VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Team Members

Rachel Kelly

Team member 2

Team member 3

Executive Summary

The project's aim was focused on identifying, exploiting and mitigating vulnerabilities found in our various codebase. The aim of this project was to improve our understanding and application of the principles that we have learned through our vulnerability research course. Our team conducted manual code reviews, static analysis and also dynamic analysis. During our analysis we uncovered critical vulnerabilities such as the use of the gets() function, weak encryption. We also created exploits such as password cracking, buffer overflows and implemented tools such as, GDB, Radare2 and Java decompliers which were essential to show our exploitation of the vulnerabilities found within our project. Once our vulnerabilities were identified and exploited, we then moved on to implementing mitigation techniques to address the vulnerabilities. We carried out our mitigation by replacing unsafe functions, application of OpenSSL, and securing key storage mechanisms. Such actions that we have identified and mitigated have ensured that the codebases have an enhanced robustness to adhere to the industry best practice as declared by OWASP and Cert C guidelines.

Manual Code Review

Methodology

For our Manual code testing we implemented multiple techniques in our methodology which will be broken down into tools that we used and the techniques we implemented;
Tools:

- Step 1 We manually reviewed our code and broke it down into sections so that we could identify the insecure functions such as gets() function, weak encryption and poor validation.
- Step 2 we then created code to exploit the vulnerabilities we identified. We created a buffer overflow and password cracking. We also used debugging tools to help us when we tested the buffer overflow.
- Step 3 we then implemented mitigation techniques such as using secure functions fgets() to replace the gets(). We used Open SSL which has secured our password encryption.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Vulnerability Identification

Within the manual code testing stage, we identified there are several vulnerabilities which we will expand on;

- Within this code the function gets() is used to read user input. This is a vulnerability as this function does not perform any bound checks so can lead to buffer overflows.

Screenshot of gets() function

```
    printf("Enter your name: ");
    gets(account->name);

    printf("Enter your password: ");
    gets(raw_password);
```

- A second vulnerability is that the user has hardcoded credentials such as admin123 and the password. This is a vulnerability as this compromises the security of the code. If an attacker finds this coding, they can exploit it.

Screenshot of hardcoded accounts

```
Account hardcoded_accounts[2] = {
    {"admin", "admin123", 10000.0},
    {"user", "password", 500.0}
};
```

- The encrypt() function uses a Caesar cipher with a fixed key. This is a vulnerability as this encryption method is weak and can easily be broken and the contents exploited.

Screenshot of encrypt() function

```
    encrypt(raw_password, encrypted_password);
    strncpy(account->password, encrypted_password, 15);
    account->password[15] = '\0';
    account->balance = 0.0;

    printf("Account created successfully!\n");
}
```

- The create_account() function has no validation for the user input. This is a vulnerability as it can allow an injection of malicious inputs

Screenshot of create_account() function

```
            create_account(&user_account);
            break;
```

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Exploit Development

We have implemented two exploitation developments to demonstrate the vulnerabilities in our code.

Buffer overflow

Screenshot of buffer overflow

```
//buffer overflow in field 'name'
char long_input[64] = "ABCDEFGHIJKLMNOPQRSTUVWXYZZYXWVUTSRQPONMLKJIHGFEDCBAABCDEFGHIJK";
```

Cracking Hardcoded Password

Screenshot on cracking encrypt function

```
//Coding to crack encrypt()
def decrypt_caesar(ciphertext, key=3):
    plaintext = ''
    for char in ciphertext:
        if char.isalpha():
            base = 'a' if char.islower() else 'A'
            plaintext += chr((ord(char) - ord(base) - key) % 26 + ord(base))
        else:
            plaintext += char
    return plaintext

print(decrypt_caesar("dplq"), "=> 'admin'")
```

Mitigation Implementation

We will now outline how to fix our vulnerabilities mentioned in our vulnerabilities identified section.

- Firstly we need to use fgets() in place of gets(). Replacing the gets() function avoids the buffer overflow (MSC34-C) which is recommended by CERT C. MISRA C also avoids the use of the gets() function.

Screenshot to show fgets()

```
//changing gets() into fgets()
printf("Enter your name: ");
fgets(account->name, sizeof(account->name), stdin);

printf("Enter your password: ");
fgets(raw_password, sizeof(raw_password), stdin);
```

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

- We would use a config_file which will be a separate file that contains our hardcoded credentials. Not only will this be a separate file but it will also be encrypted therefore, tackling this vulnerability issue.
- We would use Open SSL for hashing our passwords and keeping them encrypted. OWASP standard compliance would implement strong cryptography for password management.

Screenshot to show password encryption and use of Open SSL

```c
//encrypt password
#include <openssl/evp.h> //links to openssl library
#include <openssl/sha.h>

void hash_password(const char *input, char *output) {
    unsigned char hash[SHA256_DIGEST_LENGTH]; //hashing
    SHA256((unsigned char *)input, strlen(input), hash); //size
    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++) { //conversion into hex
        sprintf(output + (i * 2), "%02x", hash[i]);
    }
}
```

Findme_J

Methodology

1. **Install Java Runtime Environment (JRE) package**

```
┌──(rachel㉿kali)-[~]
└─$ sudo apt update
[sudo] password for rachel:
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main arm64 Packages [20.2 MB]
Get:3 http://kali.download/kali kali-rolling/main arm64 Contents (deb) [47.9 MB]
Get:4 http://kali.download/kali kali-rolling/contrib arm64 Packages [87.5 kB]
Get:5 http://kali.download/kali kali-rolling/contrib arm64 Contents (deb) [157 kB]
Get:6 http://kali.download/kali kali-rolling/non-free arm64 Packages [154 kB]
Get:7 http://kali.download/kali kali-rolling/non-free arm64 Contents (deb) [829 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware arm64 Packages [9703 B]
Get:9 http://kali.download/kali kali-rolling/non-free-firmware arm64 Contents (deb) [22.4 kB]
Fetched 69.4 MB in 20s (3416 kB/s)
2130 packages can be upgraded. Run 'apt list --upgradable' to see them.

┌──(rachel㉿kali)-[~]
└─$ sudo apt install default-jre
The following packages were automatically installed and are no longer required:
  openjdk-17-jre openjdk-17-jre-headless
Use 'sudo apt autoremove' to remove them.
```

2. **Find and Unzip the findMe_J folder**

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

```
┌──(rachel㉿kali)-[~]
└─$ cd ~/Desktop

┌──(rachel㉿kali)-[~/Desktop]
└─$ ls
'Session 9 '   VR-Final_brief.pdf   findMe_J.zip   http.cap   session_11   wrapingNumbers.zip

┌──(rachel㉿kali)-[~/Desktop]
└─$ unzip findMe_J.zip
Archive:  findMe_J.zip
   creating: findMe_J/
  inflating: findMe_J/findMe.class
  inflating: findMe_J/instruction.txt
```

