

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

Дисциплина: Информационная безопасность

Лабораторная работа №4

Анализ уязвимостей веб-приложения с помощью OWASP ZAP

Группа: Р3432

Выполнили: Готов Егор
Дмитриевич

Преподаватель: Рыбаков Степан
Дмитриевич

г. Санкт-Петербург

2025 г.

Содержание

Назначение	3
Задание	4
Выполнение.....	5
Найденные уязвимости.....	10
1. Cross Site Scripting (XSS)	10
2. SQL injection	12
3. Отсутствие токенов защиты от CSRF атака (Absence of Anti-CSRF Tokens)	14
4. Заголовок Content Security Policy (CSP) не установлен	15
Вывод.....	17

Назначение

Освоить базовые навыки динамического тестирования безопасности (DAST) на примере тестового приложения.

Задание

1. Установить OWASP ZAP (бесплатный инструмент).
2. Запустите встроенный браузер ZAP и перейдите на тестовый сайт (например, <http://testphp.vulnweb.com/>)
3. Проведите «Быстрое сканирование» (Quick Scan) сайта
4. Проанализируйте результаты сканирования: найдите 3-5 различных типов уязвимостей (например, XSS, SQLi)
5. Сделайте скриншоты найденных уязвимостей и кратко опишите суть каждой

Выполнение

Установим и запустим программу для динамического автоматизированного тестирования безопасности (DAST) Zed Attack Proxy (ZAP). При запуске программы нас встречает окно на рисунке 1

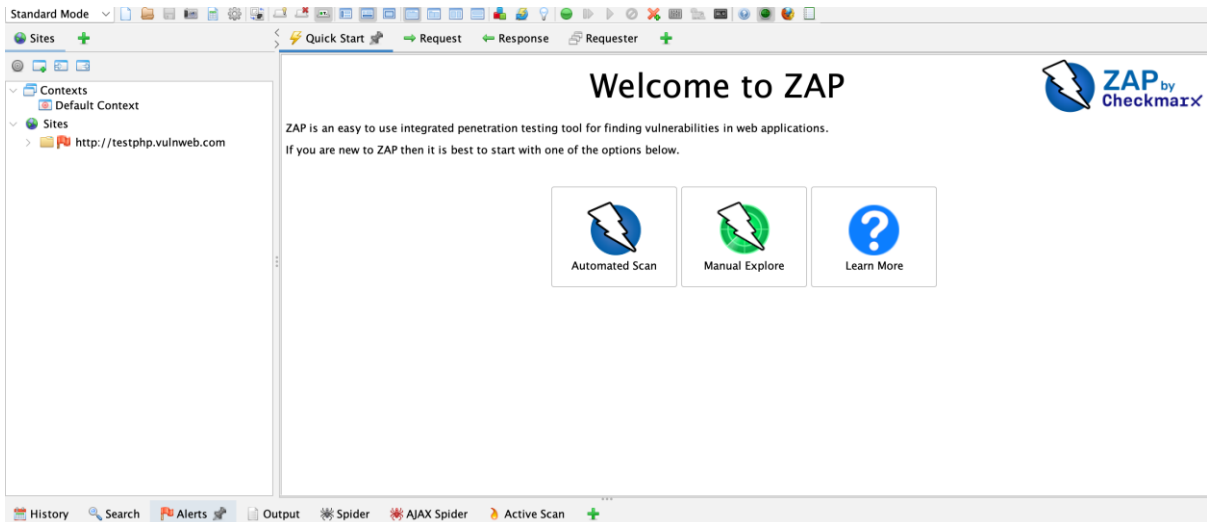


Рисунок 1 – Стартовая страница ZAP

Для тестирования выберем «Автоматизированное сканирование» (Automated Scan). В качестве тестового сайта будем использовать специализированный веб-ресурс, который намеренно содержит ряд уязвимостей и на который можно применить ZAP.

