Security in Development Environment

----Break The Linker

Keji Yu，Wei Zhao Venustech AD-Lab

X'con 2005

Foreword

- This speech is about the linker vulnerability we found in program development environment .
- This is the problem that we might be ignored. Have you ever thinking about the security problem which we might come across in our daily using development environment?

Why we need to study the develop environment security?

- **Why:**

  We found many security problems in develop environment, but we didn’t focus on this problem seriously. The software which offer its source code are considered as security ones, no one have thought that he will be attacked by the compiling process or linking process.

Environment

- **Commonly the program develop environment might include （mainly as Intel x86 under Windows and Linux）:**
  - **Compiler: CL, gcc**
  - **Linker: Link, ld**
  - **Debugger: VC debugger, GDB**
  - **Etc.**

Emphases

- Linker security is the important part of this speech.
- What is the linker? The function of the linker
- The vulnerabilities of linkers (Both ld and link)

Linker

- In order to study this vulnerability deeply, let's describe the linker process first.
- Linker's general working process

Under Linux

- **May be we all have this kind of experience:**

  gcc crashed when we compiling program.
- **One reason: The vulnerabilities in Binary File Descriptor(bfd) library**

ld linking process

- ld brief introduction:Usually the last step in compiling a program is to run ld.
- ld linking process

BFD library

- **BFD: The BFD library provides a uniform method of accessing a variety of object file formats.**
- **BFD working process: When an object file is opened, BFD subroutines automatically determine the format of the input object file. They then build a descriptor in memory with pointers to routines that will be used to access elements of the object file's data structures.**

Elf file format brief introduction 1 ELF file head

- ELF file head

char magic[4] = "\177ELF";// magic number
char class; // address size， 1 = 32 bit， 2 = 64 bit
char byteorder; // 1 = little-endian， 2 = big-endian
char hversion; // header version， always 1
char pad[9];
short filetype; // file type: 1 = relocatable， 2 = executable，
    // 3 = shared object， 4 = core image
short archtype; // 2 = SPARC， 3 = x86， 4 = 68K， etc.
int fversion; // file version， always 1
int entry; // entry point if executable

Elf file format brief introduction 2 ELF file head

Continued

int phdrpos; // file position of program header or 0
int shdrpos; // file position of section header or 0
int flags; // architecture specific flags， usually 0
short hdrsize; // size of this ELF header
short phdrent; // size of an entry in program header
short phdrcnt; // number of entries in program header or 0
short shdrent; // size of an entry in section header
short shdrcnt; // number of entries in section header or 0
short strsec; // section number that contains section name strings

Elf file format brief introduction 3 Section head

- **Section head**

int sh_name; // name， index into the string table
int sh_type; // section type
int sh_flags; // flag bits， below
int sh_addr; // base memory address， if loadable， or zero
int sh_offset; // file position of beginning of section
int sh_size; // size in bytes
int sh_link; // section number with related info or zero
int sh_info; // more section-specific info
int sh_align; // alignment granularity if section is moved
int sh_entsize; // size of entries if section is an array

Bfd vulnerability background

- We found the ld crash since 2004, but we ignored it. (Do not ignore any details around youJ)
- June 1st 2005, Gentoo code auditing team published this vulnerability.

bfd elf vulnerability 1

- **A crafted elf file head**

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  02 00 03 00 01 00 00 00  94 80 04 08 34 00 00 00  |............4...|
00000020  26 00 00 00 00 00 41 41  41 41 41 41 41 41 41 41  |&.....AAAAAAAAAA|
00000030  00 00 41 41 41 41 41 41  41 41 00 00 00 40 41 41  |..AAAAAAAA...@AA|
00000040  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41  |AAAAAAAAAAAAAAAA|
```

bfd elf vulnerability 2

- elf_object_p() function in Elfcode.h:

```
bfd_set_start_address (abfd, i_ehdrp->e_entry);
 if (i_ehdrp->e_shoff != 0)
   {
     if (bfd_seek (abfd, (file_ptr) i_ehdrp->e_shoff, SEEK_SET) != 0)  //get shoff and find
     section header table
     goto got_no_match;
     if (bfd_bread (&x_shdr, sizeof x_shdr, abfd) != sizeof (x_shdr))// read the first section
     head
     goto got_no_match;
     elf_swap_shdr_in (abfd, &x_shdr, &i_shdr); // translate to internal formate
     if (i_ehdrp->e_shnum == SHN_UNDEF) // if section number is 0, use the sh_size
     i_ehdrp->e_shnum = i_shdr.sh_size;
     if (i_ehdrp->e_shstrndx == SHN_XINDEX)
     i_ehdrp->e_shstrndx = i_shdr.sh_link;
   }
```

bfd analyze elf vulnerability 3

- Contioued

