Adaptive Android Kernel Live Patching

Tim Xia, Yulong Zhang

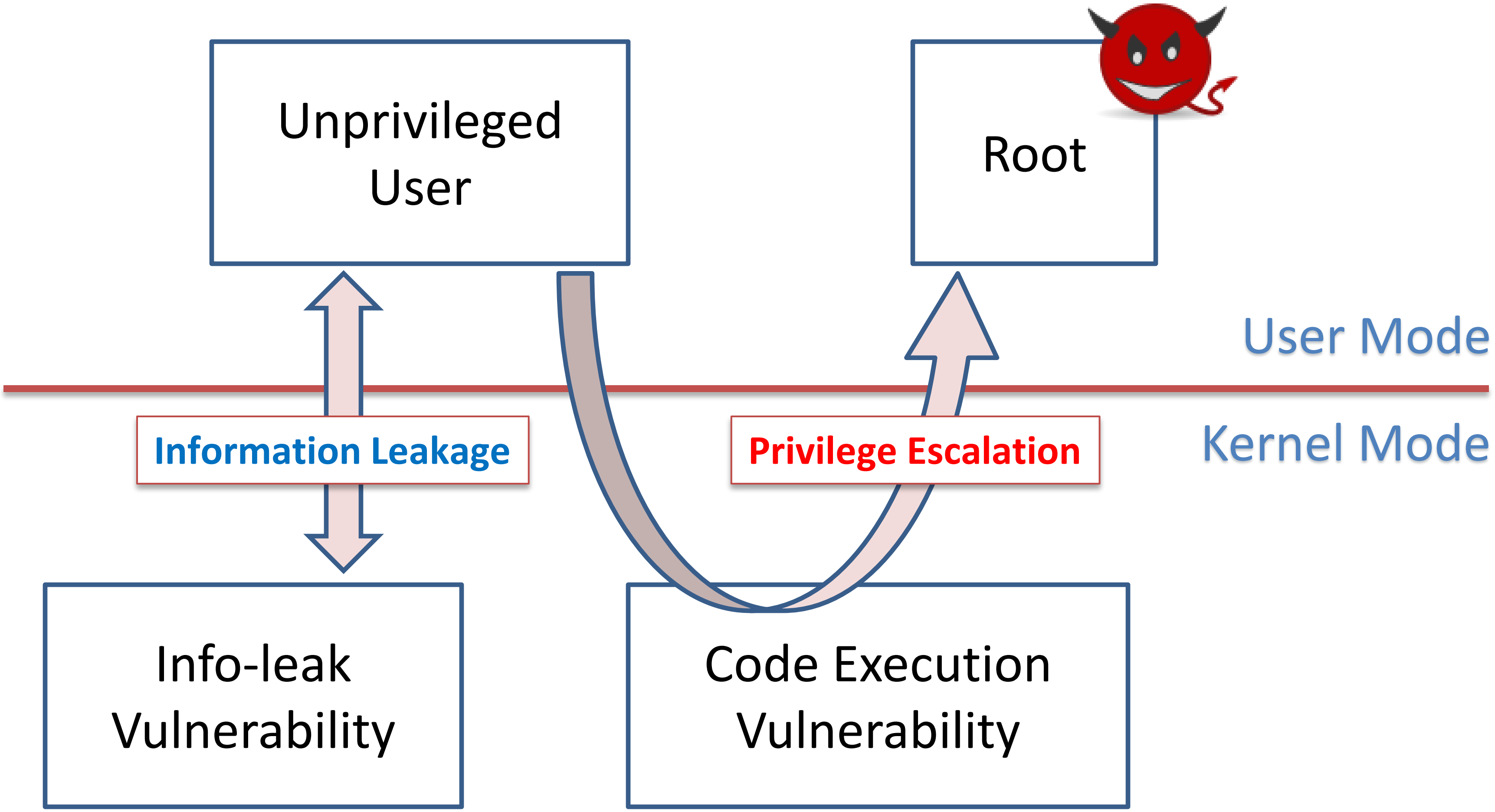
Baidu X-Lab

May 2016

Outline

- Android Kernel Vulnerability Landscape
- The Problem:
  - Devices Unpatched Forever/for A Long Period
  - Difficult to Patch due to Fragmentation
- The Solution: Adaptive Kernel Live Patching
- Establishing the Ecosystem

Threats of Kernel Vulnerabilities



Unprivileged User

Root

User Mode

Kernel Mode

Information Leakage

Privilege Escalation

Info-leak Vulnerability

Code Execution Vulnerability

Threats of Kernel Vulnerabilities

- Most security mechanisms rely on kernel integrity/trustworthiness, thus will be broken
  - Access control, app/user isolation
  - Payment/fingerprint security
  - KeyStore
  - Other Android user-land security mechanisms
- TrustZone will also be threatened
  - Attack surfaces exposed
  - Many TrustZone logic trusts kernel input

