

Finding 0-days by Using VulnHunt

Wei Wang(alert7)
Code Audit Labs of Vulnhunt Co.,Ltd



翰海源

翰海源 www.vulnhunt.com

Agenda

- About author
- Vulnhunt Internal
 - Ring3 VM
 - Tainted Propagate Analysis
 - Modules of checking vulnerability
- How to find pdf 0day(now fixed) with Vulnhunt
 - Normal steps
 - To check
 - Locate and Analysis
- How to find alzip 0day withvulnhunt (Demo)

About author

- Aka Alert7 ,XFOCUS member
- Find lot of vulnerabilities in Microsoft, ORACLE , ADOBE , QUICKTIME , LINUX KERNEL , Trend Micro , CA...
- With other XFOCUS guys wrote a book 《Network Penetration Testing》
- He was McAfee Labs Security Researcher
- Now he is CTO of Vulnhunt co.,Ltd
www.vulnhunt.com

Data Stream Analysis

- Why Data stream Analysis
 - To identify exploitable vulnerabilities which is triggered by specific user data stream
- Type of Data stream analyze
 - Static data stream analyze
 - Coverity
 - Fortify
 - Klocwork
 - Dynamic data stream analyze
 - Valgrind (for linux)
 - Immunity Debugger
 - PEIMEI
 - Vtrace

What's vulnhunt?

- Abbreviation of **Vulnerability hunt**
- Two technology
 - RING3 VM
 - Tainted Propagate Analysis

Virtualization Technology

- Data Control
 - Where does user data stream come from
 - and go to?
- Execution Control
 - Instruction execution path Prediction
 - Flow Control

RING3VM

- Normal VM
 - Cpu,FPU,MEMORY
 - IO
 - Acpi,network,cdrom,DMA,pic,BIOS,Harddisk...
 - Load OS, app run in OS
- RING3 VM
 - CPU,FPU
 - App run in RING3VM or RING3VM step execute app.

RING3VM main functions

- Framework for Instruction-level Tracing

- Instruction-level Taint Propagation

- Trace tainted data: from and to.

- example

```
mov ecx, dword ptr ds:[eax]
```

we need check if eax is tainted, check if [eax] is tainted, mark ecx as tainted.

- Instruction-level Other functions(TODO)

RING3VM

- Pros
 - Simplified design
 - Better performance
- Cons
 - Tampered by malicious code from target
 - OS Dependency
 - Bad for muti-thread

Tainted Source Marking

- Function Calls

call dword ptr ds:0x40606c

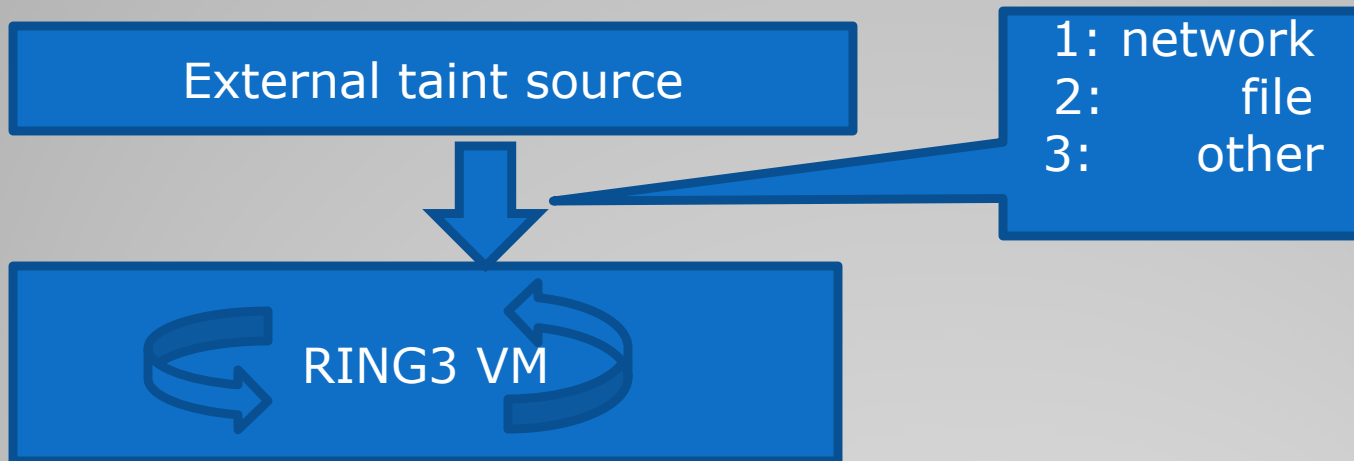
The data of Address 0x40606c is 0x7c810e17

