Міністерство освіти й науки
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

«Тестування веб-застосунків на вразливість та проникнення»

Текстова частина до курсової роботи за спеціальністю
«Комп'ютерні науки та інформаційні технології» - 122

Керівник курсової роботи:
доцент
Олецький О. В.
_____(підпис)
«___» _____ 2020 р.
Виконала студентка:
Михальова Валерія
«___» _____ 2020 р.

Київ 2020

Календарний план виконання курсової роботи
Тема: Тестування веб-застосунків на вразливість та проникнення

No п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	6 жовтня 2019 р.	
2.	Огляд літератури за темою роботи	20 грудня 2019 р.	
3.	Створення середовища для тестування	17 січня 2020 р.	
5.	Information Gathering	20 березня 2020 р.	
6.	Вивчення найпоширеніших атак на веб-застосунки	25 березня 2020 р.	
8.	Реалізація атаки SQL Injection та CSRF	26 березня 2020 р.	
9	Реалізація атаки XSS та Clickjacking	2 квітня 2020 р.	
10.	Вивчення захисту від реалізованих атак	6 квітня 2020 р.	
13.	Захист курсової роботи	19 квітня 2020 р.	

ЗМІСТ

Вступ

Актуальність теми. Прогалини в безпеці з'являються на різних стадіях процесу і залежать від безлічі факторів: помилка проектування, невдала конфігурація обладнання та програмного забезпечення, проблеми з мережею, людська помилка. Вони дають неавторизованому користувачеві можливість для атаки на систему, що впливає на її цілісність і конфіденційність. Таким чином, тестування програмних продуктів на вразливість та проникнення допомагає позбутися цих вразливостей і зробити систему достатньо компетентною для захисту від очікуваних і навіть несподіваних шкідливих загроз і атак.

Метою курсової роботи є провести тестування веб-застосунків на вразливість та проникнення. Випробувати можливості Kali Linux – ОС, розробленої для тестувальників на проникнення. Розглянути такі види атак, як File Upload, SQL Injection, CSRF, XSS, Clickjacking а також продемонструвати їх роботу на практиці та зрозуміти, як писати захищений веб-застосунок від цих та подібних атак. Провести збір інформації про веб-застосування та написати власну програму пошуку піддоменів будь-якого сайту. Дінатись, як можна поліпшити сайт від великої кількості атак за допомогою CSP.

Розділ 1. Збір інформації про веб-застосування

1.1 Information Gathering

Information Gathering – це процес збору різних видів інформації про цільовий сайт чи систему. Це перший крок і початковий етап етичного хакінгу. Це роблять і тестувальники на проникнення і хакери. Чим більше інформації зібрано про цільовий сайт, тим більша ймовірність отримати відповідні результати, тому до цього етапу потрібно підходити серйозно. Існують різні інструменти, методи та веб-сайти, включаючи публічні джерела, такі як Whois, Nslookup, які можуть допомогти хакерам зібрати інформацію. Цей крок необхідний, тому що, виконуючи атаки на будь-яку ціль, вам може знадобитися будь-яка інформація.

Найбільш важливою для нас є інформація про:

- IP адресу сайту
- доменне ім’я
- технології, які використовуються,
- інші веб-сайти на цьому ж сервері,
- піддомени, приховані директорії

Для проведження цього етапу було вибрано сайт університету – ukma.edu.ua. Через деякий час, скориставшись такими ресурсами, як whois.domaintools.com, robtex.com, sitereport.netcraft.com, отримали такі дані:

- IP: 194.44.142.7
- Реєстратор домену: whois.ua
- Хостинг-компанія: ukma.kiev.ua

- Сервер: Apache/2.2.31 (FreeBSD) mod_ssl/2.2.31 OpenSSL/0.9.8x DAV/2

Тепер нам відома дата створення сайту, IP-адреса, сервер імен, хостинг-компанія та сам сервер із технологіями, які на ньому використовуються.

1.2 Пошук піддоменів за допомогою SublistFinder

Для знаходження піддоменів існує багато різних ресурсів та застосувань. Найпростіший спосіб – це, так само, шукати їх вручну на різних сайтах та використовуючи різні пошукові системи. Але це, очевидно, займає велику кількість часу, якщо ми хочемо знайти максимальну кількість піддоменівів. Тому, ми спробуємо автоматизувати цей процес і напишемо програму SublistFinder.

Програма SublistFinder допоможе пентестерам, або тестувальникам на проникнення, знаходити велику кількість піддоменів вказаного веб-сайту чеез різні джерела Open-source intelligence. Тут використовуються різні пошукові системи, а для самого пошуку використовуються такі ресурси, як Netcraft, Virustotal, ThreatCrowd, DNSdumpster та PassiveDNS. Також тут використовується метол брутфорс,за допомогою якого перебирається більше 100 000 піддоменів та більше 2 000 серверів імен, які відповідають на декілька запитів за 1 секунду. Дана програма дозволяє отримати повний (або майже) список піддоменів менше, ніж за хвилину.

На вхід ми приймаємо домен та назву файлу де запишемо результати (Рис 1.1).

Рис 1 1 Запуск програми SublistFind

І менше, ніж ха хвилину ми отримуємо список піддоменів ukma.edu.ua (Рис 1.2)

Рис 1 2 Список піддоменів ukma.edu.ua

Розділ 2. Вразливості. Знаходження, реалізація та захист.

2.1 Створення середовища для тестування

Для тестування застосунків було використано Oracle VM VirtualBox - програмний продукт віртуалізації для операційних систем. Було встановлено віртуальні операційні системи – Kali Linux та Metasploitable. Операційна система Kali Linux призначена для користувачів і розробників, які тестують різні системи і сервіси на уразливості або можливість злому. Дистрибутив включає сотні інструментів, за допомогою яких можна перевіряти експлуатації відомих вразливостей і тестувати проекти на захищеність.

Реалізовувати атаки на проникнення не є легальною задачою. Тому для навчання та дослідження вразливостей використовується ОС Metasploitable. Це навмисно вразлива віртуальна машина Linux, яка може бути використана для проведення тренінгів з безпеки, тестування інструментів безпеки та практичних методів тестування на проникнення. Вона має 3 рівні захисту: low, medium та high, а також надає можливість подивитись код певної серверної частини.

Ми також повинні налаштувати мережу відповідним чиином. Для цього створюємо в налаштуваннях VirtualBox NatNetwork і підключаємо до неї обидва комп’ютери з ОС Kali Linux та Metasploitable. Це налаштування відповідає за створення віртуальної мережі, де наша хост-машина (наш комп’ютер) буде маршрутизатором для цієї мережі, а всі віртуальні машини будуть клієнтами, підключеними до цієї мережі.

Тепер ми можемо відкрити віртуальну машину Metasploitable. Бачимо, що її IP адреса 10.0.2.255 (Рис 2.1)

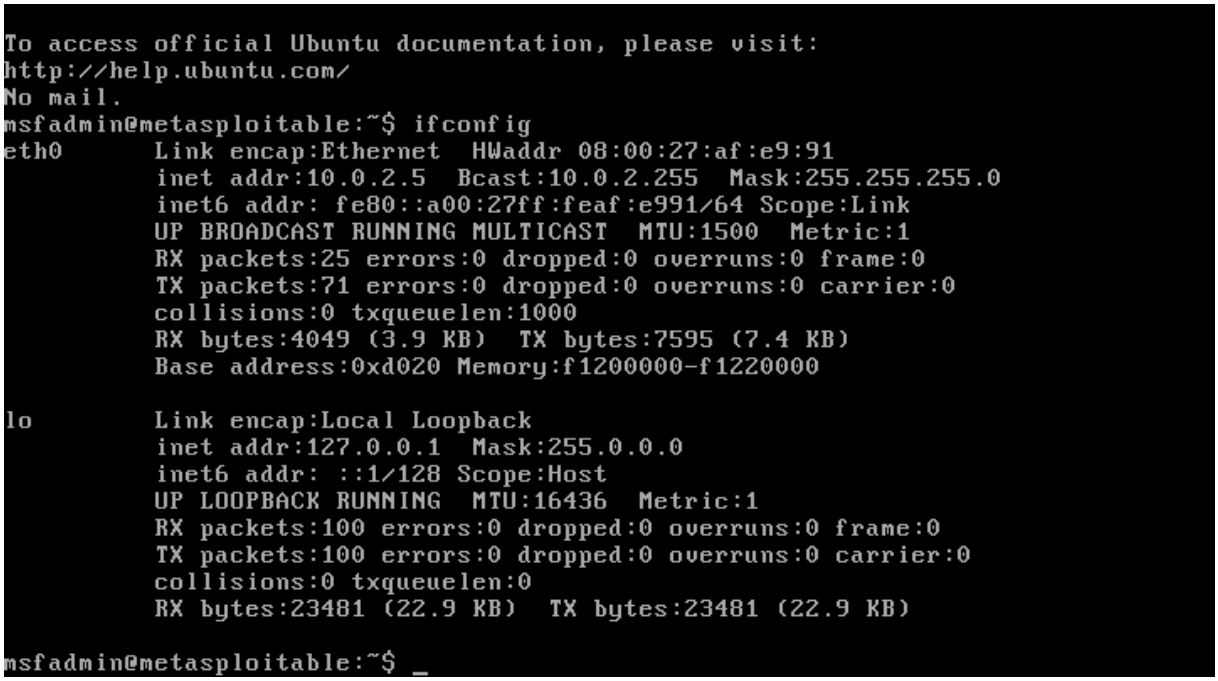

Рис 2 1 Metasploitable IP

Завдяки правильним налаштуванням мережі, якщо ми прейдемо до Kali Linux і введемо 10.0.2.255 - IP адресу веб-сервера, який встановлений на Metasploitable, то побачимо, що тут є ряд застосунків, які ми будемо використовувати для тестування на проникнення (Рис 2.2)

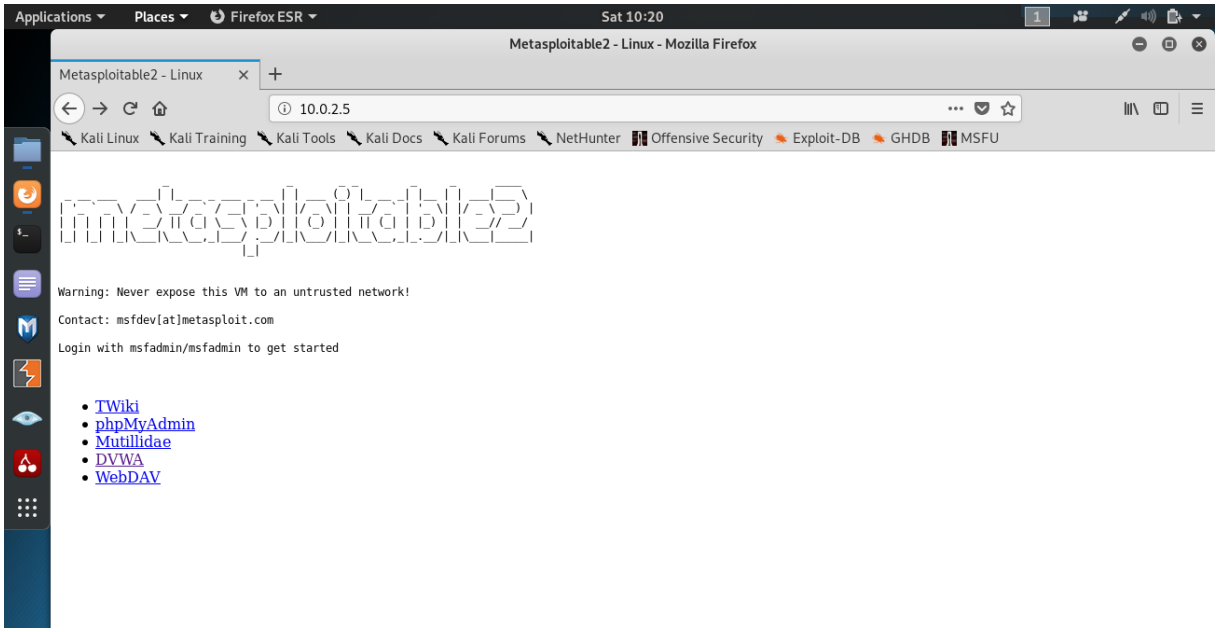

Рис 2 2 Metasplotable в браузері

2.2 File upload

File upload – це така вразливість, коли програма дозволяє користувачу безпосередньо завантажувати шкідливий файл, який потім виконується. Уразливим на таку атаку є місце веб-застосунку, де ми дозволяємо користувачу завантажити своє фото, щоб відобразити його в профілі. Зрозуміло, що ми очікуємо, що корисутувач завантажить саме файл із розширенням “jpg” чи “jpeg”, де він буде знаходитись на гарному фоні із усміхненим обличчям, але, на жаль, не завжди все відбувається по такому сценарію.

