Włamania do systemów Linux w architekturze x86

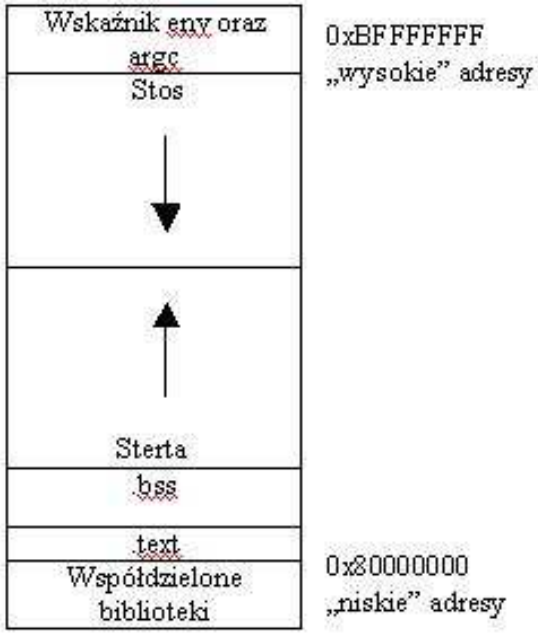
- Adam Zabrocki
- http://pi3.hack.pl
- pi3@itsec.pl (lub oficjalnie: adam.zabrocki@avet.com.pl)

Włamania do systemów Linux w architekturze x86

- Dlaczego Linux?
- Jądro 2.4 a 2.6

Włamania do systemów Linux w architekturze x86

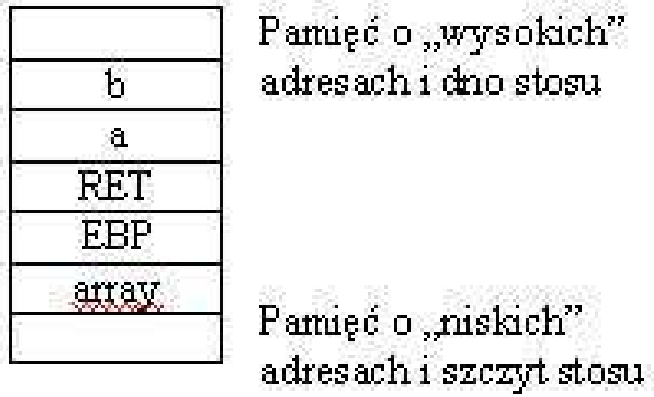
- Architektura x86
- Organizacja pamięci:



Włamania do systemów Linux w architekturze x86

- Stos...
- Wywołanie funkcji:

```
void pi3_fun(int a, int b);
int main(void) {
        pi3_fun(-1,1);
        printf("Adresik powrotu wskazuje na to miejsce :)");
}
void pi3_func(int a, int b) {
        int jakas_zmienna[666];
}
```



Włamania do systemów Linux w architekturze x86

- Początek – stack overflow:

```
int main(int argc, char *argv[]) {
        char buf[100];

        bzero(buf,sizeof(buf));
        if (argv[1])
                strcpy(buf,argv[1]);
        printf(”strlen buf = %u\n”,(unsigned int) strlen(buf));
}
```

Włamania do systemów Linux w architekturze x86

- Początek – stack overflow:

Zdebugujmy program:

```
$ gdb -q ./main
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) disass main
Dump of assembler code for function main:
0x08048444 <main+0>:    push   %ebp
0x08048445 <main+1>:    mov    %esp,%ebp
...
...
0x080484a7 <main+99>:   leave
0x080484a8 <main+100>:  ret
...
0x080484ac <main+104>:  nop
...
End of assembler dump.
(gdb) b *0x08048445
Breakpoint 1 at 0x8048445
(gdb) b *0x080484a8
Breakpoint 2 at 0x80484a8
(gdb)
```

Włamania do systemów Linux w architekturze x86

- Początek – stack overflow:

Zdebugujmy program – ciąg dalszy:

```
(gdb) r `perl -e 'print "A"x128'`
Starting program: /root/projekty/sekit/main `perl -e 'print "A"x128'`

Breakpoint 1, 0x08048445 in main ()
(gdb) x/x $esp
0xbffffc68:     0xbffffc88    <- wartość rejestru %ebp
(gdb) x/x $esp+4              <- wartość rejestru %eip

0xbffffc6c:     0x400384a0
(gdb) x/1i 0x400384a0
0x400384a0 <__libc_start_main+160>:     mov    %eax,(%esp)
(gdb) c
Continuing.
strlen buf = 128

Breakpoint 2, 0x080484a8 in main ()
(gdb) x/x 0xbffffc68
0xbffffc68:     0x41414141        <- wartość rejestru %ebp (po przepełnieniu)
(gdb) x/x 0xbffffc6c
0xbffffc6c:     0x41414141        <- wartość rejestru %eip (po przepełnieniu)
(gdb)
```

Włamania do systemów Linux w architekturze x86

- One-byte stack overflow:

```
int main(int argc, char *argv[]) {
        char buf_1[8],buf_2[4];

        bzero(buf_1,sizeof(buf_1)), bzero(buf_2,sizeof(buf_2));
        if (argv[1]) {
                strncpy(buf_1,argv[1],8);
                strncpy(buf_2,buf_1,4);
        }
        printf(”buf_2 = %s\n”,buf_2);
}
```

Włamania do systemów Linux w architekturze x86

- One-byte stack overflow:

Debug:

```
$ ./2test AAAA
buf_2 = AAAAAAAA
$ ./2test AAA
buf_2 = AAA
$ ./2test AAAA
buf_2 = AAAAAAAA
$ ./2test AAAAAAAA
buf_2 = AAAAAAAAAAAAůŹżdâ˘
$
```

Włamania do systemów Linux w architekturze x86

- One-byte stack overflow – cd.:

```
int main(void) {

   char buf[8];
   long addr = 0x12345678;

   bzero(buf, sizeof(buf)); // czyścimy buf
   printf("addr (przed) = %x\n", addr);
   buf[strlen(buf)-1]='\0'; // kończymy buf
   printf("addr (po) = %x\n", addr);

   return 0;
}
```

Włamania do systemów Linux w architekturze x86

- One-byte stack overflow – cd.:

Debug:

```
$ ./3test
addr (przed) = 12345678
addr (po) = 345678
$
```

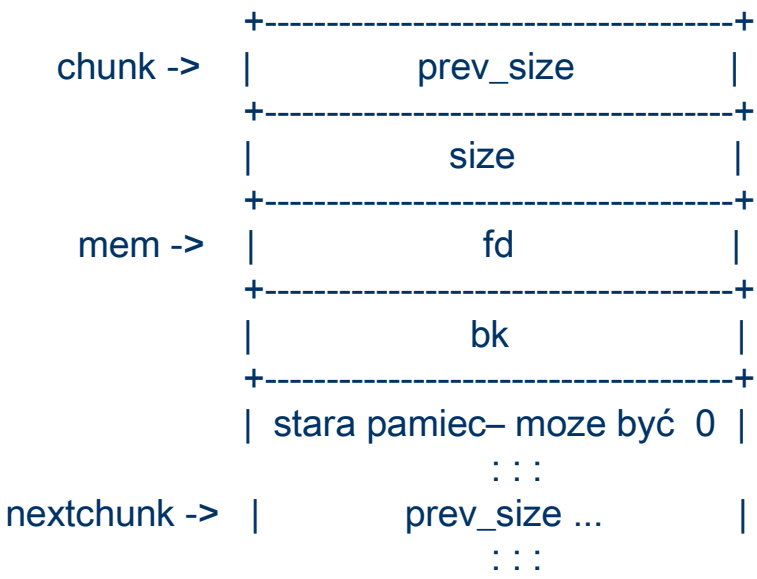
Włamania do systemów Linux w architekturze x86

- Buffer overflow != stack overflow!
- Co ze stertą?


```
Free() => unlink
#define unlink(P, BK, FD) \
{ \
 BK = P->bk; \
 FD = P->fd; \
 FD->bk = BK; \
BK->fd = FD; \
 }
```

Włamania do systemów Linux w architekturze x86

- Heap Overflow
- Co ze sterta?