```
/* allocate memory for section header table in internal format*/
 if (i_ehdrp->e_shnum != 0)
    {
      Elf_Internal_Shdr *shdrp;
      unsigned int num_sec;
      amt = sizeof (*i_shdrp) * i_ehdrp->e_shnum; // i_shdr.sh_size
      i_shdrp = bfd_alloc (abfd, amt); // allocate amt size memory
      if (!i_shdrp)
      goto got_no_match;
      num_sec = i_ehdrp->e_shnum; // i_shdr.sh_size
      if (num_sec > SHN_LORESERVE)
      num_sec += SHN_HIRESERVE + 1 - SHN_LORESERVE; //0xffff-0xff00+1
      elf_numsections (abfd) = num_sec;
      amt = sizeof (i_shdrp) * num_sec; // integer overflow
      elf_elfsections (abfd) = bfd_alloc (abfd, amt); // allocated wrong size and lead overflow
```

Threat

- Lot of program use bfd lib: objdump, gdb,nm,size,ar…etc.
- It’s not linux kernel bug
- May be used to attack programmersJ

Summary

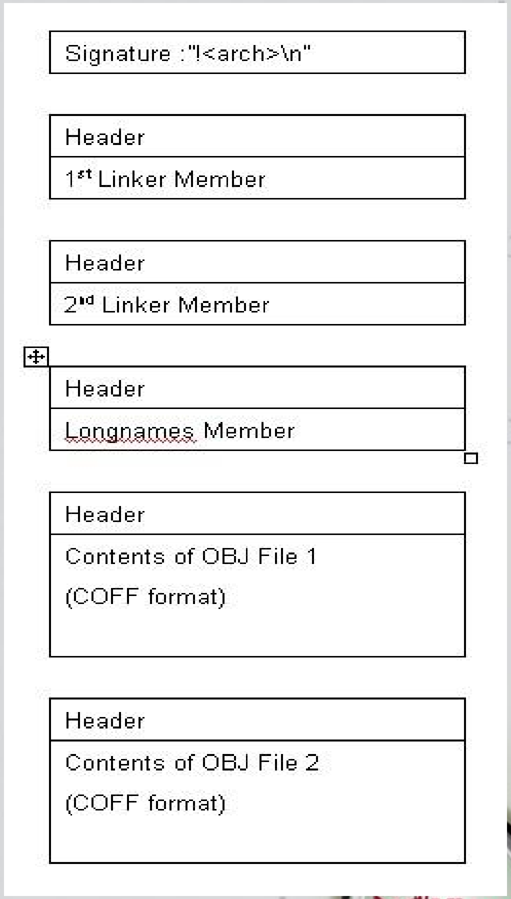
- Reliable linkers never crash J
- This is a typical integer overflow attack
- Patch suggestion
  - Gentoo 's patch
  - GNU 's patch

Windows platform

- There is an unreleased vulnerability exit in VC 6 (discovered by Keji)
- Similar with the ld problem
- Next, a brief introduction about lib file format

Lib format

- The differences between Lib file and obj file
- The format of Lib file
  - Signature
  - Header
  - First section



Lib file format brief introduction

- **First section :**

```
typedef struct {
        unsigned long SymbolNum;
    //symbol number in lib
        unsigned long SymbolOffset[n];    //
    symbol offset
        char StrTable[m];                   // symbol
    name table
     }FirstSec;
```

Link.exe vulnerability

- A craft library file, the symbol num is at offset 0x44