2.2.1 Реалізація File upload атаки

Для реалізації цієї атаки нам буде потрібен DVWA(Damn Vulnerable Web Application), який знаходиться на сервері Metasploitable. Тут є можливість завантаження файлу. Якщо все добре, до сервер нам надсилає відповідь у вигляді місця розташування цього файлу. Раніше ми вже ознайомились із етапом збору інформації, в ході якого було виявлено, що серверна частина застосунку DVWA написана мовою PHP. Це означає, що файли з рошзиреннм ".php" зможуть виконуватись. Отже, нашою ціллю буде завантаження php-файлу замість картинки.

Тут ми будемо використовувати Weevely – веб-шелл командного рядка, який динамічно розповсюджується по мережі під час виконання, призначений для віддаленого адміністрування і тестування на проникнення. Він вже є встановленим на Kali Linux. Ця програма забезпечить схожий на ssh термінал в обмеженому оточенні. Все, що потрібно зробити, це створити php-файл і встановити пароль (Рис 2.3).

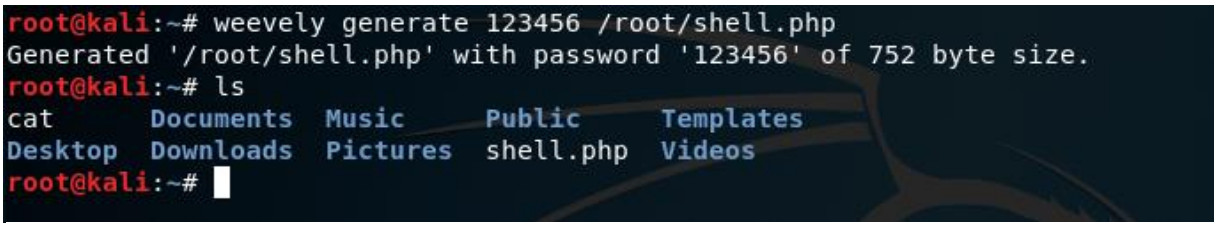

Рис 2 3 Weevely

Завантаження файлу – це POST запит і для того, щоб ми могли контролювати запити, використаємо proxy-сервер із застосунку Burp Suite (Рис 2.4)

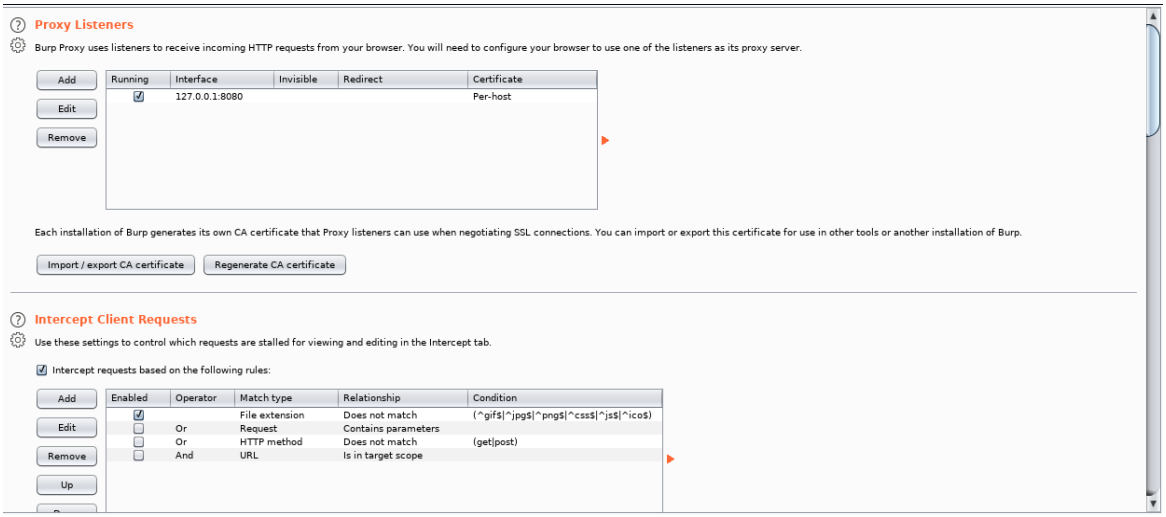

Рис 2 4 Burp Suite

Proxy Burp працює по протоколу HTTP(S) в режимі man-in-the-middle. Перебуваючи між браузером і веб-додатком він дозволить перехоплювати, вивчати і змінювати трафік в обох напрямках.

Отже, для раніше створеного файлу “shell.php” змінимо розширення – “shell.jpg”на комп’ютері. Коли ми перехопимо цей запит в proxy сервері, то його Content-Type буде “image/jpeg”, тому що ми завантажуємо файл із розширенням картинки. Але тепер, перехопивши запит, ми змінюємо його

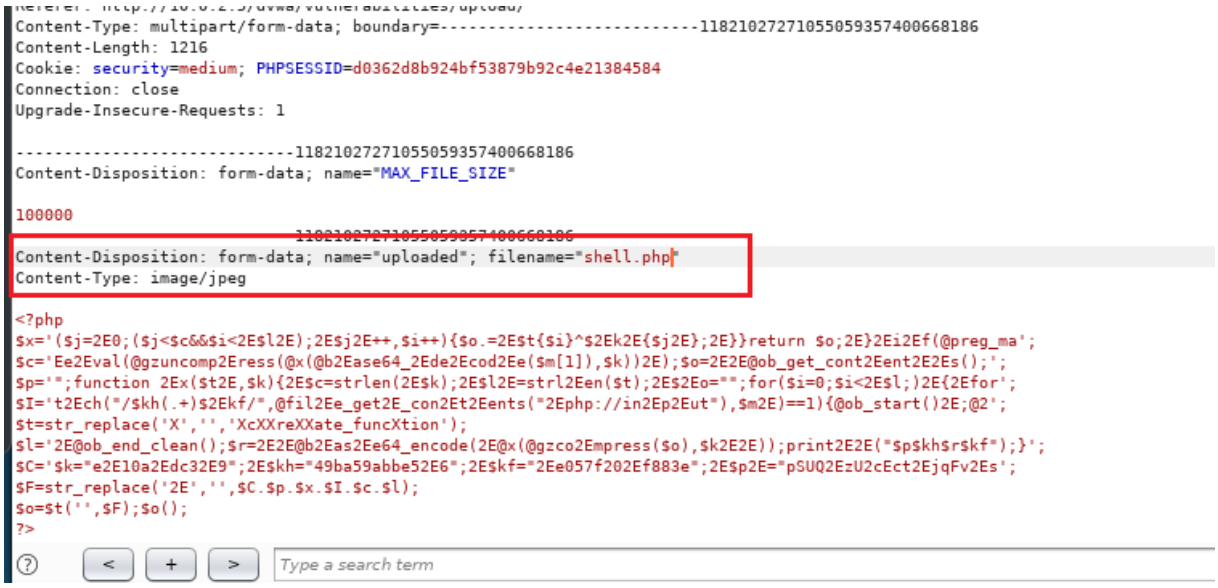

Рис 2 5 Завантаження файлу

значення “file-name”, зокрема залишаємо назву незмінною, а розширення повертаємо на “.php”. На Рис 2.5 бачимо, що в заголовку запиту Content-Type: image/jpeg, тому Metasploitable на рівні захисту middle пропускає такий файл, але тепер він вже зможе виконатись.

Звичайно в реаліях ми можемо лише здогадуватись як в серверній частині перевіряється завантажувальний користувачем файл, але Metasplotable дає можливість подивитись, як це реалізовано у них. Тут перевіряється лише наявність “Content-Type: image/jpeg”. Якщо підвищити рівень захисту, то додасться перевірка файлу на розширення. В такому випадку ми не зможемо просто змінити його розширення з “.jpg” на “.php”. Але нам ніхто не заважає перейменувати файл на “shell.php.jpg” після перехоплення його proxy сервером. Таким чином файл теж завантажиться.

Тепер, коли в нас завантажений створений нами php-файл за допомогою Weevely, все, що нам залишилось зробити – вказати місце розташування цього файлу та ввести пароль (Рис 2.6).

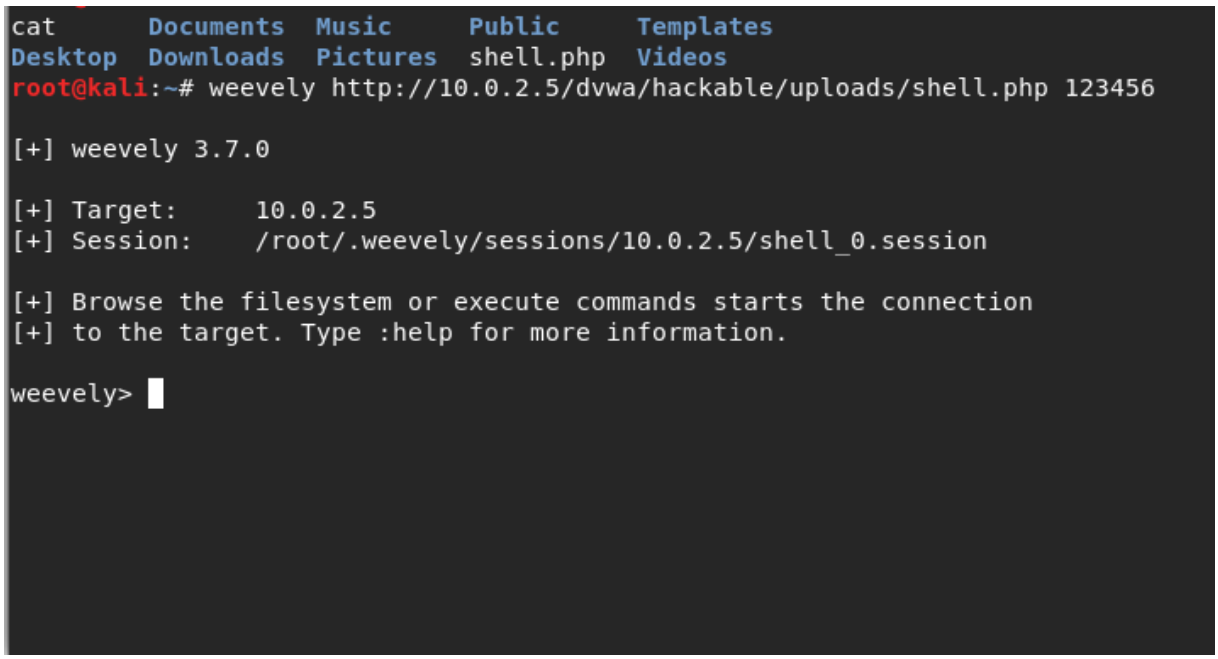

Рис 2 6 Weevely shell

Weevely – це агент та більше 30 модулів, які формують розширюваний фреймворк для адміністрування веб-акаунтів. Віддаленим агентом є маленький PHP скрипт, який може розширювати свою функціональність через мережу в реальному часі. Код агента є поліморфним і важко виявляються антивірусами. З його допомогою можна видаляти, змінювати файли, робити пошук по файлам, виконувати SQL-запити на сервері, та багато іншого. По суті, ми отримуємо повний контроль над веб-застосунком.

