Assignment 4:

Secure Systems Engineering (CS 6510)

Submitted By:

Dipanshu Kumar (CS24M019)

1. Verification of Flash Read Integrity *(5 Marks)*

- I extracted and read the contents of the flash into a binary blob "flash_dumo.bin" using **'sudo flashrom -p ch341a_spi -r flash_dump.bin'** and verified its integrity by computing its SHA-256 checksum using **'sha256sum flash_dump.bin'**.
- The computed checksum is:

```
sse@sse_vm:~/Desktop/workspace/A4$ sudo flashrom -p ch341a_spi -r flash_dump.bin
flashrom v0.9.9-rc1-r1942 on Linux 4.15.0-45-generic (x86_64)
flashrom is free software, get the source code at https://flashrom.org

Calibrating delay loop... OK.
Found Winbond flash chip "W25Q32.V" (4096 kB, SPI) on ch341a_spi.
Reading flash... done.
sse@sse_vm:~/Desktop/workspace/A4$ sha256sum flash_dump.bin
7d530283029f273601b89e4fb2b92f3c078d90a2d3c127f10b8a6e480a74a310  flash_dump.bin
sse@sse_vm:~/Desktop/workspace/A4$
```

7d530283029f273601b89e4fb2b92f3c078d90a2d3c127f10b8a6e480a74a310

Since this matches the expected checksum, I can confirm that the flash was read correctly without corruption.

2. Extracting Files from the Binary Blob

- Upon running binwalk on the flash_dump.bin using **‘binwalk -eM flash_dump.bin’**, I identified two embedded files.

```
sse@sse_vm:~/Desktop/workspace/A4$ binwalk -eM flash_dump.bin

Scan Time:     2025-03-22 09:51:44
Target File:   /home/sse/Desktop/workspace/A4/flash_dump.bin
MD5 Checksum:  81502e93d867257093ac00af0fbf9224
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             POSIX tar archive (GNU)


Scan Time:     2025-03-22 09:51:45
Target File:   /home/sse/Desktop/workspace/A4/_flash_dump.bin.extracted/enc.squashfs
MD5 Checksum:  8897654bc845f1ed5bb8b3b61a5b3df5
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------


Scan Time:     2025-03-22 09:51:46
Target File:   /home/sse/Desktop/workspace/A4/_flash_dump.bin.extracted/init
MD5 Checksum:  4853d2bca7c6f12952ed1c774cec728f
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ELF, 32-bit MSB MIPS64 executable, MIPS, version 1 (SYSV)
462624        0x70F20         Unix path: /proc/sys/vm/overcommit_memory
465884        0x71BDC         Unix path: /proc/sys/kernel/rtsig-max
469392        0x72990         Unix path: /proc/sys/kernel/osrelease
474708        0x73E54         Unix path: /usr/lib/locale/locale-archive
543500        0x84B0C         ELF, 32-bit MSB processor-specific, (GNU/Linux)
543652        0x84BA4         ELF, 32-bit MSB no file type, PowerPC or cisco 4500, (SYSV)
544802        0x85022         Unix path: /sysdeps/unix/sysv/linux/dl-origin.c
```

---

3. SHA-256 Checksum of the Correctly Processed Firmware *(10 Marks)*

Upon inspecting I came to know that the **enc.squashfs** is encrypted and I have to find a key to decrypt it.

- **Finding Key:**
  I analysed the **init** file in ghidra to find out how it is working. I came to know that a function is called after **XORing** a value with decimal **55**.

Decompile: main - (init)

```c

undefined4 main(int param_1,int param_2)

{
  if (param_1 != 3) {
                    /* WARNING: Subroutine does not return */
    exit(1);
  }
  proc_data(*(undefined4 *)(param_2 + 4),*(undefined4 *)(param_2 + 8));
  return 0;
}

```

Decompile: proc_data - (init)

```c

void proc_data(undefined4 param_1,undefined4 param_2)

{
  int iVar1;
  int iVar2;
  uint uVar3;

  iVar1 = fopen(param_1,&DAT_0046f8e0);
  iVar2 = fopen(param_2,&DAT_0046f8e4);
  if ((iVar1 != 0) && (iVar2 != 0)) {
    while( true ) {
      uVar3 = fgetc(iVar1);
      if (uVar3 == 0xffffffff) break;
      fputc(uVar3 ^ 0x55,iVar2);
    }
    fclose(iVar1);
    fclose(iVar2);
    return;
  }
  perror(&DAT_0046f8e8);
                    /* WARNING: Subroutine does not return */
  exit(1);
}

