Penetration Test Report

Track: Penetration Testing
Тестируемый ресурс: [redacted]
Методология: Black Box Testing
Статус: Завершено
Ответственный: Кропотов Денис

Введение

Целью данного тестирования является проведение комплексного анализа безопасности веб-приложения, доступного по адресу [redacted], с использованием методологии Black Box Testing. Тестирование проводилось с целью выявления потенциальных уязвимостей, которые могут быть использованы злоумышленником для получения несанкционированного доступа, компрометации данных или нарушения работы сервиса.

Тестирование проводилось без деструктивных воздействий, с соблюдением этических норм и требований курса.

Этап 1: OSINT

Цель:

Сбор информации о целевом ресурсе с использованием открытых источников.

Инструменты:

Shodan

Google Dorks

CVE Details

Действия:

1. Сканирование Shodan:

Запрос ip:[redacted] в Shodan.
Хостнеймы: 1427771-cg36175.tw1.ru
Домены: tw1.ru
Страна: Российская Федерация
Город: Санкт-Петербург
Организация/ISP: TimeWeb Ltd.
ASN: AS9123 (TimeWeb Ltd.)

Открытые порты (из Shodan):

22 (SSH) - OpenSSH 8.2p1 Ubuntu 4ubuntu0.13
8050 (HTTP) Apache 2.4.7 (Ubuntu), PHP 5.5.9-1ubuntu4.29
10050 (неизвестный сервис)

HTTP Headers (из Shodan - порт 8050):

```
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Set-Cookie: PHPSESSID=...
Content-Type: text/html
```

Обнаруженные уязвимости (из Shodan Data):

Shodan предоставляет список CVE, связанных с обнаруженным ПО (Apache 2.4.7, PHP 5.5.9). Среди них: **CVE-2015-4598** (PHP SoapFault type confusion CVSS 9.8), **CVE-2015-4601** (PHP type confusion - CVSS 9.8), **CVE-2016-4542** (PHP exif out-of-bounds read - CVSS 9.8), **CVE-2016-7126** (PHP session deserialization use-after-free - CVSS 9.8), **CVE-2015-8866** (PHP openssl_random_pseudo_bytes - CVSS 7.5), **CVE-2015-0232** (glibc "GHOST" - CVSS 10.0), и многие другие.

Вывод:

Обнаружено множество критических и высоких уязвимостей, связанных с устаревшими версиями Apache и PHP.

2. Поиск в Google (Google Dorks):

Использование поисковых запросов:

site:[redacted]

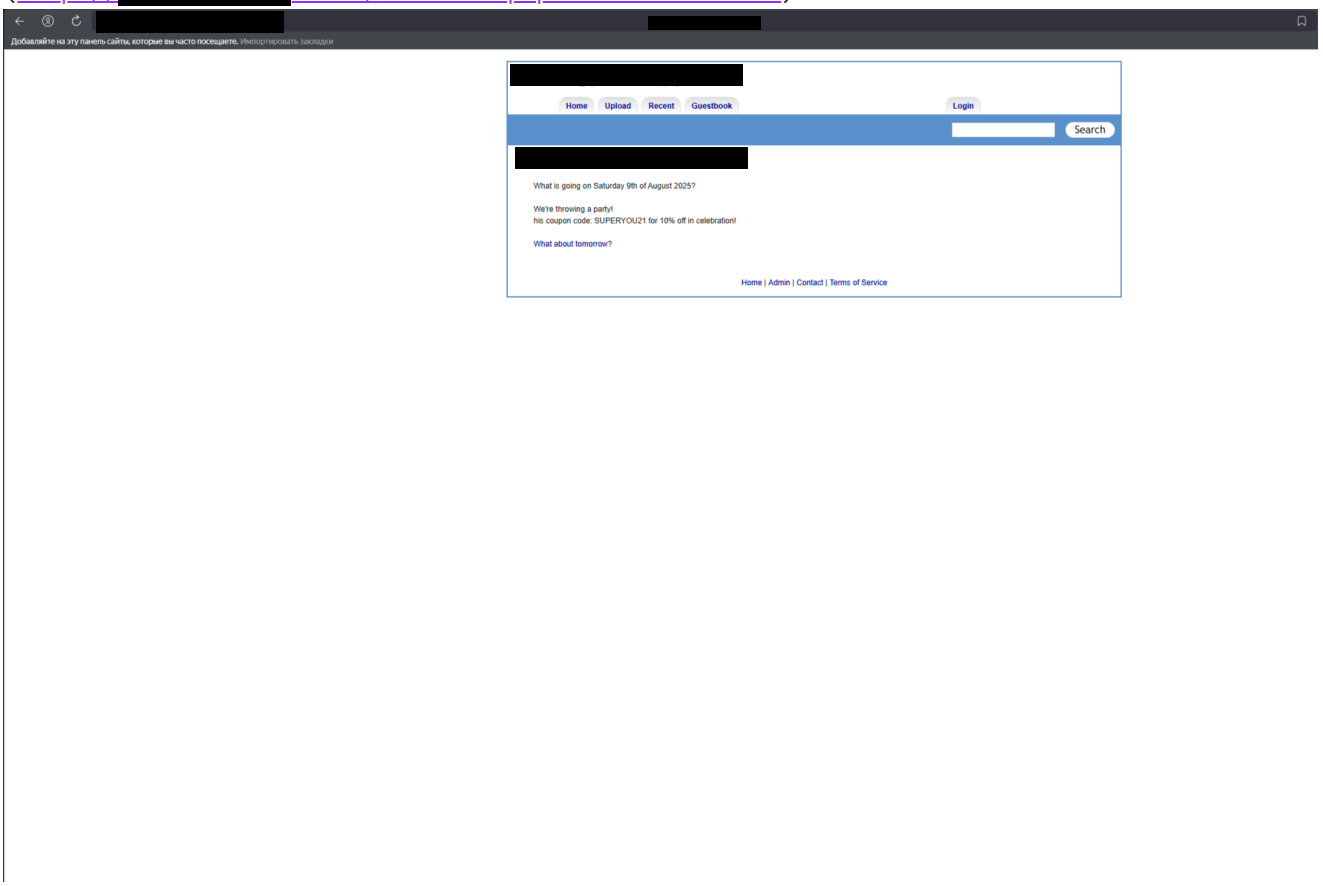
Результаты:

Поиск site:[redacted] выявил следующие URL, проиндексированные Google:

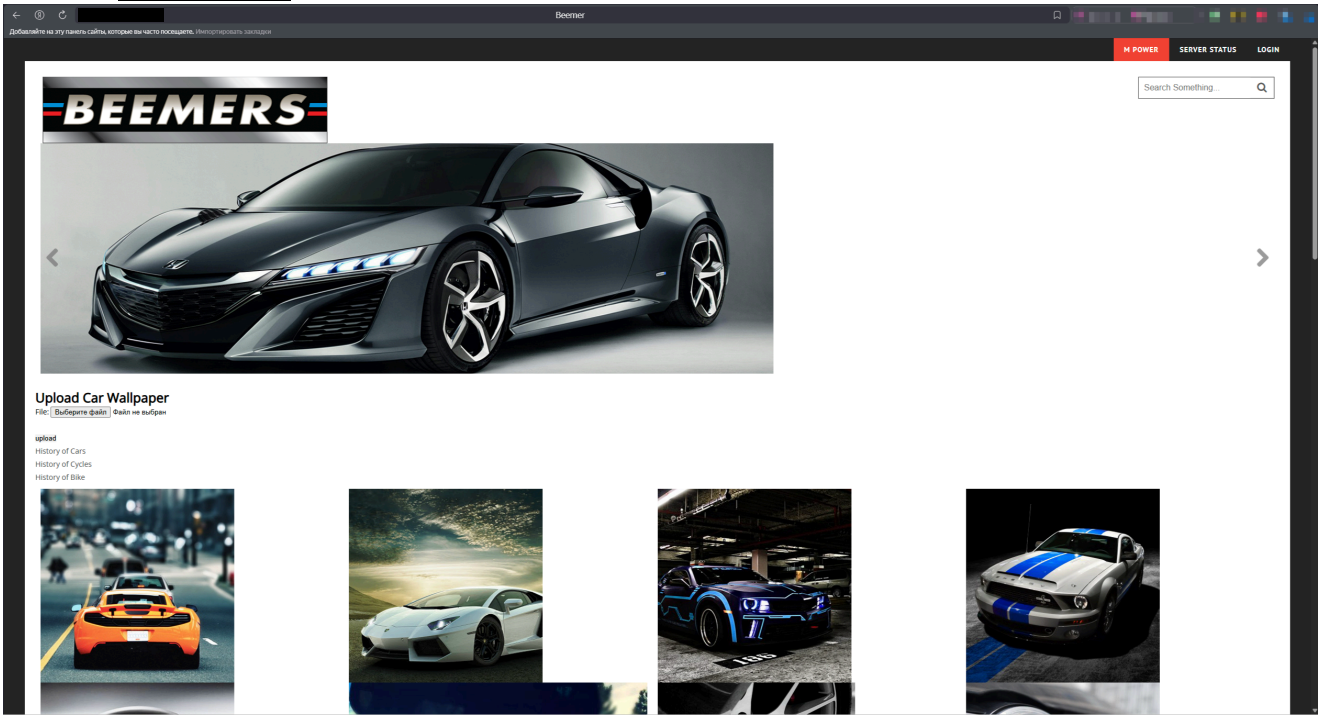
Страница административной панели
(http://[redacted]:8050/admin/index.php?page=login-)

Страница календаря с параметром даты

(http://[redacted]:8050/calendar.php?date=1754707252)



Дополнительный сервис на нестандартном порту 7788.
(Shodan не указывал порт 7788, что может означать, что он был заблокирован межсетевым экраном при сканировании Shodan.)
(http://[redacted]:7788/)



Вывод:

Это подтверждает, что на порту **8050** работает веб-приложение (вероятно, [redacted]), и часть его функционала (админка, календарь) была проиндексирована. Также указывает на наличие **административной панели**, доступ к которой может быть потенциальной целью.

3. Проверка на CVE Details:

> - **PHP 5.5.9:** Поиск версии `PHP 5.5.9` на CVE Details (cvedetails.com) показал **226 известных уязвимостей** для этой версии. Из них:
> - **38 уязвимостей** потенциально позволяют выполнение кода (Code Execution).
> - **119 уязвимостей** могут привести к отказу в обслуживании (Denial of Service).
> - **4 уязвимости** связаны с межсайтовым скриптингом (XSS).
> - Уязвимости охватывают различные годы (2015-2023), что подтверждает длительный период, в течение которого версия была уязвима.
> - Это подтверждает и расширяет информацию, найденную в Shodan, и указывает на **чрезвычайно высокий риск**, связанный с использованием этой версии PHP.
> - **Apache 2.4.7:** Поиск версии Apache 2.4.7 на CVE Details (cvedetails.com) показал **79 известных уязвимостей** для этой версии. Из них:
> - **6 уязвимостей** потенциально позволяют выполнение кода.
> - **6 уязвимостей** могут привести к отказу в обслуживании.
> - Также присутствуют уязвимости, связанные с утечкой информации (Information Leak) и обходом безопасности (Bypass).
> - Уязвимости охватывают годы с 2017 по 2025, что указывает на устаревшую версию с известными проблемами.

Вывод:

Обе версии ПО (PHP 5.5.9, Apache 2.4.7) содержат огромное количество известных уязвимостей, включая критические (RCE, DoS). Это делает целевой сервер ([redacted]:8050) высокоприоритетной целью для атак.

Этап 2: Scanning

Цель:

Подробное сканирование сервисов с использованием специализированных инструментов, валидация найденных уязвимостей и проверка на ложные срабатывания.

Инструменты:

Nmap

Nikto

Gobuster

Действия:

1. Nmap:

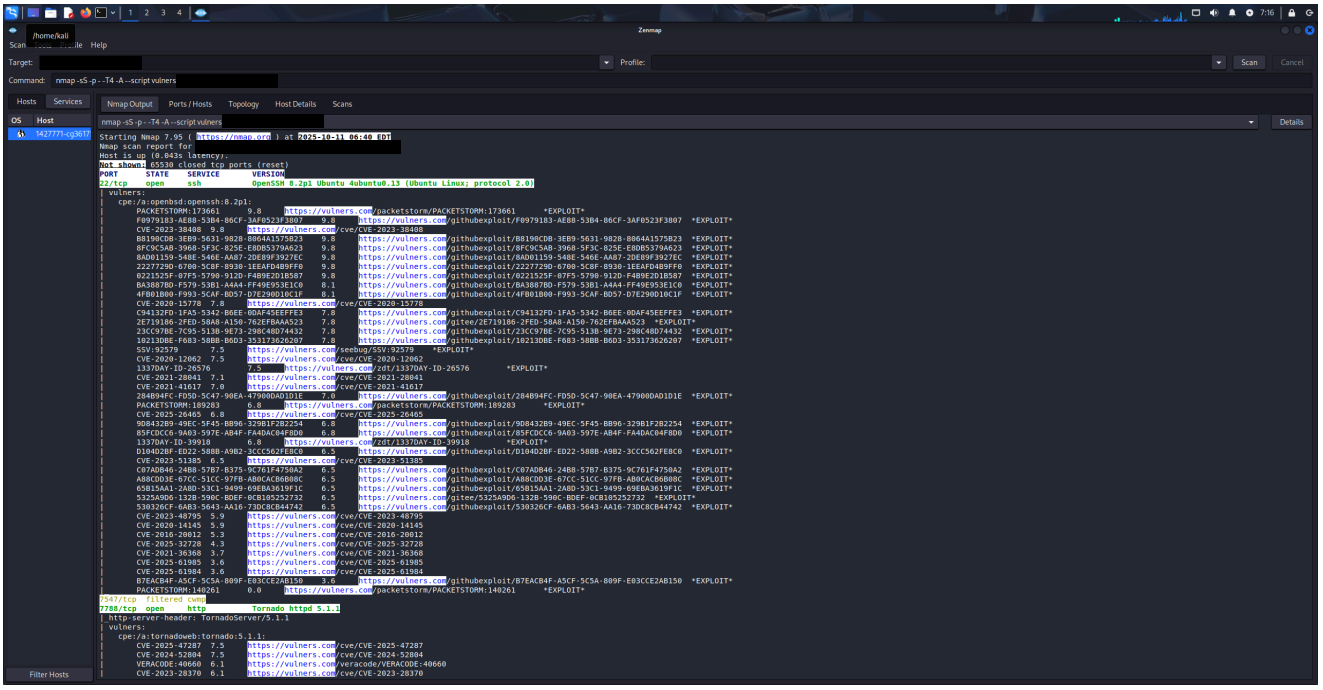
Цель:

Подтвердить порты, обнаруженные Shodan (22, 8050, 10050), и найти возможные скрытые. Определить версии сервисов и потенциальные уязвимости с помощью скриптов.