2.2.2 Захист від File upload атаки

Щоб захистити свої застосування від file upload атаки потрібно:

1. Ніколи не надавати змогу користувачу завантажувати виконувані файли
2. Перевіряти тип файлу та його розширення
3. Аналізувати завантажений файл самим, перестворювати його та переназивати.

На Рис 2.7 показано, як це повинно виглядати в нашому випадку.

```php
// Is it an image?
  if( ( strtolower( $uploaded_ext ) == 'jpg' || strtolower( $uploaded_ext ) == 'jpeg' || strtolower( $uploaded_ext ) == 'png' ) &&
    ( $uploaded_size < 100000 ) &&
    ( $uploaded_type == 'image/jpeg' || $uploaded_type == 'image/png' ) &&
    getimagesize( $uploaded_tmp ) ) {

    // Strip any metadata, by re-encoding image (Note, using php-Imagick is recommended over php-GD)
    if( $uploaded_type == 'image/jpeg' ) {
      $img = imagecreatefromjpeg( $uploaded_tmp );
      imagejpeg( $img, $temp_file, 100);
    }
    else {
      $img = imagecreatefrompng( $uploaded_tmp );
      imagepng( $img, $temp_file, 9);
    }
    imagedestroy( $img );

    // Can we move the file to the web root from the temp folder?
    if( rename( $temp_file, ( getcwd() . DIRECTORY_SEPARATOR . $target_path . $target_file ) ) ) {
      // Yes!
      $html .= "<pre><a href='${target_path}${target_file}'>${target_file}</a> succesfully uploaded!</pre>";
    }
    else {
      // No
      $html .= '<pre>Your image was not uploaded.</pre>';
    }

    // Delete any temp files
    if( file_exists( $temp_file ) )
      unlink( $temp_file );
  }
  else {
    // Invalid file
    $html .= '<pre>Your image was not uploaded. We can only accept JPEG or PNG images.</pre>';
  }
}
```

Рис 2 7 File upload захист

2.3 SQL Injection

SQL Injection – це техніка введення шкідливого коду, який може негативним чином вплинути на вашу базу даних. Це одна з найпоширеніших методик злому веб-сторінок. Як правило, вона відбувається, коли ви запитуєте користувача про введення даних, наприклад, ім’я користувача, і замість імені / ідентифікатора користувач дає вам вираз SQL , який ви несвідомо запускаєте у базі даних. Потенційно

вразливом до такого роду вразливості є не лише input поля, а й url, коли ми відправляємо GET-запит.

Коли HTTP протокол ще не був безпечним, таку атаку можна було легко реалізувати і ввійти, наприклад, в аккаунт користувача, знаючи лише його логін (Рис 2.8). Так можна було б задовільнити усі фільтри з боку фронтенду, зокрема на кількість символів, а вкінці додати через логічне “або” вірне твердження. Якщо хоча б одне твердження вірне, тобто 1 = 1, то результат буде вірним, навіть не зважаючи на неправильне перше твердження, тобто пароль.

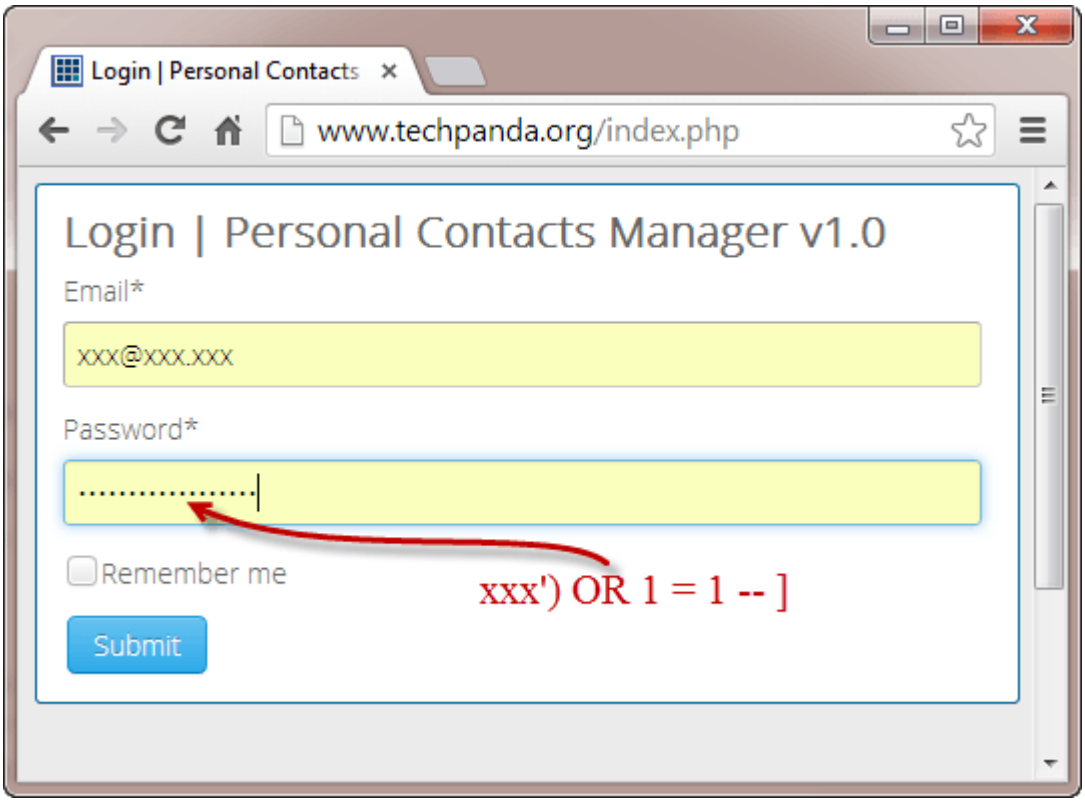

Рис 2 8 SQL Injection bypassing password

2.3.1 Реалізація SQL Injection

Для того, щоб реалізувати SQL Injection атаку, зареєструємось з ім’ям “lera” та парольом “123456”. Тепер спробуємо ввести ім’я

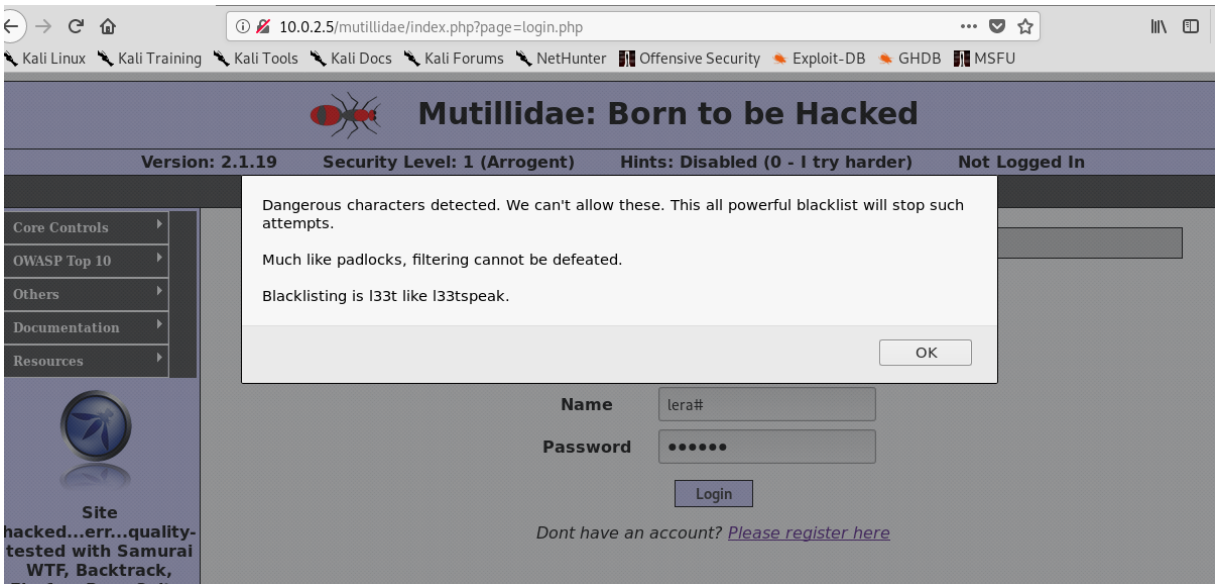

Рис 2 9 Помилка на фронтенді

“lera#” та довільний пароль з розрахуноком на те, що все, що в запиті знаходиться після “#” буде коментарем та не виконається. Але застосунок не допускає значення такого імені(Рис 2.9) Тепер потрібно зрозуміти чи нам це заважають зробити фільтри, написані на клієнтській частині сайту чи сервер. Для цього використаємо Burp Suite Proxy і розуміємо, що контроль відбувається на клієнті. Для нас це чудово, тому що такі фільтри можна обійти. Отже, вводимо ім’я користувача та довільне значення паролю. Запит пройшов (Рис 2.10) і ми можемо подивитись його вміст.

