《软件安全漏洞分析与发现》

软件漏洞机理分析

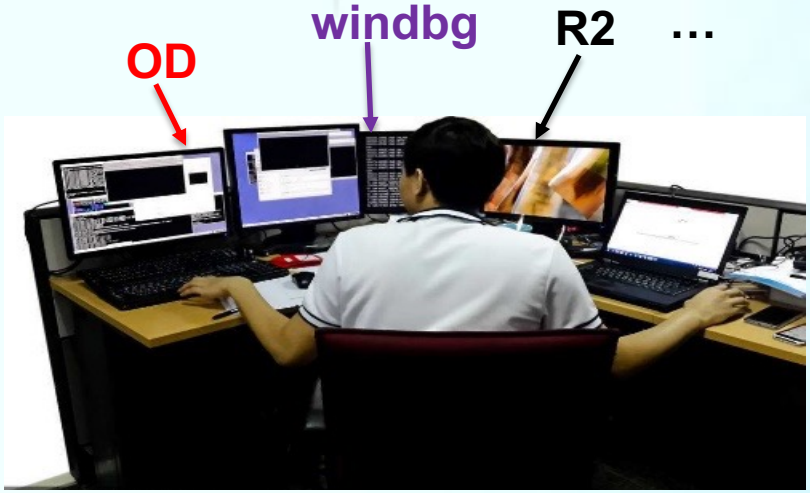
闫佳
中国科学院软件研究所
2024年5月15日

2.0 漏洞机理分析概要

- 漏洞分析
  - 软件漏洞机理分析目标
    - 了解漏洞形成原因
    - 评估漏洞可利用性
    - 为修复漏洞提供支撑
    - 指导同类型未知漏洞挖掘
  - 控制流劫持类漏洞分析
    - 漏洞脆弱点
    - 脆弱路径
    - 内存布局

2.0 漏洞机理分析概要

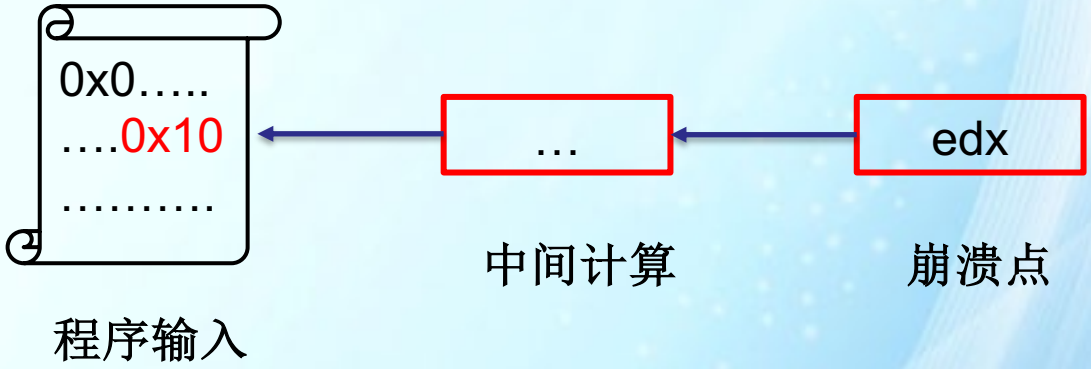
- 面临的挑战：
  - 需要面对的软件规模：10^9 LOC
  - 真正发生问题的代码：10^1 LOC
    - 缓冲区溢出：MemCopy(len) 漏检测
    - 释放重用：ObjRef 漏加减



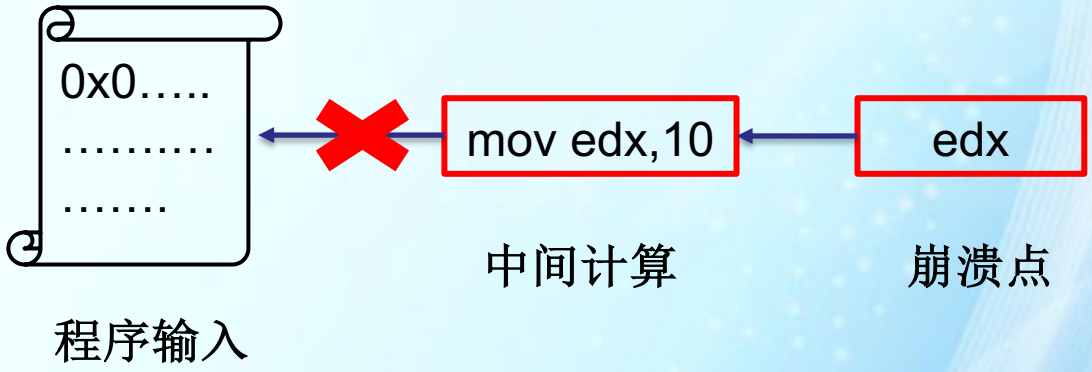
determining the cause of a crash. Crash analysis is a laborious
normally take anywhere from a few hours to a week per crash.