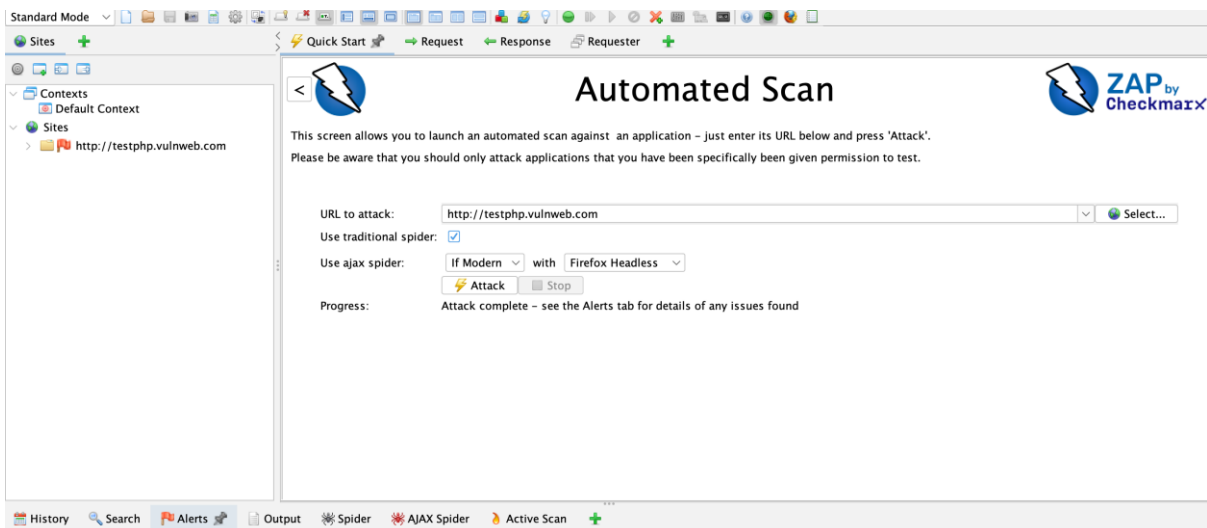


Рисунок 2 – Страницы Автоматического (Automated) сканирования в приложении ZAP

- URL to attack – указываем URL, который мы хотим атаковать/просканировать с целью поиска уязвимостей

- Use traditional spider – использовать стандартный «паук» для сканирования
- Use ajax spider – использовать ajax «паук» для сканирования

Заполняем все поля и нажимаем на Attack.

Spider (паук) – встроенный модуль для автоматического сканирования веб-приложения с целью обнаружения возможных ссылок и ресурсов. Выполняет рекурсивный обход сайта начиная с указанного URL и анализирует HTML-страницы, находит ссылки, формы, скрипты и другие элементы. Переходит по ним для построения карты сайта. Хорошо работает с сайтами, где большая часть навигации реализована через HTML-ссылки

Ajax Spider (Ajax паук) – тот же самый веб-карулер, но для современных динамических веб-приложений, которые активно используют JavaScript. Основан на Selenium и использует драйвер браузеры для эмуляции действия пользователя и анализирует динамически загружаемые части сайта с целью поиска новых ресурсов. Зачастую занимает больше времени на поиск чем классический spider, но зато находит намного больше ресурсов.

Для сравнения классический spider нашел 130 URL

Processed	Method	URI	Seed	Flags
●	GET	http://testphp.vulnweb.com	Seed	
●	GET	http://testphp.vulnweb.com/robots.txt	Seed	
●	GET	http://testphp.vulnweb.com/sitemap.xml	Seed	
●	GET	http://testphp.vulnweb.com/	Seed	
●	GET	http://testphp.vulnweb.com/AJAX	Seed	
●	GET	http://testphp.vulnweb.com/AJAX/artists.php	Seed	
●	GET	http://testphp.vulnweb.com/AJAX/categories.php	Seed	
●	GET	http://testphp.vulnweb.com/AJAX/index.php	Seed	
●	GET	http://testphp.vulnweb.com/AJAX/infoartist.php?id=3	Seed	
●	GET	http://testphp.vulnweb.com/AJAX/infocateg.php?id=4	Seed	
●	GET	http://testphp.vulnweb.com/AJAX/infofile.php	Seed	
●	GET	http://testphp.vulnweb.com/AJAX/showxml.php	Seed	
●	GET	http://testphp.vulnweb.com/AJAX/styles.css	Seed	
●	GET	http://testphp.vulnweb.com/AJAX/titles.php	Seed	
●	GET	http://testphp.vulnweb.com/Flash	Seed	
●	GET	http://testphp.vulnweb.com/Flash/add.swf	Seed	
●	GET	http://testphp.vulnweb.com/Mod_Rewrite_Shop	Seed	
●	GET	http://testphp.vulnweb.com/Mod_Rewrite_Shop/	Seed	

Рисунок 3 – Вкладка классического «паука»