3. Discovered that a key was required to access the file

```
┌──(rachel㉿kali)-[~/Desktop/findMe_J]
└─$ java findMe
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter key:

Invalid key
```

4. Read the instruction file

```
┌──(rachel㉿kali)-[~/Desktop/findMe_J]
└─$ cat instruction.txt
There is a flag for you!

You are great Reverse Engineer.

Find me out!
```

5. Download the CFR Java decompiler

```
┌──(rachel㉿kali)-[~]
└─$ java -version
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
openjdk version "21.0.5" 2024-10-15
OpenJDK Runtime Environment (build 21.0.5+11-Debian-1)
OpenJDK 64-Bit Server VM (build 21.0.5+11-Debian-1, mixed mode, sharing)

┌──(rachel㉿kali)-[~]
└─$ wget http://www.benf.org/other/cfr/cfr-0.152.jar
--2024-11-27 15:50:58--  http://www.benf.org/other/cfr/cfr-0.152.jar
Resolving www.benf.org (www.benf.org)... 217.160.0.68
Connecting to www.benf.org (www.benf.org)|217.160.0.68|:80... connected.
HTTP request sent, awaiting response ... 200 OK
Length: 2162315 (2.1M) [application/java-archive]
Saving to: ‘cfr-0.152.jar’

cfr-0.152.jar             100%[==================================================>]  2.06M  3.84MB/s    in 0.5s

2024-11-27 15:50:59 (3.84 MB/s) - ‘cfr-0.152.jar’ saved [2162315/2162315]
```

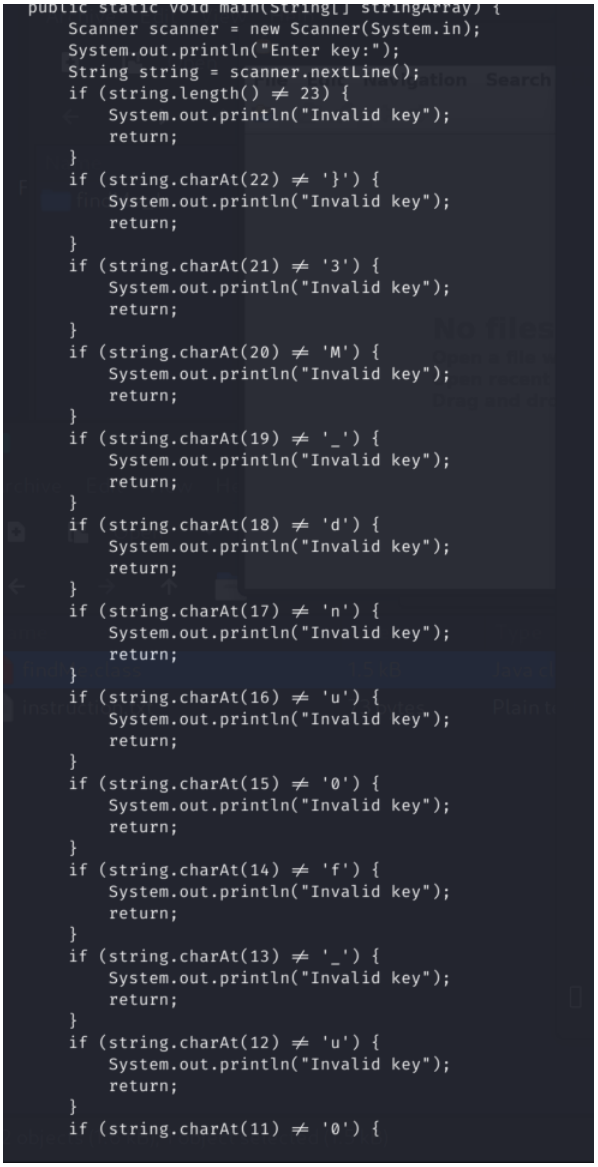
6. Decompile findMe.class file

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

```
┌──(rachel㉿kali)-[~/Desktop]
└─$ java -jar cfr-0.152.jar ./findMe_J/findMe.class
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
/*
 * Decompiled with CFR 0.152.
 */
import java.util.Scanner;
```

7. **Analyse results and find that key is 'VRctf{707_y0u_f0und_M3}'**



```
public static void main(String[] stringArray) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter key:");
    String string = scanner.nextLine();
    if (string.length() != 23) {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(22) != '}') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(21) != '3') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(20) != 'M') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(19) != '_') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(18) != 'd') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(17) != 'n') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(16) != 'u') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(15) != '0') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(14) != 'f') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(13) != '_') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(12) != 'u') {
        System.out.println("Invalid key");
        return;
    }
    if (string.charAt(11) != '0') {
```

8. **Open file using key**

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

```
┌──(rachel㉿kali)-[~/Desktop/findMe_J]
└─$ java findMe
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter key:
VRctf{707_y0u_f0und_M3}
Valid key
```

1. Vulnerability Identification

We analysed the .class file which contained key validation logic.

Steps Taken:

Reverse Engineering: Decompiled the .class file using CFR to extract the program logic.

Static Analysis: Checked the decompiled code to understand how the key was validated.

Tools Used:

- java to run the file.
- CFR to extract readable code.
- Attempted GUI-based tools like JD-GUI, but faced compatibility issues.

Finding: The program had hardcoded logic to validate keys, making it vulnerable to reverse engineering.

2. Exploit Development

How the Exploit Worked:

- Decompiled the program to find a series of if conditions that checked each character of the key.
- Manually constructed a valid key (VRctf{707_y0u_f0und_M3}) to meet all the conditions.
- Entered the key into the program, which verified it as valid.

Result: The exploit successfully bypassed the validation using the reconstructed key.

3. Mitigation Implementation

Problem: The program's validation logic was hardcoded, making it easy to reverse engineer.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Fixes:

- Instead of hardcoding, store keys securely (e.g., in environment variables or a server).
- Use code obfuscation tools to make reverse engineering harder.
- Validate the key on a secure server rather than in the program.

Impact: If left unfixed, an attacker can reverse engineer the program to extract sensitive keys.

ROP Challenge_bonus

Vulnerability Identification

Within our ROP Challenge coding we can see that the vulnerability is caused by the use of gets(buf) in the vuln() function and it has been used unsafely.

- We have carried out static analysis by reading our code which shows that the function gets(buf) does not limit input size. This can allow a buffer overflow to occur which will lead to an attacker overwriting the stack and controlling the flow.

Screenshot of gets(buf) code

- Dynamic analysis was carried out to show how the programme has run but we will test with an input which will be longer than the bufsize (128 bytes) which will cause undefined actions such as a crash. We have also used GDB to debug and observe our stack overflow.