Команда:

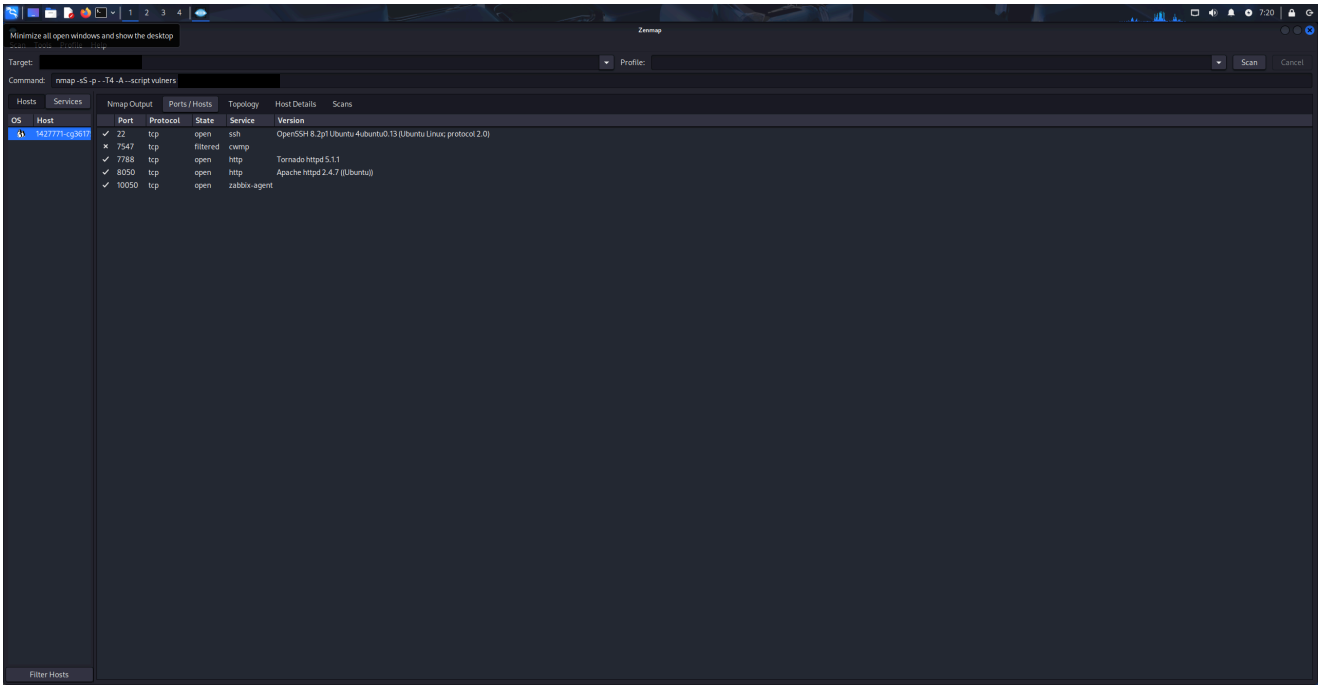
nmap -sS -p- -T4 -A --script vulners [redacted]

`-sS`: TCP SYN scan (стелс-скан).
`-p-`: Сканировать все 65535 портов.
`-T4`: Установка темпа сканирования (не слишком агрессивный).
`-A`: Включает `-sV`, `-O`, `-sC` (скрипты по умолчанию), и traceroute.
`--script vulners`: Использование скрипта `vulners` для проверки обнаруженных версий сервисов на наличие известных уязвимостей (CVE) из базы данных Vulners.



Результаты Nmap:

Открытые порты: 22 (SSH), 7788 (HTTP), 8050 (HTTP), 10050 (zabbix-agent или другой сервис). Порт 7547 (filtered).
Сервисы: OpenSSH 8.2p1 Ubuntu, Tornado httpd 5.1.1, Apache httpd 2.4.7 (Ubuntu).



Обнаруженные уязвимости (по результатам скрипта vulners):

- **Критические (CVSS 9.0 - 10.0):** Найдено **43** уязвимости. Включают CVE-2023-38408 (OpenSSH), CVE-2024-38476, CVE-2022-23943 (Apache), CVE-2017-3169 (Apache) и другие.
- **Высокие (CVSS 7.0 - 8.9):** Найдено **93** уязвимости. Включают CVE-2020-15778 (OpenSSH), CVE-2024-52804 (Tornado), CVE-2021-26690, CVE-2019 0217 (Apache) и другие.
- **Средние** (CVSS 4.0 - 6.9) и Ниже: **Найдено** 91** уязвимость.

Анализ результата:

Сканирование подтвердило наличие открытых портов и определило версии сервисов. Интеграция с базой данных vulners показала **наличие множества известных уязвимостей**, включая **43 критические** и **93 высокие**, в обнаруженных сервисах (OpenSSH, Apache, Tornado). Это указывает на высокий уровень риска.

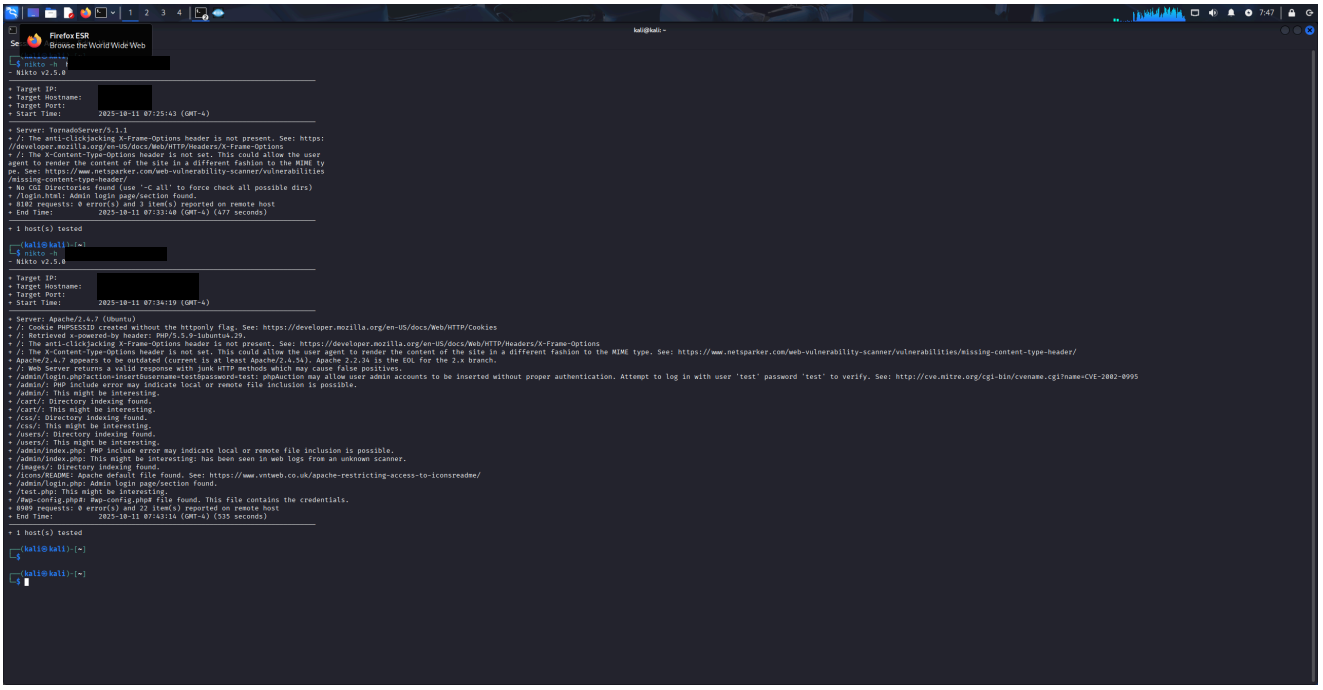
Вывод:

Сканирование Nmap **успешно** обнаружило открытые порты, определило версии сервисов и выявило **наличие** потенциальных уязвимостей, подтвержденных базой данных vulners. Наличие критических уязвимостей, таких как CVE-2023-38408 (OpenSSH) и CVE-2024-38476 (Apache).