```
000000  21 3C 61 72 63 68 3E 0A 2F 20 20 20 20 20 20 20  !<arch>./
000010  20 20 20 20 20 20 20 20 31 31 30 33 30 30 35 35          11030055
000020  34 35 20 20 20 20 20 20 20 3C 20 20 20 37 20 20  45       <   7
000030  30 20 20 20 20 20 20 20 32 34 33 20 20 20 20 20  0       243
000040  20 20 60 0A 40 00 00 07 00 00 02 62 00 00 02 62    `.@......b...b
000050  00 00 02 62 00 00 02 62 00 00 02 62 00 00 02 62  ...b...b...b...b
000060  00 00 02 62 3F 3F 5F 43 40 5F 30 42 41 43 48
    40  ...b??_C@_0BACH@
000070  4A 4B 49 43 40 43 43 AD 43 43 43 43 43 43 BD 43
    JKIC@CC.CCCCCC.C
000080  43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
    CCCCCCCCCCCCCCCC
000090  43 43 43 43 43 40 00 3F 3F 5F 5B 40 5F C5 48 41
    CCCCC@.??_[@_.HA
0000A0  48 40 45 42 42 43 40 41 41 41 55 41 41 41 41 41
    H@EBBC@AAAUAAAAA
```

Link.exe vulnerability

- **LINK.EXE allocated sybolnum*4 size memory for saving the symbol string table:**

```
0045FB54   mov        edx,[ebx+0x18] ; [EBX+18]
   symbolnum
0045FB57   shl        edx,2                  ; symbolnum * 4
0045FB5A   push       edx
0045FB5B   call       00451B20               ; allocate
   memory(malloc)
0045FB60   mov        edx,[ebx+0x18]
0045FB63   xor        ecx,ecx
0045FB65   mov        [ebx+0x28],eax ;save allocated
   memory address
```

Link.exe vulnerability

- Calculate the symbol name string address, and save to allocated memory:

```
0045FB6F   mov      edx,[ebx+0x28]         ; allocated memory address
0045FB72   mov      [edx+ecx*4],eax        ;save the symbol name
0045FB75   mov      dl,[eax]
0045FB77   inc      eax
0045FB78   test     dl,dl
0045FB7A   jz       0045FB83
0045FB7C   mov      dl,[eax]               ; next symbol name string
0045FB7E   inc      eax
0045FB7F   test     dl,dl
0045FB81   jnz      0045FB7C
0045FB83   mov      edx,[ebx+0x18]         ;[ebx+0x18] is symbol number
0045FB86   inc      ecx
0045FB87   cmp      ecx,edx                ;if not finish loop
0045FB89   jb       0045FB6F
```

Write above code in C

- This is a very “clear” vulnerability, are there any vulnerabilities like this?
- Write above code in C：

```
DWORD* pTable = (DWORD *)malloc( SymbolNum *
    4);
//integer overflow here
for(int i=0; i<SymbolNum; i++)
{
    pTable[i] = symbol name string address;
}
```

New bug in bfd

- We found another vulnerability in library format but in bfd library.
- It's a similar vulnerability as the VC one.
- The vulnerability is in Archive.c in bfd lib.

New bug in bfd

- In the do_slurp_coff_armap funciton:

```
static bfd_boolean
do_slurp_coff_armap (bfd *abfd){
…

  carsym_size = (nsymz * sizeof (carsym));  //used the nsymz from file
  ptrsize = (4 * nsymz);  //integer overflow here
…
/* Allocate and read in the raw offsets.  */
  raw_armap = bfd_alloc (abfd, ptrsize);  // allocate wrong memory size here
  if (raw_armap == NULL)
    goto release_symdefs;
…
}
```

Threat

- Programmer is still the first target, whether there’s anyone do not write program but use vcJ?
- Similar with other file format vulnerability, need the method to deliver it.
- Hiding in some open source code (New slogan: Do not compile stranger’s code)

Doubt

- **Doubt :Why this kind of problem always comes out?**
  - **Whether we have ignored some thing?**
  - **Why it is the int overflow vulnerability?**
  - **What's the essence of this kind of vulnerability?**
  - **Still exist this kind of vulnerability?**

Real reason

- Usually the security tips tell us: Do not trust user's input
- We extend it: Do not trust user's input, do not trust file's input as well, because file is the user's input either.
  - Include the develop tool's configuration files
  - Include the develop tool's project files
  - Include the develop tool's makefile files
  - All these files might hide malicious program

Security tips

- **Two new security tips:**
  - **Do not trust user's input, include file's input.**
  - **Do not compile stranger's source or similar engineering file, they are not safe.**
- **Hope you guys could do more extension.**

Summary

- This is a new area, can we meet new problem in future?
- Some questions about this:
  - What's the impact of vulnerability in development environment?
  - What kind of threat to the programmers will caused by these security problems?
  - How many attack types this will bring? Use this in injecting backdoor?

Q/A

Thanks

安全源自未雨绸缪

Venus Info Tec Inc.

Security

Trusted {Solution} Provider

Services