Kernel Vulnerabilities in Android Security Bulletin

Month	Vulnerability List	Count
2015/09	CVE-2015-3636	1
2015/12	CVE-2015-6619	1
2016/01	CVE-2015-6637 CVE-2015-6638 CVE-2015-6640 CVE-2015-6642	4
2016/02	CVE-2016-0801 CVE-2016-0802 CVE-2016-0805 CVE-2016-0806	4
2016/03	CVE-2016-0728 CVE-2016-0819 CVE-2016-0820 CVE-2016-0822 CVE-2016-0823	5
2016/04	CVE-2014-9322 CVE-2015-1805 CVE-2016-0843 CVE-2016-0844 CVE-2016-2409 CVE-2016-2410 CVE-2016-2411	7
2016/05	CVE-2015-0569 CVE-2015-0570 CVE-2016-2434 CVE-2016-2435 CVE-2016-2436 CVE-2016-2437 CVE-2015-1805 CVE-2016-2438 CVE-2016-2441 CVE-2016-2442 CVE-2016-2443 CVE-2016-2444 CVE-2016-2445 CVE-2016-2446 CVE-2016-2453	15

🟦 Information Leakage 🟥 Privilege Escalation

The Growing Trend Indicates

Month	Count
2015/08	0
2015/09	1
2015/10	0
2015/11	0
2015/12	1
2016/01	4
2016/02	4
2016/03	5
2016/04	7
2016/05	15

- More and more attentions are drawn to secure the kernel 👍
- More and more vulnerabilities are in the N-Day exploit arsenal for the underground businesses 👎

Recent Vulnerabilities with Great Impact

- CVE-2014-3153 (Towelroot)

  - The futex_requeue function in kernel/futex.c in the Linux kernel through 3.14.5 does not ensure that calls have two different futex addresses, which allows local users to gain privileges.

Recent Vulnerabilities with Great Impact

- CVE-2015-3636 (PingPong Root)

  - The ping_unhash function in net/ipv4/ping.c in the Linux kernel before 4.0.3 does not initialize a certain list data structure during an unhash operation, which allows local users to gain privileges or cause a denial of service.

Recent Vulnerabilities with Great Impact

- CVE-2015-1805 (used in KingRoot)
  - The pipe_read and pipe_write implementations in kernel before 3.16 allows local users to cause a denial of service (system crash) or possibly gain privileges via a crafted application.
  - A known issue in the upstream Linux kernel that was fixed in April 2014 but wasn’t called out as a security fix and assigned CVE-2015-1805 until February 2, 2015.

Many Vulnerabilities Have Exploit PoC Publicly Disclosed

Vulnerability/Exploit Name	CVE ID
mempodipper	CVE-2012-0056
exynos-abuse/Framaroot	CVE-2012-6422
diagexploit	CVE-2012-4221
perf_event_exploit	CVE-2013-2094
fb_mem_exploit	CVE-2013-2596
msm_acdb_exploit	CVE-2013-2597
msm_cameraconfig_exploit	CVE-2013-6123
get/put_user_exploit	CVE-2013-6282
futex_exploit/Towelroot	CVE-2014-3153
msm_vfe_read_exploit	CVE-2014-4321
pipe exploit	CVE-2015-1805
PingPong exploit	CVE-2015-3636
f2fs_exploit	CVE-2015-6619
prctl_vma_exploit	CVE-2015-6640
keyring_exploit	CVE-2016-0728
……	……

There’re also exploits made public but

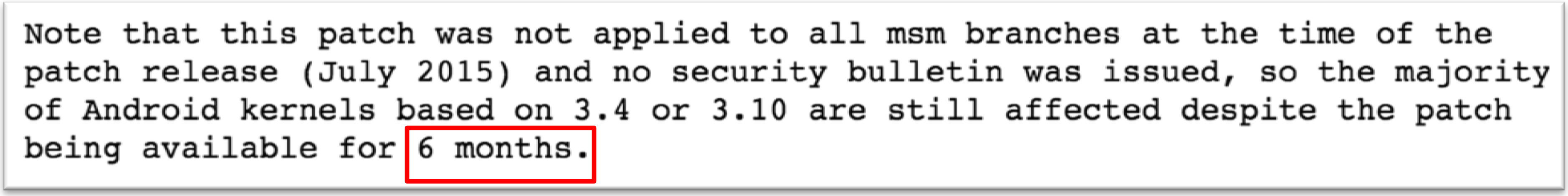
- Never got officially reported to vendors
- Disclosed before being patched
- Not getting timely fix
- ……

Exploits made public but not reported

"… We are able to identify at least **10** device driver exploits (from a famous root app) that are **never reported** in the public…"

Android Root and its Providers: A Double-Edged Sword
H. Zhang, D. She, and Z. Qian, CCS 2015