2. Nikto:

nikto -h http://[redacted]:8050

nikto -h http://[redacted]:7788



Результаты (из `Nikto):

Порт 7788 (TornadoServer/5.1.1):

> - Server: TornadoServer/5.1.1 (подтверждено).
> - **Отсутствие заголовка X-Frame-Options**: Позволяет встраивать сайт во фреймы, что может привести к Clickjacking-атакам.
> - **Отсутствие заголовка X-Content-Type-Options**: Позволяет браузеру MIME-sniffing, что может привести к выполнению кода, если файл подаётся с неправильным Content-Type.
> - **Найдена страница администратора**: /login.html.

Порт 8050 (Apache/2.4.7, PHP/5.5.9-1ubuntu4.29):

> - Server: Apache/2.4.7 (Ubuntu) (подтверждено).
> - X-Powered-By: PHP/5.5.9-1ubuntu4.29 (подтверждено).
> - **Отсутствие флага `HttpOnly` у cookie PHPSESSID**: Позволяет доступ к cookie из JavaScript, увеличивая риск XSS-атак.

- **Отсутствие заголовка X-Frame-Options**: Позволяет встраивать сайт во фреймы, что может привести к Clickjacking-атакам.
- **Отсутствие заголовка** X-Content-Type-Options : Позволяет браузеру MIME-sniffing, что может привести к выполнению кода, если файл подаётся с неправильным Content-Type.
- **Устаревшая версия Apache**: Apache/2.4.7 уязвима, текущая версия намного новее.
- **Найдены директории с индексацией**: /cart/, /css/, /users/, /images/. Это может раскрыть внутреннюю структуру или файлы.
- **Найден файл README от Apache**: /icons/README. Указывает на возможное использование стандартных файлов Apache.
- **Найдена страница администратора**: /admin/login.php.
- **Возможная уязвимость Local File Inclusion (LFI)**: Ошибки include в /admin/ и /admin/index.php могут указывать на LFI.
- **Найден файл** test.php.
- **Найден файл** wp-config.php (внимание: это может быть фейковый файл или артефакт, но указывает на возможное сканирование/атаку на WordPress или просто имя файла, оставленное разработчиком).
- **Потенциальная уязвимость phpAuction**: /admin/login.php?action=insert&username=test&password=test (требует подтверждения).

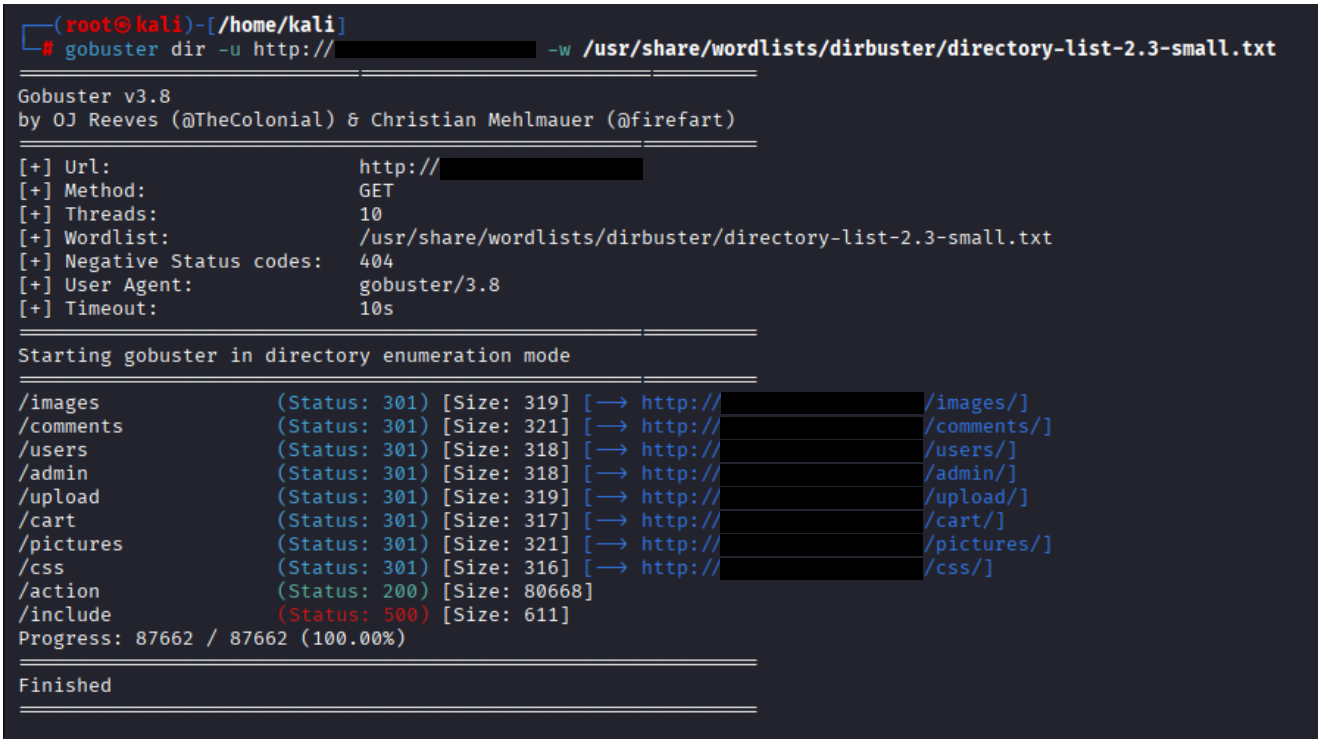
3. Gobuster:

Цель:

Перечисление директорий и файлов на веб-серверах, доступных на портах 8050 и 7788.

Действия:

Запуск сканирования порта 8050 с использованием словаря directory-list-2.3-small.txt : gobuster dir -u http://[redacted]:8050 -w /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt



Запуск сканирования порта 7788 с использованием словаря directory-list-2.3-medium.txt: gobuster dir -u http://[redacted]:7788 -w /usr/share/wordlists/dirbuster/directory-list-

2.3-medium.txt

```
┌──(root㉿kali)-[/home/kali]
└─# gobuster dir -u http://[redacted] -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
===============================================================
Gobuster v3.8
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://[redacted]
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.8
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/search               (Status: 200) [Size: 3504]
/upload               (Status: 405) [Size: 325]
/read                 (Status: 200) [Size: 6596]
/index_html           (Status: 200) [Size: 15053]
Progress: 220558 / 220558 (100.00%)
===============================================================
Finished
===============================================================
```

Результаты (из Gobuster):

Порт 8050 (Apache/2.4.7, PHP/5.5.9-1ubuntu4.29):

> - Найдены директории с редиректом (Status 301): `/images`, `/comments`, `/users`, `/admin`, `/upload`, `/cart`, `/pictures`, `/css`. Это подтверждает и расширяет находки Nikto.
> - Найден файл/путь с Status 200: `/action`.
> - Найден путь с Status 500 (Internal Server Error): `/include`. Это может указывать на проблему в обработке или на уязвимость (например, LFI, если путь принимает параметры).

Порт 7788 (TornadoServer/5.1.1):

> - Найден файл/путь с Status 200: `/search`.
> - Найден файл/путь с Status 200: `/read`.
> - Найден файл/путь с Status 200: `/index_html`.
> - Найден файл/путь с Status 405 (Method Not Allowed): `/upload`.