Screenshot of bufsize (128 bytes)

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

```c
#define BUFSIZE 128

void give_shell(){
    gid_t gid = getegid();
    setresgid(gid, gid, gid);
    system("/bin/sh -i");
}

void vuln(){
  char buf[BUFSIZE];
  puts("Give me some text:");
  fflush(stdout);
  gets(buf);
  puts(buf);
  fflush(stdout);
}
```

- Application of using tools like Ghidra will reveal the binary structure and confirm gets() as a vulnerability.
- Finally, implementing debugging using GDB, will identify give_shell()'s address so that it can be exploited.
- Using tools such as AFL will automatically input generation to identify any vulnerabilities and how the program will handle data.

Exploitation

- Our first exploit is that we have created a buffer flow which contains more characters than the allocated buffer size of 128 bytes.
- We have implemented chaining small parts of our code to control the programme flow. We have carried this our on our coding that contains the ret () function
- We have redirected the return address to our give_shell function which is our final exploitation.

Steps to exploit:

Bella could you add this here

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Mitigation Implementation

- Our first mitigation is to change the gets () function into an fgets() function which will prevent a buffer overflow.

- Use of stack canaries to detect buffer overflow attacks

- Randomise the memory addresses of the ROP gadgets so that it is harder to find by the use of address space layout randomisation (ASLR)

- Ensuring that the stack is non-executable which will prevent the executing shellcode being placed on the stack. This will increase the complexity for ROP based exploits alongside the ASLR.

- Finally, making sure our program does not run with any elevated privileges and will limit the impact of any exploitation.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Wrapping Numbers

```
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ls
flag.txt  instructions.txt  sum
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ cat instructions.txt
I know you love Maths :)
Can you think about two different numbers that satisfies this condition? (x > x + y OR y > x + y)
```

Executable file sum can be seen with the ls command along with flag.txt and the instructions.txt.

Integer Overflow

This vulnerability was identified when discovering the CTF. The addIntOvf function should be checking for integer overflow when adding two integers (num1 and num2). However, the current code has incorrect logic. If the sum of a and b is greater than -1, an overflow is incorrectly indicated. Overflow occurs when the sum exceeds the maximum or minimum values that a data type can hold.

Decompile: addIntOvf - (sum)

```
1
2  int addIntOvf(int result,int a,int b)
3
4  {
5    int iVar1;
6    int b_local;
7    int a_local;
8    int result_local;
9
10   if (((a < 1) || (b < 1)) || (-1 < b + a)) {
11     if (((a < 0) && (b < 0)) && (0 < b + a)) {
12       iVar1 = -1;
13     }
14     else {
15       iVar1 = 0;
16     }
17   }
18   else {
19     iVar1 = -1;
20   }
21   return iVar1;
22 }
23
```

The default overflow behaviour in the else block, where it sets iVar1 = -1, can lead to false positives in overflow detection because it indicates an overflow has occurred despite the actual value of a and b. The use of comparisons like < in the addIntOvf function shows incorrect logic, as it can lead to errors. It should ensure that the values exceed the minimum and maximum values properly. The function currently returns -1 when an overflow is detected, along with an

'no overflow' message. Additional error handling is required to provide the necessary information to understand the error.

```
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ./sum
num1 > num1 + num2 OR num2 > num1 + num2
What are two pisitive numbers that satisfies this condition? Provide them like x y
2147483647
10
You entered 2147483647 and 10
You have an integer overflow
YOUR FLAG IS: yourCFT{You_smashed_!t}

cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ./sum
num1 > num1 + num2 OR num2 > num1 + num2
What are two pisitive numbers that satisfies this condition? Provide them like x y
21346775873585478527642 4
237924785825736984703u4
You entered -1 and -1
No overflow
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ./sum
num1 > num1 + num2 OR num2 > num1 + num2
What are two pisitive numbers that satisfies this condition? Provide them like x y
198733421
123
You entered 198733421 and 123
No overflow
```

This vulnerability was exploited by exploiting the overflow check, which occurs by providing large value input for num1 and num2 that exceeds the 32-bit integer limit.

2147483647 + 1 was input once the application was run. 2147483647 + 1 = 2147483648, which exceeds the maximum value a 32-bit integer can store. This triggered the overflow, which was also printed by the application.

```
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ./sum
num1 > num1 + num2 OR num2 > num1 + num2
What are two pisitive numbers that satisfies this condition? Provide them like x y
2147483647
1
You entered 2147483647 and 1
You have an integer overflow
YOUR FLAG IS: yourCFT{You_smashed_!t}
```

yourCFT{You_smashed_!t}

Mitigation

```
if (a > INT_MAX - b) {
    // Handle overflow
    printf("Overflow detected\n");
    exit(1);  // Or handle error accordingly
}
int result = a + b;
```

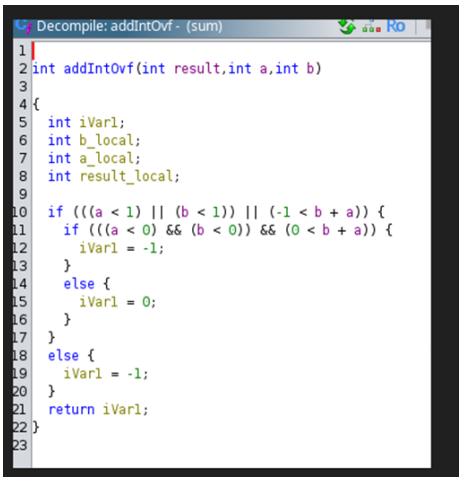
Checking for overflow before performing the arithmetic calculation/operation.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

```c
#include <limits.h>
int safe_add(int a, int b) {
    if (a > INT_MAX - b || a < INT_MIN - b) {
        return -1;  // Overflow detected
    }
    return a + b;
}
```

To mitigate integer overflow in the code, it's best practice to set the integer limit within the code.

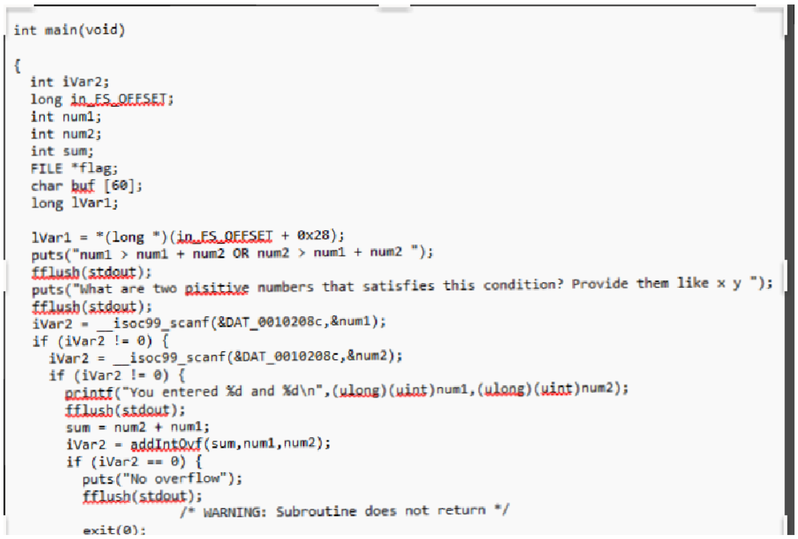


VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Input Validation

The scanf function reads the user's input (num1 and num2) without any form of input validation. This can lead to issues when invalid and unexpected inputs are entered.