Crash analysis with BitBlaze, BlackHat, 2010

- 软件漏洞机理分析
  - 污点逆向回溯分析
    - 崩溃点： mov [edx+0x4], eax　　(edx = 0x10)
    - !exploitable：访问非法地址，但针对非法地址的形成过程，无法进一步给出结论

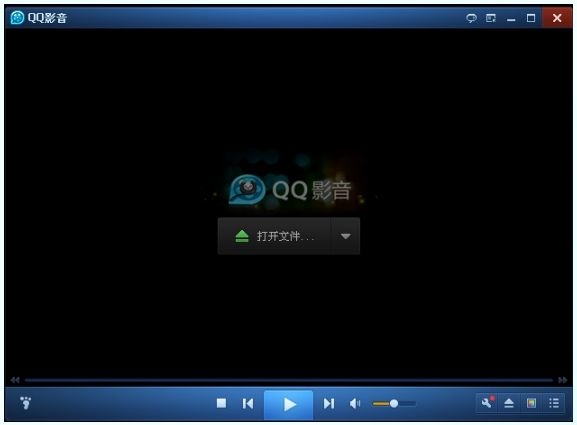


- 软件漏洞机理分析
  - 污点逆向回溯分析
    - 崩溃点： mov [edx+0x4], eax　　(edx = 0x10)
    - !exploitable：访问非法地址，但针对非法地址的形成过程，无法进一步给出结论



2.0 漏洞机理分析概要

- 软件漏洞机理分析
  - 基于逆向污点分析的堆溢出漏洞类型推断：
    - 某资深漏洞分析专家：2-3天，手工+windbg
    - 拥有逆向污点分析的博士生：30分钟，其中25分钟搭建漏洞验证环境，5分钟分析结果




```
windhl@windhl-u16:/data/exheaper/qqplayer/1$ aread DSR_1_238990188_0
| eip:0x0377197c | id:238990186 | mov ebx, [ebp+0x8]          | [
| eip:0x03514248 | id:238990178 | push ecx                    | [
| eip:0x03513e02 | id:238990172 | mov ecx, [esi]              | [
| eip:0x03771b33 | id:238989915 | rep movsd                   | [
| eip:0x03771b33 | id:238989914 | rep movsd                   | [
| eip:0x0377102c | id:238840914 | rep stosd                   | [
| eip:0x0377102c | id:238840913 | rep stosd                   | [
| eip:0x03771025 | id:238840100 | xor eax, eax                | [
[i] it takes time: 1
```

QQPlayer 堆溢出0Day漏洞

2.1 软件漏洞脆弱点分析

- 栈溢出漏洞脆弱点分析
  - 对于栈溢出漏洞，其脆弱点是覆盖栈空间的函数或指令，通常情况下是strcpy函数或movs指令

```
#include<stdio.h>
#include<string.h>
void overflow(char *buf)
{
    char des[5]="";
    strcpy(des,buf);
    return;
}

void main(int argc,char *argc[])
{
    LoadLibrary("user32.dll");
    char longbuf[100]="aaaaaaaaaaaabbbbccccccccccc";
    overflow(longbuf);
    return;
}
```

2.1 软件漏洞脆弱点分析

- 栈溢出漏洞脆弱点分析
  - 在该段代码中，漏洞脆弱点是overflow函数，由于该函数调用strcpy向缓冲区中写入了超长的数据，导致程序漏洞被触发

```
.text:00401000          push    ebp
.text:00401001          mov     ebp, esp
.text:00401003          sub     esp, 0Ch
.text:00401006          mov     eax, ___security_cookie
.text:0040100B          xor     eax, ebp
.text:0040100D          mov     [ebp+var_4], eax
.text:00401010          mov     al, byte_4120E0
.text:00401015          mov     [ebp+var_C], al
.text:00401018          xor     ecx, ecx
.text:0040101A          mov     [ebp+var_B], ecx
.text:0040101D          mov     edx, [ebp+arg_0]
.text:00401020          push    edx             ; char *
.text:00401021          lea     eax, [ebp+var_C]
.text:00401024          push    eax             ; char *
.text:00401025          call    _strcpy
.text:0040102A          add     esp, 8
.text:0040102D          mov     ecx, [ebp+var_4]
.text:00401030          xor     ecx, ebp
```

- 整数溢出脆弱点分析
  - 整数溢出的脆弱点分析

```
.text:00401000 var_8         = dword ptr -8
.text:00401000 var_4         = dword ptr -4
.text:00401000 arg_0         = dword ptr  8
.text:00401000 arg_4         = dword ptr  0Ch
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 sub     esp, 8
.text:00401006                 push    esi
.text:00401007                 mov     eax, [ebp+arg_4]
.text:0040100A                 shl     eax, 2
.text:0040100D                 push    eax             ; size_t
.text:0040100E                 call    _malloc
.text:00401013                 add     esp, 4
.text:00401016                 mov     [ebp+var_8], eax
.text:00401019                 cmp     [ebp+var_8], 0
.text:0040101D                 jnz     short loc_401024
.text:0040101F                 or      eax, 0FFFFFFFFh
.text:00401022                 jmp     short loc_401055
.text:00401024                 mov     [ebp+var_4], 0
```