Вывод:

Gobuster подтвердил и расширил список обнаруженных директорий и файлов, найденных с помощью Nikto на порту 8050. Также были обнаружены новые потенциально интересные точки входа на обоих портах, особенно `/action` (8050), `/include` (8050 - ошибка 500), `/search` (7788), `/read` (7788). Путь `/include` с ошибкой 500 требует дополнительного внимания.

Этап 3: Testing

1. Автоматизированное тестирование (OWASP ZAP):

Цель:

использовать OWASP ZAP для поиска критических и часто эксплуатируемых уязвимостей на портах 8050 и 7788.

Инструменты:

OWASP ZAP (Zed Attack Proxy).

Результаты OWASP ZAP:

Порт 7788 (TornadoServer/5.1.1):

- **Высокий уровень обход пути:** возможность чтения произвольных файлов, подтверждена ZAP (`/etc/passwd`).
- **Высокий уровень удалённое внедрение команд в ОС (обычное и по расписанию):** возможность выполнения команд ОС, подтверждённая ZAP (выполнение `cat /etc/passwd`, `sleep`).
- **Высокий уровень межсайтовый скриптинг (на основе отражения и DOM):** возможность выполнения JS-кода в браузере пользователя, подтвержденная ZAP.
- **Высокий уровень SQL-инъекция:** возможность манипулирования базой данных, подтвержденная ZAP (ошибка 500) и ручным тестированием (`' OR 1=1 --`).
- **Medium отсутствует заголовок X-Content-Type-Options:** повышает риск межсайтового скриптинга с использованием MIME. Подтверждено ZAP.
- **Средний уровень отсутствует заголовок для защиты от кликджекинга:** допускает атаки кликджекинга. Подтверждено ZAP.
- **Medium Cookie без флага HttpOnly / Cookie без атрибута SameSite:** Повышает риск кражи `PHPSESSID` через XSS и CSRF. Подтверждено ZAP.

Порт 8050 (`Apache/2.4.7`, `PHP/5.5.9-1ubuntu4.29`):

- **Высокий уровень SQL-инъекция (на основе булевых значений/времени/ошибок):** множественные векторы SQLi, подтвержденные ZAP (ошибки MySQL, булевы условия).
- **Высокий уровень межсайтовый скриптинг (отраженный и на основе DOM):** множественные случаи межсайтового скриптинга, подтвержденные ZAP. Межсайтовый скриптинг на основе DOM использует сложные полезные нагрузки во фрагменте URL (`#jaVas...`).
- **Medium отсутствует заголовок X-Content-Type-Options:** повышает риск межсайтового скриптинга с использованием MIME. Подтверждено ZAP.
- Средний уровень: отсутствует заголовок для защиты от кликджекинга:** допускает атаки кликджекинга. Подтверждено ZAP.
- Medium: Cookie без флага HttpOnly / Cookie без атрибута SameSite:** Повышает риск кражи `PHPSESSID` через XSS и CSRF. Подтверждено ZAP.
- **Medium значение атрибута HTML, управляемое пользователем:** потенциальный вектор для XSS. Подтверждено ZAP.

Вывод по результатам автоматизированного тестирования:

OWASP ZAP успешно обнаружил критические уязвимости на **обоих портах**. На **порту 7788** это **обход пути**, **внедрение команд**, **внедрение SQL-кода**, **межсайтовый скриптинг**. На **порту 8050** это **внедрение SQL-кода**, **межсайтовый скриптинг**. Также подтверждены **средние** уязвимости, повышающие риск (**отсутствие заголовков безопасности**, **небезопасные настройки Cookie**).

2. Ручное тестирование:

Подтверждение SQL Injection:

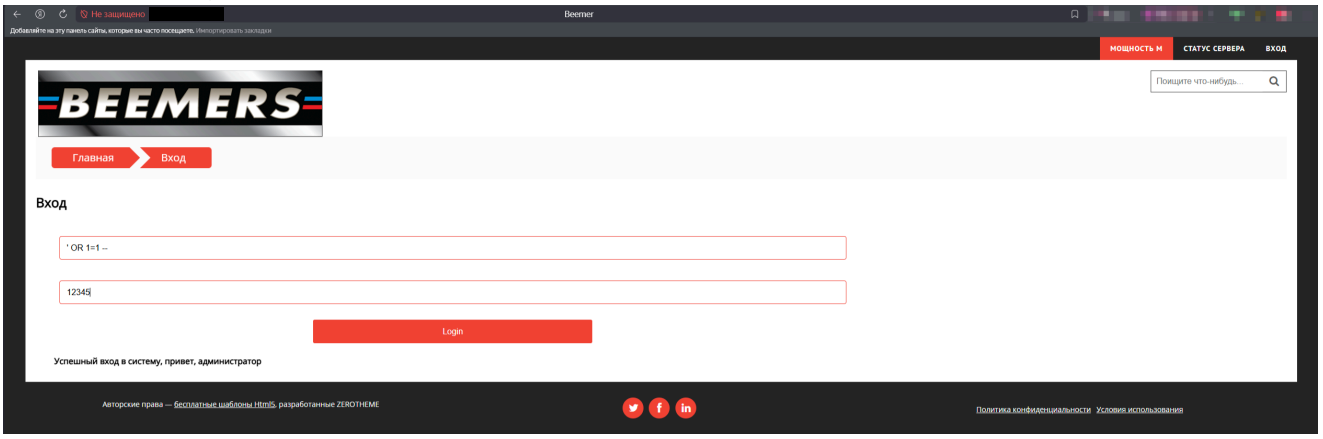
Открыть форму входа http://[redacted]:7788/login.html в веб-браузере.
В поле **Login** ввести `' OR 1=1 --`.
В поле **Password** ввести любое значение, например, `12345`.
Нажать кнопку входа.

Результаты SQL Injection:

Успешный вход:

Был успешно выполнен вход в учетную запись администратора.
Подтверждение: **Успешный вход при использовании полезной нагрузки** `' OR 1=1 --` **в поле логина** подтверждает **эксплуатацию уязвимости** SQL Injection** на уровне аутентификации. Условие `1=1`

всегда истинно, что позволило обойти проверку логина и пароля.



Подтверждение Remote OS Command Injection:

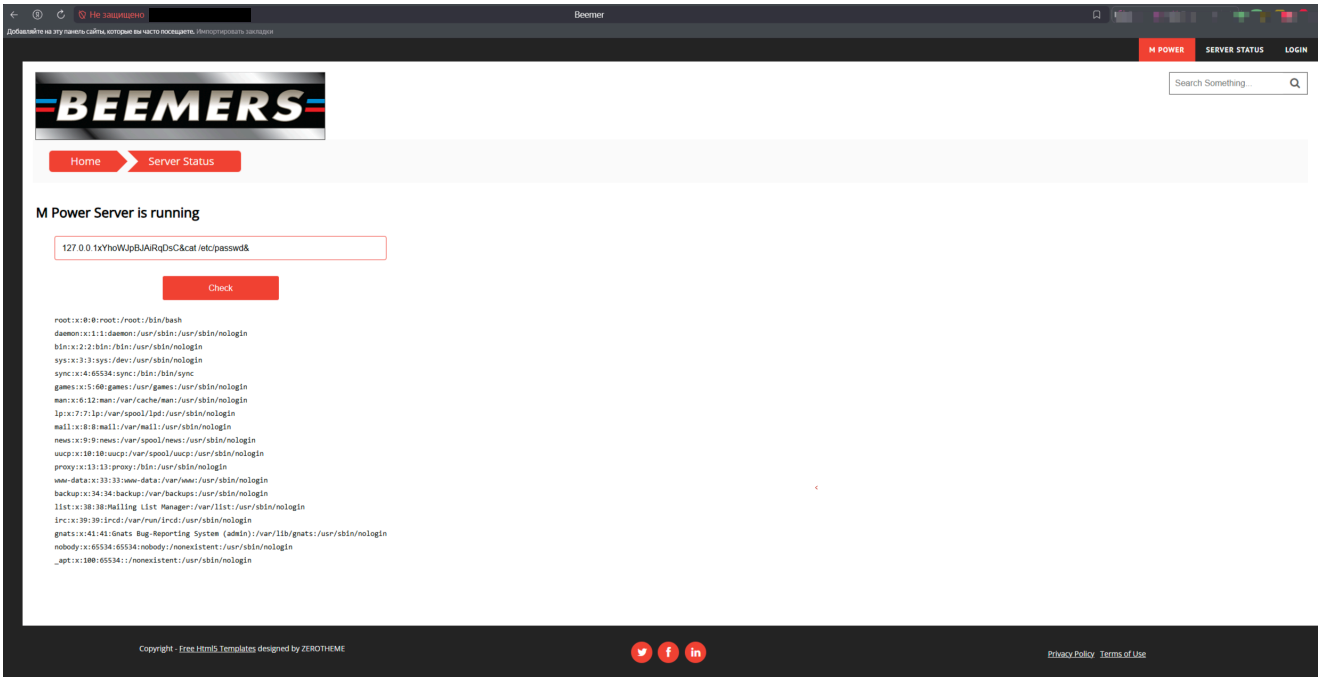
Открыть http://[redacted]:7788/server.html в веб-браузере.
В поле ввода (предположительно, для проверки доступности сервера) ввести: `127.0.0.1xYhoWJpBJAiRqDsC&cat /etc/passwd&`.
Отправить форму (нажать кнопку Check или аналогичную).

Результаты Command Injection:

Успешное выполнение команды:** В ответе от сервера было получено содержимое файла `/etc/passwd`:

Подтверждение:

Получение содержимого системного файла `/etc/passwd` в результате отправки специально сформированной строки в поле ввода **подтверждает** эксплуатацию уязвимости **Remote OS Command Injection**. Введённая строка была интерпретирована сервером как команда ОС.



Подтверждение Reflected XSS на `search?q=...`:

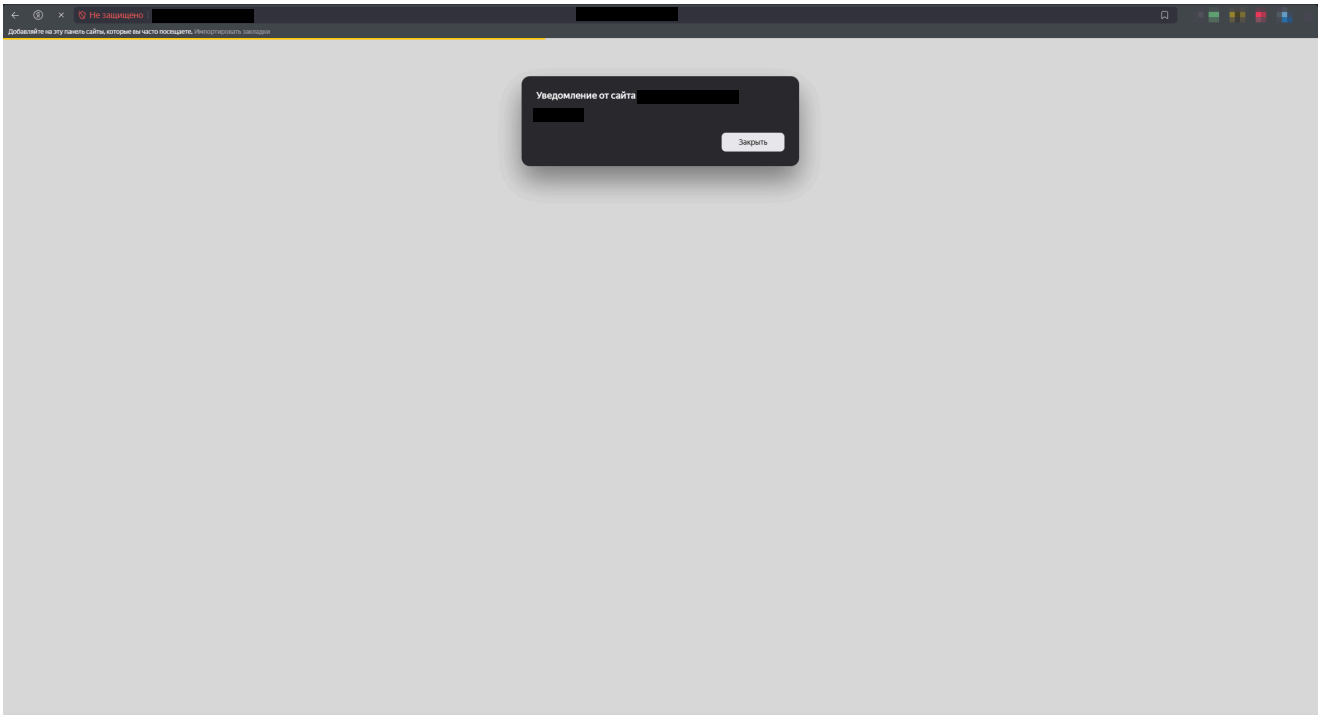
Действия:

Перешёл на http://[redacted]:8050/. Ввёл в поле поиска `<script>alert(document.domain)</script>`. Отправил поиск.

Результаты: Успешно всплыло окно `alert` с IP-адресом сервера ([redacted]).

Вывод:

Уязвимость **Reflected Cross-Site Scripting (XSS)** на странице поиска (http://[redacted]:8050/search?q=...) **подтверждена вручную.** Это доказывает, что пользовательский ввод не проходит надлежащую фильтрацию/экранирование и может быть использован для выполнения произвольного JavaScript в браузере жертвы.



Подтверждение Stored Cross Site Scripting (XSS) на `guestbook.php`:

Действия:

Перешёл на `http://[redacted]:8050/guestbook.php`.
В форме добавления сообщения ввёл полезную нагрузку из отчёта ZAP: `</p><scrIpt>alert(”XSS-PWNED”);</scRipt><p>`.
Отправил сообщение.
Обновил страницу `guestbook.php`.