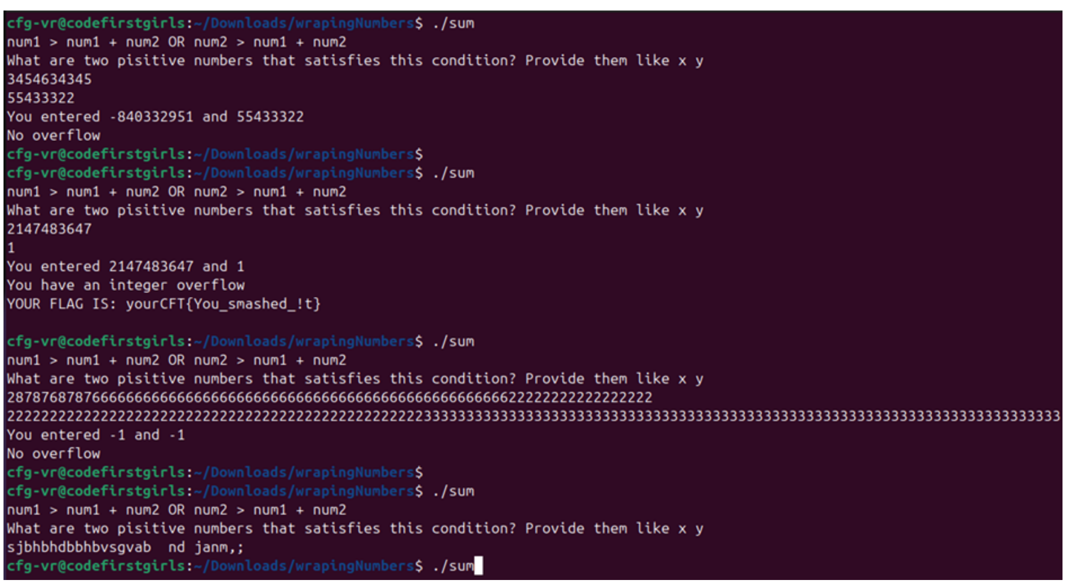


```c
int main(void)

{
  int iVar2;
  long in_FS_OFFSET;
  int num1;
  int num2;
  int sum;
  FILE *flag;
  char buf [60];
  long lVar1;

  lVar1 = *(long *)(in_FS_OFFSET + 0x28);
  puts("num1 > num1 + num2 OR num2 > num1 + num2 ");
  fflush(stdout);
  puts("What are two pisitive numbers that satisfies this condition? Provide them like x y ");
  fflush(stdout);
  iVar2 = __isoc99_scanf(&DAT_0010208c,&num1);
  if (iVar2 != 0) {
    iVar2 = __isoc99_scanf(&DAT_0010208c,&num2);
    if (iVar2 != 0) {
      printf("You entered %d and %d\n",(ulong)(uint)num1,(ulong)(uint)num2);
      fflush(stdout);
      sum = num2 + num1;
      iVar2 = addIntOvf(sum,num1,num2);
      if (iVar2 == 0) {
        puts("No overflow");
        fflush(stdout);
                    /* WARNING: Subroutine does not return */
        exit(0);
```

There is no bound checking; the program does not check if the user input is within the range of the 32-bit integer limit. If the user inputs different data types or other invalid inputs, it could lead to undefined behaviour. Additionally, this could result in arbitrary memory access. num1 and num2 are uninitialized in the case of invalid input, which can be manipulated and lead to logic flaws. Ensuring all variables are initialised, validating the input format, and using a width specifier with scanf for strings would help mitigate these issues. User inputs should be sanitized before being used in calculations.



```
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ./sum
num1 > num1 + num2 OR num2 > num1 + num2
What are two pisitive numbers that satisfies this condition? Provide them like x y
3454634345
55433322
You entered -840332951 and 55433322
No overflow
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ./sum
num1 > num1 + num2 OR num2 > num1 + num2
What are two pisitive numbers that satisfies this condition? Provide them like x y
2147483647
1
You entered 2147483647 and 1
You have an integer overflow
YOUR FLAG IS: yourCFT{You_smashed_!t}

cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ./sum
num1 > num1 + num2 OR num2 > num1 + num2
What are two pisitive numbers that satisfies this condition? Provide them like x y
2878768787666666666666666666666666666666666666666666666666666222222222222222
22222222222222222222222222222222222222222222222222333333333333333333333333333333333333333333333333333333333333333333333333333333
You entered -1 and -1
No overflow
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ./sum
num1 > num1 + num2 OR num2 > num1 + num2
What are two pisitive numbers that satisfies this condition? Provide them like x y
sjbhbhdbbhbvsgvab  nd janm,;
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ ./sum
```

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

The Flag.txt file was accessible prior via the terminal.

```
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ printf 'A%.0s' {1..50000} > flag.txt
cfg-vr@codefirstgirls:~/Downloads/wrapingNumbers$ cat flag.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

The code uses fgets(buf, 0x3b, (FILE *)flag) to read data from flag.txt into a buffer of 60 bytes (buf[60]).

The code checks the size by using 0x3b (59 bytes), which is appropriate for the buf[60] buffer.

No buffer overflow occurs when the file content is 59 characters or less because fgets respects the size limit.

<u>Potential Risk:</u>

Hardcoded Size: The size of 59 bytes is hardcoded. If the buffer size (buf[60]) is changed, the size passed to fgets would still be 59, leading to a potential mismatch and buffer overflow.

No Input Validation: There is no validation on the content size beyond reading it. If flag.txt contains more than 59 characters, the excess will be ignored, but further processing of the data may lead to vulnerabilities. The buffer can overflow which then overwrites adjacent memory. An attacker could potentially inject malicious code and take full control of the execution flow.

<u>Mitigation:</u>

Use fgets(buf, sizeof(buf), (FILE *)flag) to automatically adjust the buffer size, ensuring consistency between the buffer and the size passed to fgets:

```
    fgets(buf,0x3b,(FILE *)flag);
    printf("YOUR FLAG IS: %s\n",buf);
    fflush(stdout);
```

Debug_me Challenge

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

GDB (GNU debugger) and Pwndbg tool were used for this challenge Debug_me. It has benefits when working with binary analysis and reverse engineering. GDB runs the program and disassembles the binary and allows for inspection at assembly level.