А Ajax Spider нашел 1371 URL

Processed	ID	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	Note	Tags
●	13,755	9/30/25, 5:02:45 PM	GET	http://testphp.vulnweb.com/guestbook.php	200	OK	325...	222 bytes	5,390 bytes	Medium		Form, Hidden, Ob...
●	13,756	9/30/25, 5:02:45 PM	GET	http://testphp.vulnweb.com/	200	OK	326...	222 bytes	4,958 bytes	Medium		Form, Object, Scri...
●	13,757	9/30/25, 5:02:45 PM	GET	http://testphp.vulnweb.com/categories.php	200	OK	328...	222 bytes	6,115 bytes	Medium		Form, Object, Scri...
●	13,758	9/30/25, 5:02:46 PM	GET	http://testphp.vulnweb.com/	200	OK	327...	222 bytes	4,958 bytes	Medium		Form, Object, Scri...
●	13,759	9/30/25, 5:02:46 PM	GET	http://testphp.vulnweb.com/categories.php	200	OK	327...	222 bytes	6,115 bytes	Medium		Form, Object, Scri...
●	13,760	9/30/25, 5:02:46 PM	GET	http://testphp.vulnweb.com/guestbook.php	200	OK	326...	222 bytes	5,390 bytes	Medium		Form, Hidden, Ob...
●	13,761	9/30/25, 5:02:46 PM	GET	http://testphp.vulnweb.com/categories.php	200	OK	327...	222 bytes	6,115 bytes	Medium		Form, Object, Scri...
●	13,762	9/30/25, 5:02:47 PM	POST	http://testphp.vulnweb.com/search.php?test=...	200	OK	330...	222 bytes	4,777 bytes	Medium		Form, Object, Scri...
●	13,763	9/30/25, 5:02:47 PM	GET	http://testphp.vulnweb.com/guestbook.php	200	OK	330...	222 bytes	5,390 bytes	Medium		Form, Hidden, Ob...
●	13,764	9/30/25, 5:02:47 PM	GET	http://testphp.vulnweb.com/categories.php	200	OK	340...	222 bytes	6,115 bytes	Medium		Form, Object, Scri...
●	13,765	9/30/25, 5:02:47 PM	POST	http://testphp.vulnweb.com/search.php?test=...	200	OK	330...	222 bytes	4,777 bytes	Medium		Form, Object, Scri...
●	13,766	9/30/25, 5:02:47 PM	GET	http://testphp.vulnweb.com/categories.php	200	OK	329...	222 bytes	6,115 bytes	Medium		Form, Object, Scri...
●	13,767	9/30/25, 5:02:47 PM	GET	http://testphp.vulnweb.com/guestbook.php	200	OK	325...	222 bytes	5,390 bytes	Medium		Form, Hidden, Ob...
●	13,768	9/30/25, 5:02:47 PM	GET	http://testphp.vulnweb.com/guestbook.php	200	OK	328...	222 bytes	5,390 bytes	Medium		Form, Hidden, Ob...
●	13,769	9/30/25, 5:02:48 PM	POST	http://testphp.vulnweb.com/search.php?test=...	200	OK	327...	222 bytes	4,777 bytes	Medium		Form, Object, Scri...
●	13,770	9/30/25, 5:02:48 PM	GET	http://testphp.vulnweb.com/guestbook.php	200	OK	331...	222 bytes	5,390 bytes	Medium		Form, Hidden, Ob...
●	13,771	9/30/25, 5:02:48 PM	POST	http://testphp.vulnweb.com/guestbook.php	200	OK	330...	222 bytes	5,412 bytes	Medium		Form, Hidden, Ob...
●	13,772	9/30/25, 5:02:48 PM	GET	http://testphp.vulnweb.com/guestbook.php	200	OK	325...	222 bytes	5,390 bytes	Medium		Form, Hidden, Ob...
●	13,773	9/30/25, 5:02:48 PM	POST	http://testphp.vulnweb.com/search.php?test=...	200	OK	330...	222 bytes	4,777 bytes	Medium		Form, Object, Scri...
●	13,774	9/30/25, 5:02:48 PM	POST	http://testphp.vulnweb.com/search.php?test=...	200	OK	330...	222 bytes	4,777 bytes	Medium		Form, Object, Scri...

Рисунок 4 – Вкладка Ajax Spider

Но классический spider занял около 2 секунд, а Ajax spider – около 4 минут.