Which function call we need Identify address 0x7c810e17 for? (loop every module to check export table)

- Mark tainted source

```
recv(SOCKET s,char *buf,int len,int flags)
```

```
set_buff_tainted(GET_ARG(a->esp,1),GET_ARG(a->esp,2));
```



Two level API

- Function-Level API
 - API-- Taint Tracking
 - API-- HOOK Functions
 - API- alert,log
- Instruction-level API
 - Instruction-level check

Taint Tracking API

- Hook a function

```
1:
FUNC_INFO f;
    f.comment = NULL;
    f.enter_func= false;
    f.func_name="malloc";
    f.handler = malloc_handler;
    regist_func_handler(&f);

2:
static int malloc_handler(void * p,unsigned int eip,struct allregs * a)
{
FUNC_INFO * info = (FUNC_INFO *) p;
    checker_log(CHECK_MODULE"%s_handler called at %p %s(0x%x)\n",
        info->func_name,
        eip,
        info->func_name,
        GET_ARG(a->esp,0));
    bool t = buffer_from_tainted(GET_ARG_OFFSET(a->esp,0));
    if ( t )
        checker_log(CHECK_MODULE SECURITY_WARNING"%s(0x%x) arg0 is %s",
            info->func_name,
            GET_ARG(a->esp,0),
            "[tainted]");

    return 0;
}
```

Taint Tracking API

- Mark tainted source

```
static int recv_handler(void * p,unsigned int eip,struct allregs * a)
{
    set_buff_tainted(GET_ARG(a->esp,1),GET_ARG(a->esp,2));
    return 0;
}
```

Taint Tracking API

- Goal : let behind false positives, discover as many tainted check point as possible
 - 100 to 10 ? Not bad!
- Thoughts about reducing false+
 - Future work.

Embedded hundreds of checker and function Abstract

- Memcpy
- memcpy_s
- vsprintf_l
- Syslog
- Printf
- Open
- Fread
- Read
- ReadFile
- ...

Case Study -- MALLOC MEMORY

- example
 - `Len=get_int_from_net();`
 - `Len+=1;`
 - `Char *buf = malloc(Len);`
- **Malloc, RtlAllocateHeap, HeapAlloc**
 - `buffer_from_tainted(GET_ARG_OFFSET(a->esp,x));`
 - Check if length param is tainted, alert for manual review

Case Study -Format string

- example
 - `printf(get_buf_from_net());`
- `Printf,fprintf,sprintf...`
 - If format string param is tainted, alert as potential format string vulnerability
 - `Printf("%s%n")`, it's a real format string vulnerability, but due to the param is not tainted, so it's a unexploitable . So ignore.

Case Study - rep movs

- example
 - `Memcpy(dst,src,get_int_from_net());`
- Alert if ECX register is tainted

Case Study - signed integer

- Example
 - `Int a;`
 - `char *buf = Get_buf_from_net();`
 - `a = buf[0]; // signed extend`
- `movsx ecx, byte ptr ss:[esp+0x4]`
 - Check `movsx`, alert if `op_2` is tainted.

