EXECUTIVE SUMMARY:
The target server given during the Capture Flag Game was extremely vulnerable to attack! There were a number of vulnerabilities considered high priority by the OWASP, and even several on the OWASP Top 10 list. These include, but are not limited to, falsifiable admin authentication techniques, allowing users to run their own code on the server pages, database attacks that reveal sensitive information, and giving users access to server files that should not be public. These weaknesses have led to huge problems in real world applications, including the 2016 election when hackers were able to access sensitive information in databases of over 200,000 Illinois voters!\(^1\) To create a more secure server we recommend that the owner of the server verifies all user input!

INTRODUCTION:
Our team had the opportunity of finding and exploiting vulnerabilities in the target server located at 35.196.64.43. In accordance with the cyber kill chain, we started by doing reconnaissance on the target. This was done through nmap and nikto, which collected information for us indicating the availability of various files and folders that should have been private. We then proceeded by trial and error to access these files and folders using a web browser, employing methods detailed below. Our Findings section details how we successfully gained access to sensitive information and its significance.

TOOLS AND METHODS:
The tools used to attack the server are common tools that are free and came pre-installed on our Kali Linux machines. It did not take many tools to find holes in the server through which we could access information in unintended ways

- Nmap: Using the command “nmap -p 35.196.64.43”, we discovered that there were only 2 ports open, 22 and 80. This meant that the server was running a webserver which we could access using a browser. Using the command “nmap -O 35.196.64.43”, the operating system was guessed to be running Windows XP, although this may be inaccurate (see Findings section).

\(^1\) https://www.wired.com/2016/08/hack-brief-fbi-warns-election-sites-got-hacked-eyes-russia/
- Developer Tools: While browsing, these inspection tools allowed us to view the source code of files being served to users on port 80.

- Nikto: Using the command “nikto -host=35.196.64.43”, we were able to see a number of vulnerabilities on the page, such as exploitable cookies, possible XSS, a php admin login page, a readme.html file, a wordpress login page, and a directory open for traversal at wp-content/uploads/. Also, it revealed that the server was an nginx server.

- SQLmap (and SQL Injection): sqlmap allowed us to access information in the databases on the server, including usernames, password hashes and other information. We used sqlmap in conjunction with the SQLi vulnerable url
“http://35.196.64.43/board.php?id=1”. Below an example attack to get the available databases is shown.

MacBook-Pro~ Emily2$ sqlmap -u "http://35.196.64.43/board.php?id=1" --dbs

[12:05:39] [INFO] resuming back-end DBMS 'mysql'
[12:05:39] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:

---
Parameter: id (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: id=1 AND 7660=7660

Type: AND/OR time-based blind
Title: MySQL >= 5.0.12 AND time-based blind
Payload: id=1 AND SLEEP(5)

Type: UNION query
Title: Generic UNION query (NULL) - 5 columns
Payload: id=1 UNION ALL SELECT NULL,NULL,NULL,NULL,CONCAT('0x71787a6278190x707
54456262705657774a74786a866246845d6616784b7657264744549666b51a4a7f706b75,0x
7187b627671')-- 1Fri

---
[12:06:39] [INFO] the back-end DBMS is MySQL
web application technology: Nginx
back-end DBMS: MySQL > 5.0.12

[12:05:39] [INFO] fetching database names
available databases [2]:
[+] board
[+] information_schema

[12:06:39] [INFO] fetched data logged to text files under '/Users/Emily2/.sqlmap/output/35.196.64.43'

[+] shutting down at 12:05:39

- BurpSuite: using burpsuite we were able to see cookie information, meaning we were able to change our credentials to grant admin privileges easily.
FINDINGS:

Server and Operating System
- Windows XP
- running nginx server (version 1.10.3)

Services Open:
- 22 (ssh)
- 80 (http/web server, running php and wordpress version 4.1)
Databases and Tables:

- Technology: running MySQL 5
- Two Databases, “board” and “information_schema”
- Board contained 17 tables
- Information_schema contained 78 tables
The most sensitive information was found in the users and wp_users tables of board. These contained personal information such as usernames and password hashes of accounts. Below is a sample of information found in the users table, which contained 1001 users information.
Vulnerabilities:

- **CWE-311**: Missing Encryption of Sensitive Data
- **CWE-319**: Cleartext Transmission of Sensitive Information
- **CWE-798**: Use of Hard-coded Credentials
  - These CWE records are all applicable to the page being served called main.php. Using developer tools to analyze the source code, it was easy to notice that a sensitive key had been left in plain sight, transmitted over the unencrypted port 80 (http).

- **CWE-89**: Improper Neutralization of Special Elements used in an SQL Command
- **CWE-312**: Cleartext Storage of Sensitive Information
- **CWE-319**: Cleartext Transmission of Sensitive Information
  - There were two web pages discovered vulnerable to SQL injection (CWE-89). One such webpage was admin.php, where an unexpected escaping of strings led to an error page instead of just allowing another login attempt.

