволяющий запускать сканирования приложения с помощью ZAP.

Например, возможен такой сценарий использования:

1. Реализовав новую возможность в приложении, разработчик отправляет код в систему контроля версий.
2. Сервер автоматизации Jenkins, взаимодействую с системой контроля версий, запускает автоматическую сборку приложения в maven, в ходе которой проводятся unit-тесты.
3. В результате сборки создается Docker-образ со всеми необходимыми инфраструктурными приложениями, такие как база данных и сервер приложений.
4. Сервер автоматизации запускает приложение внутри платформы виртуализации Docker и запускает тестирование безопасности с помощью сканера безопасности ZAP.
5. Отчет о выполненном тестировании рассылается сервером автоматизации разработчику, работавшему над новой функциональностью и остальным заинтересованным лицам.

```
#ZAP Scanning Report

##Summary of Alerts

| Risk Level | Number of Alerts |
| --- | --- |
| High | 4 |
| Medium | 1 |
| Low | 4 |
| Informational | 0 |

##Alert Detail


### SQL-инъекция
##### High (Medium)

#### Description
<p>Возможна SQL-инъекция.</p>

* URL: [http://192.168.99.100:32771/index.php?page=login.php](http://192.168.99.100:32771/index.php?page=login.php)
  * Method: `POST`
  * Parameter: `username`
  * Attack: `' OR '1'='1' -- `
* URL: [http://192.168.99.100:32771/index.php?page=login.php](http://192.168.99.100:32771/index.php?page=login.php)
  * Method: `POST`
  * Parameter: `password`
  * Attack: `' OR '1'='1' -- `
Instances: 2

### Solution
<p>Do not trust client side input, even if there is client side validation in place.  </p><p>In general, type check all data o

### Other information
<p>The page results were successfully manipulated using the boolean conditions [' AND '1'='1' -- ] and [' OR '1'='1' -- ]</p><

### Reference
* https://www.owasp.org/index.php/Top_10_2010-A1
* https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

#### CWE Id : 89
#### WASC Id : 19
#### Source ID : 1


### Cross Site Scripting (отражённый)
##### High (Medium)
#### Description
<p>Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instan
* URL: [http://192.168.99.100:32771/index.php?page=login.php](http://192.168.99.100:32771/index.php?page=login.php)
  * Method: `GET`
  * Parameter: `User-Agent`
  * Attack: `</div><script>alert(1);</script><div>`
  * Evidence: `</div><script>alert(1);</script><div>`
* URL: [http://192.168.99.100:32771/index.php?page=login.php](http://192.168.99.100:32771/index.php?page=login.php)
  * Method: `GET`
  * Parameter: `showhints`
```

Рис. 1. Пример отчета о выполнении сканирования страницы входа в приложение OWASP Mutillidae

VI. Заключение

В данной статье:

- Проанализированы существующие инструментальные средства тестирования веб-приложений.
- Рассмотрены приложения, реализующие различные уязвимости.
- Предложены как ручные, так и автоматизированные варианты использования инструментального средства анализа уязвимостей ZAP для тестирования безопасности приложений.

VII. Библиография

[1] О.Р. Лапонина «Основы сетевой безопасности. Часть 1. Межсетевые экраны», Учебное пособие // М. Национальный Открытый Университет «ИНТУИТ» , 2014, с. 378.

[2] А.С. Марков, В.Л. Цирлов «Аудит программного кода по требованиям безопасности», //Информационная безопасность, 2008, №2, с.46-47.

[3] M. Fauler «UnitTest», https://martinfowler.com/bliki/UnitTest.html, 2014.

[4] ANSI/IEEE 1059. Guide for Software Verification and Validation Plans. Approved 1994-06-03.

[5] A. Stock, D. Cuthbert, «Application Security Verification Standard», v3.0.1 //OWASP Foundation, 2016, с.70.

[6] J. Williams, D. Wichers, «The Ten Most Critical Web Application Security Risks», rc1, //OWASP Foundation, 2017, с.23.

[7] M. Meucci, A. Muller, «OWASP Testing Guide», v4.0, //OWASP Foundation, 2014, с.453.

[8] J. Williams, «OWASP Code Review Guide», //OWASP Foundation, 2013, с.191.

