HW/Lab-4

信息安全导论 – Spring 2022

Homework/Lab #4

Due: Tuesday, May 17, 2022

Highlights

- Expected contribution towards the final score: 6%.
- **You should work on this homework individually or in teams of up to 3 members (highly recommended). One submission per team.**
- **Submit your work as a pdf** through the USTC Blackboard

1. **(10 points)** Describe what a NOP sled is and how it is used in a buffer overflow attack.

2. (**15 points)** Look into different shellcodes released in Packet Stream, and summarize different operations an attacker may design shellcode to perform.

3. (**30 points)**Below is a simple C code with a buffer overflow issue.

   a) **(20 points)** Craft a simple buffer overflow exploit, and circumvent the password checking logic. Include in your submission necessary step-by-step screenshots or descriptions to demonstrate how you carry out the attack.

b)**(10 points)** Describe how to fix this buffer overflow issue.

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

int main(int argc, char *argv[]) {
    int valid = false;
    char str1[9] = "fdalfakl";
    char str2[9];
    printf("Input your password:\n");
    gets(str2);
    if (strncmp(str1, str2, 8) == 0) {
        valid = true;
        printf("Your exploit succeeds!\n");
    }
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

4. **(25 points)** Alice is attacking a buggy application. She has found a vulnerability that allows her to control the values of the registers **ecx, edx, and eip**, and also allows her to control the contents of memory locations **0x9000 to 0x9014**. She wants to use return-oriented programming, but discovers that the application was compiled without any ret instructions! Nonetheless, by analyzing the application, she learns that the application has the following code fragments (gadgets) in memory:

```
0x3000: add edx, 4      ; edx = edx + 4
        jmp [edx]       ; jump to *edx

0x4000: add edx, 4      ; edx = edx + 4
        mov eax, [edx]  ; eax = *edx
        jmp ecx         ; jump to ecx

0x5000: mov ebx, eax    ; ebx = eax
        jmp ecx         ; jump to ecx

0x6000: mov [eax], ebx  ; *eax = ebx
        ...             ; don't worry about what happens after this
```

Show how Elizabeth can set the values of the registers and memory so that the vulnerable application writes the value 0x2222 to memory address 0x8888.

ecx	
edx	
eip	0x4000

0x9000	
0x9004	
0x9008	
0x900c	
0x9010	
0x9014	

5. **(20 points)** Consider the following simplified code that was used earlier this year in a widely deployed router. If hdr->ndata = "ab" and hdr->vdata = "cd" then this code is intended to write "ab:cd" into buf. Suppose that the attacker has full control of the contents of hdr. Explain how this code can lead to an overflow of the local buffer buf.

```
uint32_t nlen, vlen;    /*  values in 0 to 2^32-1  */
char buf[8264];

nlen = 8192;
if ( hdr->nlen <= 8192 )
    nlen = hdr->nlen;

memcpy(buf, hdr->ndata, nlen);
buf[nlen] = ':';

vlen = hdr->vlen;
if (8192 - (nlen+1) <= vlen )    /*  DANGER  */
```

```
    vlen = 8192 - (nlen+1);

memcpy(&buf[nlen+1], hdr->vdata, vlen);
buf[nlen + vlen + 1] = 0;
```