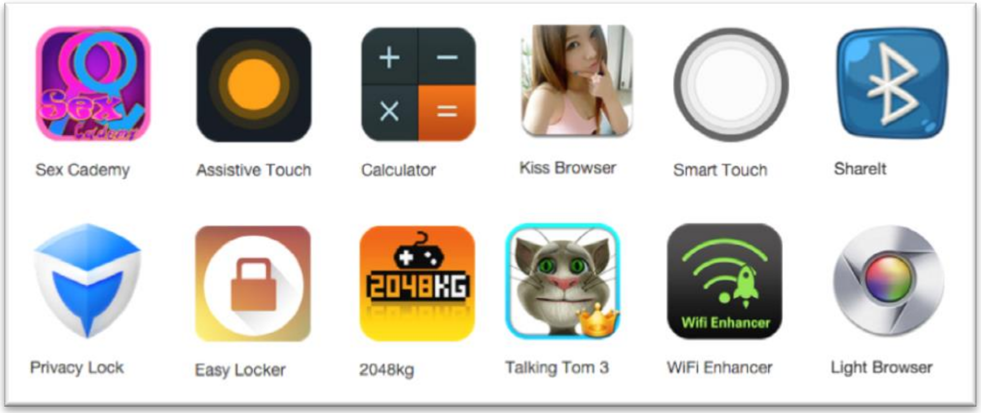
Exploits disclosed but not timely patched

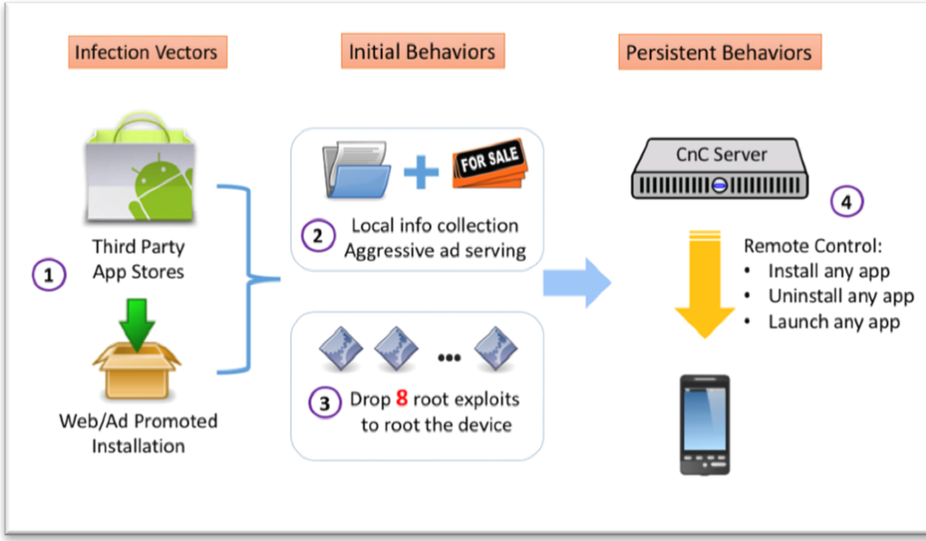


Note that this patch was not applied to all msm branches at the time of the patch release (July 2015) and no security bulletin was issued, so the majority of Android kernels based on 3.4 or 3.10 are still affected despite the patch being available for 6 months.

https://bugs.chromium.org/p/project-zero/issues/detail?id=734&can=1&sort=-id

Malware/Adware with Root Exploits





Infection Vectors

Initial Behaviors

Persistent Behaviors

1 Third Party App Stores

Web/Ad Promoted Installation

2 Local info collection Aggressive ad serving

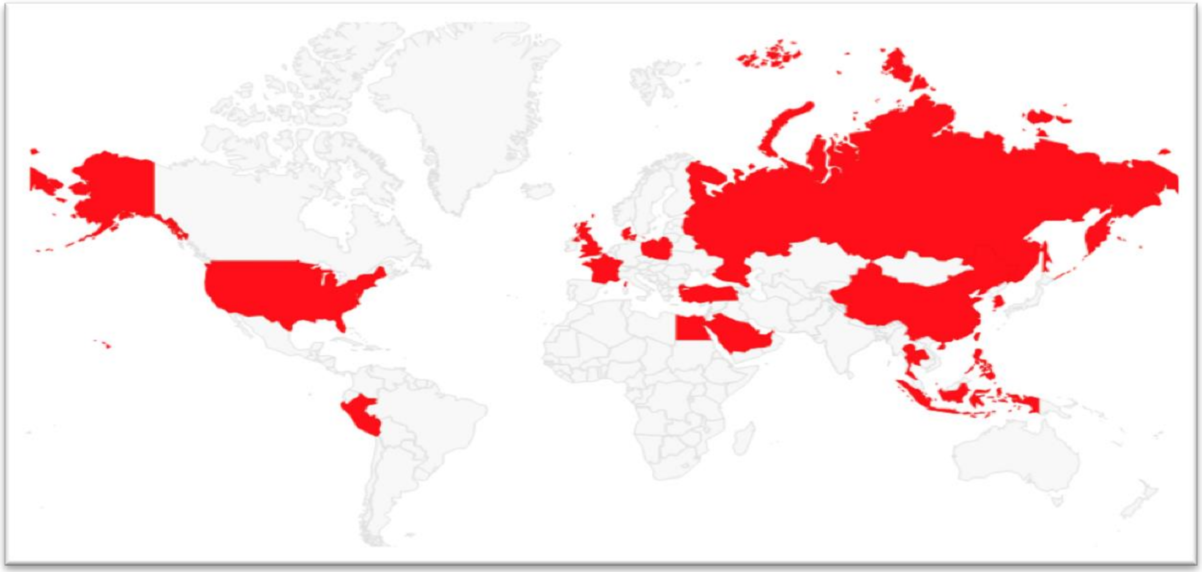
3 Drop 8 root exploits to root the device