Włamania do systemów Linux w architekturze x86

- Heap Overflow
- Co ze stertą? – double free()

```
int main(int argc, char *argv[]) {
char *ptr1 = malloc(10);
char *ptr2 = malloc(8);

If (argv[1])
      strcpy(ptr2,argv[1]);
printf(”ptr2 = %s\n”,ptr2);
free(ptr2);
free(ptr1);
return 0;
}
```

Włamania do systemów Linux w architekturze x86

- Heap Overflow
- Co ze stertą? – double free()

```
$ ltrace ./4test `perl -e 'print "A"x8'`
__libc_start_main(0x8048428, 2, 0xbf8fb9a4, 0x80484b0, 0x8048510 <unfinished ...>
malloc(10)                                                = 0x8049718
malloc(8)                                                 = 0x8049728
strcpy(0x8049718, "AAAAAAAA")                                     = 0x8049718
printf("ptr1 = %s\n", "AAAAAAAA"ptr1 = AAAAAAAA
)                                  = 16
free(0x8049728)                                               = <void>
+++ exited (status 0) +++
$ ltrace ./4test `perl -e 'print "A"x10'`
__libc_start_main(0x8048428, 2, 0xbfde49a4, 0x80484b0, 0x8048510 <unfinished ...>
malloc(10)                                                = 0x8049718
malloc(8)                                                 = 0x8049728
strcpy(0x8049718, "AAAAAAAAAA")                                   = 0x8049718
printf("ptr1 = %s\n", "AAAAAAAAAA"ptr1 = AAAAAAAAAA
)                                  = 18
free(0x8049728)                                               = <void>
+++ exited (status 0) +++
$
```

Włamania do systemów Linux w architekturze x86

- Heap Overflow
- Co ze stertą? – double free()

```
$ ltrace ./4test `perl -e 'print "A"x16'`
__libc_start_main(0x8048428, 2, 0xbfb749a4, 0x80484b0, 0x8048510 <unfinished ...>
malloc(10)                                                    = 0x8049718
malloc(8)                                                     = 0x8049728
strcpy(0x8049718, "AAAAAAAAAAAAAAAA")                                 = 0x8049718
printf("ptr1 = %s\n", "AAAAAAAAAAAAAAAA"ptr1 = AAAAAAAAAAAAAAAA
)                                 = 24
free(0x8049728*** glibc detected *** double free or corruption (out): 0x08049728 ***
 <unfinished ...>
--- SIGABRT (Aborted) ---
+++ killed by SIGABRT +++
$
```

Włamania do systemów Linux w architekturze x86

- Format Strings:
- Przykład:

```
int main(int argc, char *argv[]) {
    if (argv[1])
        printf(argv[1]);
    printf("\n");
    return 0;
}
```

Włamania do systemów Linux w architekturze x86

- Format Strings:
- Przykład:

$ ./format ”Ala ma kota :)”
Ala ma kota :)
$ ./format AAAA%x(...)%x
AAAA40013760(...)414141
$

- %n twoim (nie)przyjacielem

Włamania do systemów Linux w architekturze x86

- Inne ataki

- Integer overflow!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
- Signal race
- Race conditional
- TMP file (symlink attack)
- Zmienne środowiskowe
- Błędy logiczne...
- Pozostałe...

Włamania do systemów Linux w architekturze x86

- Real-life:

Przyjrzyjmy się ostatniemu błędowi w oprogramowaniu ISC DHCP :)

```
int
cons_options(struct packet *inpacket, struct dhcp_packet *outpacket,
    int mms, struct tree_cache **options,
    int overload, /* Overload flags that may be set. */
    int terminate, int bootpp, u_int8_t *prl, int prl_len)
{
        ...
        unsigned char buffer[4096];        /* Really big buffer... */
       int main_buffer_size;
        ...
        if(..)
             mms = getUShort(inpacket->options[DHO_DHCP_MAX_MESSAGE_SIZE].data);
         if (mms)
              main_buffer_size = mms - DHCP_FIXED_LEN;
        ...
         if (main_buffer_size > sizeof(buffer))
              main_buffer_size = sizeof(buffer);
        ...
```