左移2位，截断后
直接申请内存

2.1 软件漏洞脆弱点分析

- 整数溢出脆弱点分析

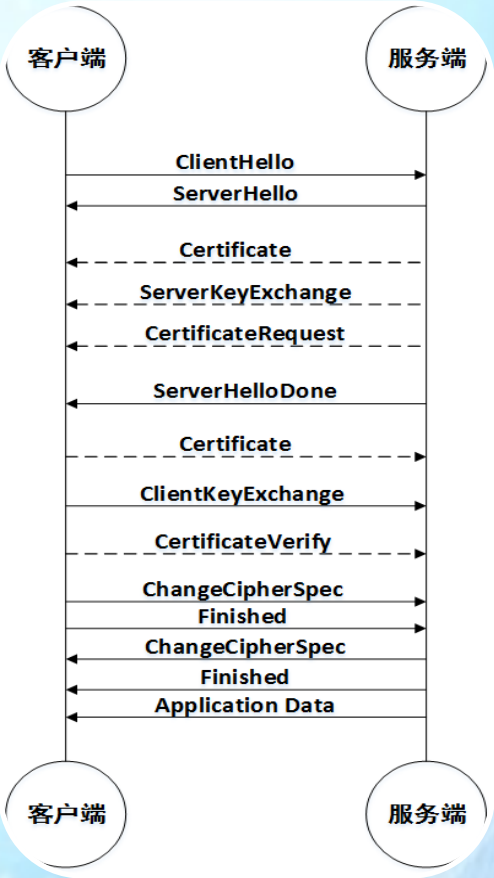
```
.text:0040102B    jmp     short loc_401036
.text:0040102D    mov     ecx, [ebp+var_4]
.text:00401030    add     ecx, 1
.text:00401033    mov     [ebp+var_4], ecx
.text:00401036    mov     edx, [ebp+var_4]
.text:00401039    cmp     edx, [ebp+arg_4]
.text:0040103C    jge     short loc_401052
.text:0040103E    mov     eax, [ebp+var_4]
.text:00401041    mov     ecx, [ebp+var_8]
.text:00401044    mov     edx, [ebp+var_4]
.text:00401047    mov     esi, [ebp+arg_0]
.text:0040104A    mov     edx, [esi+edx*4]
.text:0040104D    mov     [ecx+eax*4], edx
.text:00401050    jmp     short loc_40102D
.text:00401052    mov     eax, [ebp+var_8]
.text:00401055    pop     esi
.text:00401056    mov     esp, ebp
.text:00401058    pop     ebp
.text:00401059    retn
```

此处的长度引用了输入参数，而非截取后的值

2.1 软件漏洞脆弱点分析

- 心脏滴血漏洞脆弱点分析
  - 漏洞软件OPENSSL库：
    - 客户端需要不断的发送心跳信息到服务器，以确保服务器是可用的
  - 心跳握手过程
    - 客户端发送一段固定长度的字符串到服务器，服务器接收后，返回该固定长度的字符串
    - 客户端发送消息A到服务器，服务器接受后，原样返回A，客户端根据返回的数据判定openssl服务器是可用的

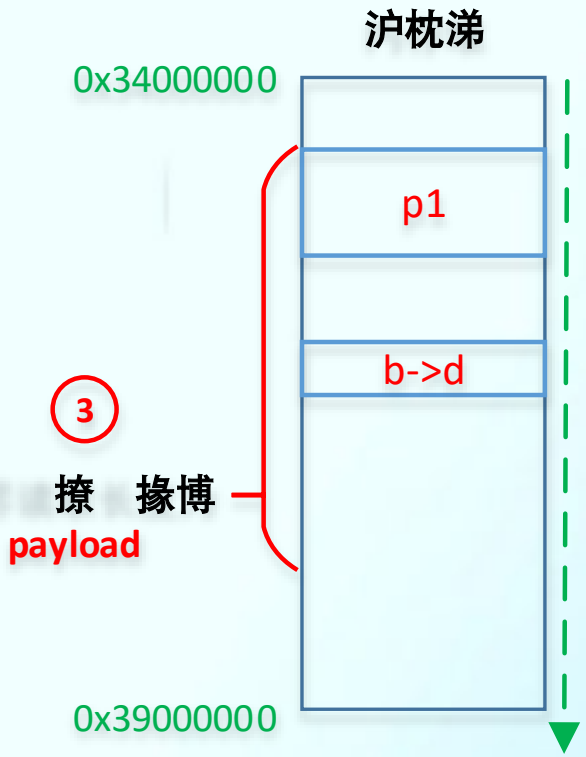
```
struct hb {
        int type;
        int length;
        unsigned char
*data;
};
```



最快10秒
泄露服务器根证书私钥

2.1 软件漏洞脆弱点分析

- 心脏滴血漏洞脆弱点分析
  - 心脏滴血漏洞其脆弱点在于数据拷贝的函数
  - 心脏滴血没有控制流劫持点




```
bn_div.c: bn_expand2
...
443: if(words > b->dmax)
...
446:    if(!a) return NULL;
447:    if(b->d)  OPENSSL_free(b->d);  (1)
448:    b->d=a;
...
```

```
d1_both.c: dtls1_process_heartbeat
...
1352: buffer = OPENSSL_malloc(1+2+payload+padding);
1353: bp = buffer;
...
1358: memcpy(bp, p1 , payload);  (2)
1358: r = dtls1_write_bytes(s,TLS1_RT_HEARTBEAT,buffer...)
...
```