4 CnC Server

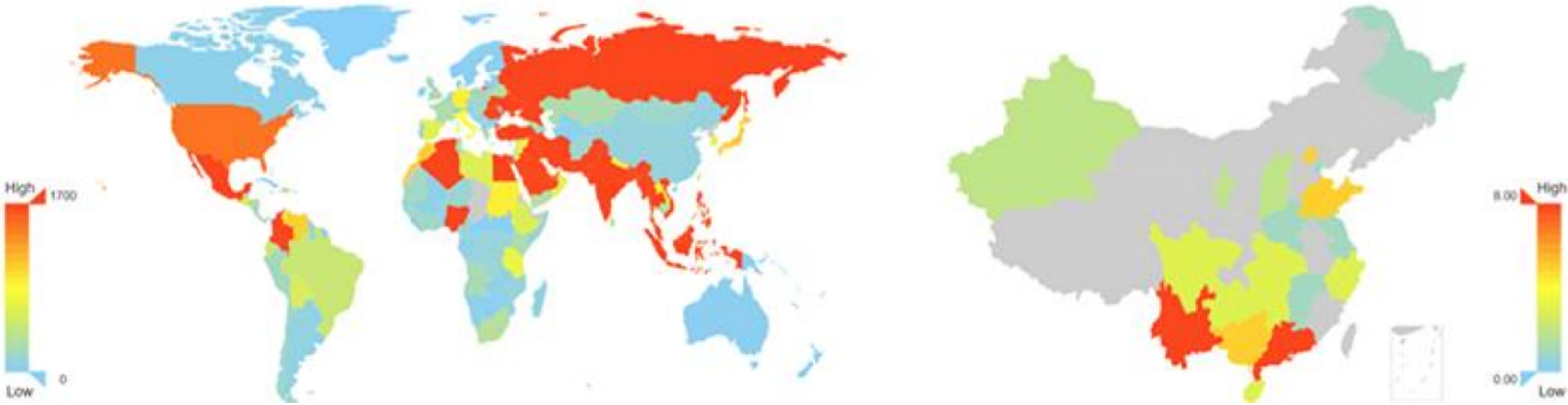
Remote Control:

- Install any app
- Uninstall any app
- Launch any app

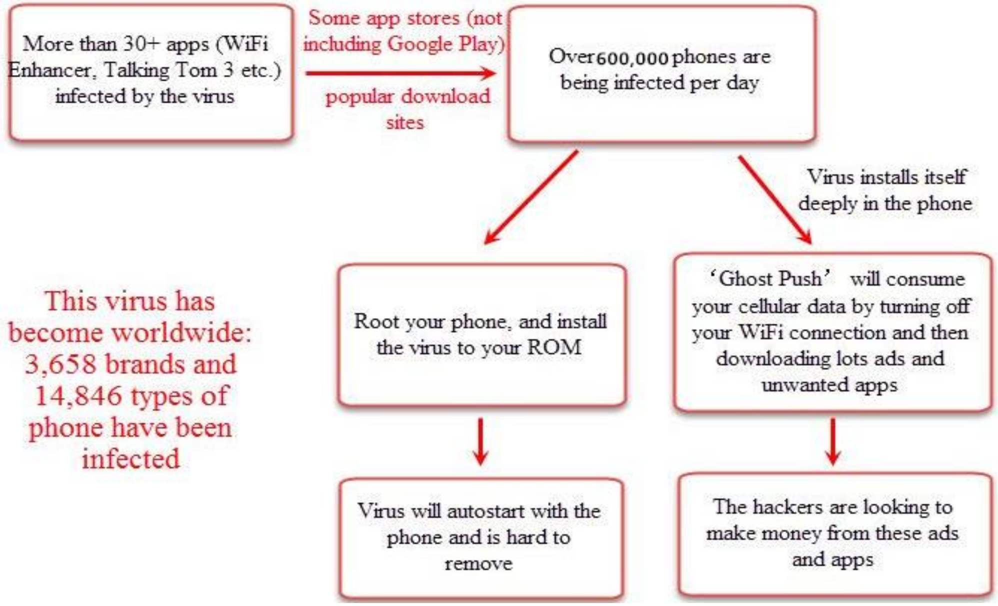
KEMOGE



Malware/Adware with Root Exploits



GHOSTPUSH



More than 30+ apps (WiFi Enhancer, Talking Tom 3 etc.) infected by the virus

Some app stores (not including Google Play)

popular download sites

Over 600,000 phones are being infected per day

Virus installs itself deeply in the phone

Root your phone, and install the virus to your ROM

'Ghost Push' will consume your cellular data by turning off your WiFi connection and then downloading lots ads and unwanted apps