Włamania do systemów Linux w architekturze x86

- Real-life:

Przyjrzyjmy się ostatniemu błędowi w oprogramowaniu ISC DHCP :)

```
...
option_size = store_options(
            buffer,
            (main_buffer_size - 7 + ((overload & 1) ? DHCP_FILE_LEN : 0)+
                ((overload & 2) ? DHCP_SNAME_LEN : 0)),
            options, priority_list, priority_len, main_buffer_size,
            (main_buffer_size + ((overload & 1) ? DHCP_FILE_LEN : 0)),
            terminate);
/* Put the cookie up front... */
        memcpy(outpacket->options, DHCP_OPTIONS_COOKIE, 4);
        mainbufix = 4;
...
```

Włamania do systemów Linux w architekturze x86

- Real-life:

Przyjrzyjmy się ostatniemu błędowi w oprogramowaniu ISC DHCP :)

```
...
if (option_size <= main_buffer_size - mainbufix) {
        ...
} else {
                outpacket->options[mainbufix++] = DHO_DHCP_OPTION_OVERLOAD;
                outpacket->options[mainbufix++] = 1;
                if (option_size >
                    main_buffer_size - mainbufix + DHCP_FILE_LEN)
                        outpacket->options[mainbufix++] = 3;
                else
                        outpacket->options[mainbufix++] = 1;

                memcpy(&outpacket->options[mainbufix],
                    buffer, main_buffer_size - mainbufix);
...
```

Włamania do systemów Linux w architekturze x86

- Real-life:

Przyjrzyjmy się bardzo ciekawemu błędowi w jądrze systemu Linux na architekturze x86_64:

arch/x86_64/ia32/ia32entry.S:

```
---8<---
        sysenter_do_call:
        cmpl $(IA32_NR_syscalls-1),%eax
        ja ia32_badsys
        IA32_ARG_FIXUP 1
        call *ia32_sys_call_table(,%rax,8)
        ---8<---
        cstar_do_call:
        cmpl $IA32_NR_syscalls-1,%eax
        ja ia32_badsys
        IA32_ARG_FIXUP 1
        call *ia32_sys_call_table(,%rax,8)
        ---8<---
        ia32_do_syscall:
        cmpl $(IA32_NR_syscalls-1),%eax
        ja ia32_badsys
        IA32_ARG_FIXUP
        call *ia32_sys_call_table(,%rax,8) # xxx: rip relative
        ---8<---
```

Włamania do systemów Linux w architekturze x86

- Real-life:

Głupota nie zna granic... a my mamy root'a :)

Włamania do systemów Linux w architekturze x86

- Czas na błędy, które oznaczyłem jako „pozostałe” ;)

NULL Pointers

Włamania do systemów Linux w architekturze x86

- NULL Pointers

- „Nieexploitowalne ponieważ adres 0x00000000 nie jest zamapowany” – NIEPRAWDA!

1. Nie zawsze potrzebujemy by adres był zamapowany (NULL pointer + [offset]).
2. Niektóre systemy standardowo mapuja owy adres.
3. Przemycenie NULL do funkcji, które interpretują owy argument ze specjalnym znaczeniem (np.. Strtok(), get/set timeofday() itp..)

Włamania do systemów Linux w architekturze x86

- NULL Pointers

1. NULL + offset (przykład procmail):

```
void*app_val_(sp,size)
structdyna_array*const sp;
intsize;
{
  if(sp->filled==sp->tspace) /* need to grow ? */
  {
    size_tlen=(sp->tspace+=4)*size;
    sp->vals=sp->vals?realloc(sp->vals,len):malloc(len);
  }
  return &sp->vals[size*sp->filled++];
}
```

Włamania do systemów Linux w architekturze x86

- NULL Pointers

1. NULL – specjalne znaczenie (strtok()):

```
void *funkcja(void *arg) {
…
char *ptr1,*tmp,*last_tok;
…

ptr1 = strdup(jakis_argument);
for (tmp=strtok_r(ptr1,":", &last_tok);tmp!=NULL;tmp=strtok_r(NULL,":",&last_tok)) {
   …
}
}
```

Włamania do systemów Linux w architekturze x86

- NULL Pointers
  - Kernel!