2.2 软件漏洞路径分析

- 软件漏洞路径分析
  - 目标

软件漏洞路径分析的目标是提取程序由数据输入点或程序执行入口到漏洞脆弱点的路径，包括抽取相关指令序列、内存上下文数据等信息，为漏洞成因分析、漏洞利用生成等工作提供支撑

  - 分析方法
    - 基于调试工具的分析
      - 通常情况下，针对软件漏洞路径的分析主要基于Windbg(Time Travel Debug)或Ollydbg、Mozilla RR(Record and Replay)手工开展，需要大量的人力和物力
    - 基于动态污点传播的分析
      - 追踪输入数据在漏洞程序中的处理过程，大致包括污点源标记、污点传播计算、污点传播过程回溯等阶段

2.2 软件漏洞路径分析

- 软件漏洞路径分析方法
  - 主要方法
    - 污点源引入
      - 漏洞与数据数据相关，标记输入数据作为分析的起点
    - 污点传播规则设计
      - 根据指令的语义，制订规则进行污点传播计算
    - 传播终止条件确定
      - 当输入数据的操作方式满足漏洞条件时，停止污点传播
    - 污点传播回溯分析
      - 以引发漏洞的关键指令为起点，确定触发漏洞的关键数据在输入数据中的位置和长度

2.2 软件漏洞路径分析

- 软件漏洞路径分析
  - 针对漏洞的污点源引入
    - 文件污点源引入
      - Ring3级文件读写API：ReadFile
      - Ring3文件映射操作API：MapViewOfFile
      - Ring0级文件读写API：NtReadFile、ZwReadFile
      - Ring0级文件操作API：DeviceIoControl
    - 网络污点源引入
      - Ring3级网络API：Recv，WSARecv
      - Ring0级网络API：DeviceIoControl
    - 注册表污点源引入
      - Ring3级注册表API：RtlQueryRegistryValues
      - Ring0级注册表API：CMRegister***

2.2 软件漏洞路径分析

- 污点传播规则
  - 将指令分组，针对不同的指令组制定传播规则
  - 对于浮点指令，需要在污点传播计算时考虑其实际值
  - 针对其他特殊指令，需要根据针对的问题和分析过程改变规则

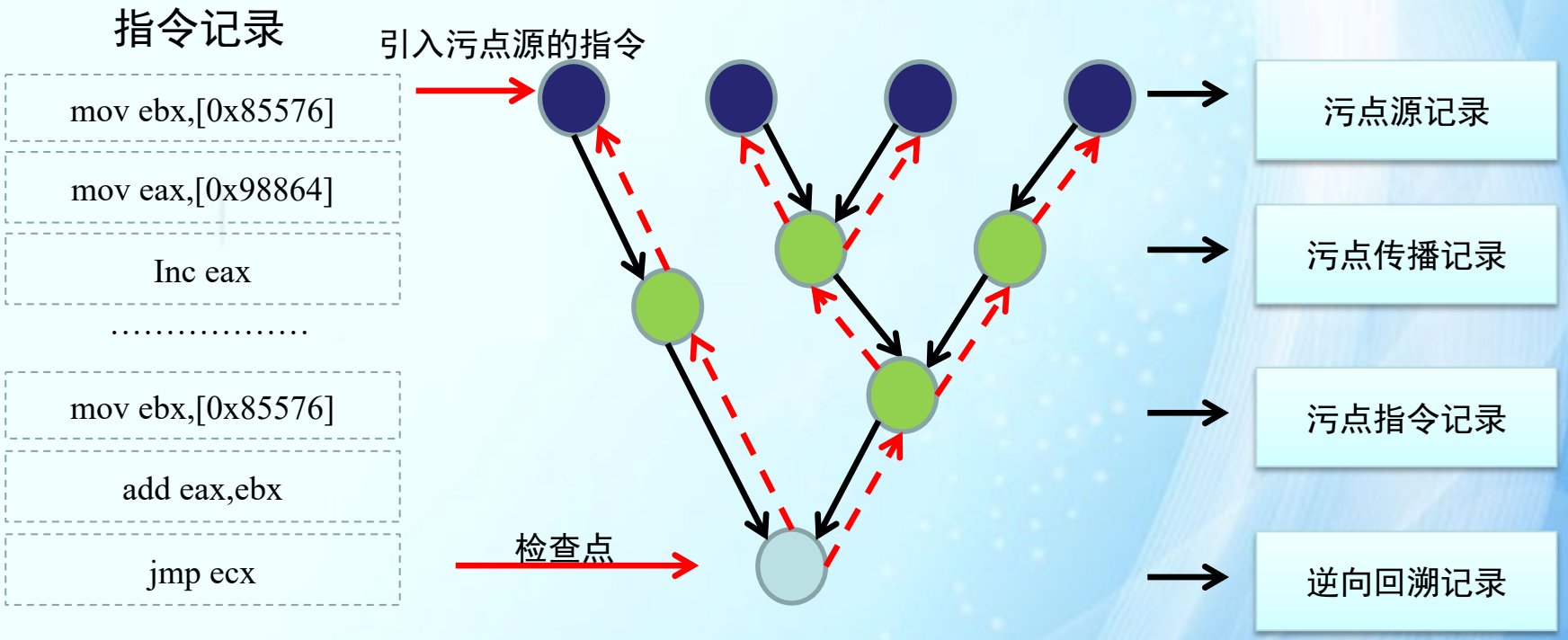
指令类型	传播规则	举例说明
数据移动指令	dst_ts = src_ts	mov eax, ebx
数学运算指令	dst_ts = op(src_ts_1,…src_ts_n)	add eax, ebx
堆栈操作指令	esp_ts = src_ts **or** dst_ts = esp_ts	push eax / pop eax
函数跳转指令	eip_ts = src_ts **or** dst_ts = eip_ts	call [ecx] / ret