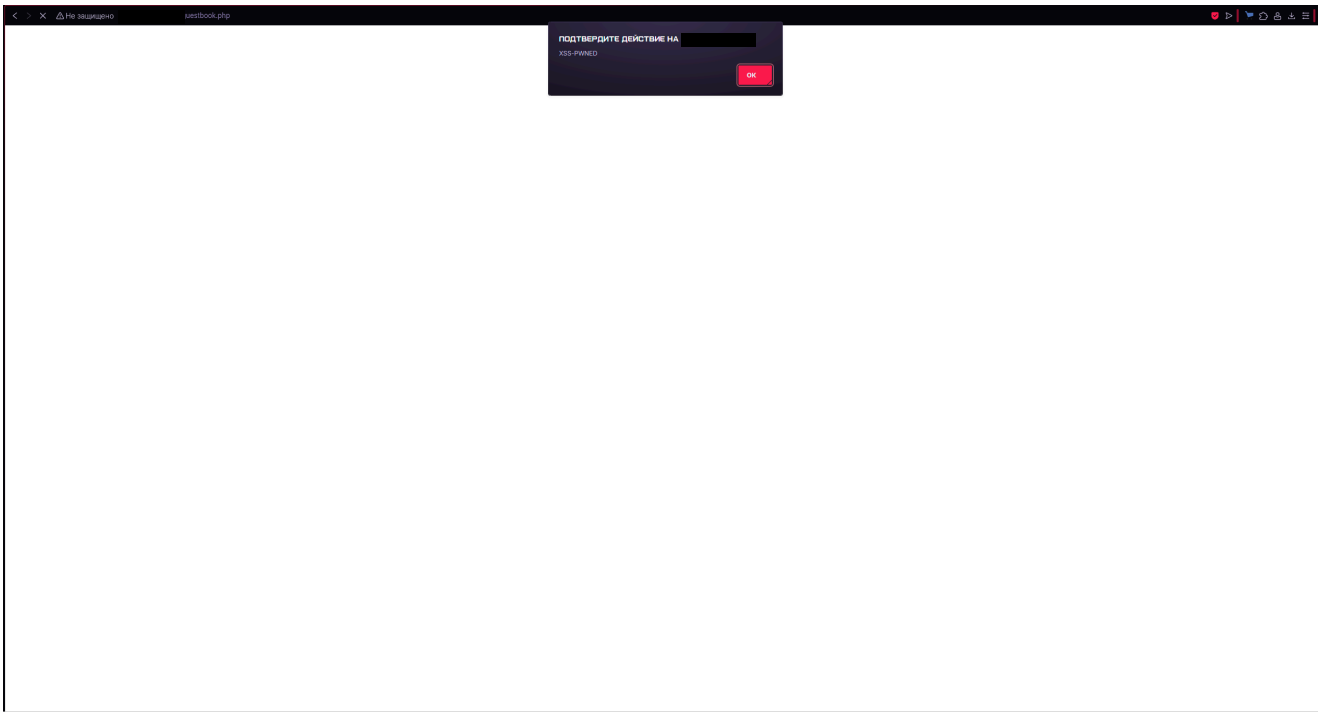
Результаты:

Появилось всплывающее окно `alert` с надписью `XSS-PWNED`.

Вывод:

Уязвимость **Stored XSS** на `http://[redacted]:8050/guestbook.php` **подтверждена вручную.** Это означает, что любой пользователь, открывший страницу гостевой книги, выполнит вредоносный

JavaScript-код.



Подтверждение SQL Injection / Обнаружение Error-Based SQL Injection на `users/login.php`:

Действия:

Перешёл на http://[redacted]:8050/users/login.php.
В поле **Username** ввёл полезную нагрузку из отчёта ZAP, предназначенную для XSS: `'"><scRipt>alert("XSS-PWNED-login");</scRipt>`.
Ввёл произвольный пароль.
Нажал кнопку входа.

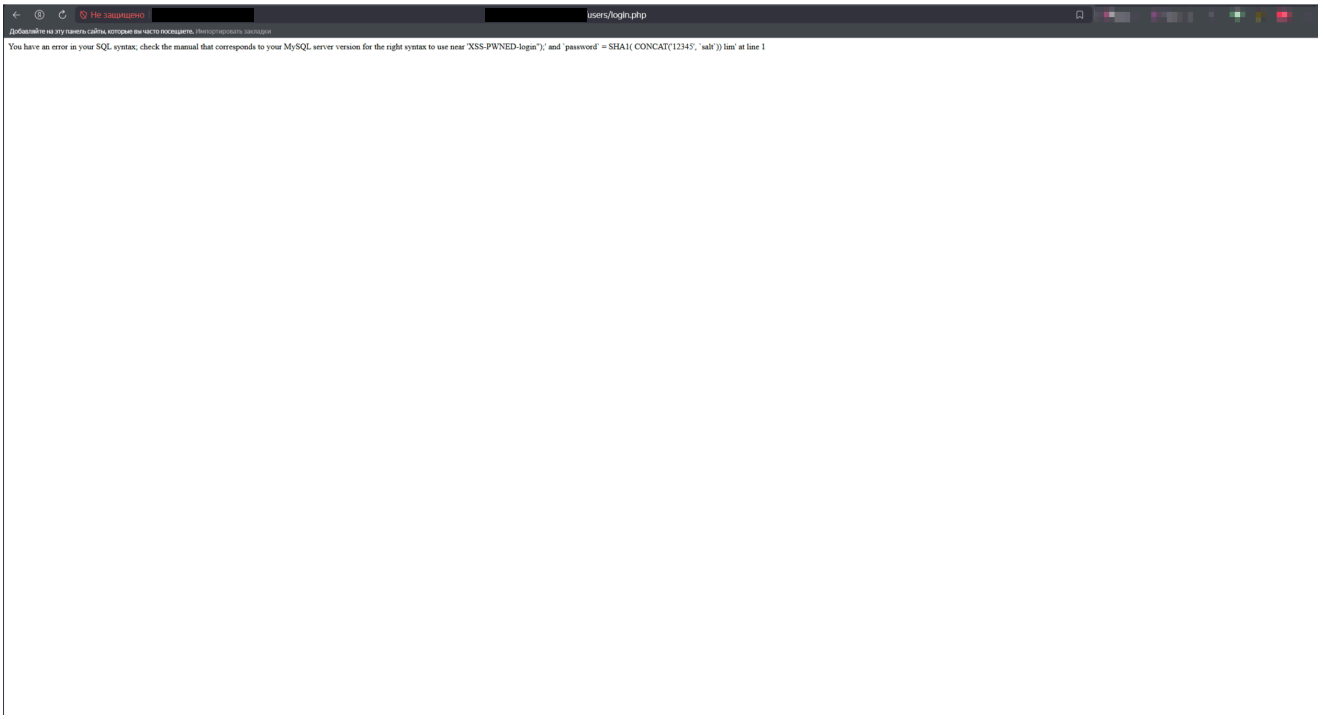
Результаты:

Вместо ожидаемого `alert` или сообщения об ошибке входа, была получена **ошибка SQL** от сервера: `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 'XSS-PWNED-login");' and password = SHA1( CONCAT('12345', salt)) lim' at line 1`

Вывод:

Попытка эксплуатации **Reflected XSS** привела к **ошибке SQL**.
Эта ошибка **доказывает наличие уязвимости SQL Injection** в логике аутентификации на http://[redacted]:8050/users/login.php. Специальные символы из XSS-полезной нагрузки (`'`, `)`, `;`) нарушили синтаксис SQL-запроса.
Факт наличия XSS-полезной нагрузки (`<scRipt>alert(...)`) внутри SQL-запроса также **косвенно подтверждает** точку отражения, необходимую для **Reflected XSS**, но сама ошибка помешала её

срабатыванию в этот раз. Это указывает на **две потенциальные уязвимости в одной точке входа.**



Вывод по ручному тестированию:

Ручное тестирование успешно **подтвердило эксплуатацию критических уязвимостей** на **обоих исследуемых портах** целевого хоста [redacted].

Порт 7788 (TornadoServer/5.1.1):

- **SQL Injection** на http://[redacted]:7788/login.html была подтверждена с помощью полезной нагрузки `' OR 1=1 --`, что позволило **обойти аутентификацию** и получить доступ к учетной записи администратора.
- **Remote OS Command Injection** на http://[redacted]:7788/server.html была подтверждена с помощью полезной нагрузки `127.0.0.1xYhoWJpBJAiRqDsC&cat /etc/passwd&`, что привело к **чтению содержимого системного файла /etc/passwd** и доказало возможность **выполнения произвольных команд операционной системы** на сервере.