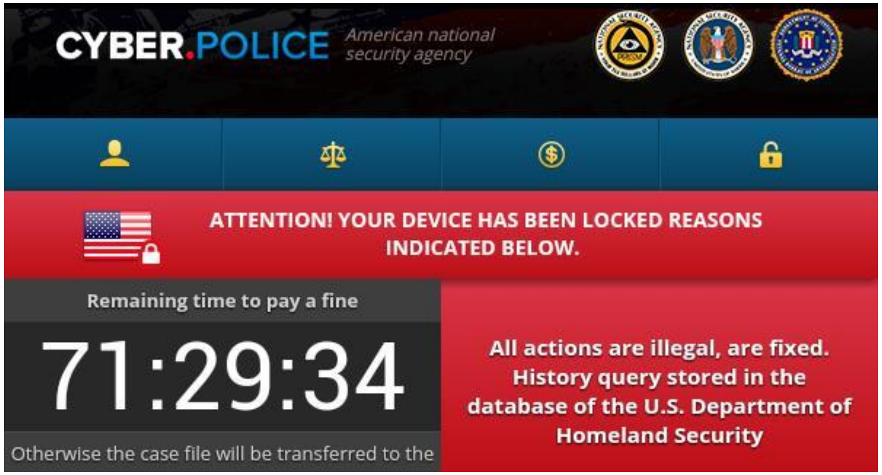
Virus will autostart with the phone and is hard to remove

The hackers are looking to make money from these ads and apps

This virus has become worldwide: 3,658 brands and 14,846 types of phone have been infected

Malware/Adware with Root Exploits

> “This is the first time, to my knowledge; an exploit kit has been able to successfully install malicious apps on a mobile device without any user interaction on the part of the victim... the payload of that exploit, a Linux ELF executable named module.so, contains the code for **the futex or Towelroot exploit** that was first disclosed at the end of 2014.”



CYBER.POLICE *American national security agency*

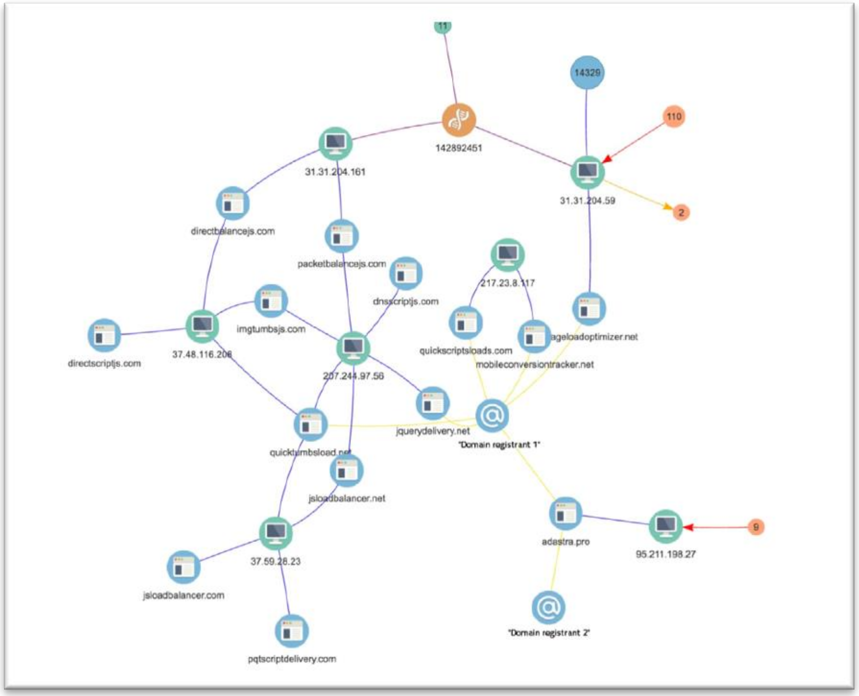
ATTENTION! YOUR DEVICE HAS BEEN LOCKED REASONS INDICATED BELOW.

Remaining time to pay a fine

71:29:34

Otherwise the case file will be transferred to the

All actions are illegal, are fixed. History query stored in the database of the U.S. Department of Homeland Security



DOGSPECTUS

Outline

- Android Kernel Vulnerability Landscape
- The Problem
  - – Devices Unpatched Forever/for A Long Period
  - – Difficult to Patch due to Fragmentation
- The Solution: Adaptive Kernel Live Patching
- Establishing the Ecosystem

iOS More Secure?