2.2 软件漏洞路径分析

- 污点传播停止规则
  - 污点数据影响控制流转移方向
    - JMP　TAINT
    - RET　TAINT
    - 能够实现控制流劫持
  - 污点数据被同时影响写入指针和写入内容
    - MOV　[$TAINT_1$]，$TAINT_2$
    - 能够向任意地址写入双字内容的DWORD SHOOT
  - 污点数据引发页面访问异常
    - MOV　REG，[TAINT]
    - 当SEH能够被覆盖时，该指令能够触发漏洞

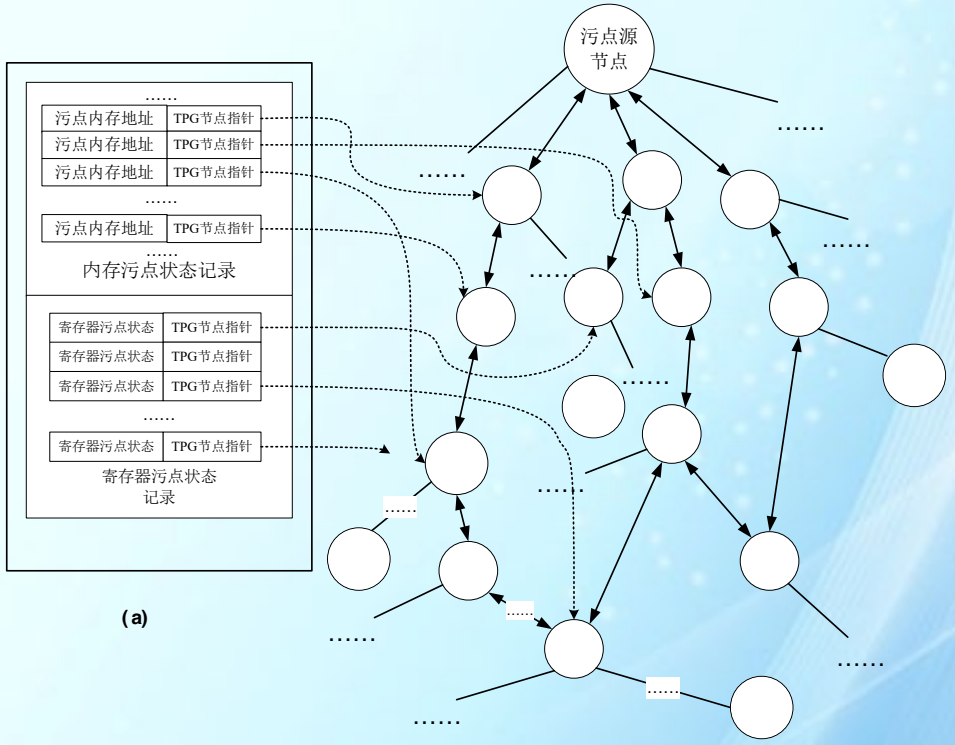
2.2 软件漏洞路径分析

- 污点传播回溯规则
  - 由用户指定逆向回溯的起点，沿动态污点传播过程记录，逆向查找上一级节点，直至回溯过程到达输入数据，提取输入数据的偏移、长度



2.2 软件漏洞路径分析

- 路径提取
  - 从污点源（引入污点数据的API）开始，
  - 通过动态污点传播分析，计算指令之间的数据依赖关系
  - 当污点传播过程满足终止条件时，停止传播
  - 从污点源引入的节点到终止节点之间的部分，即为漏洞的脆弱路径



2.2 软件漏洞路径分析

- 路径提取中存在的特殊情况
  - 浮点数：异常传播，两个数差值过大导致其中小的浮点数被忽略
  - 指针：可能引起污点爆炸，全部不标记，则污点漏标
  - 控制依赖：需要计算控制依赖范围并在特定条件下标记污点

浮点数引发异常传播

变量A “吞掉” 变量B

```
A = 3*10^15;
B = 5;
C = A + B;      // 实际上 C==A
if( C > A )     // 永远不可能满足的条件
{
    ……
}
```

基于指针的污点传播

```
mov eax, [edx+ecx*2] ;        //字符串转换
call [ecx+0x18];              //虚函数指针
```

控制依赖引起的污点传播

污点数据被用作控制流转移

```
if ( input ==‘a’)
{
        x = ‘A’;
}
```

2.2 软件漏洞路径分析

- 路径提取中存在的特殊情况
  - 浮点数：异常传播，两个数差值过大导致其中小的浮点数被忽略
  - 指针：可能引起污点爆炸，全部不标记，则污点漏标
  - 控制依赖：需要计算控制依赖范围并在特定条件下标记污点