```

- **Decryption Process:**

  The **enc.squashfs** file was decrypted using a custom Python script, which processed each byte by applying an XOR operation with 0x55. This transformation produced a new file named **firmware_decrypted.bin**. The script used for decryption is as follows:

```python
# Read the encrypted firmware file
def decrypt_firmware(input_file, output_file, key=0x55):
    with open(input_file, "rb") as enc_file:
        content = enc_file.read()

    # XOR decryption
    decrypted_content = bytes(byte ^ key for byte in content)

    # Save the decrypted firmware
    with open(output_file, "wb") as dec_file:
        dec_file.write(decrypted_content)

    print("Decryption successful. Processed firmware saved.")

# File paths
input_firmware = "enc.squashfs"
output_firmware = "firmware_decrypted.bin"

# Perform decryption
decrypt_firmware(input_firmware, output_firmware)

```

- **Verification:**

The SHA-256 hash of the decrypted firmware (firmware_decrypted.bin) was computed to confirm the successful decryption. The resulting checksum was:

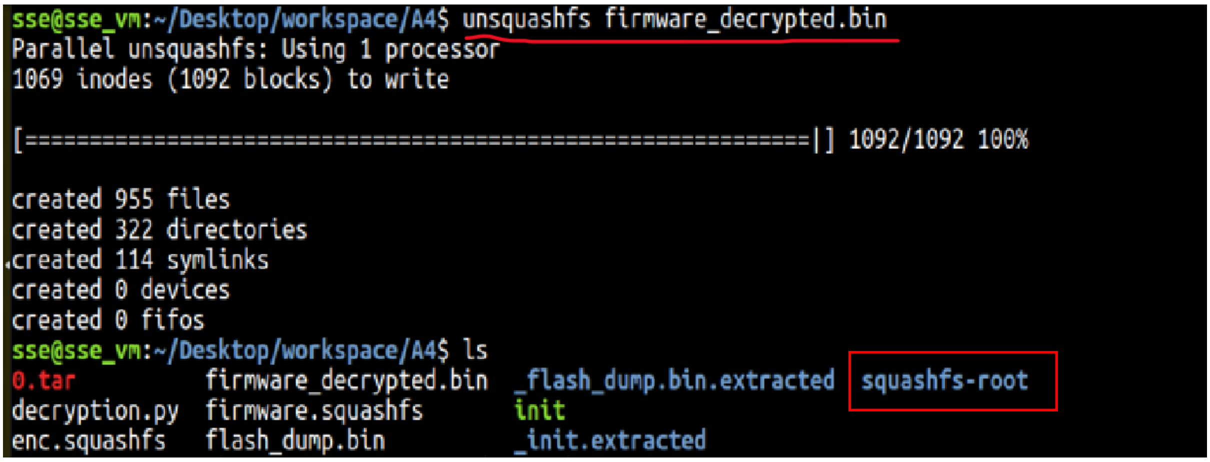
dc8d9c00a19af54e349bbd569e92801aee3713feb7298ed9d81c8da7a7478a96

```
sse@sse_vm:~/Desktop/workspace/A4$ python3 decryption.py
Decryption successful. Processed firmware saved.
sse@sse_vm:~/Desktop/workspace/A4$ sha256sum firmware_decrypted.bin
dc8d9c00a19af54e349bbd569e92801aee3713feb7298ed9d81c8da7a7478a96  firmware_decrypted.bin
sse@sse_vm:~/Desktop/workspace/A4$
```

- **Extraction:**

Once decrypted, the firmware was extracted using the unsquashfs command:

unsquashfs firmware_decrypted.bin



This operation generated a directory named **squashfs-root**, which contains the router's full filesystem, enabling further analysis.

---

4. Identifying CPU Architecture and Endianness *(5 Marks)*

I previously used **binwalk -eM flash_dump.bin**, upon analysing the firmware:

- Running file on the extracted firmware binary revealed the **CPU architecture** and **Endianness**

```
sse@sse_vm:~/Desktop/workspace/A4$ binwalk -eM flash_dump.bin

Scan Time:     2025-03-22 09:51:44
Target File:   /home/sse/Desktop/workspace/A4/flash_dump.bin
MD5 Checksum:  81502e93d867257093ac00af0fbf9224
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             POSIX tar archive (GNU)


Scan Time:     2025-03-22 09:51:45
Target File:   /home/sse/Desktop/workspace/A4/_flash_dump.bin.extracted/enc.squashfs
MD5 Checksum:  8897654bc845f1ed5bb8b3b61a5b3df5
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------


Scan Time:     2025-03-22 09:51:46
Target File:   /home/sse/Desktop/workspace/A4/_flash_dump.bin.extracted/init
MD5 Checksum:  4853d2bca7c6f12952ed1c774cec728f
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ELF, 32-bit MSB MIPS64 executable, MIPS, version 1 (SYSV)
462624        0x70F20         Unix path: /proc/sys/vm/overcommit_memory
465884        0x71BDC         Unix path: /proc/sys/kernel/rtsig-max
469392        0x72990         Unix path: /proc/sys/kernel/osrelease
474708        0x73E54         Unix path: /usr/lib/locale/locale-archive
543500        0x84B0C         ELF, 32-bit MSB processor-specific, (GNU/Linux)
543652        0x84BA4         ELF, 32-bit MSB no file type, PowerPC or cisco 4500, (SYSV)
544802        0x85022         Unix path: /sysdeps/unix/sysv/linux/dl-origin.c
```

- I also used command **'file init'** for confirmation.

```
sse@sse_vm:~/Desktop/workspace/A4$ file init
init: ELF 32-bit MSB executable, MIPS, MIPS32 rel2 version 1 (SYSV), statically linked, BuildID[sha1]=e2291d3dac3
972e592b718d0fa88e432e8f82bd8, for GNU/Linux 3.2.0, not stripped
sse@sse_vm:~/Desktop/workspace/A4$
```

The output indicates that the executable is for a **MIPS64 CPU but in 32-bit mode**. The presence of **MSB (Most Significant Byte first) means it follows big-endian** byte order. This suggests that the firmware runs on a 32-bit MIPS64 architecture with big-endian encoding.

Findings:

- **CPU Architecture:** 32-bit MIPS64 architecture
- **Endianness:** Big-endian encoding

5. Root Password Change Investigation *(10 Marks)*

During our testing phase, the router warned that "the root password was changed." To find the new password:

- The **/etc/shadow** file stores **hashed** user passwords in Linux-based systems, providing better security than **/etc/passwd**, which only contains user account information.

shadow | passwd

```
root:$6$miekpK4m$wLI8N9ROKZRirodBAOJLzy2zWib0k7EIZE2caOrgOu3XdVB76jt84x.3VPSfzTtDYEQUli1Jueiw1QBszWEff.:0:0:0:0:::
daemon:*:0:0:0:0:::
www-data:!:0:0:0:0:::
mysql:!!:0:0:0:0:::
```

passwd | shadow

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
mysql:x:112:117:MySQL Server:/nonexistent:/bin/false
```

- I analysed the extracted firmware's **/etc/shadow and /etc/passwd** file. And combined the both file in a text file **john_input.txt**

```
sse@sse_vm:~/Desktop/workspace/A4$ sudo cp squashfs-root/etc/shadow .
sse@sse_vm:~/Desktop/workspace/A4$ sudo cp squashfs-root/etc/passwd .
sse@sse_vm:~/Desktop/workspace/A4$ unshadow passwd shadow > john_input.txt
```

- Since passwords are stored in a hashed format, they need to be cracked using tools like **John the Ripper**. I downloaded the common password from the internet and made a text file **password.txt**

- Used John the Ripper with a wordlist attack to crack the hashed password.

```
sse@sse_vm:~/Desktop/workspace/A4$ john --wordlist=password.txt john_input.txt
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Press 'q' or Ctrl-C to abort, almost any other key for status
hacker           (root)
1g 0:00:00:07 100% 0.1424g/s 492.3p/s 492.3c/s 492.3C/s holly1..wolf359
Use the "--show" option to display all of the cracked passwords reliably
Session completed
sse@sse_vm:~/Desktop/workspace/A4$ john --show john_input.txt
root:hacker:0:0:root:/root:/bin/bash