```
cfg-vr@codefirstgirls:~/Downloads/Debug_me$ cat inst.txt
Dissassemble ME!

I am the last value of the eax register at the end of the main function?

Your flag should be in this format: diss_CTF{dddddd} where dddddd is the contents of the eax register in the decimal number base.

If the answer was 0x123 your flag would be diss_CTF{291}. You can use calculator to do convertions.
cfg-vr@codefirstgirls:~/Downloads/Debug_me$
```

Image 1

```
cfg-vr@codefirstgirls:~/Downloads/Debug_me$ gdb debug_me
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 175 pwndbg commands and 45 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $base, $hex2ptr, $bn_sym, $bn_var, $bn_eval, $ida GDB functions (can be used with print/break)
Reading symbols from debug_me...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
(No debugging symbols found in debug_me)
------- tip of the day (disable with set show-tips off) -------
Use patch <address> '<assembly>' to patch an address with given assembly code
pwndbg> disassemble main
Dump of assembler code for function main:
   0x0000000000001129 <+0>:     endbr64
   0x000000000000112d <+4>:     push   rbp
   0x000000000000112e <+5>:     mov    rbp,rsp
   0x0000000000001131 <+8>:     mov    DWORD PTR [rbp-0x4],edi
   0x0000000000001134 <+11>:    mov    QWORD PTR [rbp-0x10],rsi
```

- gdb debug_me- runs the file in GDB mode.
- Shows no debugging symbols found in the file.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Image 2

```
pwndbg> disassemble main
Dump of assembler code for function main:
   0x0000000000001129 <+0>:     endbr64
   0x000000000000112d <+4>:     push   rbp
   0x000000000000112e <+5>:     mov    rbp,rsp
   0x0000000000001131 <+8>:     mov    DWORD PTR [rbp-0x4],edi
   0x0000000000001134 <+11>:    mov    QWORD PTR [rbp-0x10],rsi
   0x0000000000001138 <+15>:    mov    eax,0x86342
   0x000000000000113d <+20>:    pop    rbp
   0x000000000000113e <+21>:    ret
End of assembler dump.
```

- Command to disassemble main was run displaying the assembly language.

Image 3

```
0x0000000000001138 <+15>:    mov    eax,0x86342
```

- Identifies the value being moved into the eax register at the end of the main function can be seen as a hexadecimal value (0x86342).

Image 4

```
──────────[ DISASM / x86-64 / set emulate on ]──────────
► 0x555555555129 <main>                    endbr64
  0x55555555512d <main+4>                  push   rbp
  0x55555555512e <main+5>                  mov    rbp, rsp                    RBP => 0x7fffffffdbf0 → 0x7fffffffdc90 → 0x7fffffffdcf0 ← ...
  0x555555555131 <main+8>                  mov    dword ptr [rbp - 4], edi    [0x7fffffffdbec] <= 1
  0x555555555134 <main+11>                 mov    qword ptr [rbp - 0x10], rsi [0x7fffffffdbe0] <= 0x7fffffffdd18 → 0x7fffffffe06c ← '/home/c
oads/Debug_me/debug_me'
  0x555555555138 <main+15>                 mov    eax, 0x86342                EAX => 0x86342
  0x55555555513d <main+20>                 pop    rbp                         RBP => 0x7fffffffdc90
  0x55555555513e <main+21>                 ret                              <__libc_start_call_main+122>
   ↓
  0x7ffff7c2a1ca <__libc_start_call_main+122>    mov    edi, eax             EDI => 0x86342
  0x7ffff7c2a1cc <__libc_start_call_main+124>    call   exit                 <exit>

  0x7ffff7c2a1d1 <__libc_start_call_main+129>    call   __nptl_deallocate_tsd <__nptl_deallocate_tsd>
```

- Shows the value being moved into the eax register, as well as the value being moved from the eax register to the edi register.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Image 5

```
cfg-vr@codefirstgirls:~/Downloads/Debug_me$ python3 -c "print(int('0x86342', 16))"
549698
cfg-vr@codefirstgirls:~/Downloads/Debug_me$
```

- Python code was used to convert the hexadecimal value into a decimal.
- The hexadecimal string is converted using the int () function with base 16.
- The int () function is used to convert a value into an integer in Python.
- The output of the conversion is 549698.
- Flag - diss_CTF{549698}

Image 6

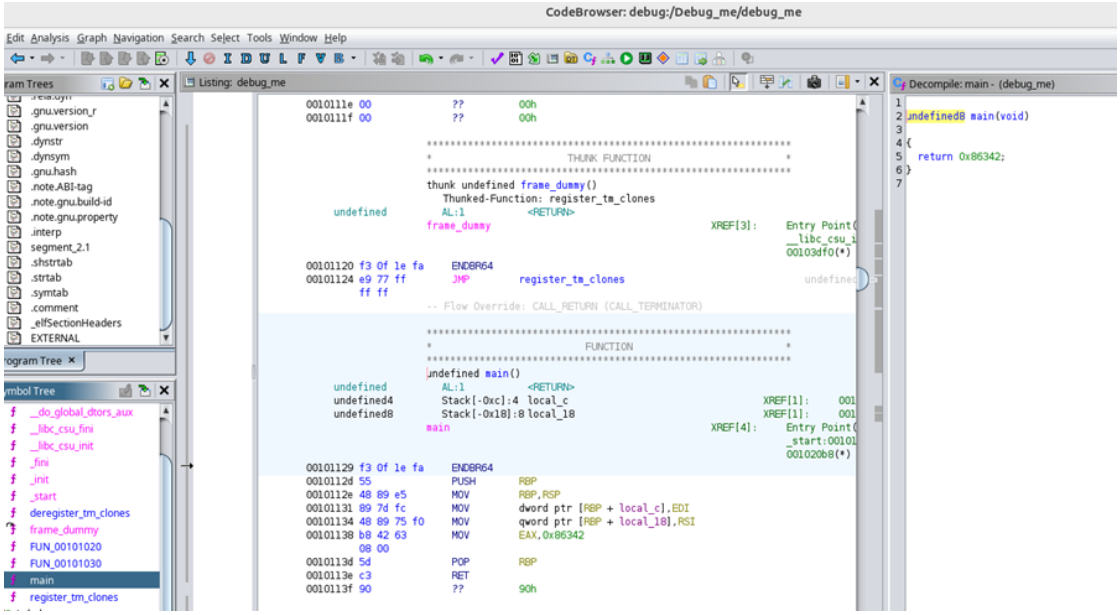
```
pwndbg> print /d 0x86342
$1 = 549698
```

- The pwndbg command (print /d 0x86342) prints the value of the register in decimal. /d option tells pwndbg to print in decimal format.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Image 7



GDB was a simpler tool to use in starting the program and disassembling the main function in comparison to Ghidra for example which requires additional steps in accessing the assembly language. GBD provides a more streamlined approach for certain tasks such as starting the program, quick disassembling functions.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

DumpMe-Ascii Challenge