控制依赖引起的污点传播

污点数据被用作控制流转移

if (input ==‘a’) { x = ‘A’;}

```
jz target
mov bl,’a’
mov bh,’b’
sub ax, bx
```

1. 静态预分析
2. 建立目标代码的小范围CFG
3. 计算CFG的前后必经节点
4. 在回溯时，如果污点传播停止，则尝试把控制依赖变量加入污点中

基于指针的污点传播

```
mov eax, [edx+ecx*2] ;      //字符串转换
call [ecx+0x18];            //虚函数指针
```

有可能引起污点数据爆炸

1. 在正向污点传播计算时，将该指令作为普通指令处理
2. 在污点回溯截断考虑基于指针的污点传播情况
3. 如果污点传播回溯无法到达污点源，则标记指针索引为污点逆向回溯

2.2 软件漏洞路径分析

- 路径条件分析
  - 目标
    - 基于漏洞脆弱路径，分析输入数据与漏洞触发的依赖关系，确定触发脆弱路径的输入数据条件
  - 基本思路

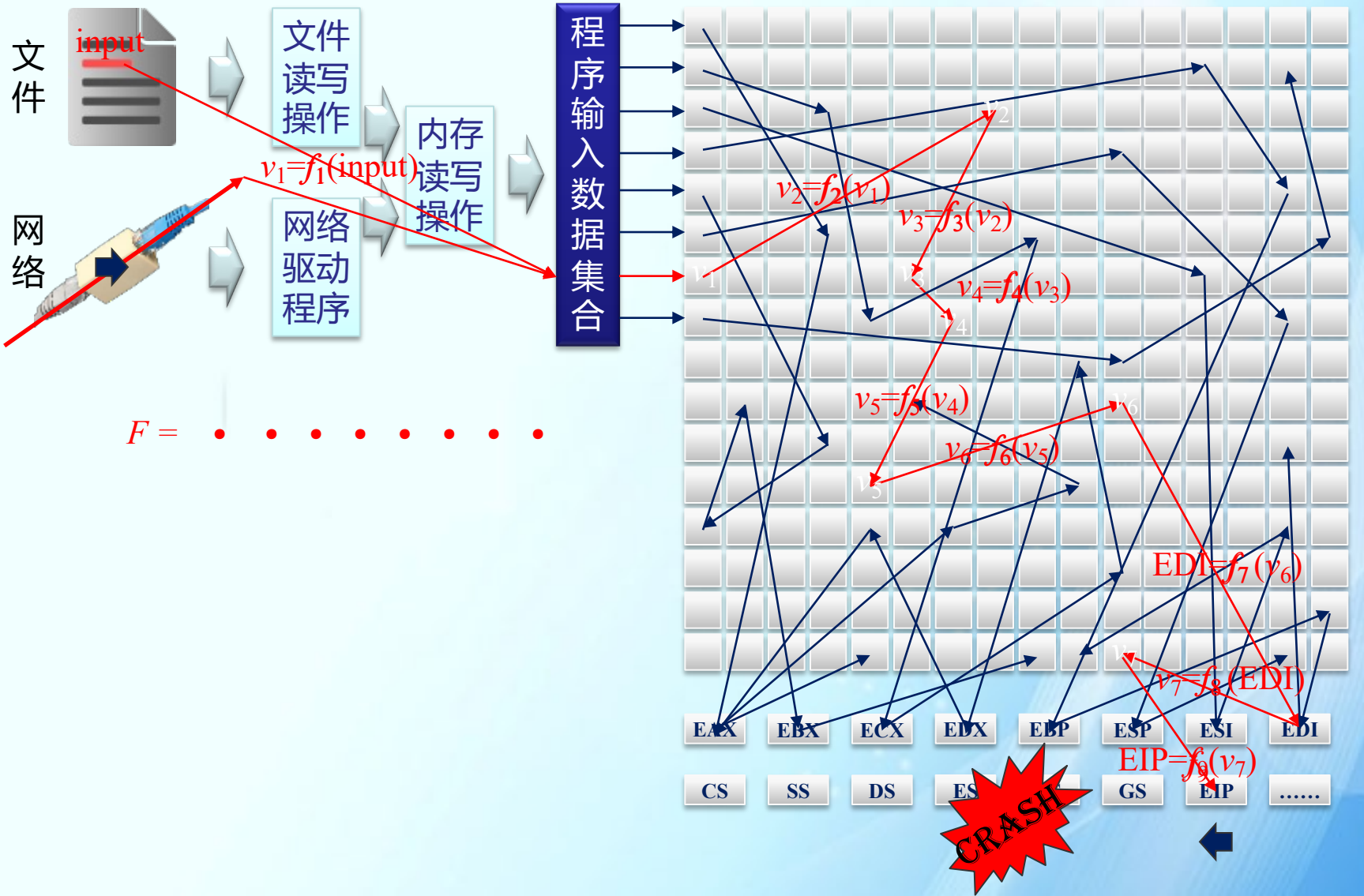
① 控制流指令数据依赖提取：抽取脆弱路径上的控制流转移指令，提取数据取值范围

② 数据流回溯：以控制流转移指令为起点，基于污点传播的数据依赖关系，逆向回溯控制流转移所依赖的输入数据

③ 获取路径条件：根据回溯路径上的运算关系，结合控制流转移点数据取值范围综合求解，确定输入数据的取值范围

2.2 软件漏洞路径分析

■路径条件分析



2.3 软件漏洞内存布局分析

- 内存布局分析
  - 分析目标
    - 当程序的脆弱点被触发时，确定程序读入数据在内存中的布局情况，分析漏洞利用所需的关键数据所在位置，为漏洞成因分析、漏洞利用生成提供支撑
  - 主要思路