Во время работы ZAP происходит пассивное и активное сканирование сайта. Пассивное сканирование начинает работу во время поиска URL, то есть во время работы «пауков». Является безопасным для целевого приложения и не создает дополнительной нагрузки. Активное сканирование начинается после того, как все веб-краулеры закончили свою работу. В этом случае ZAP сам генерирует специальные запросы к сайту с целью обнаружения таких уязвимостей как SQLi, XSS, CSRF и так далее. Является уже более агрессивным и глубоким сканированием, которое может нарушить работу сайта.

http://testphp.vulnweb.com Scan Progress						
Progress		Response Chart				
Host:		http://testphp.vulnweb.com				
	Strength	Progress	Elapsed	Reqs	Alerts	Status
Analyser			00:07.007	24		
Plugin						
Path Traversal	Medium		00:47.004	882	0	✓
Remote File Inclusion	Medium		00:29.314	520	0	✓
Source Code Disclosure – /WEB-INF Folder	Medium		00:00.673	3	0	✓
Heartbleed OpenSSL Vulnerability	Medium		00:03.005	0	0	✓
Source Code Disclosure – CVE-2012-1823	Medium		00:03.005	59	0	✓
Remote Code Execution – CVE-2012-1823	Medium		00:04.310	180	0	✓
External Redirect	Medium		00:26.494	468	0	✓
Server Side Include	Medium		00:12.799	208	0	✓
Cross Site Scripting (Reflected)	Medium		00:08.744	191	22	✓
Cross Site Scripting (Persistent) – Prime	Medium		00:03.138	52	0	✓
Cross Site Scripting (Persistent) – Spider	Medium		00:02.925	90	0	✓
Cross Site Scripting (Persistent)	Medium		00:01.197	0	0	✓
SQL Injection	Medium		01:00.532	903	14	✓
SQL Injection – MySQL (Time Based)	Medium		01:09.082	484	7	✓
SQL Injection – Hypersonic SQL (Time Based)	Medium		00:30.412	520	0	✓
SQL Injection – Oracle (Time Based)	Medium		00:14.543	260	0	✓
SQL Injection – PostgreSQL (Time Based)	Medium		00:14.536	260	0	✓
SQL Injection – SQLite (Time Based)	Medium		00:27.638	478	0	✓
Cross Site Scripting (DOM Based)	Medium		06:51.159	0	18	✓
SQL Injection – MsSQL (Time Based)	Medium		00:29.914	520	0	✓
Log4Shell	Medium		00:00.003	0	0	✗
Spring4Shell	Medium		00:05.170	181	0	✓
Server Side Code Injection	Medium		00:23.539	416	0	✓
Remote OS Command Injection	Medium		00:55.771	988	0	✓
XPath Injection	Medium		00:08.815	156	0	✓
XML External Entity Attack	Medium		00:03.142	10	0	✓
Generic Padding Oracle	Medium		00:01.347	0	0	✓
Cloud Metadata Potentially Exposed	Medium		00:11.620	0	0	✓
Server Side Template Injection	Medium		00:41.189	728	0	✓
Server Side Template Injection (Blind)	Medium		00:35.452	624	0	✓
Remote OS Command Injection (Time Based)	Medium		00:46.976	832	0	✓
Directory Browsing	Medium		00:02.367	90	0	✓
Buffer Overflow	Medium		00:02.995	52	0	✓
Format String Error	Medium		00:08.843	156	0	✓
CRLF Injection	Medium		00:20.583	364	0	✓
Parameter Tampering	Medium		00:20.574	358	0	✓
ELMAH Information Leak	Medium		00:00.349	1	0	✓
Trace.axd Information Leak	Medium		00:01.522	18	0	✓
.htaccess Information Leak	Medium		00:01.330	18	0	✓
.env Information Leak	Medium		00:01.339	18	0	✓
Spring Actuator Information Leak	Medium		00:00.327	2	0	✓
Hidden File Finder	Medium		00:09.962	52	0	✓
XSLT Injection	Medium		00:09.955	238	2	✓
GET for POST	Medium		00:01.351	7	4	✓
User Agent Fuzzer	Medium		00:27.166	1080	247	✓
Script Active Scan Rules	Medium		00:00.001	0	0	✗
SOAP Action Spoofing	Medium		00:01.326	0	0	✓
SOAP XML Injection	Medium		00:01.168	0	0	✓
Totals			18:53.411	12771	314	
		Copy	Close			

Рисунок 5 – Прогресс активного сканирования

На рисунке 5 представлен детальный прогресс активного сканирования веб-приложения. Можно заметить, что процесс сканирования достаточно долгий (18:53 минуты). После завершения сканирования во вкладке Alerts находятся все найденные уязвимости, сгруппированные по различным типам и по уровню критичности.