```
cfg-vr@codefirstgirls:~/Downloads/dump ascii (2)$ cat instructions.txt
Dump the file and decode your flag. It should be somthing like xxxx{yyyyyy}.
Good luck.
cfg-vr@codefirstgirls:~/Downloads/dump ascii (2)$
```

Image 1

```
cfg-vr@codefirstgirls:~/Downloads/dump ascii$ file dumpMe
dumpMe: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, int
rpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=e1c32dace8ac1516160b771e493f5ebffcac9
55, for GNU/Linux 3.2.0, not stripped
cfg-vr@codefirstgirls:~/Downloads/dump ascii$ ./ dumpMe
bash: ./: Is a directory
cfg-vr@codefirstgirls:~/Downloads/dump ascii$ ./dumpMe
The flag starts with 63
cfg-vr@codefirstgirls:~/Downloads/dump ascii$ cat dumpMe
@@@@◆XX◆◆   pp◆-◆=◆=`h◆-◆=◆=◆888 XXXDDS◆td888 P◆td      DDQ◆tdR◆td◆-◆=◆=PP/lib64/ld-linu
-x86-64.so.2GNU◆GNU◆◆-◆◆
                            wI?^◆◆◆◆UGNU◆◆e◆mZ
                                                  2v ◆ #"libc.so.6__stack_chk_failprintf__
xa_finalize__libc_start_mainGLIBC_2.2.5GLIBC_2.4_ITM_deregisterTMCloneTable__gmP◆`@◆?◆?◆
◆?◆?◆?◆?◆◆H◆H◆◆/H◆◆t◆◆H◆◆◆5◆/◆◆%◆/◆◆h◆◆◆◆◆◆◆◆◆h◆◆◆◆◆◆◆◆◆◆%◆/D◆◆◆◆%]/D◆◆◆◆◆H◆=◆◆2/◆◆H◆=Y
H◆R/H9◆tH◆/H◆◆t  ◆◆◆◆◆H◆=)/H◆5"/H)◆H◆◆H◆◆?H◆◆H◆H◆◆tH◆◆.H◆◆◆◆fD◆◆◆◆◆=◆.u+UH◆=◆.H◆◆t
             H◆=◆.◆     ◆◆◆◆d◆◆◆◆◆.]◆◆◆◆◆◆w◆◆◆◆◆UH◆◆H◆◆◆0dH◆%(H◆E◆1◆◆E◆c◆E◆f◆E◆g◆E◆-◆E◆C◆
◆T◆E◆F◆E◆{◆E◆A◆E◆S◆E◆C◆E◆I◆E◆I◆E◆_◆E◆I◆E◆S◆E◆_◆E◆E◆E◆A◆E◆S◆E◆Y◆E◆_◆E◆V◆E◆U◆E◆L◆◆◆V◆◆◆◆H◆
◆dH3%(t◆1◆◆◆◆◆f.◆D◆◆AWL◆=c+AVI◆◆AUI◆◆ATA◆◆UH◆-T+SL)◆H◆◆◆◆H◆◆t1◆◆L◆◆L◆◆D◆◆A◆◆H◆◆H9◆u◆H◆[
A\A]A^A_◆ff.◆◆◆◆◆◆H◆H◆◆The flag starts with %x
D◆◆◆x0◆◆◆◆@◆◆◆◆`◆◆◆`I◆◆◆◆ ◆◆◆◆◆◆◆◆8zRx
                                      ◆◆◆◆/D$4◆◆◆◆0FJ
◆                                                  ◆?◆:*3$"\◆◆◆◆t◆◆◆◆ ◆q◆◆◆◆E◆C
D◆(◆◆◆eF◆I◆E ◆E(◆D0◆H8◆G@n8A0A(B BB◆P◆◆◆`
◆◆◆◆◆◆◆◆o◆◆◆◆
◆
 ◆?0(◆ ◆◆◆◆◆◆◆o8◆◆◆◆o◆◆◆◆o(◆◆◆◆o◆=0@GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.08X|◆◆(    8
h
 (
 P`◆◆    h ◆=◆=◆=◆?@◆@◆
                      ◆◆! 7◆@F◆=m`y◆=◆◆◆◆◆◆l!◆◆◆◆=◆◆=◆◆=◆   ◆◆?◆
                                                                ◆ ◆ @3@◆:Vj◆@◆ @◆ ◆@e◆◆@◆
```

1. file dumpMe- identifies what type of file

- Shows ELF: that it is in linkable and executable format.
- 64bit indicating that the binary file is completed for 64-bit architecture.
- The binary file has shared libraries which load at run time.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

- Not stripped- indicates the binary file has debug symbols which is useful for reverse engineering making it easier to identify functions and variables.
2. ./ dumpMe runs the file- output showing 'The flag starts with 63'.
3. Cat dumpMe- attempts to read the file however the content is obfuscated. The file content has been made hard to understand. The file contacts binary data and readable string (The flag starts with %x) can be seen within the file data.

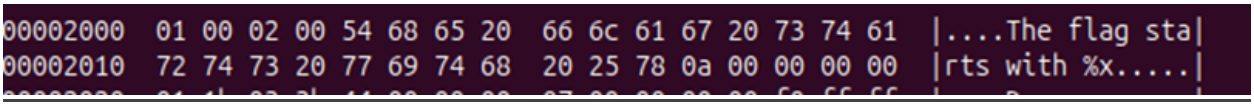
Image 2

- Stings dumpMe shows the file has stack canaries, the libraries used. 'The flag starts with %x can be seen.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Image 3

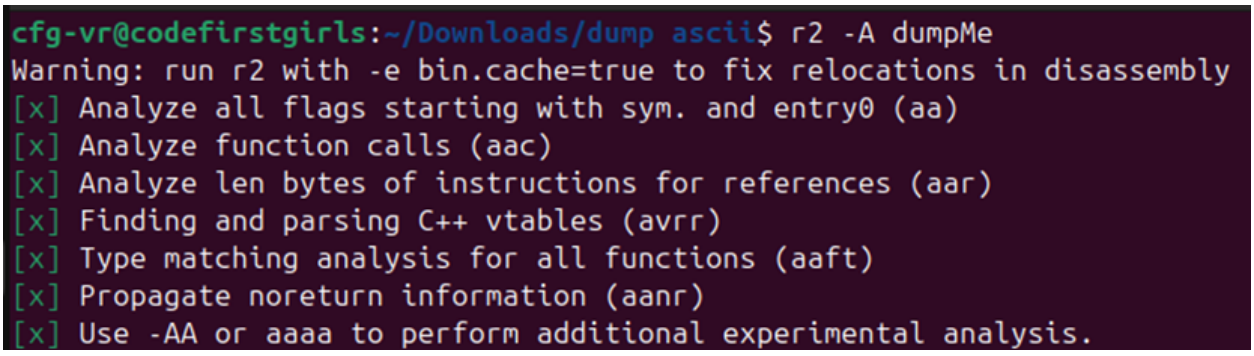


- hexdump -C dumpMe command used to inspect raw binary data -Again 'The flag starts with %x can be seen.

Radare2 tool was the chosen tool, the file was obfuscated, it would then need to be reverse engineered to find how the flag is generated or stored. Radare2 supports reading strings within binary and helps identify areas of code that references the flag.

Radare2 also supports analysing raw bytes in hexadecimal formats and their equivalent ASCII character. In addition to the analysis of external calls, the binary file calls external libraries and allows for the inspection of the calls.

Image 3



- r2 – A dumpMe opens the binary file for analysis.

Image 4