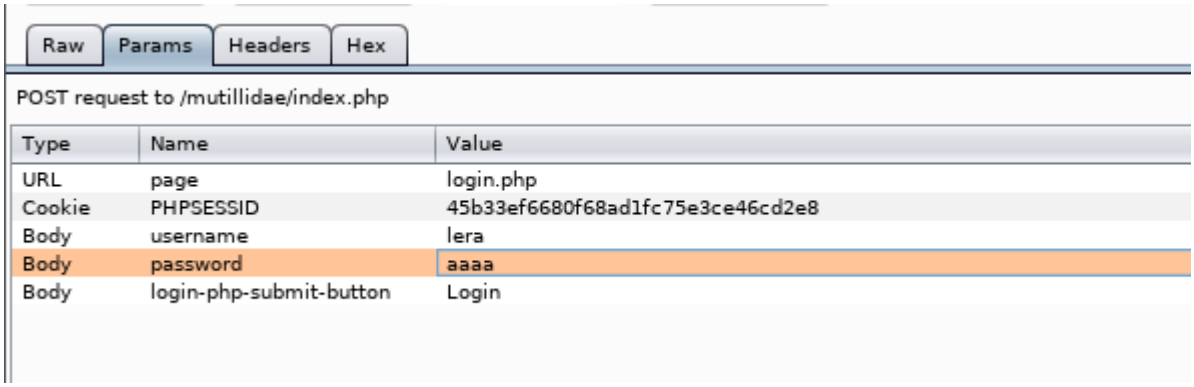

Рис 2 10 Burp Suite SQL Injection

Burp Suite нам дозволяє змінити вміст запиту, що ми і робимо (Рис 2.11)

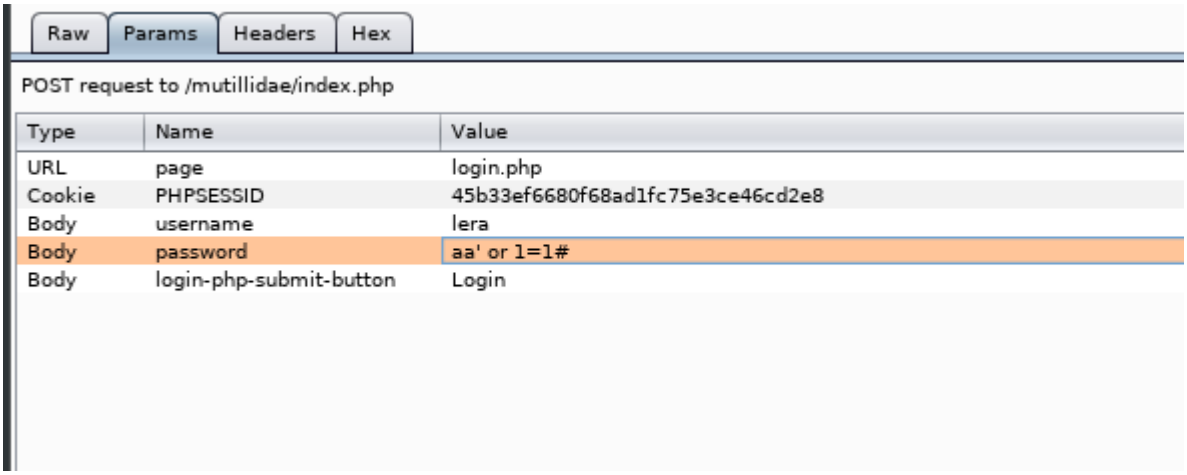

Рис 2 11 Зміна паролю через Burp Suite

Таким чином значення паролю буде true в будь-якому випадку, тому що 1=1 і ми зайшли у застосунок під ім’ям “lera”.

Нам вдалось це обійти, тому що значення полів в коді безпосередньо вставляється в текст, як змінна (Рис 2.12), де змінною може бути шкідливий SQL-вираз.

```
$query =        "SELECT * FROM accounts WHERE username='".
                $username.
                "' AND password='".
                $password
```

Рис 2 12 Souce code

```
$query  =       "SELECT * FROM accounts WHERE username='".
                $conn->real_escape_string($username) .
                "' AND password='".
                $conn->real_escape_string($password).
                "'";
```

Рис 2 13 Source code impoved

В даному випадку ми можемо покращити цей код таким чином (Рис 2.13). Це дає нам гарнтію, що значення відповідного поля буде саме стрічкою і не зможе виконатись.

SQL Injection атака може бути присутньою і в GET запитах. Для її реалізації перейдемо на сторінку, де за тими ж полями ми отримуємо іноформацію про користувача (Рис 2.14)

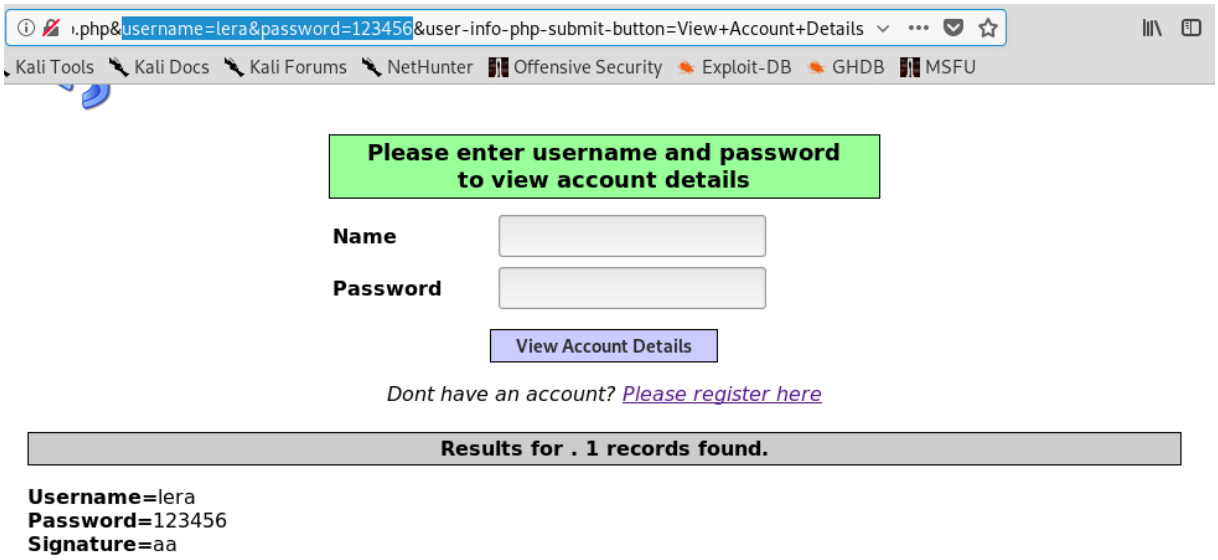

Рис 2 14 Login page

Тепер змінимо значення url-параметрів з “username=lera&password= 123456” на “username=lera' order by 1 %23&password=123456”. “%23” – це та сама “#”. Результатом будуть усі користувачі з ім’ям “lera” які задовільняють наш order by (Рис 2.15).

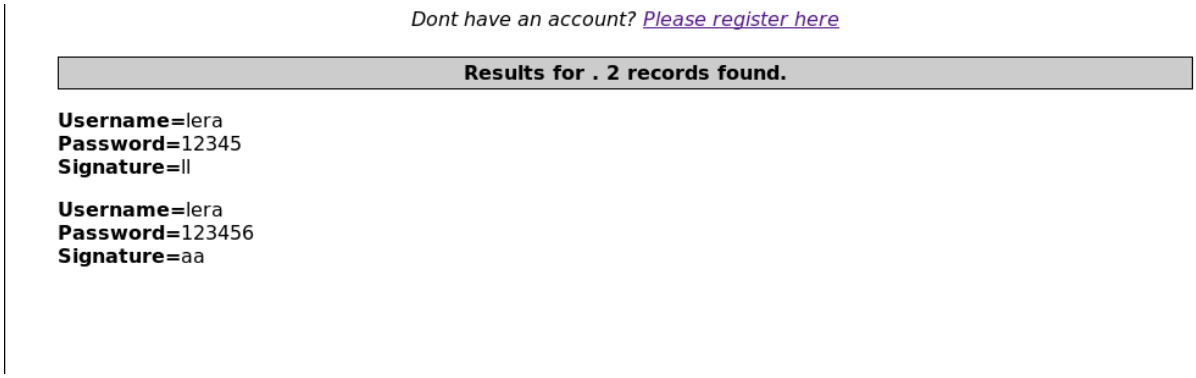

Рис 2 15 Результат SQL Injection

Потрібно врахувати, що тут теж можуть стояти різного роду фільтри на допустимі значення параметрів URL, але самий шкідливий текст теж можна написати по-різному, наприклад ось так “ uNiOn+SeLecT+1,2+%23”.

Всі ці шкідливі SQL маніпуляції зазвичай не роблять вручну. Для цьогог є набір готових рішень. Тут ми розгялнемо одне із них. SQLmap - це інструмент з відкритим вихідним кодом для тестування на проникнення, який автоматизує процес виявлення і експлуатування SQL injection вразливостей і захоплення серверів баз даних.

Спробуємо протестувати наш сайт (Рис 2.16)

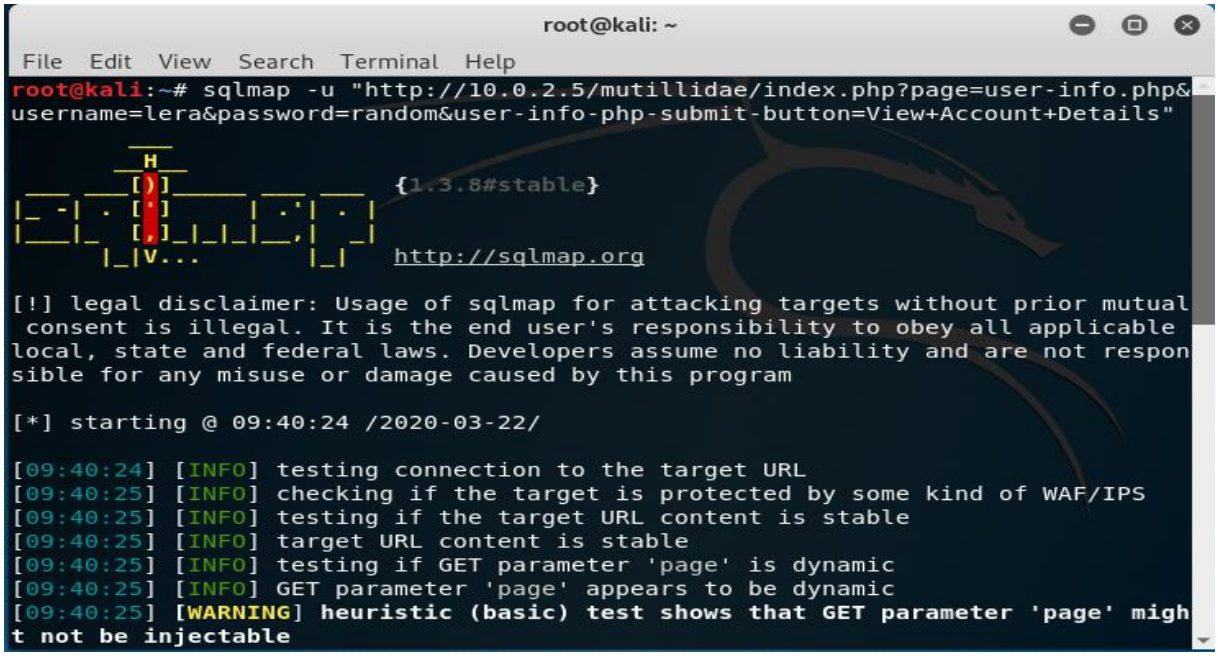

Рис 2 16 SQLmap

Значення параметрів може бути довільним. З Рис 2.17 видно, що поле з іменем “username” вразливе до SQL-injection.