1. Jeżeli jesteśmy w kernel mode, ale z kontekstu użytkownika, adresy w większości przypadków są współdzielone!
2. Wystarczy zamapować pamięć o aresie 0x00000000:

   mmap(0x0,1000 /* długość */,PROT_READ | PROT_WRITE, MAP_FIXED |
   MAP_ANONYMOUS | MAP_PRIVATE, 0, 0);
3. Błędy w jądrze stają się exploitowalne! – przykład CVE: CVE-2007-1000 BID: 22904 (Linux Kernel IPV6_Getsockopt_Sticky Memory Leak)
4. Pierwszy exploit wydany przez 8lgm z 1994 roku (chown()) :)

Włamania do systemów Linux w architekturze x86

- NULL Pointers
  - Przykład kontekstowy :)

```
#define TEST 0xdeadbabe

int main(int argc, char *argv[]) {
...
void *ptr;
...
  if ( (ptr = mmap(0x0, 1000, PROT_READ | PROT_WRITE, MAP_FIXED | MAP_ANONYMOUS | MAP_PRIVATE, 0, 0)) != NULL)
        {
    printf("Kupa kwika!\n");
    exit(-1);
  }
...
  ptr = (char *) ( (char *) ptr);
  *((int *)ptr) = TEST;
  *((int *)ptr+1) = TEST;
  *((int *)ptr+2) = TEST;
  *((int *)ptr+3) = TEST;
  *((int *)ptr+4) = TEST;
...
}
```

Włamania do systemów Linux w architekturze x86

- NULL Pointers
  - Przykład kontekstowy :)

```
[root@lost-coder null-pointer]# gdb -q ./mmap
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) b execle
Function "execle" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (execle) pending.
(gdb) r a
Starting program: /root/projekty/null-pointer/mmap a
Breakpoint 2 at 0x400aa6f0
Pending breakpoint "execle" resolved
Before...

Breakpoint 2, 0x400aa6f0 in execle () from /lib/libc.so.6
(gdb) x/x 0x0
0x0:    0xdeadbabe
(gdb) x/x 0x0+4
0x4:    0xdeadbabe
(gdb) x/x 0x0+4+4
0x8:    0xdeadbabe
(gdb)
```

Włamania do systemów Linux w architekturze x86

- Czas na błędy, które oznaczyłem jako „pozostałe” ;)

Stack exhaustion (rekurencje)

Włamania do systemów Linux w architekturze x86

- Czas na błędy, które oznaczyłem jako „pozostałe” ;)

  Shifting the Stackpointer (zmienne lokalne)

Włamania do systemów Linux w architekturze x86

- Shifting the Stackpointer

```
char *func(int len, char *arg) {
   …
   char buf[len];
   …
}
```

Włamania do systemów Linux w architekturze x86

- Czas na błędy, które oznaczyłem jako „pozostałe” ;)

Pisanie do stderr

Włamania do systemów Linux w architekturze x86

- Stderr:

1. Wiele bibliotek wysyła informacje o błędach do strumienia stderr
2. Co się stanie gdy zamienimy deskryptor o numerze 2?
3. … a w późniejszym czasie zostanie otwarty plik w trybie Read/Write np. /etc/passwd?

Włamania do systemów Linux w architekturze x86

- Zabezpieczenia:
- Non-Executable Stack (pierwsza idea w sierpniu 1996 roku – DEC's Digital Unix on Alpha. Nx-stacks dla systemow Solaris 2.4/2.5/2.5.1 – 19 listopada 1996. Pierwsza implementacja dla Linux – Solar Designer 12 kwiecień 1997)
- W^X (W xor X – OpenBSD, Theo Dde Raadt. Pierwsza idea 1972 ☺)
- AAAS (ASCII Armored Address Space)
- ASLR

Grsecurity, OpenWall, itd……

Włamania do systemów Linux w architekturze x86

- Obejścia:

- Ret2data (inne bufory)
- Ret2libc
- Ret2strcpy
- Ret2gets
- Ret2code
- Chained ret2code (pop-pop-pop-…-pop)
- Ret2syscall
- Ret2text
- Ret2plt (Procedure Linkage Table)
- Ret2dl-resolve
- Mutacje… (W^X to W+X; W^X to X after W)

Włamania do systemów Linux w architekturze x86

- Obejścia – przykłady z .text:

Adresowanie argumentów – glibc 2.3.3:

System:
<system+17>: mov 0x8(%ebp),%esi ; refers %ebp + 8

Execve:
<execve+9>: mov 0xc(%ebp),%ecx ; second argument of execve()
<execve+27>: mov 0x10(%ebp),%edx ; third argumet of execve()
<execve+30>: mov 0x8(%ebp),%edi ; first argument of execve()

Obecnie – glibc 2.3.6:
Execve:
0x400aa551 <execve+17>: mov %edi,0x8(%esp)
0x400aa555 <execve+21>: mov 0xfffffec4(%ebx),%eax
0x400aa55b <execve+27>: mov 0x10(%esp),%edi
0x400aa55f <execve+31>: mov %esi,0x4(%esp)

System:
0x40058257 <system+7>: mov 0x14(%esp),%esi

Włamania do systemów Linux w architekturze x86

- **Obejścia – przykłady z .text (chained ret2code):**

```
0x400385be <gnu_get_libc_version+126>:  add    $0x8c,%esp
0x400385c4 <gnu_get_libc_version+132>:  pop    %ebx
0x400385c5 <gnu_get_libc_version+133>:  pop    %esi
0x400385c6 <gnu_get_libc_version+134>:  pop    %edi
0x400385c7 <gnu_get_libc_version+135>:  pop    %ebp
0x400385c8 <gnu_get_libc_version+136>:  ret

0x804847c <__libc_csu_init+76>: add    $0xc,%esp
0x804847f <__libc_csu_init+79>: pop    %ebx
0x8048480 <__libc_csu_init+80>: pop    %esi
0x8048481 <__libc_csu_init+81>: pop    %edi
0x8048482 <__libc_csu_init+82>: pop    %ebp
0x8048483 <__libc_csu_init+83>: ret

0x804851c <__do_global_ctors_aux+44>:   pop    %eax
0x804851d <__do_global_ctors_aux+45>:   pop    %ebx
0x804851e <__do_global_ctors_aux+46>:   pop    %ebp
0x804851f <__do_global_ctors_aux+47>:   ret
```

Włamania do systemów Linux w architekturze x86

- Obejścia – przykłady z .text (chained ret2code – continue):

```
0x40038770 <__errno_location+192>:    add   $0x30,%esp
0x40038773 <__errno_location+195>:    pop   %esi
0x40038774 <__errno_location+196>:    pop   %edi
0x40038775 <__errno_location+197>:    pop   %ebp
0x40038776 <__errno_location+198>:    ret

0x40038ac7 <__umoddi3+183>:    add   $0x8,%esp
0x40038aca <__umoddi3+186>:    pop   %ebx
0x40038acb <__umoddi3+187>:    pop   %esi
0x40038acc <__umoddi3+188>:    pop   %edi
0x40038acd <__umoddi3+189>:    pop   %ebp
0x40038ace <__umoddi3+190>:    ret

0x40038c12 <iconv_open+258>:   pop   %ebx
0x40038c13 <iconv_open+259>:   pop   %esi
0x40038c14 <iconv_open+260>:   pop   %edi
0x40038c15 <iconv_open+261>:   pop   %ebp
0x40038c16 <iconv_open+262>:   ret
```

Włamania do systemów Linux w architekturze x86

- Obejścia – przykłady z .text (chained ret2syscall – praktyka):