1 password hash cracked, 0 left
sse@sse_vm:~/Desktop/workspace/A4$
```

Recovered Root Password:

hacker

This confirms that the root password was modified, and I successfully identified its new value.

6. Malicious Activity Report ***(5 Marks)***

IP Address	Malicious Activity	Timestamp
192.168.1.103	**Unauthorized access attempt to /etc/passwd** – This device tried to access a sensitive system file without the necessary permissions, which could indicate an attempt to gather critical system information for further exploitation.	06:00:30, 06:01:00
192.168.1.104	**Sensitive file /etc/shadow downloaded via SCP** – This is a severe breach since /etc/shadow contains hashed passwords of system users. If cracked, it could allow unauthorized access.	06:00:45
172.16.0.10	**Brute-force login attempts** – This external IP attempted multiple failed logins, likely using automated credential-guessing techniques to gain unauthorized access to the system.	06:02:45
192.168.1.113	**Suspicious command execution (Remote Shell Attempt)** – This device accessed a router URL containing a command to start a Telnet server with an unrestricted shell, indicating an attempt to gain full remote control of the system.	06:03:15
192.168.1.50	**Port scanning followed by a Telnet connection attempt** – This device scanned multiple ports, likely probing for vulnerabilities, before specifically attempting to connect to port 23 (commonly used for Telnet), which is often targeted due to weak security.	06:05:15, 06:05:19

---

7. Remote Shell Execution Attempt ***(5 Marks)***

- **IP Address: 192.168.1.113**
- **Program Executed:**
- telnetd -l /bin/sh -p 2333 -b 0.0.0.0
- **Port Number: 2333**
- **Threat Description:** This command attempts to launch a Telnet daemon (telnetd) with an unrestricted shell (/bin/sh), making it possible for an attacker to remotely execute commands on the system. By binding it to **port 2333**, they are likely trying to avoid detection, as standard Telnet runs on port **23**. If successful, the attacker would gain full control over the compromised machine.

8. Finding the Flag *(10 Marks)*

- To trace the execution of the malicious command, I first analyzed the **log file** located at /squashfs_root/dev/log. I observed the following entry:

2025-02-28 06:03:15 [**INFO**] [192.168.1.113] User 'guest' accessed 'www.router.local/showdevices=?new&telnetd -l /bin/sh -p 2333 -b 0.0.0.0'

  This indicated that an attacker injected a command via the **web interface** to start a **Telnet daemon** (telnetd) with a shell (/bin/sh) on **port 2333**, exposing the system to remote access. Shortly after, another log entry showed:

2025-02-28 06:05:15 [**WARNING**] [192.168.1.121] Port scanning detected from IP 192.168.1.50

  This suggested that the attacker or another entity scanned the network, likely discovering the **open Telnet port**.

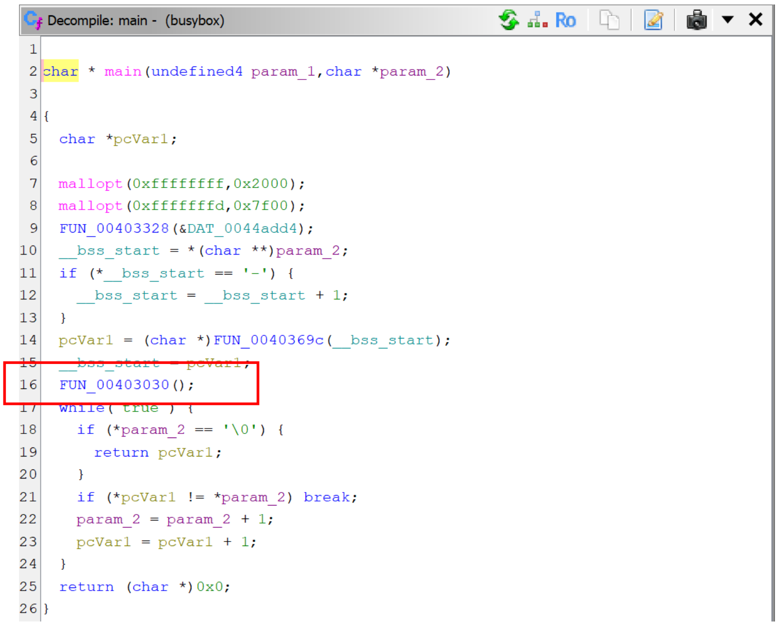
- Since many embedded systems use **BusyBox** for core utilities, I suspected that the attacker's command execution leveraged **BusyBox's built-in telnetd**. **BusyBox** is commonly found in router firmware, and it provides a minimalistic UNIX environment, making it a prime target for privilege escalation or remote access attacks.

To confirm this, I:

1. **Extracted the BusyBox binary** from the firmware from path:

   /squashfs_root/bin/busybox

2. **Decompiled the binary in Ghidra**, where I identified a function named FUN_00403030 that was responsible for handling suspicious command execution.



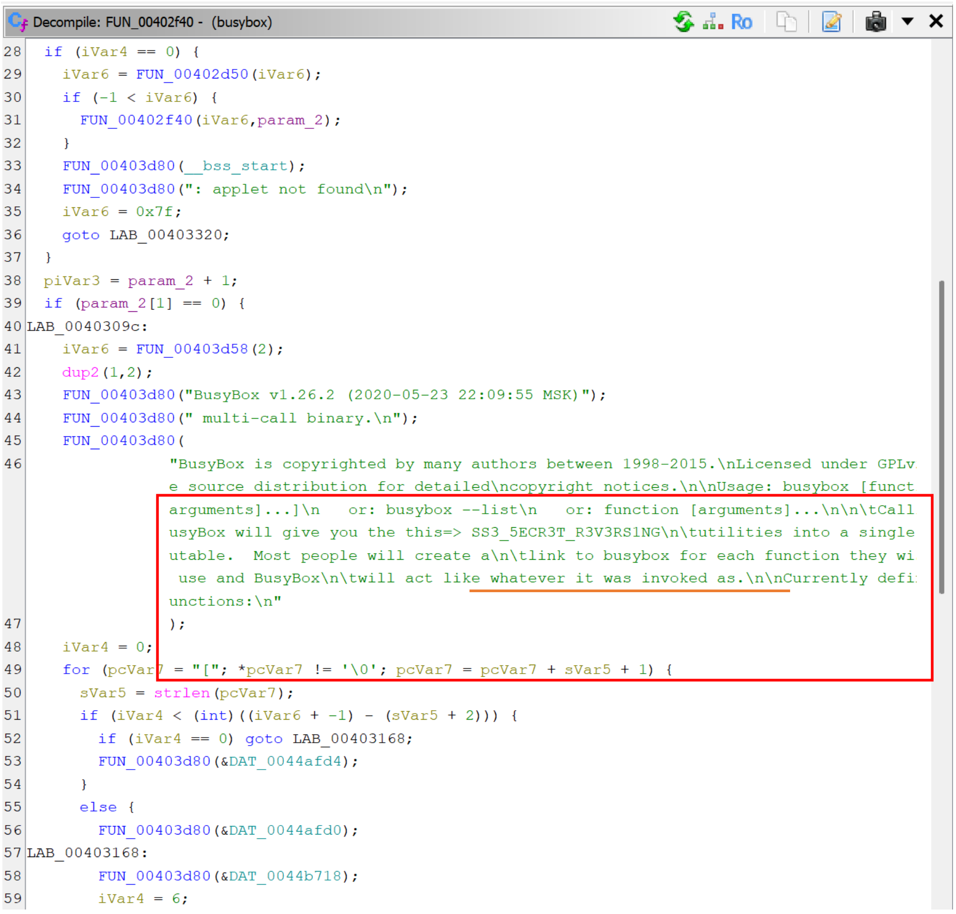