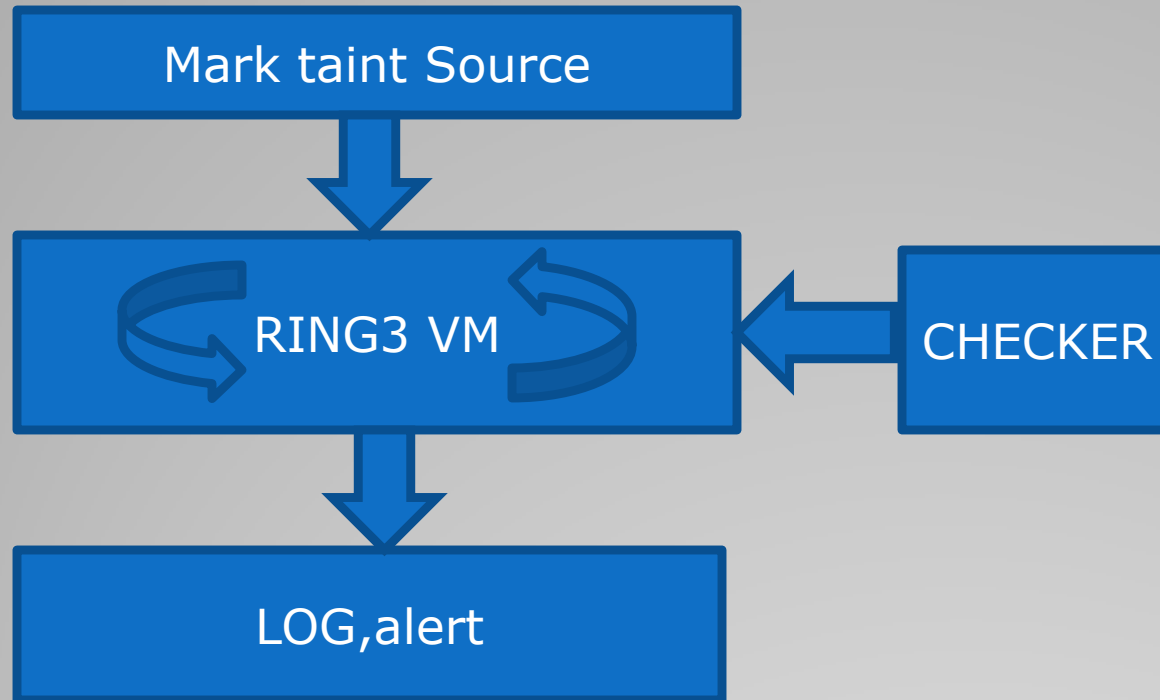
Case Study – DIV ZERO

- Example
 - Int a;
 - a/=get_int_from_net();
- Idiv/div eax, ecx
- Check idiv/div, alert if op_2 is tainted

Case Study – Integer Overflow

- example
 - Int a;
 - a*=get_int_from_net();
- Imul/mul edi, [ebp+arg_8]
 - Alert if one of op1,op2 is tainted
- lea ecx, dword ptr ds:[eax*8]
 - Sometime,using lea express multiplication

Components



Perf Optimization

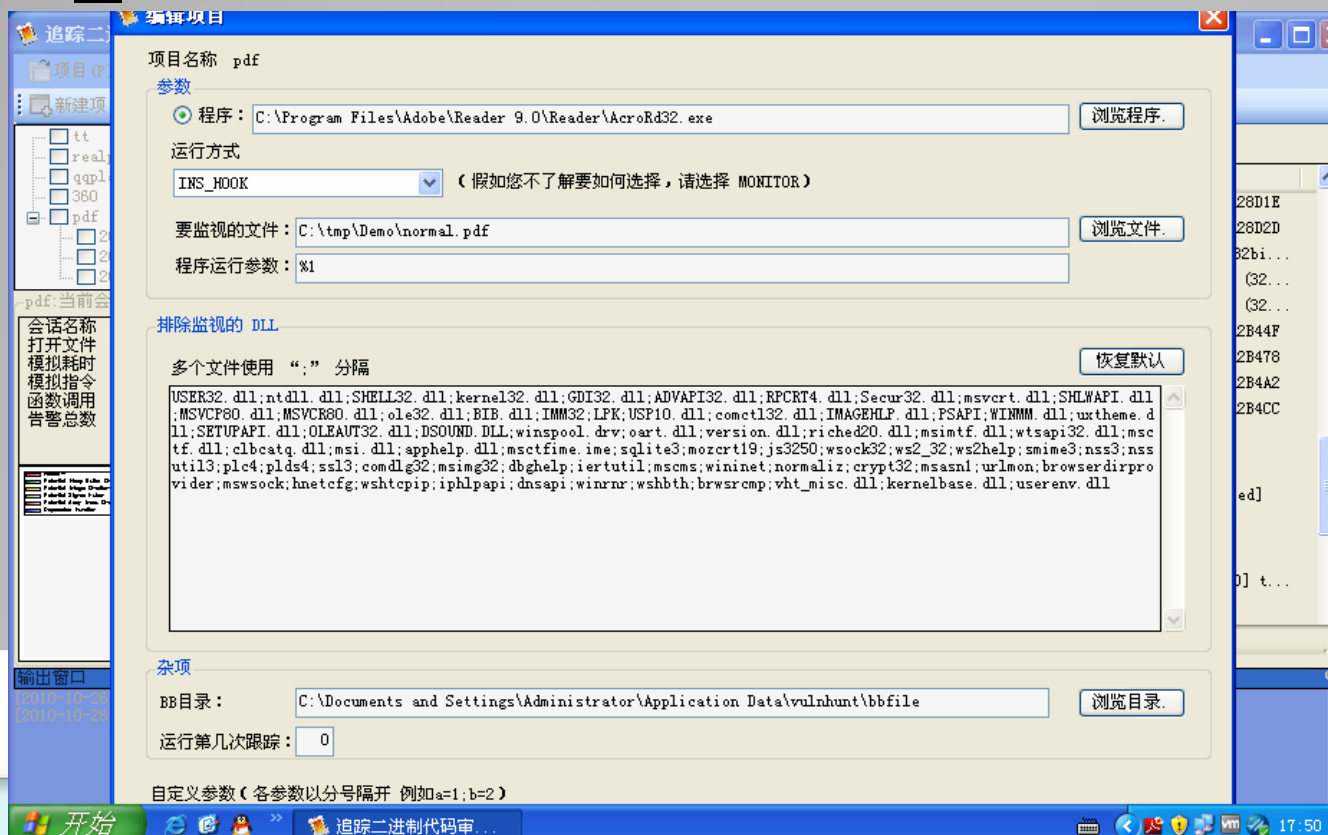
- Rep movsd dword ptr es:[edi], dword ptr ds:[esi]
 - Exece once , eq memcpy(edi,esi,ecx*4)
- Skip irrelevant instructions
 - I.e: Skip instructions in ntdll.dll user32.dll