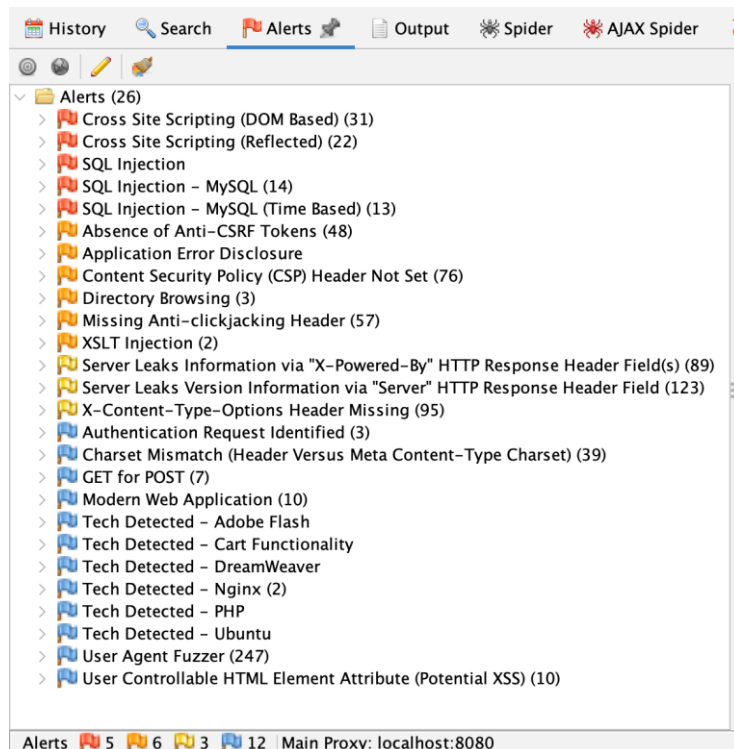


Рисунок 6 – Вкладка Alerts

После двух запусках сканирования системы было найдено 26 уязвимостей. Из них 5 критических (High), 6 средних (Medium), 3 низких (Low), 12 информационных (Informational). Каждую уязвимость можно рассмотреть, подробнее кликнув два раза на название.

Найденные уязвимости

1. Cross Site Scripting (XSS)

- CWE ID: 79
- Risk: High

Cross-Site Scripting (XSS) — это разновидность инъекций, при которых вредоносные скрипты внедряются в изначально безвредные и доверенные веб-сайты. XSS-атаки происходят, когда злоумышленник использует веб-приложение, чтобы отправить вредоносный код (обычно в виде скрипта, выполняемого в браузере) другому пользователю. Уязвимости, которые позволяют этим атакам быть успешными, довольно распространены и встречаются там, где веб-приложение использует пользовательский ввод в создаваемом им выводе, не проверяя его и не выполняя кодирование или санитизацию. Так злоумышленник получает доступ к cookies, токенам сессий и другой конфиденциальной информации.

В частности, reflected XSS (отраженный XSS) — это тип XSS, при котором злоумышленник отправляет вредоносный ввод (например, в параметре URL или в теле запроса), сервер немедленно вставляет этот ввод в формируемый HTML-ответ без корректного экранирования или валидации, и браузер жертвы выполняет вставленный скрипт.

ZAP нашел 22 ресурса, где применима эта уязвимость. Рассмотрим одну из них.

Edit Alert

Cross Site Scripting (Reflected)

URL: `http://testphp.vulnweb.com/product.php?pic=%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E`

Risk: High

Confidence: Medium

Parameter: pic

Attack: `<script>alert(1);</script>`

Evidence: `<script>alert(1);</script>`

CWE ID: 79

WASC ID: 8

Description:
Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually

Other Info:

Solution:
Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Reference:
<https://owasp.org/www-community/attacks/xss/>
<https://cwe.mitre.org/data/definitions/79.html>

Alert Tags:

Key	Value
POLICY_QA_FULL	
POLICY_PENTEST	
HIPAA	https://www.zaproxy.org/docs/desktop/addons/com...
OWASP_2017_A07	https://owasp.org/www-project-top-ten/2017/A7_2...
POLICY_DEV_STD	

Cancel Save

Рисунок 7 – Подробный отчет найденной уязвимости

В данной атаке нам необходимо перейти по указанному URL, который содержит в себе JavaScript код. После перехода на URL видим, что срабатывает alert