Kernel Vulnerability Disclosure Frequency Is Comparable

iOS Version	Date	Count
8.4.1	8/13/15	3
9	9/16/15	12
9.1	10/21/15	6
9.2	12/8/15	5
9.2.1	1/19/16	4
9.3	3/21/16	9

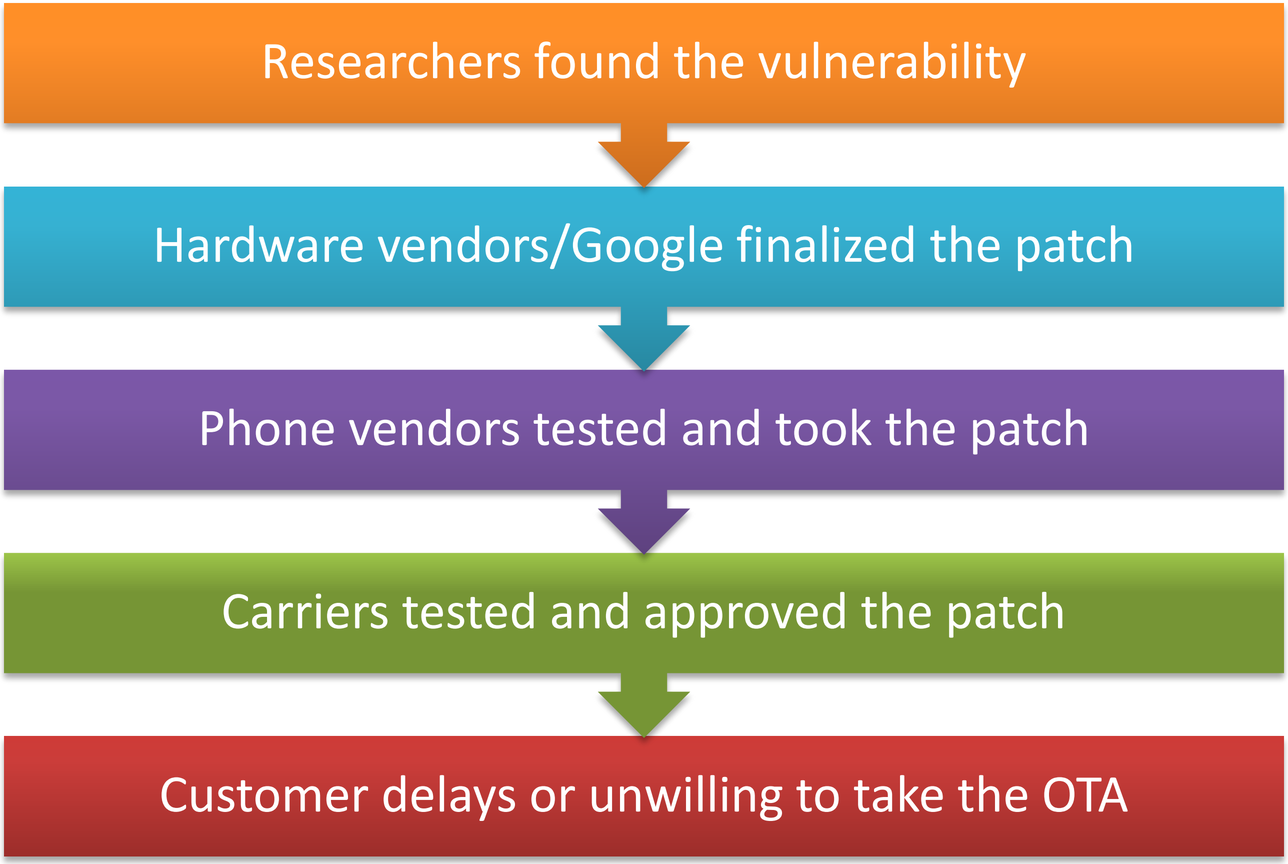
Month	Count
2015/09	1
2015/12	1
2016/01	4
2016/02	4
2016/03	5
2016/04	7
2016/05	15

However…

- If Apple wants to patch a vulnerability
  - Apple controls the entire (mostly) supply chain
  - Apple has the source code
  - Apple refuses to sign old versions, forcing one-direction upgrade
  - All the iOS devices will get update in a timely manner
- Android
  - Many devices stay unpatched forever/for a long period...