① 标记输入数据为污点，进行正向动态污点传播

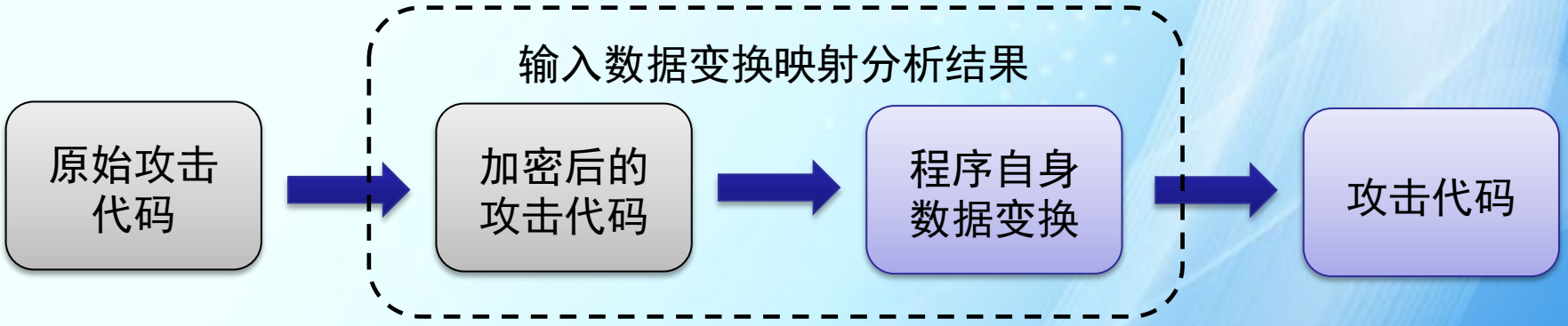
② 使用污点数据映射表，计算每一条指令执行后的内存布局情况

③ 污点传播到达脆弱点时终止，分析当前程序的污点布局情况

2.3 软件漏洞内存布局分析

- 内存布局情况分析
  - 输入数据直接映射
    - 分析污点映射表，确定崩溃时内存中的污点布局
    - 由每个污点数据的最后一次使用开始逆向回溯，直到到达污点源时停止
    - 分析污点回溯路径上是否有运算，如果无运算，则为输入数据直接映射的内存布局
    - 对于直接映射的内存布局，收集并分析其相邻污点的情况，如果相邻污点的污点源也相邻，则将连续的污点源视为一个整体

- 内存布局分析
  - 输入数据变换映射
    - 如果有运算，则为输入数据变换映射的内存布局
    - 输入数据变换映射的内存布局，同样需要收集并分析其相邻污点的情况
    - 通过人工分析或公式求解的方式，判定输入数据的变换过程是否可逆，如可逆，则该变换过程可视为一次代码解密，在编写攻击代码时，使用逆算法对攻击代码进行加密



2.4 实例分析

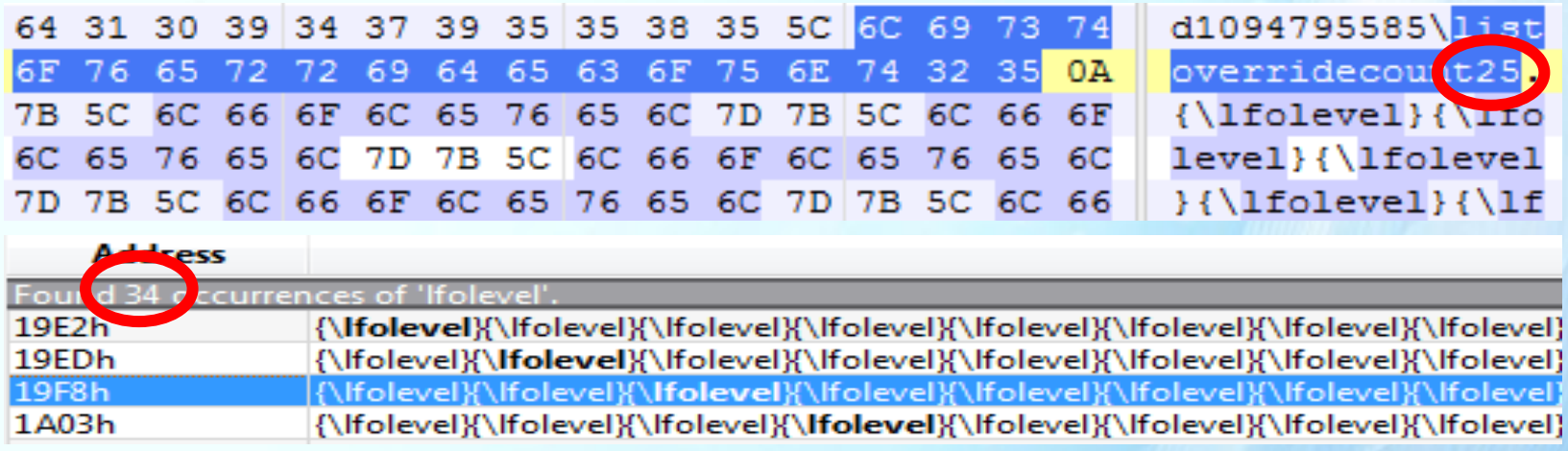
- 实例分析(CVE-2014-1761)
  - 引发该漏洞的根本原因与RTF 控制字“ overridetable ”
  - 一个“overridetable”结构可能包括“listoverride”，“listoverridecount”，和“lfolevel”域
  - “listoverridecount”表明结构所包含的“lfolevel”的数目
  - 根据微软的官方格式文档，合法参数应为0,1 或9。然而在该溢出样本中，该参数值为34