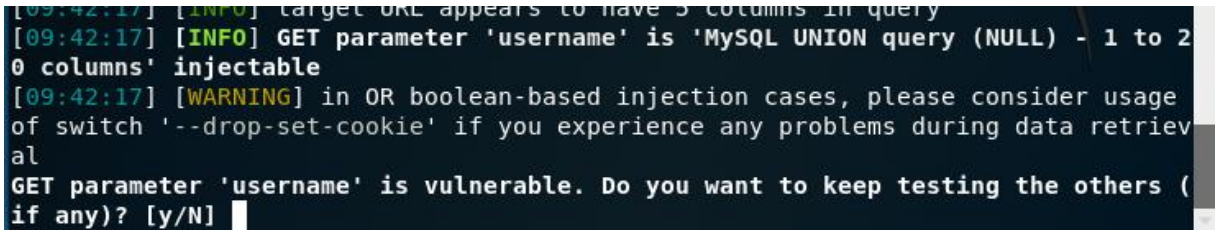

Рис 2 17 Результат SQLmap

А це означає, що ми маємо повний контроль над базою даних сервера. Для прикладу, ми можемо переглянути усі бази даних сервера (Рис 2.18) і маніпулювати ними довільним чином.

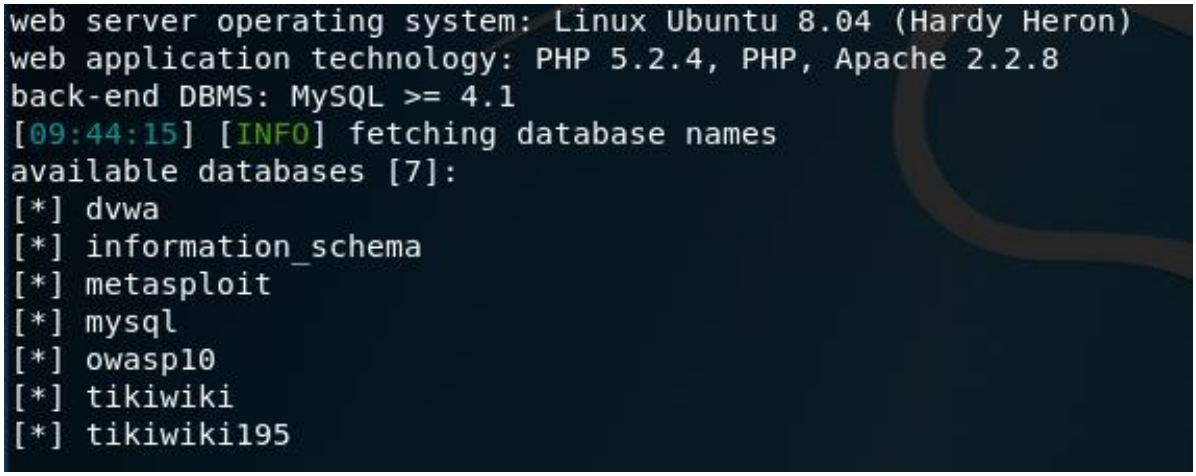

Рис 2 18 БД сервера

2.3.2 Захист від SQL Injection

Більшість успішних атак грунтується на коді, написаному без урахування відповідних вимог безпеки. Не потрібно довіряти ніяким даним, особливо якщо вони надходять з боку клієнта, навіть якщо це приховані поля або куки. Можна узагальнити правила безпеки таким чином:

- Не можна спілкуватись з базою даних, використовуючи обліковий запис власника бази даних або суперкористувача.
- Завжди портібно використовувати спеціально створених користувачів з максимально обмеженими правами.
- Потрібно використовувати підготовлені вирази з прив'язаними змінними у коді.
- Завжди потрібно перевіряти введені дані на відповідність очікуваному типу.

2.4 Cross Site Request Forgery

CSRF (Cross Site Request Forgery) – вид атаки на відвідувачів веб-сайтів, який використовує недоліки протоколу HTTP. Якщо жертва заходить на сайт, створений зловмисником, від її особи таємно відправляється запит на інший сервер, який здійснює якусь шкідливу операцію. Для здійснення даної атаки жертва повинна бути аутентифікованою на тому сервері, на який відправляється запит, і цей запит не повинен вимагати будь-якого підтвердження з боку користувача, яке не може бути проігноровано або підроблено атакуючим скриптом.

Основне застосування CSRF - змушення виконання будь-яких дій на уразливому сайті від імені жертви (зміна пароля, секретного питання для відновлення пароля, пошти, додавання адміністратора).

Рис 2 19 Форма логіну

2.4.1 Реалізація CSRF

Ми спробуємо змінити пароль адміністратора використовуючи CSRF атаку. У нас є форма зміни паролю (Рис 2.19). Давайте відкриємо панель розробника і скопіюємо все, що знаходиться між тегами <form> і </form> в окремий файл (Рис 2.20)

```
<form action="#" method="GET">
    New password:<br>
    <input autocomplete="off" name="password_new"
type="password"><br>
    Confirm new password: <br>
    <input autocomplete="off" name="password_conf"
type="password">
    <br>
    <input value="Change" name="Change" type="submit">
 </form>
```

Рис 2 20 Форма HTML

Тепер додаму до <input> тегів атрибут “type” із значенням “hidden”, додамо значення “value” нового паролю , приберемо весь текст, додамо до форми атрибут id, а значення атрибуту action змінтимо на URL сторінки, де знаходиться оригінальна форма (Рис 2.21).

```
<form id="form1" action="http://10.0.2.5/dvwa/vulnerabilities/
csrf/" method="GET">
    <input type="hidden" autocomplete="off" name="password_new"
type="password" value="newpassword">
    <input type="hidden" autocomplete="off" name="password_conf"
type="password" value="newpassword">
    <input type="hidden" value="Change" name="Change"
type="submit">
 </form>

<script>document.getElementById('form1').submit();</script>
```

Рис 2 21 Модифікуємо фрому

Також напишемо простий скрипт, який сабмітить нашу форму і збережемо файл, як “index.html”. Якщо ми відкриємо його через браузер, то побачимо

пусту сторінку, але якщо повернуитсь назад на сайт з формою, то побачимо повідомлення, що пароль було змінено (Рис 2.22)

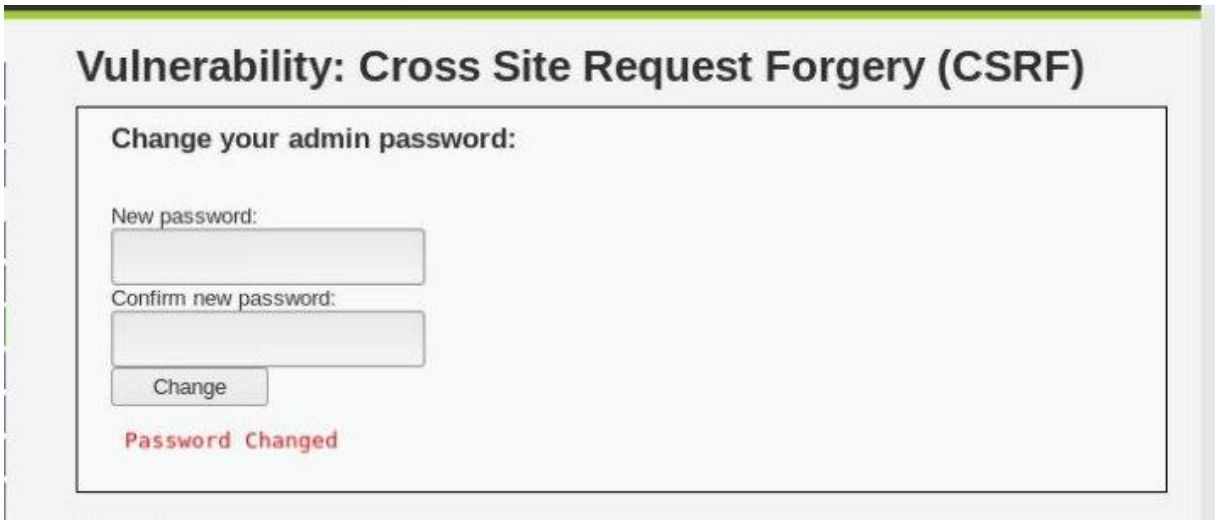

Рис 2 22 Пароль змінено

Звісно, що для реалізації атаки в звичайних умовах цю сторінку повинен відкрити адміністатор. Можна використати веб-хостинг, розмістити там сайт і за допомогою соціальної інженерії змусити адміністратора перейти за посиланням.

2.4.2 Захист від CSRF

Щоб зловмисник не зміг змінити пароль адміністоратора так, як це зробили ми, достатньо додати ще одне поле – “current password”, яке зможе заповнити лише адміністратор. Але цей метод допомагає захистити сайт від CSRF атаки лише в одному конкретному випадку. Щодо загальних правил, то є простий спосіб, який часто використовується – створення токену.

При старті сесії на стороні сервера генерується токен. Токен зберігається на стороні сервера для подальшої перевірки. У відповідь на запит клієнту повертається токен. Якщо рендеринг відбувається на сервері, то токен може повертатися всередині HTML, як, наприклад, одне з полів форми, або всередині <meta> тегу. Якщо відповідь повертається через клієнта, токен можна передавати в хедер. Часто для цього використовують X-CSRF-Token. При повторних запитах клієнт зобов'язаний передати токен сервера для перевірки. При отриманні запиту небезпечним методом (POST,

PUT, DELETE, PATCH) сервер зобов'язаний перевірити на ідентичність токен з даних сесії і токен, який надіслав клієнт. Якщо обидва токена збігаються, то пропускаємо запит, в іншому випадку - логуємо подію і відхиляємо запит.

2.5 XSS атаки

XSS (Cross Site Scripting) атака - це тип атаки, в якій шкідливі скрипти вводяться в іншу, доброякісну і надійну веб-сторінку. Атаки XSS виникають, коли зловмисник використовує веб-додаток для надсилання шкідливого коду, як правило, у вигляді стороннього скрипту, до іншого кінцевого користувача. Недоліки, які дозволяють цим атакам досягти успіху, є досить поширеними і відбуваються в будь-якому місці, де веб-додаток використовує вхідні дані від користувача у вихідних даних, які він генерує без перевірки чи кодування.

Зловмисник може використовувати XSS для відправки шкідливого скрипту нічого не підозрюючому користувачеві. Браузер кінцевого користувача не має можливості дізнатися, що скрипту не можна довіряти, і виконає сценарій. Оскільки він вважає, що сценарій надходив з надійного джерела, шкідливий скрипт може отримати доступ до будь-яких файлів. Ці сценарії можуть навіть переписати вміст HTML-сторінки.

XSS атаки можуть бути розділені на дві категорії: збережені (Stored XSS attacks) та відображені (Reflected XSS attacks).

1) Stored XSS атаки - це ті, де введений скрипт постійно зберігається на цільових серверах, наприклад, у базі даних, у форумі повідомлень, у журналі відвідувачів, у полі коментарів тощо. Потім жертва отримує з сервера шкідливий скрипт, кола вона надсилає запит. Такий тип атаки ще називають постійним.

2) Reflected XSS атаки - це ті, де введений скрипт відображається на веб-сторінці, наприклад, в повідомленні про помилку, в результатах пошуку або в будь-яких інших відповідях, які включають деякі або всі вхідні дані, надіслані на сервер як частина запиту. Так користувач може натиснути на зловмисне посилання, подаючи спеціально створену форму або навіть просто переглядаючи зловмисний сайт, введений код переходить до вразливого веб-сайту, який відображає атаку у веб-переглядачі користувача. Потім браузер виконує код, оскільки він надходить з "надійного" сервера. Такий тип атаки ще називають непостійним.