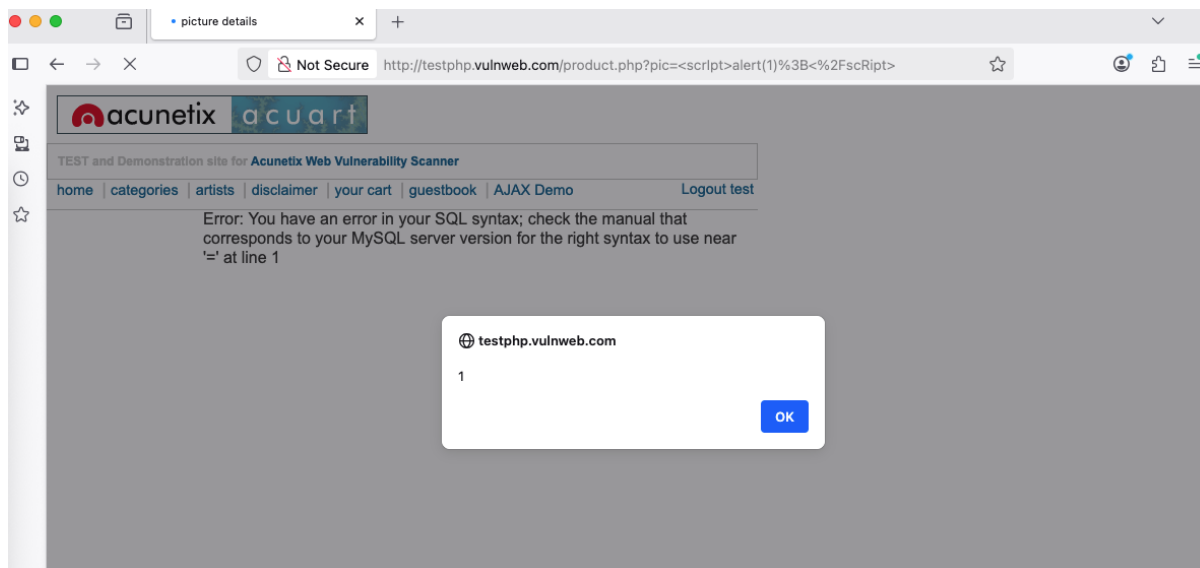
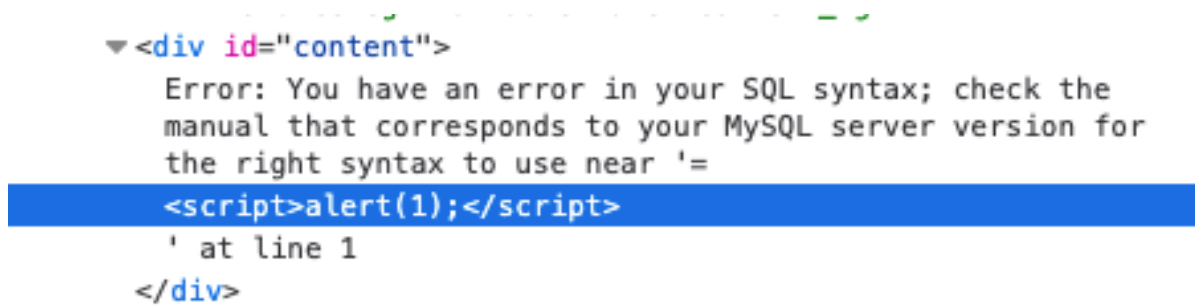


Рисунок 8 – Демонстрация уязвимости в браузере

И мы также видим сообщение об ошибке. В самом сообщении не видно вредоносный код, но если мы посмотрим на DOM, то увидим там следующее



```
<div id="content">
  Error: You have an error in your SQL syntax; check the
  manual that corresponds to your MySQL server version for
  the right syntax to use near '='
  <script>alert(1);</script>
  ' at line 1
</div>
```

Рисунок 9 – фрагмент DOM, содержащий уязвимость

Как раз тот самый вредоносный код, который мы отправили как параметр URL. Такое поведение возникает скорее всего из-за отсутствия экранирования пользовательского ввода и встраивания параметра запроса в ответ ошибки, что и приводит к вызову alert.

Также ZAP нашел XSS типа DOM Based. Данная уязвимость является разновидностью XSS, но она возникает на стороне клиента из-за небезопасного клиентского кода. Однако, воспроизвести в браузере данную уязвимость не получилось. Возможно, это false positive результат, потому что сама атака состоит во встраивании фрагментного идентификатора в HTML, однако нигде в DOM нет использования фрагмента URL.