Порт 8050 (Apache/2.4.7 , PHP/5.5.9-1ubuntu4.29):

- **Stored Cross-Site Scripting (XSS)** на http://[redacted]:8050/guestbook.php была подтверждена с помощью полезной нагрузки `</p><scRipt>alert(”XSS-PWNED”);</scRipt><p>`, что доказывает возможность **сохранения и последующего выполнения вредоносного JavaScript-кода** для всех пользователей, просматривающих гостевую книгу.
- **Reflected Cross-Site Scripting (XSS)** на http://[redacted]:8050/search?q=... была подтверждена с помощью полезной нагрузки `<script>alert(document.domain)</script>`, что доказывает возможность **выполнения JavaScript-кода в браузере жертвы** при переходе по специально сформированной ссылке или вводе данных в форму поиска.
- **Error-Based SQL Injection** на http://[redacted]:8050/users/login.php была **обнаружена косвенно** при попытке эксплуатации Reflected XSS. Ввод полезной нагрузки, содержащей специальные символы (`'"<scRipt>...`), привёл к **ошибке SQL синтаксиса**, раскрывающей структуру запроса. Это **доказывает наличие уязвимости SQL Injection** и **косвенно подтверждает точку отражения для XSS**.

Итог:

Ручное тестирование **полностью подтвердило** критический характер уязвимостей, обнаруженных на **Этапе 2 (Scanning)**. На обоих портах подтверждены уязвимости, позволяющие:

- **Обойти аутентификацию и получить несанкционированный доступ** (SQL Injection на порту 7788).
- **Выполнить произвольные команды на сервере ОС** (Command Injection на порту 7788).
- **Выполнить вредоносный JavaScript-код в браузере пользователей** (Stored и Reflected XSS на порту 8050).
- **Потенциально обойти аутентификацию** (Error-Based SQL Injection на порту 8050).

Эти результаты подтверждают **высокий уровень критичности** исследуемого хоста и реальный риск компрометации как данных приложений, так и самого сервера.

Рекомендации по устранению уязвимостей:

Важно: Ниже приведены общие рекомендации по устранению классов уязвимостей, подтверждённых в ходе тестирования. Конкретная реализация зависит от кода приложения.

1. SQL Injection (на портах 7788 и 8050)

Проблема:

- Входные данные пользователя напрямую конкатенируются в SQL-запросы без надлежащей валидации и экранирования.

Решение:

- **Использовать параметризованные запросы (Prepared Statements) или ORM.** Это самый надёжный способ. Все пользовательские данные должны передаваться в запрос как параметры, а не подставляться напрямую.
- **Валидация входных данных.** Проверять тип, длину, формат и диапазон входных данных на стороне сервера.
- **Экранирование специальных символов.** Если параметризованные запросы невозможны, использовать функции экранирования, предоставляемые СУБД, для всех пользовательских данных.
- **Принцип наименьших привилегий.** Учетная запись, используемая приложением для доступа к БД, должна иметь минимально необходимые права (не `root` / `sa` / `db_owner`).
- **Для порта 7788 (`login.html`):** Убедиться, что логика аутентификации использует параметризованные запросы при проверке логина и пароля.
- **Для порта 8050 (`users/login.php`, `pictures/search.php` и др.):** Проверить все точки, где пользовательский ввод используется в SQL-запросах, и применить параметризованные запросы.

2. Command Injection (на порту 7788)

Проблема:

- Пользовательский ввод передаётся напрямую в функции выполнения системных команд (например, `os.system`, `subprocess.call` с `shell=True` в Python).

Решение:

- **Избегать вызова системных команд с пользовательским вводом.** Если возможно, переписать функциональность, используя встроенные библиотечные функции (например, `socket.gethostbyname()` вместо `nslookup`).
- **Если вызов команды необходим:**
- **Никогда не использовать** `shell=True`.
- **Передавать аргументы как массив (список), а не одну строку.** Это предотвращает интерпретацию специальных символов оболочкой.

- **Строгая валидация входных данных.** Использовать белый список разрешённых символов/команд. Отклонять любой ввод, содержащий символы оболочки (|, &, ;, <, >, $, , \n, \r, (,), {, }, , , , ?, ~, #, `, ^, =, %).
- **Экранирование.** Если валидация недостаточна, использовать соответствующие функции экранирования для конкретной ОС и оболочки (например, shlex.quote() в Python).

3. Cross-Site Scripting (XSS) - Reflected и Stored (на порту 8050)

Проблема:

- Пользовательский ввод отражается или сохраняется и выводится в HTML-страницу без надлежащего экранирования.

Решение:

- **Контекстно-зависимое экранирование (Escaping/Encoding).** Это ключевой принцип. Перед выводом любого пользовательского (и любого другого непроверенного) данных в HTML, необходимо **экранировать** специальные символы в зависимости от **контекста** их использования:
- **Контекст HTML (** ... **):** Экранировать символы < , > , & , " , ' . Использовать функции типа htmlspecialchars() в PHP или аналоги.
- **Контекст атрибута HTML (** <input value="..."> **):** Экранировать " (если значение в двойных кавычках), ' (если в одинарных), & . Лучше использовать HTML-кодирование (например, ").
- **Контекст JavaScript (** <script>var x = "...";</script> **):** Использовать JSON-энкодинг (например, json_encode() в PHP) и выводить данные внутри кавычек. Никогда не вставлять пользовательские данные непосредственно в JavaScript.
- **Контекст URL (** <a href="?q=..."> **):** URL-энкодировать данные (например, urlencode() в PHP).
- **Content Security Policy (CSP).** Реализовать строгую политику CSP в HTTP-заголовке Content-Security-Policy . Это может предотвратить выполнение скриптов, даже если они каким-то образом проникли на страницу.
- **Не использовать** innerHTML **для вставки пользовательских данных в DOM.** Использовать безопасные методы, такие как textContent .
- **Для Stored XSS (** guestbook.php **):** Экранировать данные при **выводе** из БД/хранилища, а не при сохранении. Это гарантирует, что данные экранируются в правильном контексте.
- **Для Reflected XSS (** search?q=... **):** Экранировать данные при **выводе** в HTML-ответ.

4. Общие рекомендации по безопасности (на основе обнаруженных недостатков):

Обновление ПО:

- **Apache 2.4.7 и PHP 5.5.9** являются **крайне устаревшими** и содержат **множество известных критических уязвимостей** (подтверждено на Этапе 1 и Этапе 2). **Немедленно обновить** до актуальных, поддерживаемых версий. Это устранит большинство автоматически обнаруженных уязвимостей (vulners , nikto).

HTTP-заголовки безопасности:

- X-Frame-Options: DENY **или** SAMEORIGIN : Предотвращает Clickjacking. (Найдено отсутствующим на портах 7788 и 8050).
- X-Content-Type-Options: nosniff : Предотвращает MIME-sniffing браузером. (Найдено отсутствующим на портах 7788 и 8050).

- `Content-Security-Policy`: Определяет, откуда можно загружать ресурсы и выполнять скрипты. Сильно снижает риск XSS. (Найдено отсутствующим).
- `Strict-Transport-Security` **(HSTS)**: Принудительно использовать HTTPS. (Рекомендуется, если HTTPS используется).

Флаги Cookie:

- `HttpOnly`: Запрещает доступ к cookie через `document.cookie` в JavaScript. (Найдено отсутствующим на порту 7788).
- `Secure`: Cookie передаются только по HTTPS. (Рекомендуется, если HTTPS используется).
- `SameSite`: Предотвращает отправку cookie с запросами, инициированными другими сайтами (CSRF). Рекомендуется `SameSite=Lax` или `SameSite=Strict`.

Индексация каталогов:

- **Отключить** возможность просмотра содержимого каталогов (Directory Listing) для всех директорий на веб-сервере (`/cart/`, `/css/`, `/users/`, `/images/` на порту 8050). Это настраивается в конфигурации Apache (`Options -Indexes`).

Удаление ненужных файлов:

- Удалить тестовые файлы (`test.php`, `#wp-config.php#`), файлы README (`/icons/README`), и другие ненужные артефакты разработки с production-сервера.