  - The other page was board.php where the query string id could be used to access information. For instance, setting id equal to `1 or 1=1` without the tick marks seems to grant access to all board posts with all ids, even if those posts would normally be hidden. This allowed access to more information passed over http in cleartext (CWE-312, CWE-319). One might look at the image below and wonder where the sensitive information is, since the “keys” all look like garbage. However, the keys when run through a base64 decoder sometimes provide sensible information. According to CWE-312, “Cleartext is any information that
is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding)."

- **CWE-548: Information Exposure Through Directory Listing**
  - A directory listing is given if one goes to the page http://35.196.64.43/wp-content/uploads/. This is a problem because a user should not be allowed to traverse a directory system, as they might find sensitive information. In this case, http://35.196.64.43/wp-content/uploads/2015/10/Elif Olmez's COMP 40 Assignment.txt was an executable file that contained a hard coded plaintext string that appeared to contain sensitive information shown
CWE-521: Weak Password Requirements
- Even after a competing team had gained access to the bobo WordPress account and changed the password, our team was able to take control of the bobo account by cracking the password hash we retrieved from the wp_users table in the board database. The competing team had changed bobo’s password to “mingisawesome”, which was not difficult to crack. Although we found no sensitive information on bobo’s account, we still had the ability to change bobo’s files and account settings, so we changed the password to i321lovE432cats543!

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- There is a board page located at 35.196.64.43/board.php which allows users to write blog posts that are public to other users. This page is vulnerable to Cross Site Scripting, as can be seen below. Many alerts from malicious users pop up when accessing the site! Also, javascript runs on the page that changes the main picture of the board page, so users are also able to deface the webpage and possibly remove important information from the board itself.
CWE-598: Information Exposure Through Query Strings in GET Request

CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')

- At another webpage, which 35.196.64.43 contains an HTML link to, there is an interesting query parameter that can be manipulated in malicious ways. By setting the query string “id” to a system command, we were able to execute commands to manipulate the file system of the server. We were even able to find some sensitive information by searching the system’s directory.

```
key{FB9D8263F48A82BEEC4068B2BB6114A2D2A241BC14A14AEEF56F6C6B80B76FA4}
```

**Evil Homer**

Set the ID parameter in query string to homer...

**id parameter is**

**QUESTIONS:**
After completing the CTF game, we had a few questions remaining. Although we looked up many vulnerabilities relating to the software being used by the system, in the end all of the
exploits we performed were pretty common vulnerabilities that do not require a lot of specialty technologies. What we were wondering is whether or not more advanced exploits that can be found in exploit databases such as Exploit-DB are more difficult to use, more difficult to know when to use, or are just scripts that script kiddies can use.

POLICY IMPLICATIONS:
Despite how well-known an exploit SQL injection is, the target site was still susceptible. This vulnerability allows for serious data breaches. Our team was able to retrieve all users’ usernames, first and last names, and password hashes from the board database alone. A vulnerability of this level should not be ignored. According to scmagazine, SQL injection is likely the exploit behind the Equifax data breach\(^2\) which exposed consumer names, Social Security numbers, birth dates, and addresses. From the disclosure of the breach in September, to November 2017, Equifax shares dropped about 25%.

Another big exploit found was XSS, which is debatably even more well known than SQL Injection. XSS is also a vulnerability that continues to plague users on commonly used websites. For instance, everyone has been on eBay, the online commerce corporation, to buy pokemon cards or to buy a ghost in a jar, and yet even as recently as 2017 malicious users have been finding ways to steal other user’s credentials through these attacks.\(^3\) This is problematic especially on an eCommerce website, because then users can redirect goods to their own houses and use other’s accounts to buy the goods, potentially running up quite a high bill!

CONCLUSION:
This game was beneficial due to the hands on learning it provided. It is one thing to learn in a class how to utilize scanning programs and methods of attack. It is a completely different thing to be able to apply our skills in a real-ish world scenario. The tools that we learned the most from were sqlmap, burp suite, and nikto. These were programs we had not gone over that extensively until this assignment. It was also an exercise in teaching ourselves how to interpret program documentation.

Our largest take away from the project was to **verify all user input**. No matter how much any input is saved or used on the server side, it must be validated and sterilized, or it will become a route into your program for a black hat to exploit. This lack of sanitation is what enables the SQL injection, command injection and cross site scripting that we heavily used.

Q: If you had to do this challenge again, what would you differently?
A: We would try to do the more difficult vulnerabilities first so that we could find sensitive information and gain control of the system before other competing attackers could. The more easily attainable flags would likely still be around to get later, after all.