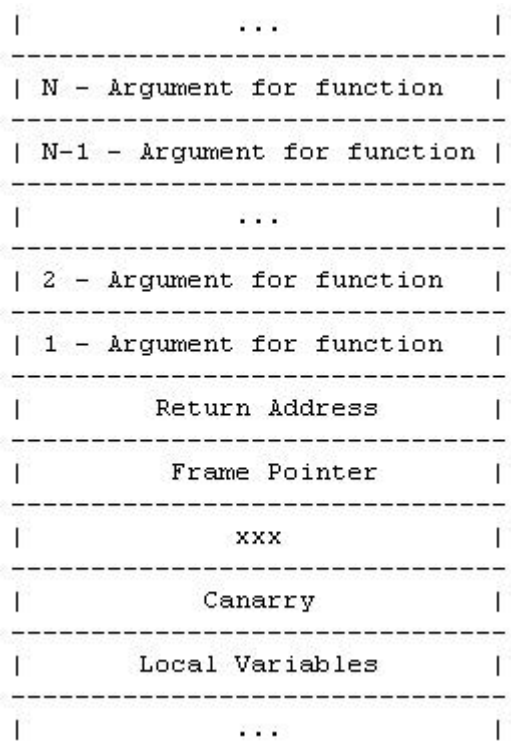
```
0x40113f72 <__libc_freeres+194>:      pop    %eax
0x40113f73 <__libc_freeres+195>:      pop    %ebx
0x40113f74 <__libc_freeres+196>:      pop    %ebp
0x40113f75 <__libc_freeres+197>:      ret

0x4004b8c7 <setjmp+55>: pop    %ecx
0x4004b8c8 <setjmp+56>: pop    %edx
0x4004b8c9 <setjmp+57>: ret

0x40041142 <__gconv_get_cache+98>:     int    $0x80
```

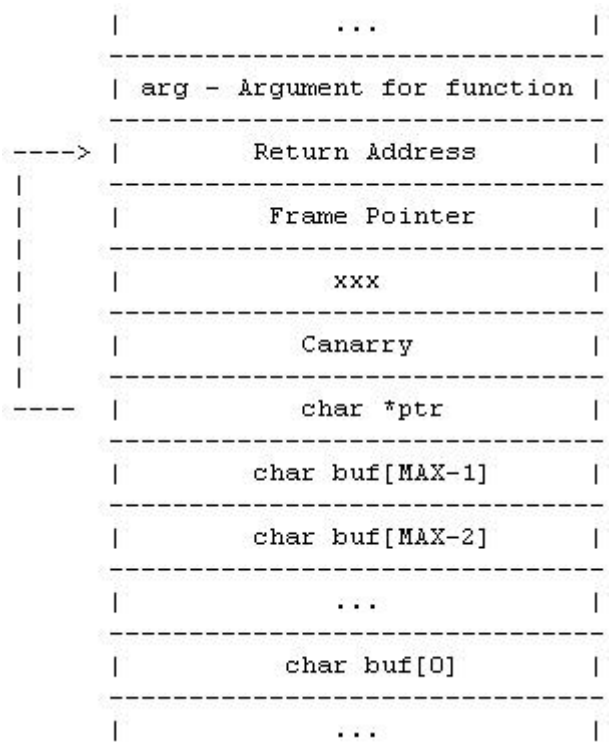
Włamania do systemów Linux w architekturze x86

- „Lepsze” zabezpieczenia – kanarek śmierci:
- StackGuard



Włamania do systemów Linux w architekturze x86

- „Lepsze” zabezpieczenia – kanarek śmierci:
- Bulba i Kil3r :)



Włamania do systemów Linux w architekturze x86

- „Lepsze” zabezpieczenia – kanarek śmierci:
- Pro-Police (SSP – Stack Smashing Protector)
  - Inteligentne rozkladanie zmiennych na stosie….
  - Kopiuje argumenty do funkcji
  - Tworzy tzw. Ramkę chronioną
- Znane „uchybienia”:
  - Ramki o dużej ilości buforów mogą być nadpisywane – możliwość nadpisania zmiennych sterujących
  - Struktury oraz klasy nie są rozdzielane, czyli chronione
  - Funkcje ze zmienna ilością argumentów muszą być wrzucone w strefę niechronioną
  - Wirtualne funkcje również w strefie niechronionej

Włamania do systemów Linux w architekturze x86

- „Lepsze” zabezpieczenia – kanarek śmierci:
- Pro-Police (SSP – Stack Smashing Protector)
  - Jednym słowem nie jest zbyt miło…