Існує третій, набагато менш відомий тип XSS атаки під назвою DOM Based XSS. Така атака відбувається в результаті модифікації середовища DOM в браузері жертви. Сама сторінка (тобто відповідь HTTP) не змінюється, але код клієнта, що міститься на сторінці, виконується інакше через шкідливі модифікації, які відбулися в середовищі DOM.
Цю атаку можна реалізувати, використовуючи url та input поля, якщо допустимо ввести не текст, який від нас очікують, а скрипт.

2.5.1 Реалізація XSS атаки

Тестувати сайт на цю вразливість можна спробувавши помістити в поле введення даних html тег текст. Якщо слово “текст” стане bold шрифтом, то це поле э вразливим до XSS. Також ми можемо помістити скрипт: <ScrIpt>alert(“test”)</sCriPt>, результат якого бачимо на Рис 2.23, але тут ми подивимось скільки контролю зловмиснику дає XSS-вразливий сайт.

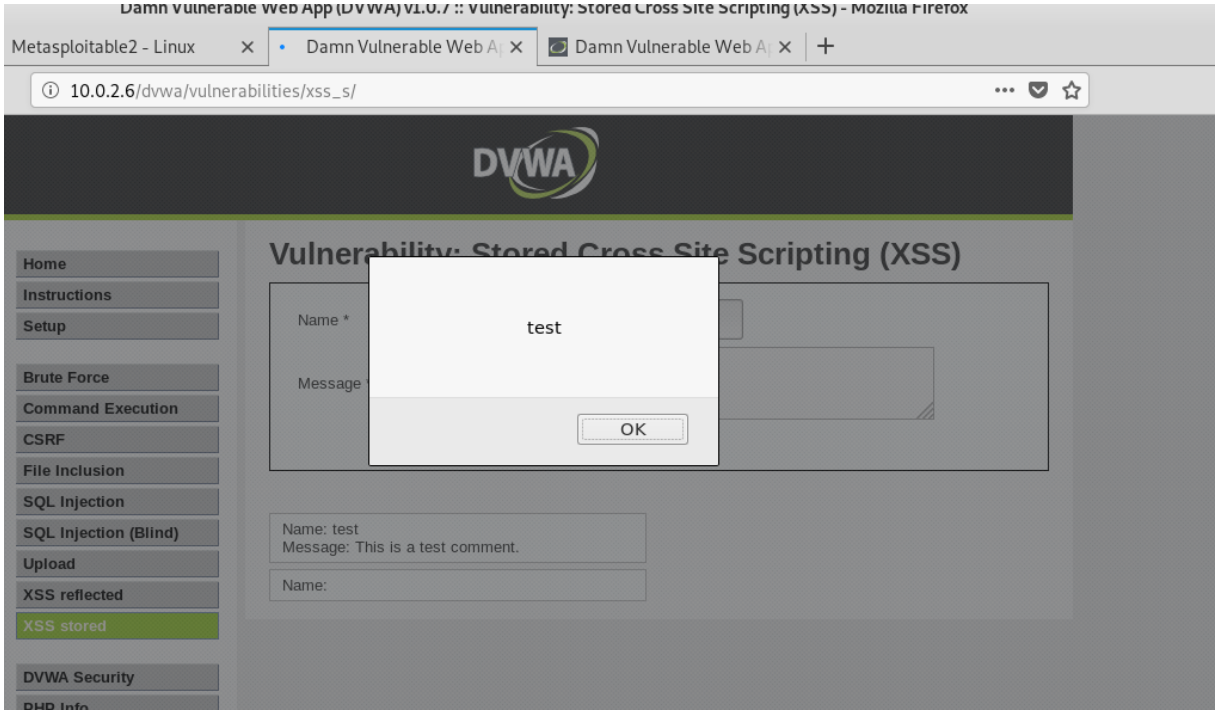

Рис 2 23 Приклад XSS

Ми будемо реалізовувати Reflected XSS атаку і використовувати BeEF (Browser Exploitation Framework). Це інструмент для тестування на проникнення, який фокусується на веб-браузерах. Для початку ми запускає BeEf в терміналі Kali Linux і отримуємо скрипт, який потрібно помістити на XSS-вразливий сайт (Рис 2.24).

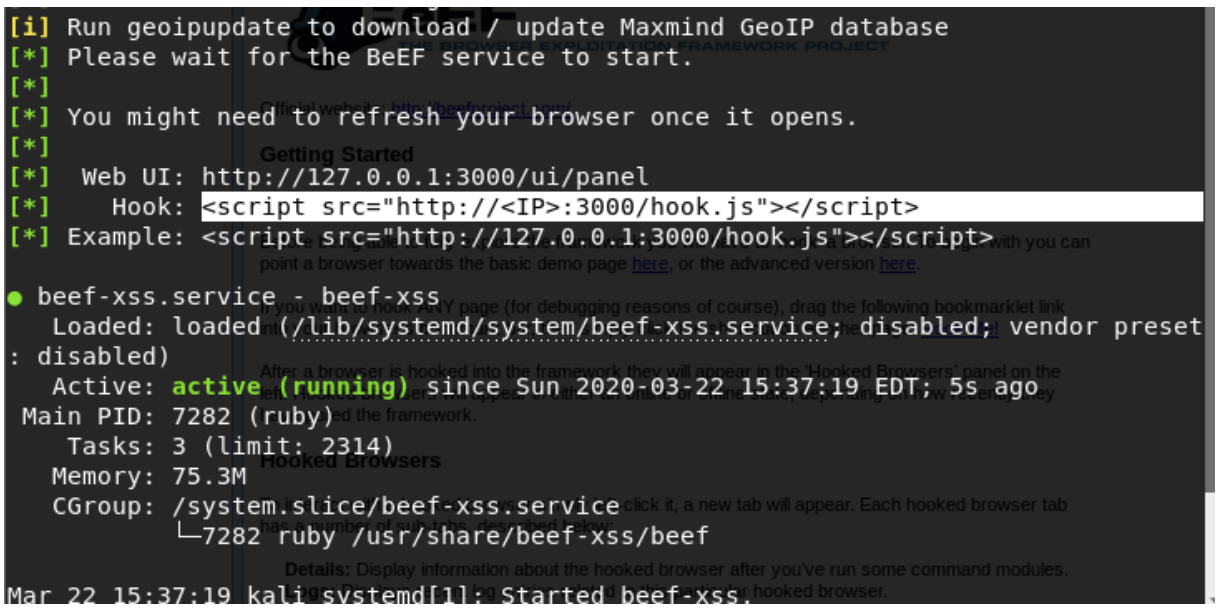

Рис 2 24 Запуск BeEF

Будемо робити атаку на цій сторінці (Рис 2.25), використовуючи url.

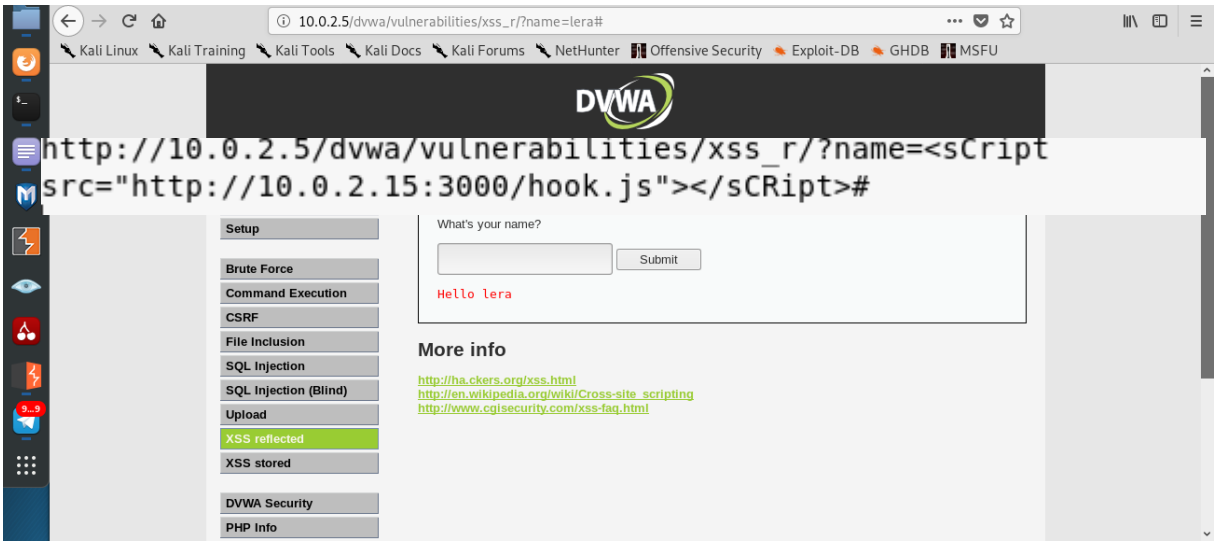

Рис 2 26 Сторінка для XSS атаки

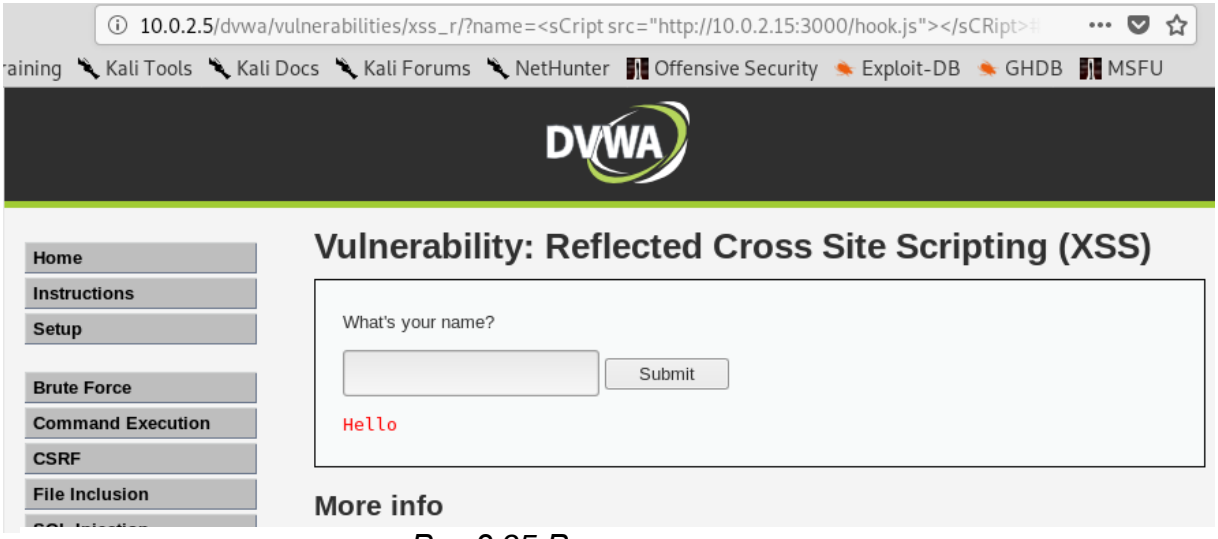

Рис 2 25 Результат запиту

Модифікуємо url вказавши нашу IP адресу в необхідне місце у скрипті.