Для предотвращения XSS-атак необходимо санитизировать весь пользовательский ввод, экранировать все недопустимые символы при помощи библиотек (OWASP Sanitizers/HtmlUtils), а также внедрять CSP заголовок.

2. SQL injection

- CWE ID: 89
- Risk: High

SQLi – уязвимость веб-приложения, при которой злоумышленник может встав произвольные фрагменты SQL- кода в запросы к базе данных через

некорректно обработанные входные данные. В результате атакующий может получить доступ ко всей базе данных и выполнять там любые операции.

Сканирование приложения показало, что, вставляя в различные параметры запросов (к примеру, pass, id, cat, artist) одинарные кавычки (‘), можно увидеть сообщение об ошибке

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '’’ at line 1

Рисунок 10 – Сообщение об ошибке при попытке SQLi

Сообщение на рисунке 10 свидетельствует, что входные данные от пользователя никак не обрабатываются и непосредственно вставляются в SQL-запрос или конкатенируются со строкой SQL-запроса.

К тому же ZAP нашел Time Based (слепой) SQLi. Это разновидность SQLi, при которой сервер базы данных не возвращает данные атакующему напрямую, но позволяет ему косвенно получить информацию. Атакующий формирует условные запросы и по наблюдаемым отличиям (да/нет) восстанавливает нужные данные по одному биту/символу. В нашем случае было найдено 13 ресурсов, где можно применить слепую SQLi

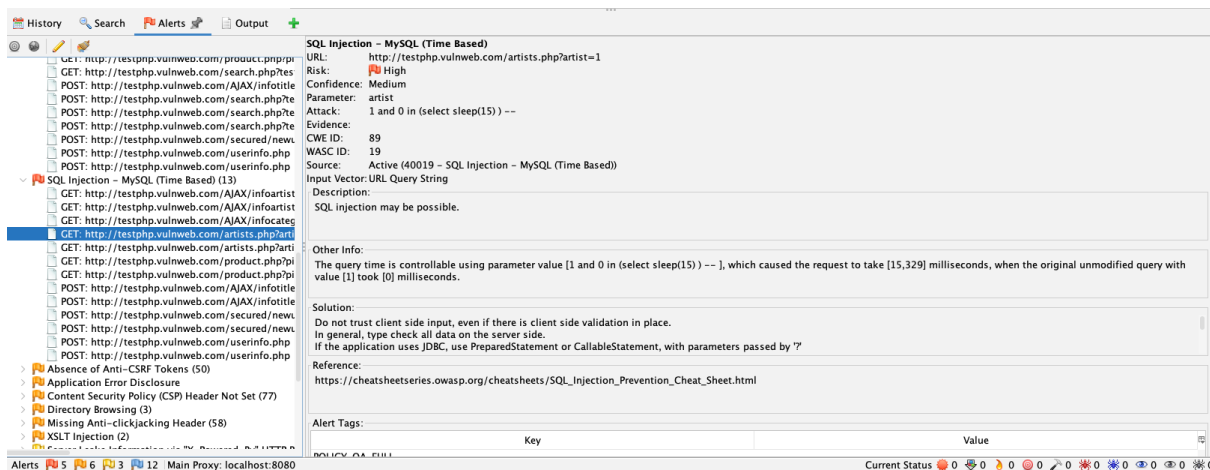


Рисунок 11 – Time Based SQLi

Для предотвращения SQLi следует никогда не использовать конкатенацию строк для создания запросов, а использовать

PreparedStatement, ORM (Hibernate, SQLAlchemy и другие) и организовать доступ к БД с минимальными привилегиями.

3. Отсутствие токенов защиты от CSRF атака (Absence of Anti-CSRF Tokens)

- CWE ID: 352
- Risk: Medium

Отсутствие токенов защиты от CSRF-атак описывает уязвимость, при которой злоумышленник без ведома аутентифицированного в целевом веб-сайте пользователя выполняет нежелательные действия, используя его текущую сессию и права. Атакующему даже не нужна знать учетные данные жертвы – достаточно, чтобы жертва была залогинена в целевом приложении и перешла на злонамеренную страницу.

ZAP нашел 50 ресурсов, а точнее форм, в которых отсутствует скрытое поле anticrsrf, CSRFToken, csrf_token и так далее. Это означает, что атакующий беспрепятственно может подделать сайт или создать специальную ссылку, которая выполнит вредоносные действия от лица пользователя (изменение пароля, перевод денежных средств, отправка сообщений, удаление аккаунта и другое).