    .....
    .....
    .....
    .....
    .....
  - Jednak spójrzmy na test :)

Włamania do systemów Linux w architekturze x86

- Przyjżyjmy się plikom ELF:

```
0x08048114->0x08048127 at 0x00000114: .interp ALLOC LOAD READONLY DATA HAS_CONTENTS
0x08048128->0x08048148 at 0x00000128: .note.ABI-tag ALLOC LOAD READONLY DATA HAS_CONTENTS
0x08048148->0x0804817c at 0x00000148: .hash ALLOC LOAD READONLY DATA HAS_CONTENTS
0x0804817c->0x080481fc at 0x0000017c: .dynsym ALLOC LOAD READONLY DATA HAS_CONTENTS
0x080481fc->0x0804826a at 0x000001fc: .dynstr ALLOC LOAD READONLY DATA HAS_CONTENTS
0x0804826a->0x0804827a at 0x0000026a: .gnu.version ALLOC LOAD READONLY DATA HAS_CONTENTS
0x0804827c->0x0804829c at 0x0000027c: .gnu.version_r ALLOC LOAD READONLY DATA HAS_CONTENTS
0x0804829c->0x080482a4 at 0x0000029c: .rel.dyn ALLOC LOAD READONLY DATA HAS_CONTENTS
0x080482a4->0x080482c4 at 0x000002a4: .rel.plt ALLOC LOAD READONLY DATA HAS_CONTENTS
0x080482d0->0x080482e7 at 0x000002d0: .init ALLOC LOAD READONLY CODE HAS_CONTENTS
0x080482e8->0x08048338 at 0x000002e8: .plt ALLOC LOAD READONLY CODE HAS_CONTENTS
0x08048340->0x08048560 at 0x00000340: .text ALLOC LOAD READONLY CODE HAS_CONTENTS
0x08048560->0x0804857b at 0x00000560: .fini ALLOC LOAD READONLY CODE HAS_CONTENTS
0x0804857c->0x08048590 at 0x0000057c: .rodata ALLOC LOAD READONLY DATA HAS_CONTENTS
0x08048590->0x08048594 at 0x00000590: .eh_frame ALLOC LOAD READONLY DATA HAS_CONTENTS
0x08049594->0x0804959c at 0x00000594: .ctors ALLOC LOAD DATA HAS_CONTENTS
0x0804959c->0x080495a4 at 0x0000059c: .dtors ALLOC LOAD DATA HAS_CONTENTS
0x080495a4->0x080495a8 at 0x000005a4: .jcr ALLOC LOAD DATA HAS_CONTENTS
0x080495a8->0x08049670 at 0x000005a8: .dynamic ALLOC LOAD DATA HAS_CONTENTS
0x08049670->0x08049674 at 0x00000670: .got ALLOC LOAD DATA HAS_CONTENTS
0x08049674->0x08049690 at 0x00000674: .got.plt ALLOC LOAD DATA HAS_CONTENTS
0x08049690->0x0804969c at 0x00000690: .data ALLOC LOAD DATA HAS_CONTENTS
0x0804969c->0x080496a0 at 0x0000069c: .bss ALLOC
```

Włamania do systemów Linux w architekturze x86

- IP spoofing:
- słynny atak wykorzystany przez Kevina Mitnicka w celu dostania się do komputera Tsutomu Shimomury
- Musimy znać:
  - SQN number
  - Port źródłowy
  - Port docelowy
- Czy aby na pewno atak „martwy” ? (newtcp – lcamtuf)
- Zagrożenia (SMTP, DNS, POP3, IMAP, FTP, RPC, The X Window System, R services, HTTP responses, Bypass firewall auth)
- Obejrzyjmy test… (oczywiście jeżeli będę miał dostęp do neta :))

Włamania do systemów Linux w architekturze x86

- Dziękuję za uwagę

- Adam Zabrocki
- http://pi3.hack.pl
- pi3@itsec.pl (adam.zabrocki@avet.com.pl)

Włamania do systemów Linux w architekturze x86

- Pytania ???

Włamania do systemów Linux w architekturze x86