Devices Unpatched Forever/for A Long Period

- Cause A: The long patching chain



Researchers found the vulnerability

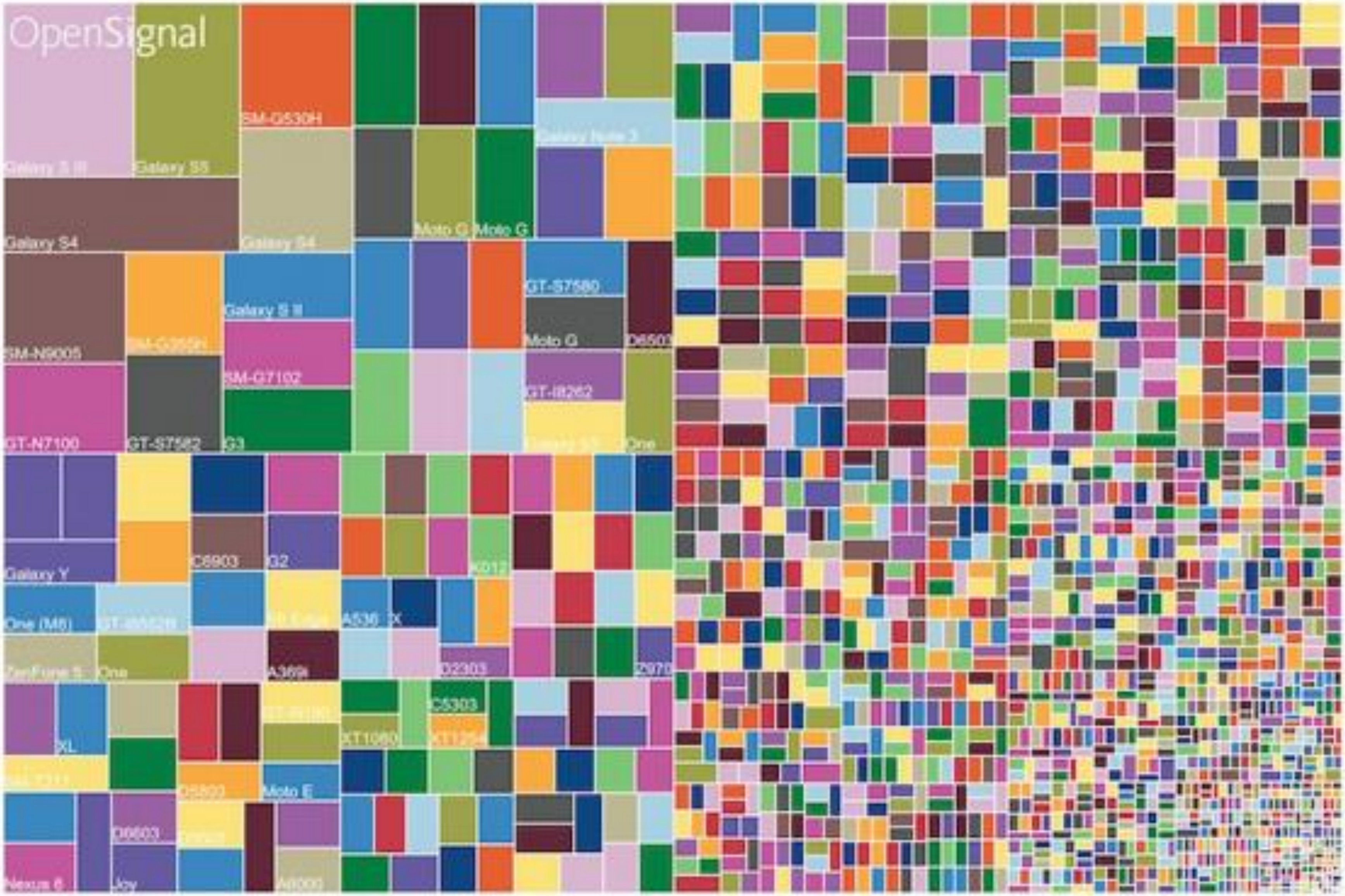
Hardware vendors/Google finalized the patch

Phone vendors tested and took the patch

Carriers tested and approved the patch

Customer delays or unwilling to take the OTA

DEVICE FRAGMENTATION

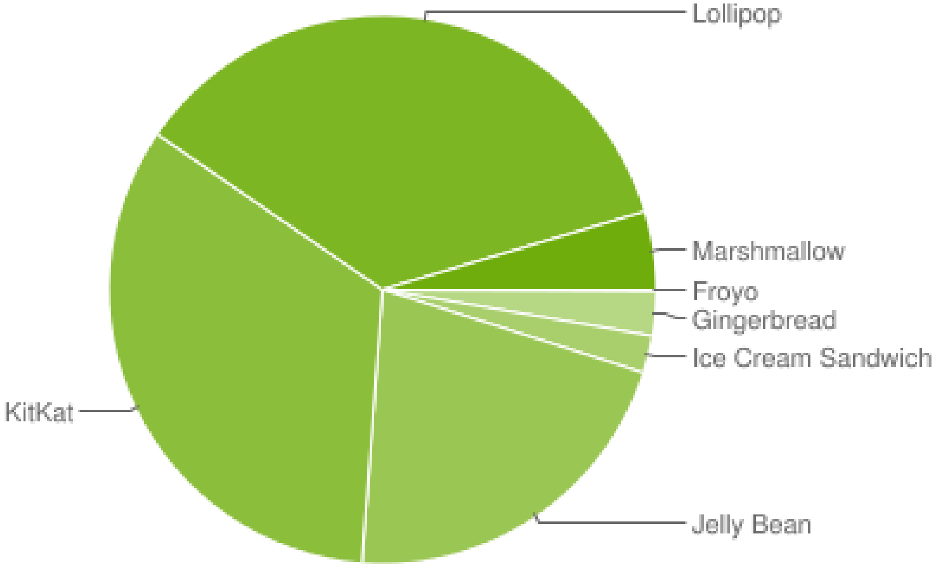


http://opensignal.com/reports/2015/08/android-fragmentation

Device Fragmentation

Google Dashboard (2016/04/04)

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.x	Gingerbread	10	2.6%
4.0.x	Ice Cream Sandwich	15	2.2%
4.1.x	Jelly Bean	16	7.8%
4.2.x		17	10.5%
4.3		18	3.0%
4.4	KitKat	19	33.4%
5.0	Lollipop	21	16.4%
5.1		22	19.4%
6.0	Marshmallow	23	4.6%