```c

char * main(undefined4 param_1,char *param_2)

{
  char *pcVar1;

  mallopt(0xffffffff,0x2000);
  mallopt(0xfffffffd,0x7f00);
  FUN_00403328(&DAT_0044add4);
  __bss_start = *(char **)param_2;
  if (*__bss_start == '-') {
    __bss_start = __bss_start + 1;
  }
  pcVar1 = (char *)FUN_0040369c(__bss_start);
  __bss_start = pcVar1;
  FUN_00403030();
  while( true ) {
    if (*param_2 == '\0') {
      return pcVar1;
    }
    if (*pcVar1 != *param_2) break;
    param_2 = param_2 + 1;
    pcVar1 = pcVar1 + 1;
  }
  return (char *)0x0;
}
```

3. **Inside FUN_00403030, I found the flag**:



9. Conclusion

This report provides a **comprehensive security analysis** of the router firmware, covering **flash integrity verification, firmware extraction, decryption, password recovery, and threat investigation**.

1. **Flash Integrity & Extraction:** The firmware was successfully extracted and verified using SHA-256 checksums. Encrypted files were identified and decrypted using an XOR-based approach.

2. **CPU & Endianness Identification:** The system was confirmed to run a **32-bit MIPS64 architecture with big-endian encoding**.

3. **Root Password Recovery:** The hashed password from /etc/shadow was cracked using John the Ripper, revealing the **root password: "hacker"**.

4. **Threat Analysis:** Logs revealed **unauthorized access, sensitive file downloads, brute-force attempts, and remote shell execution** via **BusyBox's telnetd**.

5. **Flag Discovery:** By decompiling the **BusyBox binary in Ghidra**, the attack execution was traced to **FUN_00403030**, where the flag was found.

This analysis underscores the **importance of secure authentication, monitoring suspicious activities, and restricting unnecessary network services** to mitigate vulnerabilities and enhance system security.

10. Reference

- Shared the Hardware with Team Decompiler (CS24M017)
- Prof. Chester Sir's YouTube lectures
- ChatGPT.com
- Perplexity.ai

11. Contribution

I independently conducted this assignment and compiled the entire report on my own.