З Рис 2.26 бачимо, що запит пройшов, повертаємось в BeEF і бачимо, що IP адреса нашого target стала онлайн клієнтом застосунку BeEF (Рис 2.27)

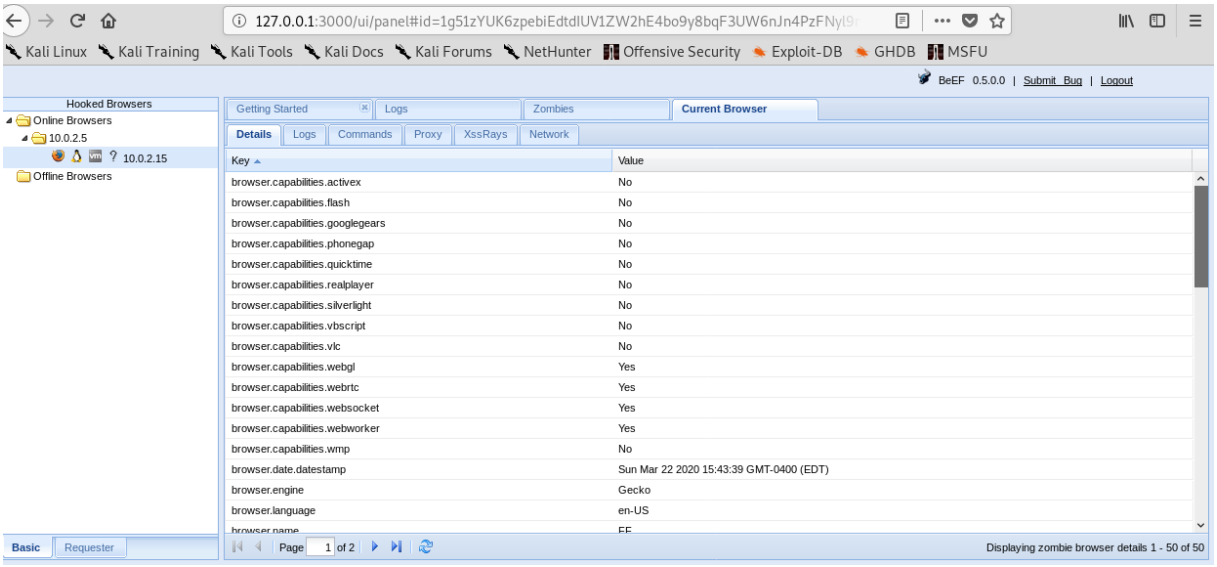

Рис 2 27 Інтерфейс BeEF

Тут ми в повній мірі можемо використовувати можливості Javascript. Через консоль BeEF можна робити скріншоти браузера, дивитись на встановлені плагіни жертви, надсилати повідмлення і багато іншого. Спробуємо надіслати модальне вікно із повідомленням на варазливий сайт (Рис 2.28)

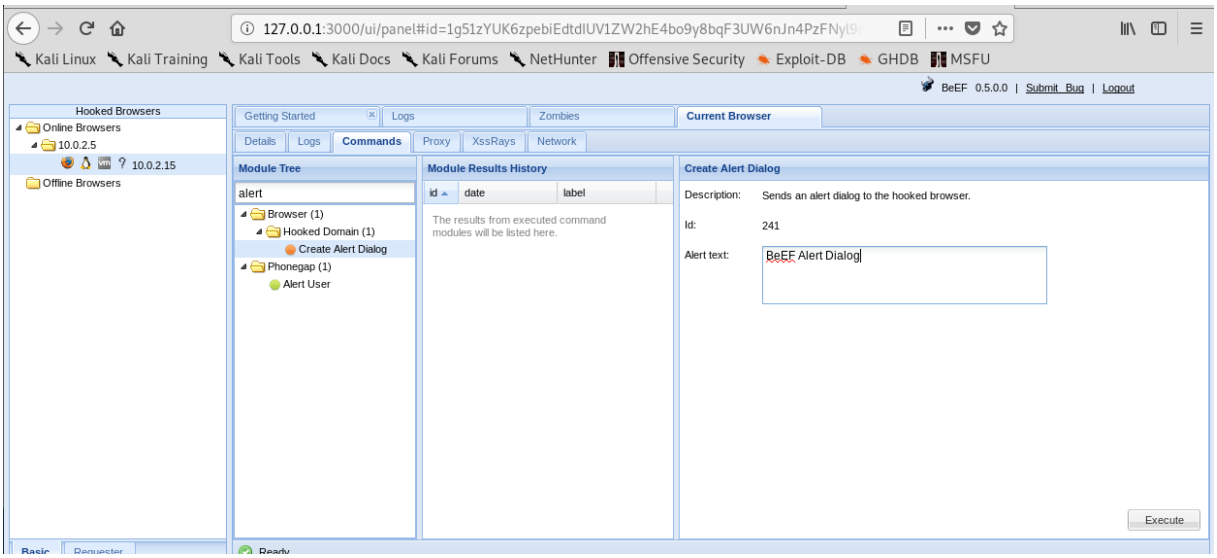

Рис 2 28 Надсилання модального вікна

Повідомлення наіслано в онлайн режимі (Рис 2.29).

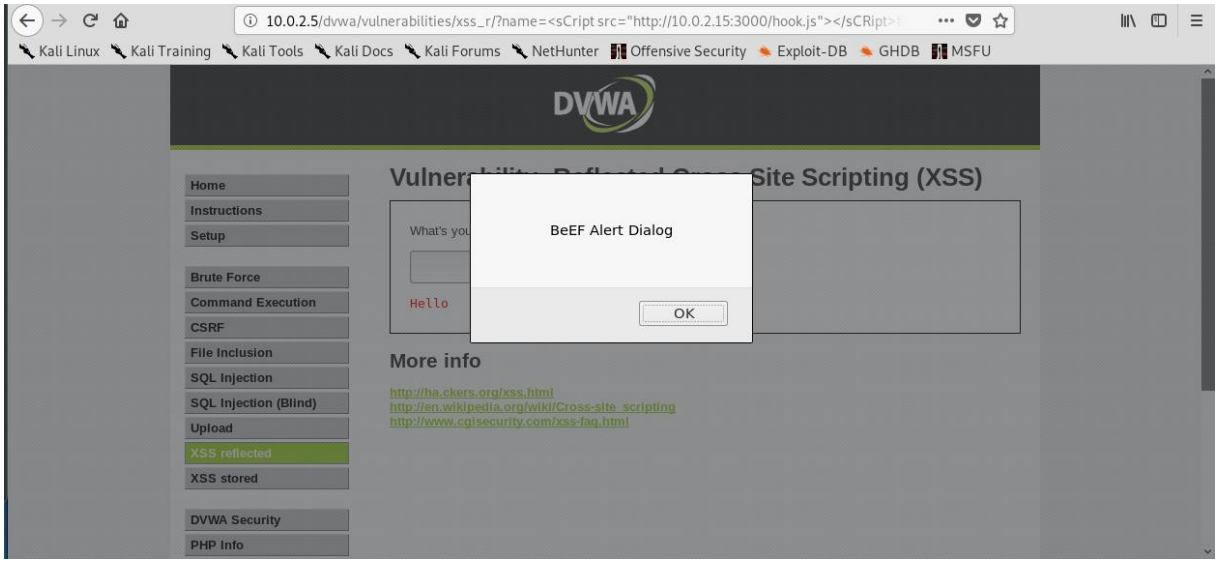

Рис 2 29 Результат надсилання

Сценаріїв взаємодії з користувачем може бути багато, але для цього сайт повинен бути вразливим на такого роду атаку.

2.5.2 Захист від XSS атаки

Якщо на сайті присутні поля для введення, то має виконуватися кодування. Якщо кодування неможливе або недоречне в деяких ситуаціях, то потрібно замінювати його або доповнювати валідацією. На стороні клієнта небхідно використовувати фільтри і враховувати різні комбінації символів. Безпечна обробка даних повинна виконуватися в коді не тільки на стороні клієнта, але і на стороні web-сервера. При використанні популярних CMS, наприклад Wordpress, Bitrix, Joomla, потрібно регулярно оновлювати версії движка і всіх встановлених модулів і плагінів. За замовчуванням більшість найпоширеніших систем для управління сайтів захищені від використання XSS, а ось сторонні плагіни з неперевірених джерел можуть містити уразливості.

2.6 Clickjacking

Clickjacking – механізм обману користувачів, при якому зловмисник може отримати доступ до конфіденційної інформації або навіть отримати доступ до комп'ютера користувача, заманивши його на зовні нешкідливу сторінку або запровадивши шкідливий код на безпечну сторінку. Багато сайтів були зламані подібнимчином, включаючи Twitter, Facebook, Paypal і інші. Всі вони, звичайно ж, зараз захищені.

Ідея цієї атаки дуже проста. Ось як вона відбувалась у Facebook:

1. Відвідувача заманюють на шкідливу сторінку (неважливо як).
2. На сторінці є посилання, яка виглядає безневинно (наприклад, «Розбагатій прямо зараз» або «Натисни тут, це дуже смішно»).
3. Поверх цього посиланнч шкідлива сторінка розміщує прозорий <iframe> з src з сайту facebook.com таким чином, що кнопка «like» знаходиться прямо над цим посиланням. Зазвичай це робиться за допомогою z-index в CSS.
4. При спробі кліка на це посилання відвідувач насправді натискає на кнопку.

2.6.1 Реалізація Clickjacking атаки

Для реалізації clickjacking атаки була написана програма clickjacking_tester.py, яка на вхід приймає txt-файл із списком веб-сайтів (Рис 2.30), які ми хочемо протестувати, а на виході ми бачимо які з сайти із

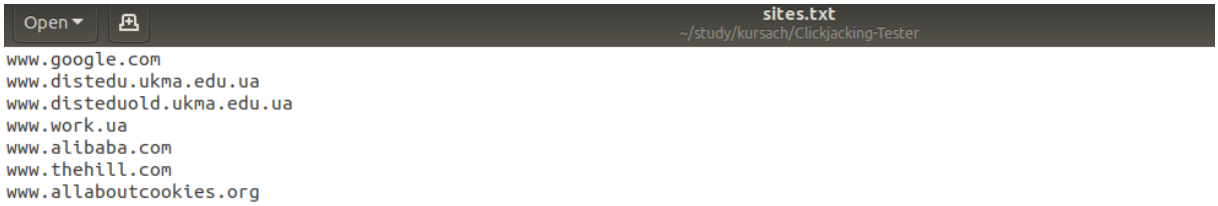

Рис 2 30 Вміст текстового файлу

цього списку є вразивими до clickjacking атаки (Рис 2.31)