```
[0x00001080]> aa
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00001080]> aaaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information (aanr)
[x] Finding function preludes
[x] Enable constraint types analysis for variables
```

- aa (analyse all) command was used to analyse the entire binary.
- aaaa (analyse all recursively) was used for a deeper analysis of the binary functions, strings and control flow.
- afl (analyses function list) listing all the functions used in the binary.

Image 5

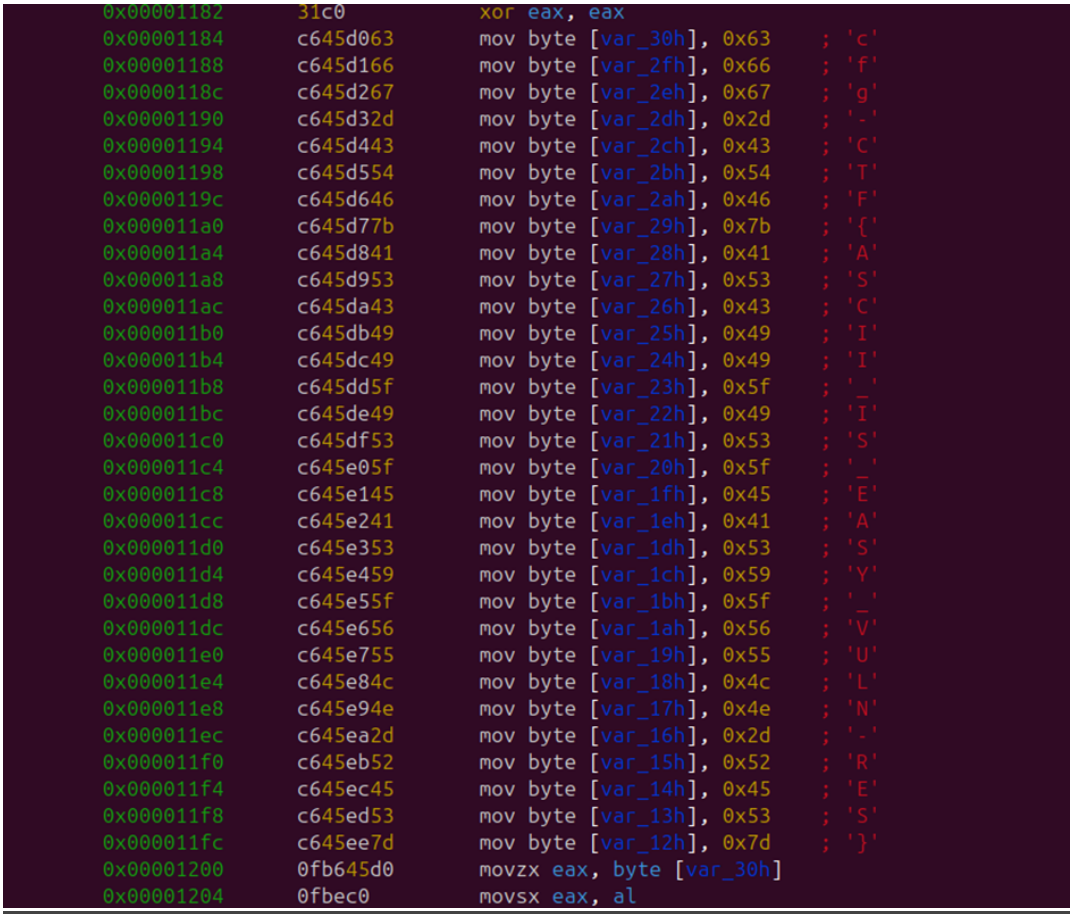
```
[0x00001080]> pd @main
```

Print Disassembly- pd @main command disassembles the main function at the main address.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Image 6



- The Pd @main command identifies that the flag is being stored in the memory at locations corresponding to var_30, var_2eh and so on.
- The values are moved to specific memory locations using .mov instruction.
- The column in yellow represents the byte values in hexadecimal. The byte values are then converted into ascii characters which are being stored.

0x63 is the hexadecimal value that represents the ASII character ‘c’

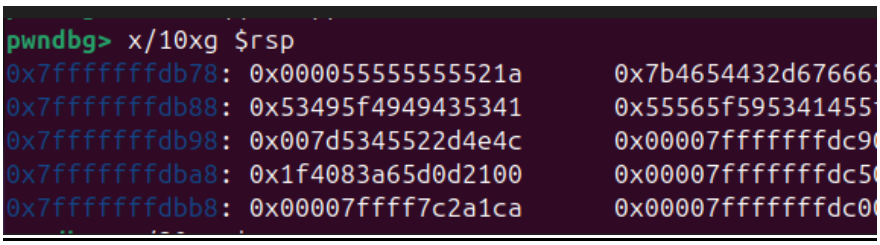
0x66 is the hexadecimal value that represents the ASII character ‘f’

0x67 is the hexadecimal value that represents the ASII character ‘g’

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

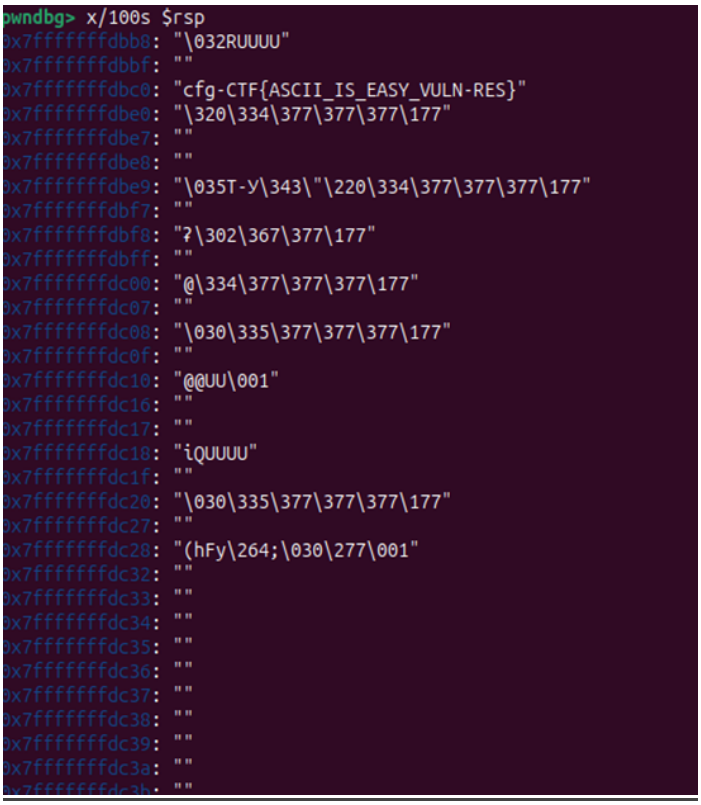
Image 7



x/10xg $rsp GDB command to examine at the stack pointer.

Highlighting the value in memory shown in hexadecimal and in 64-bit chucks.

Image 8



- x/100s $rsp examined the memory location of the stack pointer displaying 100 values of strings which also help see readable text/flag.

The flag cfg-CTF{ASCII_IS_EASY_VULN_RES} can also be seen with this command.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Conclusion

Our project has successfully completed its aim to identify common vulnerabilities found in software systems. We have demonstrated effective methodology to identify, exploit and mitigate the vulnerabilities found. By applying tools and techniques such as static and dynamic analysis, reverse engineering and cryptographic techniques our team has showcased the importance of needing to review security thoroughly to make sure our software is safeguarded. The mitigations that were implemented aimed at addressing the immediate risk identified but also provided the benchmark for developing a more secure codebase in the future. This project has emphasised the critical and practical role that we must place to add security measures to protect our codebases against the ever evolving cyber threats we see today and will continue to see.

References

Manual Code Review:

- CFG degree Vulnerability Research Static Analysis Techniques slides 25-26
- CFG degree Vulnerability Research Static and Dynamic Analysis slides 7-11, 14
- https://www.blackduck.com/glossary/what-is-code-review.html#:~:text=Manual%20code%20review%20involves%20a,general%20business%20logic%20into%20consideration.
- https://www.blackduck.com/glossary/what-is-code-review.html#:~:text=Manual%20code%20review%20involves%20a,general%20business%20logic%20into%20consideration.
- https://stackoverflow.com/questions/574159/what-is-a-buffer-overflow-and-how-do-i-cause-one
- https://www.youtube.com/watch?v=ytGATjX3nqc
- https://www.geeksforgeeks.org/blockchain-encrypt-decrypt-files-with-password-using-openssl/
- https://stackoverflow.com/questions/46053182/caesar-cipher-in-c-encryption-and-decryption