Finding pdf vuln(CVE-2009-3955)

- One example, all (Potential)vulnerabilities of this execution path
- Show how to use vulnhunt find CVE-2009-3955 Adobe Reader and Acrobat JpxDecode Memory Corruption Vulnerability。

Finding pdf vuln(configure)

- Our configure, AcroRd32.exe open a normal example normal.pdf , run mode INS_HOOK.



Finding pdf vuln(result)

追踪二进制代码审计系统

项目(F) 会话(S) 引擎(E) 查看(V) 系统(M) 帮助(H)

新建项目 打开项目 选择运行模式 运行 停止 调用图 报表

tt
realplay
qqplayer
360
pdf
2010-09-27 14:24:09
2010-09-27 14:16:16
2010-09-27 14:09:55

pdf:当前会话

会话名称	2010-09-27 14:24:09
打开文件	C:\tmp\Demo\normal.pdf
模拟耗时	73.16 秒
模拟指令	2601326 条
函数调用	124884 次
告警总数	65 个

警告窗口

编号	日期	EIP	信心	内容
<input type="checkbox"/> 74	2010-9-27 14:24:14	0x04A28D1E	2	LEA may integer overflow at EIP 04A28D1E
<input type="checkbox"/> 75	2010-9-27 14:24:14	0x04A28D2D	2	LEA may integer overflow at EIP 04A28D2D
<input type="checkbox"/> 77	2010-9-27 14:24:14	0x023D14BE	4	memset(len 0x3) arg len is tainted (32bi...
<input type="checkbox"/> 79	2010-9-27 14:24:14	0x023D3E25	4	malloc(len 0x23e) arg len is tainted (32...
<input type="checkbox"/> 83	2010-9-27 14:24:15	0x02A8DC98	4	memcpy(len 0x222) arg len is tainted (32...
<input type="checkbox"/> 99	2010-9-27 14:24:16	0x04A2B44F	2	LEA may integer overflow at EIP 04A2B44F
<input type="checkbox"/> 100	2010-9-27 14:24:16	0x04A2B478	2	LEA may integer overflow at EIP 04A2B478
<input type="checkbox"/> 102	2010-9-27 14:24:16	0x04A2B4A2	2	LEA may integer overflow at EIP 04A2B4A2
<input type="checkbox"/> 104	2010-9-27 14:24:16	0x04A2B4CC	2	LEA may integer overflow at EIP 04A2B4CC

Potential Signed Extern (1)

<input type="checkbox"/> 93	2010-9-27 14:24:15	0x04A2D830	5	MOVZX_2T04 0x1 sign extend [tainted]
-----------------------------	--------------------	------------	---	--------------------------------------

Potential Array Index Overflow (1)

<input type="checkbox"/> 88	2010-9-27 14:24:15	0x049FA717	4	Resolve [eax(0x42d0533:0x1010101)+0x0] t...
-----------------------------	--------------------	------------	---	---

警告 65

输出窗口

[2010-10-26 17:43:32] 欢迎使用追踪二进制代码审计系统: 旗舰版1.0
[2010-10-26 17:43:39] 打开项目 [pdf] 完成

开始 追踪二进制代码审... 17:47

Finding pdf vuln(checking)

- Currently , output many alerts , need manual checking.
- Goal : let behind false positives, discover as many tainted check point as possible
 - 100 to 10 ? Not bad!

Finding pdf vuln(Analysis 1)

- **confirm Potential signed extern**
 - Checking No93 ,type is Potential signed extern, content is MOV SX_2TO4 0x1 sign extern.