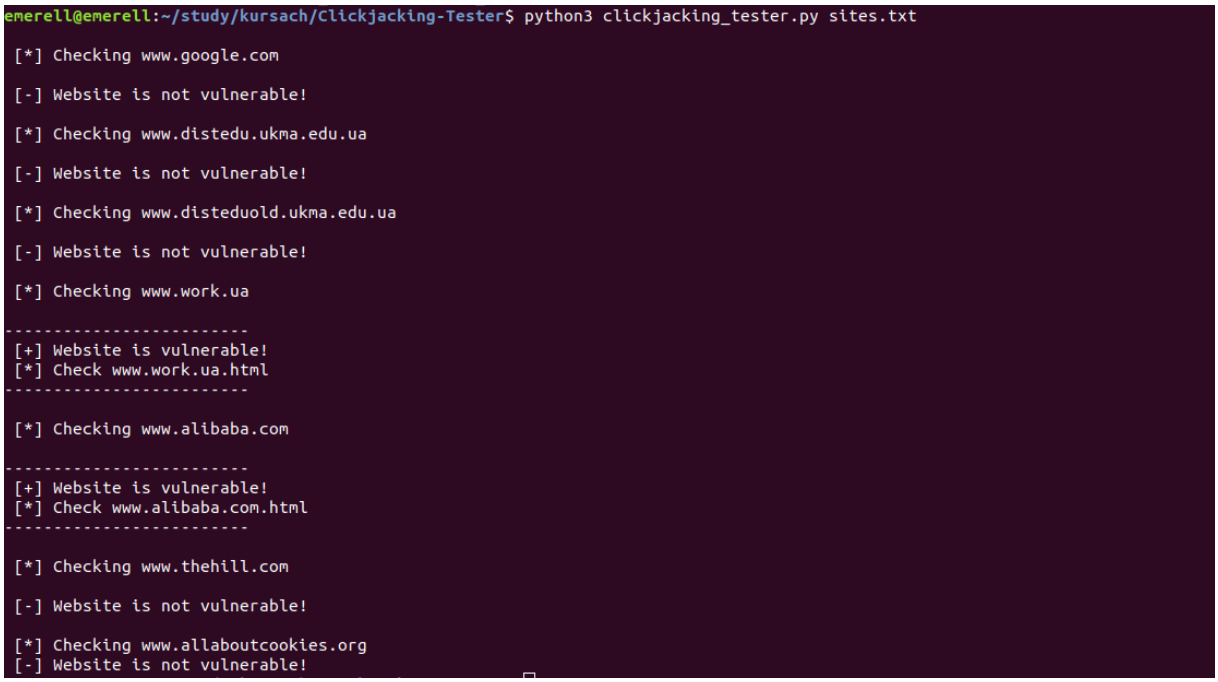

Рис 2 31 Результат виконання програми

Тут ми бачимо, що сайти work.ua та alibababa.com є вразливим до clickjacking-атаки. Отже, якщо сайт є вразливим, зокрема якщо він не містить заголовка X-Frame-Options, я створюю сторінку із <iframe>, де в “src” атрибуті знаходиться посилання на вразливий сайт. Таким чином, після деяких css маніпуляцій із строінкою з iframe можна реаілзувати clickjacking атаку (Рис 2.31)

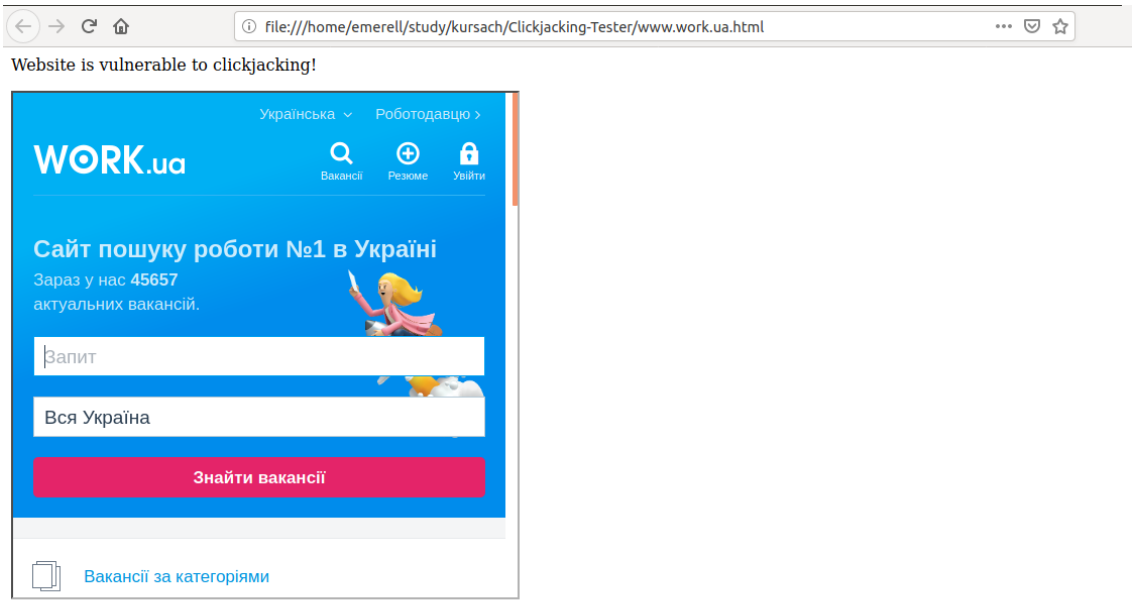

Рис 2 32 Clickjacking атака work.ua

2.6.2 Захист від Clickjacking атаки

Для захисту від цієї атаки рекомендується використовувати X-Frame-Options: SAMEORIGIN на сторінках, які не призначені для перегляду у фреймі.

Розділ 3. Поліпшення мережевої безпеки

3.1 Content Security Policy

Content Security Policy (CSP) - це додатковий рівень безпеки, що дозволяє розпізнавати і усувати певні типи атак, таких як Cross Site Scripting (XSS), Clickjacking і атаки впровадження даних. Спектр

застосування цих атак включає, але не обмежується крадіжкою даних, підміною сторінок і поширенням шкідливого ПО.

Основною метою створення CSP є усунення XSS атак і збір даних про спробі зробити це. CSP дає можливість адміністраторам серверів знизити або повністю усунути шляхи, за якими зловмисники можуть провести XSS, за допомогою визначення доменів, які браузер клієнта повинен вважати довіреними джерелами виконуваних скриптів. В такому випадку, браузер, сумісний з CSP, виконуватиме тільки ті скрипти, які були отримані зі списку дозволених джерел, і ігнорувати інші (в т.ч. вбудовуються скрипти і обробники подій, зазначені безпосередньо в HTML-атрибутах). Як крайній захід захисту, сайти, які хочуть заборонити виконання скриптів, можуть налаштувати цю поведінку глобально, за допомогою відповідної опції.

Для використання цієї політики сторінка повинна містити HTTP-заголовок Content-Security-Policy з одного і більше директивами, які представляють собою «білі списки». У версії 1.0 підтримуються наступні директиви: default-src, script-src, object-src, style-src, img-src, media-src, frame-src, font-src, connect-src. У другій версії цей список більший. В default-src перераховуються дозволені джерела за замовчуванням для інших директив. Якщо якась директива не зазначена в заголовку, то політика застосовується згідно зі списком default-src.

Для всіх директив діють наступні правила:

1. Для посилання на поточний домен використовується self.
2. У переліку URL адреси розділяються пробілами.
3. Якщо в рамках даної директиви нічого не повинно завантажуватися, то застосовується none. Наприклад, object-src 'none' забороняє завантаження будь-яких плагінів, включаючи Java і Flash.
4. Найпростіший приклад політики, що дозволяє завантаження ресурсів тільки зазначеного домену: Content-Security-Policy: default-src 'self';

5. Спроба завантаження ресурсів з інших доменів будуть припинятися браузером з видачею повідомлення в консолі (Рис 3.1)

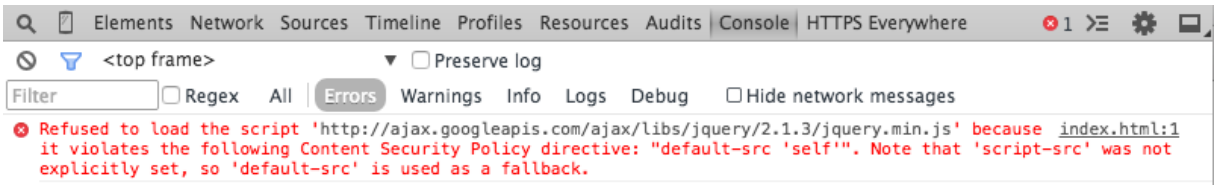

Рис 3 1 Помилка завантаження

Завантаження зовнішніх скриптів, які не включені в CSP, також буде припинено.

Давайте подивимося, як CSP впроваджений в Twitter (Рис 3.2):

```
default-src https:;
connect-src https:;
font-src https: data:;
frame-src https: twitter:;
frame-ancestors https:;
img-src https: data:;
media-src https:;
object-src https:;
script-src 'unsafe-inline' 'unsafe-eval' https:;
style-src 'unsafe-inline' https:;
report-uri https://twitter.com/i/csp_report?a=NVQWGYLXFVZXO2LGOQ%3D%3D%3D%3D%3D%3D%3D&ro=false;
```

Рис 3 2 CSP в Twitter

Тут всюди прописані https :, тобто примусово використовується SSL.

Також в CSP версії 2.0 з'явилася можливість дозволяти інлайн-скрипти і стилі за допомогою nonce-значень і хеш. Nonce - це згенерований випадковим чином на сервері змінна. Вона додається в CSP-заголовок (Рис):

```
Content-Security-Policy:    default-src 'self';
                            script-src 'self' 'nonce-Xiojd98a8jd3s9kFiDi29Uijwdu';
```

Рис 3 3 CSP Nonce

В такий же спосіб додається хеш. Перед рендерингом сторінки браузер обчислює хеш скрипта чи стилю, і якщо він співпадає, то виконання дозволяється.

Варто додати, що велика кількість різного виду атак відбувалась через брfузерні екстеншени, або плагіни. На сьогоднішній день, в Chrome Browser, при створенні Chrome extension не дозволяється використовувати inline скрипти. Але при цьому, якщо ми хочемо використати сторонню бібліотеку, наприклад React, для написання плагіну, достатньо зазначити згенерований хеш в Content-Security-Policy.

Висновок

Виконуючи дану роботу, було проведено тестування веб-застосунків на вразливість та проникнення. Було розгялнуто такі види мережевих атак, як File Upload, SQL Injection, CSRF, XSS, Clickjacking а також продемонстовано їх роботу та виявлено чому такі атаки є можливими. Було написано програму, яка виявляє чи є сайт вразливим до Clickjacking атаки.

Було знайдено найоптимальніший спосіб захисту від кожної з них. При проведенні тестувань використовувалась віртуальна ОС Kali Linux, яка виступила, як ОС зловимсника, та віртуальна ОС Metasploitable, яка була нашою “жертвою”.

Було розглянуті та використані на практиці застосунки Kali Linux для кращого тестування, зокрема Burp Suite Proxy, Weevely, SQLMap, Browser Exploitation Framework.

Було проведено збір інформації про веб-застосування та написано власну програму пошуку піддоменів будь-якого сайту. Було розглянуто переваги Content Security Policy та його спосіб використання.

Отже, актуальність тестування програмних продуктів на вразливість та проникнення залишається й досі, це допомагає нам, як розробникам, створювати надійний веб-застосунок та бути готовими до несподіваних атак.

Список використаних джерел

1. Information Gathering [Електронний ресурс] – Режим доступу до ресурсу: https://www.erdalozkaya.com/information-gathering/
2. File Upload Attack [Електронний ресурс] – Режим доступу до ресурсу: https://www.securitylab.ru/analytics/456285.php
3. Burp Suit [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/328382/
4. SQL Injection [Електронний ресурс] – Режим доступу до ресурсу: https://portswigger.net/web-security/sql-injection
5. Cross Site Request Forgery [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/csrf
6. XSS [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/66057/
7. Clickjacking [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/clickjacking
8. Content Security Policy [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/nix/blog/271575/