- https://stackoverflow.com/questions/16213546/secure-configuration-file-in-clients
- https://wiki.sei.cmu.edu/confluence/display/c/MSC24-C.+Do+not+use+deprecated+or+obsolescent+functions
- https://uk.mathworks.com/discovery/misra-c.html
- https://www.blackduck.com/static-analysis-tools-sast/misra.html
- https://ldra.com/sei-cert/

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

·https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/#:~:text=It%20includes%20an%20introduction%20to,quickly%20understand%20secure%20coding%20practices.

Findme_J:

https://www.youtube.com/watch?v=dFbMdo8peF8
https://forums.kali.org/archived/showthread.php?41-Installing-Java-on-Kali-Linux
http://java-decompiler.github.io/
https://www.jetbrains.com/help/idea/decompiler.html#show-bytecode
Youtube
Class slides

ROP Challenge_bonus:

- CFG degree Vulnerability Research Exploitation: Return Object Programming. Slides 5-38
- https://dl.acm.org/doi/pdf/10.1145/3029806.3029812
- https://bluegoatcyber.com/blog/unpacking-rop-vulnerabilities/#:~:text=Defining%20ROP%20Vulnerabilities,the%20improper%20handling%20of%20input.
- https://engineering.backtrace.io/2016-08-31-rop-exploitation-detection/
- https://www.youtube.com/watch?v=8zRoMAkGYQE

. https://ctf101.org/binary-exploitation/return-oriented-programming/

Wrapping Numbers:

CWE - CWE-738: CWE CATEGORY: CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT) (4.16) (no date). https://cwe.mitre.org/data/definitions/738.html.

CWE - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (4.16) (no date). https://cwe.mitre.org/data/definitions/120.html.

Buffer Overflow | OWASP Foundation (no date). https://owasp.org/www-community/vulnerabilities/Buffer_Overflow.

VR-Final Project Group 2

Vulnerability Research and Mitigation Project

Dowd, M., McDonald, J. and Schuh, J. (2006) The Art of Software Security Assessment: Identifying and Preventing software Vulnerabilities. https://dl.acm.org/citation.cfm?id=1196394.

W3Schools.com (no date). https://www.w3schools.com/c/c_operators.php.

CWE - CWE-680: Integer overflow to buffer overflow (4.16) (no date). https://cwe.mitre.org/data/definitions/680.html.

Ghidra (no date). https://ghidra-sre.org/.

Presentation slides

Interpreted language

Exploitation_bufferoverflow

Dynamic analysis

Vulnerability

Dump ASCII:

hexdump(1) - Linux manual page (no date). https://man7.org/linux/man-pages/man1/hexdump.1.html.

radare2 Cheatsheet (no date). https://scoding.de/uploads/r2_cs.pdf.

Top (Debugging with GDB) (no date). https://sourceware.org/gdb/current/onlinedocs/gdb.html/.

Shell Commands (Debugging with GDB) (no date).

https://sourceware.org/gdb/current/onlinedocs/gdb.html/Shell-Commands.html#Shell-Commands.

Injosoft AB, http://www.injosoft.se (no date) *ASCII table - Table of ASCII codes, characters and symbols*. https://www.ascii-code.com/.

Ghidra (no date). https://ghidra-sre.org/.

Presentation slides

Reverse engineering

Understanding assembly

Decomplication

Debug_me:

Internals - GDB Wiki (no date). https://sourceware.org/gdb/wiki/Internals.

PWNDBG CHEATSHEET (no date). https://pwndbg.re/CheatSheet.pdf.

Built-in functions (no date). https://docs.python.org/3/library/functions.html#int.

Registers (Debugging with GDB) (no date).

https://sourceware.org/gdb/current/onlinedocs/gdb.html/Registers.html.

Presentation slides

Reverse engineering

Understanding assembly

Decomplication

VR-Final Project Group 2

Vulnerability Research and Mitigation Project