VIII. Приложение. Исходный код для работы со сканером тестирования безопасности ZAP через Java API

```
/**
* Программа, реализующая управление сканером безопасности
* OWASP ZAP через доступное Java API.
*/
public class ZAPLauncher {

   // Интервал для проверки
   private static final long CHECK_PERIOD = 1000L;
   // URL для сканирования
```

```
   private static final String TEST_URL =
"http://192.168.99.100:32771/index.php?page=lo
gin.php";

   // API KEY для работы со сканером OWASP ZAP
   private static final String ZAP_API_KEY =
"111111111111111111";
   // адрес сканера OWASP ZAP
   private static final String ZAP_ADDRESS =
"192.168.99.101";
   // порт, на котором запущен сканер
   private static final int ZAP_PORT = 8090;

/**
* Точка входа в приложение.
*/
   public static void main(String[] args) {

      // Подготавливаем url для сканирования.
      List<String> urls = ArrayList<>();
      urls.add(TEST_URL);

      ScanLauncher scanLauncher = new
ZAPLauncher(ZAP_ADDRES, ZAP_PORT,
ZAP_API_KEY);

      // Запуск сканирования.
      foreach(String url : urls) {
         scanLauncher.startScan(url);
      }

      while (scanLauncher.isScanContinue()) {
         Thread.sleep(CHECK_PERIOD);
      }
      // Получаем и выводим отчет о найденных
уязвимостях.
      String scanReport =
scanLauncher.getReport();
      printReport(printReport);
   }

   private static void printReport(String
report) {
      System.out.println(report);
   }
}

/**
* Интерфейс для работы со сканерами
безопасности.
*/
public interface ScanLauncher {

   /**
   * Запуск сканирования на наличие
уязвимостей страниц веб-
   * приложения, расположенных по заданному
url.
   *
   * @param List<String> url страницы для
сканирования.
   */
   void startScan(List<String> urls);

   /**
   * Проверка, завершено ли сканирование.
   */
   boolean isScanContinue();

   /**
   * Получение отчета о сканировании.
   */
   String getReport();
}

/**
* Реализация интерфейса ScanLauncher для
работы с OWASP ZAP.
*/
public class ZAPLauncher implements
ScanLauncher {

   private ClientApi zapClient;
   private List<String> scainIds;

   public ZAPLauncher(String addres, String
port, String apiKey) {
      zapClient = new ClientApi(addres, port,
apiKey);
   }

   /**
   * @see ScanLauncher::startScan
   */
   void startScan(List<String> urls) {
      for(String url : urls) {
         ApiResponse resp =
api.ascan.scan(TARGET, "True", "False",
         null, null, null);
         String id = ((ApiResponseElement)
resp).getValue();
         System.out.println("Запускаем
сканиорования url " + url
               + ", id = " + id);

         // Сохраняем id сканирования для
дальнейшей обработки.
         scanIds.add(id);
      }
   }

   /**
   * @see ScanLauncher::isScanContinue
   */
   boolean isScanContinue() {
      boolean flagFinish = true;
      for(String id : scanIds) {
         byte progress =
Byte.parseByte(((ApiResponseElement)

   api.spider.status(id)).getValue());
         System.out.println("Сканирование с
id= " + id + " : "
               + progress + "%");
         if (progress < 100) {
               flagFinish = false;
               break;
         }
         System.out.println("\n\n");
      }
      return flagFinish;
   }

   /**
   * @see getReport
   */
   String getReport() {
      return new
String(api.core.htmlreport());
   }
}
```

Using the ZAP Vulnerability Scanner to Test Web Applications

Olga R. Laponina, Sergey A. Malakhovsky

***Abstract* – The article examines tools for testing security and searching for various types of vulnerabilities in web applications. The main functionality and main components of the vulnerability testing tools W3AF, Burp Suite and OWASP ZAP are considered. Also discussed are the deployment of the environment for testing application security and implemented various options for using the vulnerability scanner OWASP ZAP. The vulnerability test was conducted in the OWASP Mutillidae application. Mutillidae implements all types of vulnerabilities listed in OWASP Top 10 2007, 2010 and 2013. The application provides the program code for working with the vulnerability scanner via the Java API.**

***Keywords* – vulnerability scanners, web application testing, OWASP ZAP, testing environment.**