Finding pdf vuln(Analysis 2)

- ALT_5 run Ida jumper, Send to Ida jumper

The screenshot displays two main windows on a Windows desktop:

- IDA Pro:** The main window shows the assembly code for a PDF file. The code is as follows:

```
.text:1003D7D4      push    2
.text:1003D7D6      mov     ecx, ebx
.text:1003D7D8      call   get_bytes
.text:1003D7DD      sub    [ebp+var_8]
.text:1003D7E1      push  2
.text:1003D7E3      mov     ecx, ebx
.text:1003D7E5      mov     byte ptr [
```
- 追踪二进制代码审计系统 (Binary Code Audit System):** This window provides a detailed view of the audit process. It includes:
 - 项目 (Project):** A tree view showing the project structure, including 'pdf' and its sessions.
 - 会话名称 (Session Name):** 2010-09-27 14:24:09
 - 打开文件 (Open File):** C:\tmp\Demo\normal.pdf
 - 模拟耗时 (Simulation Time):** 73.16 秒
 - 模拟指令 (Simulation Instructions):** 2801326 条
 - 函数调用 (Function Calls):** 124884 次
 - 警告总数 (Total Warnings):** 65 个
 - 警告列表 (Warning List):** A table listing warnings with columns for ID, Date, Time, and EIP.

编号	日期	EIP
75	2010-09-27 14:24:14	0x04A28D
77	2010-09-27 14:24:14	0x023D14
79	2010-09-27 14:24:14	0x023D3E
83	2010-09-27 14:24:15	0x02A8DC
99	2010-09-27 14:24:16	0x04A2B4
100	2010-09-27 14:24:16	0x04A2B4
102	2010-09-27 14:24:16	0x04A2B4
104	2010-09-27 14:24:16	0x04A2B4
 - 警告窗口 (Warning Window):** A list of warnings with actions like '加入误报列表' (Add to false positive list), '发消息到 Ida Jumper' (Send message to Ida Jumper), and '查看污染图' (View pollution map).
 - 输出窗口 (Output Window):** Shows system logs and messages, including '欢迎使用追踪二进制代码审计系统: 旗舰版 1.0' and '打开项目 [pdf] 完成'.

Finding pdf vuln(Analysis 3)

- Auto locate to Ida ASM from alert

The screenshot displays the IDA Pro interface. The main window shows assembly code for a function named `loc_1003D830`. The code includes instructions like `mov ecx, ebx`, `call get_bytes`, `sub [ebp+var_8], 4`, `movsx eax, ax`, and `cmp eax, [edi+4Ch]`. A console window at the bottom shows the IDA Pro startup log, including the path to the processor metapc file and the loading of the `JP2KLib.dll` file.

On the right side, a Windows taskbar is visible, showing the system tray with a clock at 19:05 and several icons. A window titled "系统 (S) 帮助 (H)" is open, displaying a table with columns for "日期" (Date) and "EIP". The table contains several rows of data, including dates from 2010-9-27 and EIP addresses like 0x04A28D, 0x023D14, 0x023D3E, 0x02A8DC, 0x04A2B4, 0x04A2B4, 0x04A2B4, 0x04A2B4, 0x04A2B4, 0x04A2B4, 0x04A2D8, 0x049FAT, and 0x04A2B4.

日期	EIP
2010-9-27 14:24:14	0x04A28D
2010-9-27 14:24:14	0x023D14
2010-9-27 14:24:14	0x023D3E
2010-9-27 14:24:15	0x02A8DC
2010-9-27 14:24:16	0x04A2B4
2010-9-27 14:24:16	0x04A2B4
2010-9-27 14:24:16	0x04A2B4
2010-9-27 14:24:16	0x04A2B4
2010-9-27 14:24:16	0x04A2B4
2010-9-27 14:24:16	0x04A2B4
2010-9-27 14:24:15	0x04A2D8
2010-9-27 14:24:15	0x049FAT
2010-9-27 14:24:15	0x04A2B4

Finding pdf vuln(Analysis 4)

- `.text:1003D830 loc_1003D830: ; CODE`
`XREF: sub_1003CB13+D04j`
- `.text:1003D830 movsx eax, ax`
- `.text:1003D833 cmp eax, [edi+4Ch]`
- `.text:1003D836 jge loc_1003DC8A`
- `.text:1003D83C mov ecx, [edi+2Ch]`
- `.text:1003D83F mov esi, eax`
- `.text:1003D841 imul esi, 0Ch`
- `.text:1003D844 mov [esi+ecx], eax`
- `.text:1003D847 mov eax, [edi+2Ch]`
- Ax is tainted , movsx extern to eax,so eax can be negative.
bypass jpe checking.
- `mov [esi+ecx], eax` , can put eax value to limited
arbitrary address. Cause memory Corruption.

Finding alzip 0day(Demo)

THANK YOU

Question?