2.4 实例分析

- 实例分析(CVE-2014-1761)
  - 污点误用检测：污点源字符串“22873”当做十进制数后转换成十六进制“0x5959”，并最终作为跳转目标地址

2.4 实例分析

- 实例分析(CVE-2014-1761)
  - 污点分析与手工逆向结果：漏洞利用程序声称最多使用25个数据，但未进行长度检测，实际上填充了34项数据。因此，漏洞利用程序通过越界拷贝覆盖虚函数指针，完成控制流劫持



2.4 实例分析

- 视频演示
  - 基于动态污点传播的CVE-2014-1761样本分析

2.4 实例分析

■ 实例：Office公式编辑器漏洞 CVE-2018-0798

目标：基于程序切片追溯Shellcode的来源

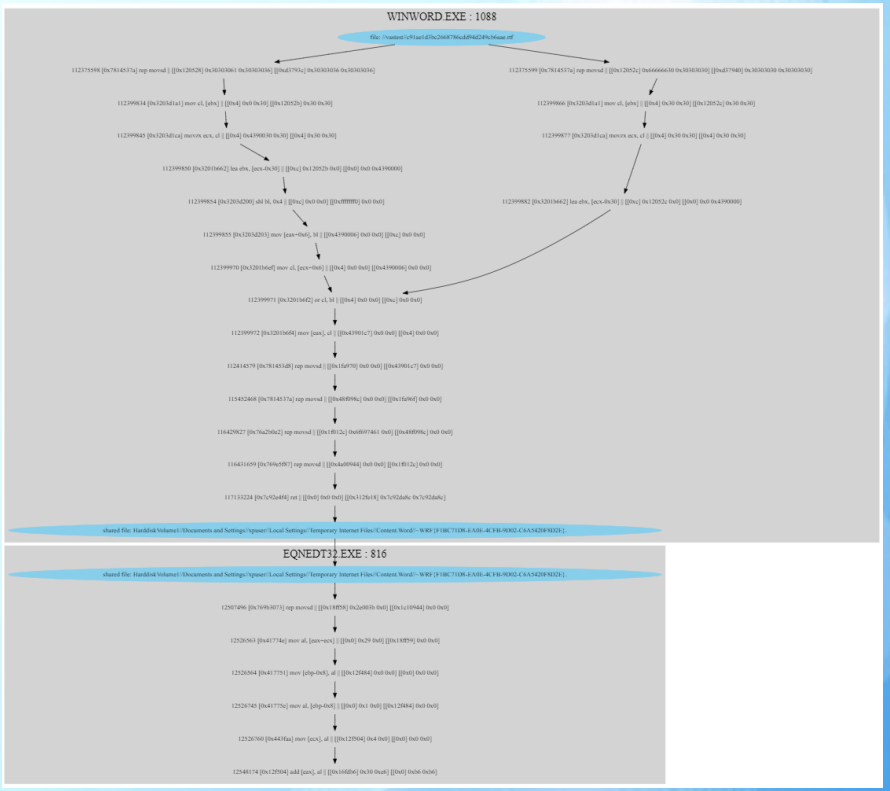
攻击者注入的Shellcode

选择进程 EQNEDT32.EXE:816 读取记录

起始ID 12548174 读取数量 100

ID	指令地址	指令
12548174	12f504	add byte ptr [eax], al
12548175	12f506	add byte ptr [eax], al
12548176	12f508	push 1
12548177	12f50a	call 5
12548178	12f50f	pop eax
12548179	12f510	add eax, 0x10
12548180	12f513	push eax
12548181	12f514	mov eax, dword ptr [0x46681c]
12548182	12f519	call eax

Shellcode的后向切片



2.5 参考资料

- 技术书籍
  - 0 day 安全 软件漏洞分析技术（第1版/第2版）
  - 漏洞战争：软件漏洞分析精要
  - 有趣的二进制软件安全与逆向分析
- 综述论文
  - L. Szekeres, M. Payer, Tao Wei, Dawn Song. SoK: Eternal War in Memory. IEEE S&P'13

谢谢