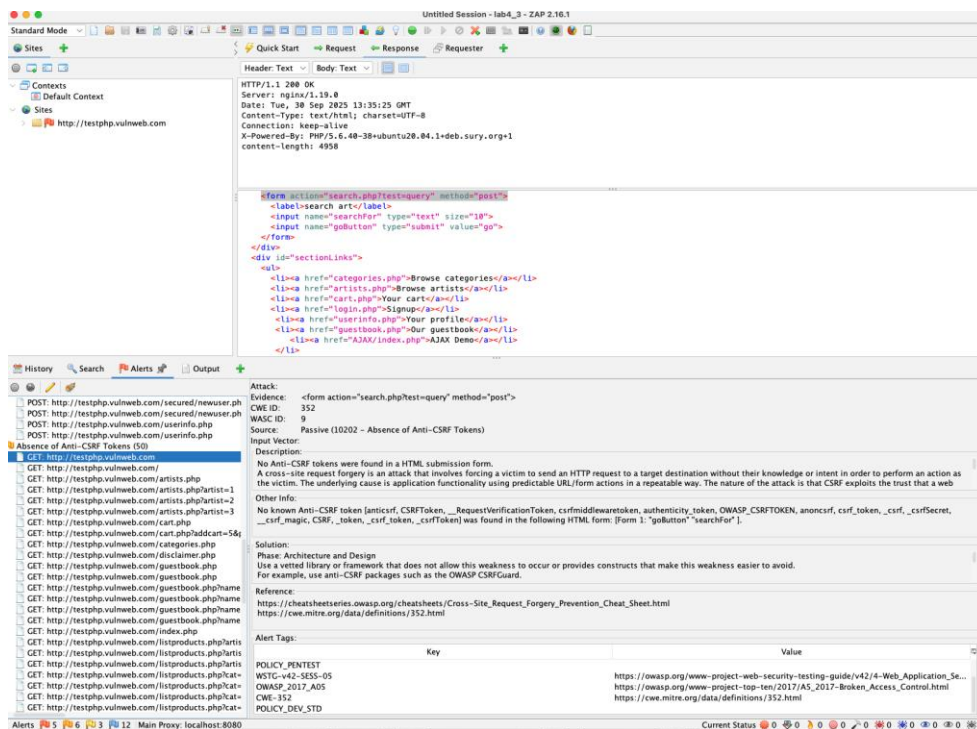


Рисунок 12 – Отсутствие токена защиты от CSRF-атак

Для предотвращения CSRF-атак необходимо внедрять скрытые поля, содержащие csrf-токен, который будет проверяться сервером. Также необходимо сначала обеспечить защиту от XSS-атак, так как, если этого не сделать, то csrf-токен может быть украден

4. Заголовок Content Security Policy (CSP) не установлен

- CWE ID: 693
- Risk: Medium

Заголовок CSP сообщает браузеру откуда разрешено загружать ресурсы (script-src, font-src, style-src, media-src и так далее) и какие типы выполнения контента допустимы (inline-скрипты, CSS, JavaScript и так далее). Данный заголовок создает дополнительный уровень защиты, даже если XSS-уязвимости присутствуют в приложении. К тому же можно отправлять отчеты о попытках XSS-атак при помощи параметра CSP заголовка report-uri.

ZAP нашел 78 ресурсов, где отсутствует данный заголовок. Обычно такой заголовок вставляется сервером при отправке веб-страниц пользователю.

Вывод

В результате выполнения лабораторной работы было проведено динамические тестирование безопасности тестового приложения (<http://testphp.vulnweb.com/>). Было найдено 26 уязвимостей, из которых 5 критических, 6 средних, 3 низких и 12 информационных.

В рамках работы было рассмотрено всего 4 уязвимости: XSS (DOM Based, Reflected), SQLi (In-Band, Time Based), отсутствие CSRF-токена и отсутствие CSP-заголовка. Также, были приведены некоторые шаги для предотвращения этих уязвимостей.

Стоит отметить, что работа приложения ZAP сильно зависит от браузера и в целом результат может разниться от запуска к запуску даже на одном браузере. К примеру, при первом запуске сканирования при помощи Chrome было найдено всего 11 уязвимостей, из которых не было ни SQLi, ни XSS. Позже при использовании браузера FireFox было найдено значительно больше уязвимостей, а при повторном запуске было найдено еще несколько. Поэтому, исходя из опыта работы с ZAP, стоит запускать DAST-инструменты чаще, на разных браузерах и на разных версиях для построения более полной картины наличия проблем в разрабатываемом приложении.