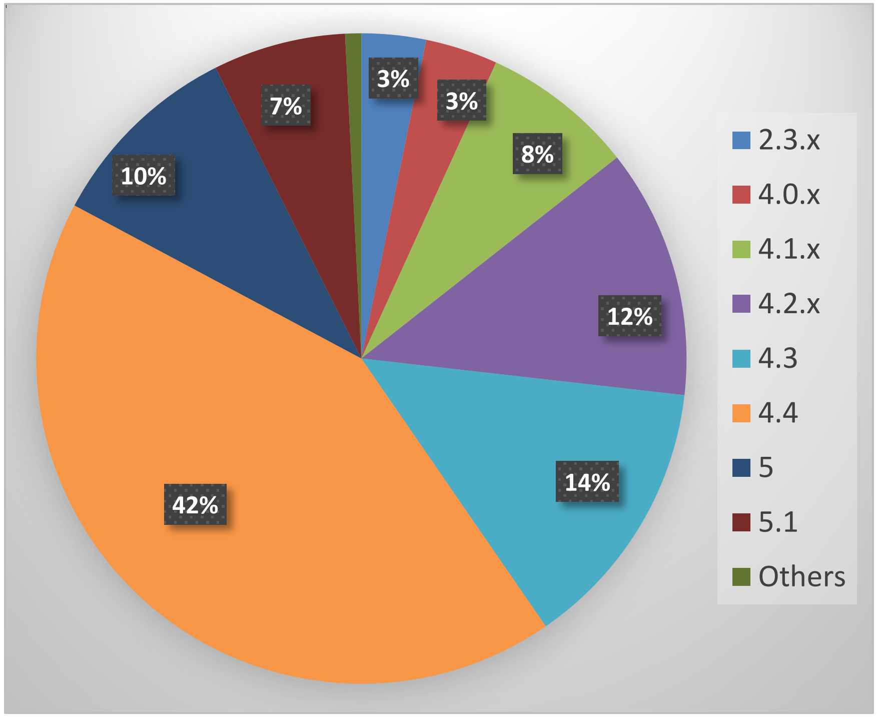
Lollipop was released in November 12, 2014, but **60%** of the devices are still older than that!

Google stopped patching for Android older than 4.4, but **26.2%** of the devices are still older than that!

Chinese Market Is Even Worse

(Stats from devices with Baidu apps installed, 03/21/2016-04/21/2016)

Version	Codename	API	Rate
2.3.x	Gingerbread	10	3.2%
4.0.x	Ice Cream Sandwich	15	3.6%
4.1.x	Jelly Bean	16	7.6%
4.2.x		17	12.4%
4.3		18	13.6%
4.4	KitKat	19	42.4%
5	Lollipop	21	9.8%
5.1		22	6.6%
Others	-	-	0.8%



Lollipop was released in November 12, 2014, but **82.8%** of the devices are still older than that!

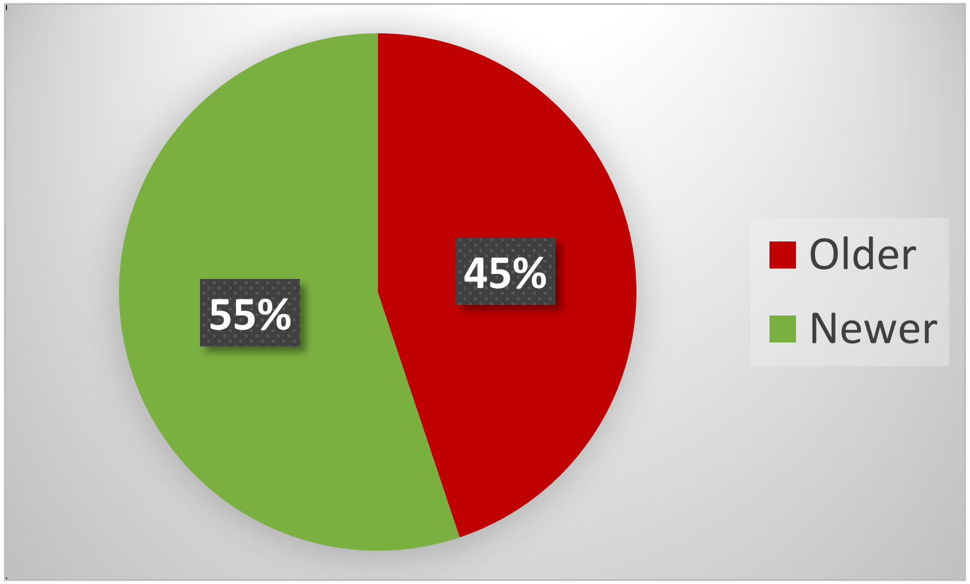
40.4% of the devices are <4.4!

And China **blocks** Google....

Devices with Unpatched Kernels

(Stats from devices with Baidu apps installed, May 2016)

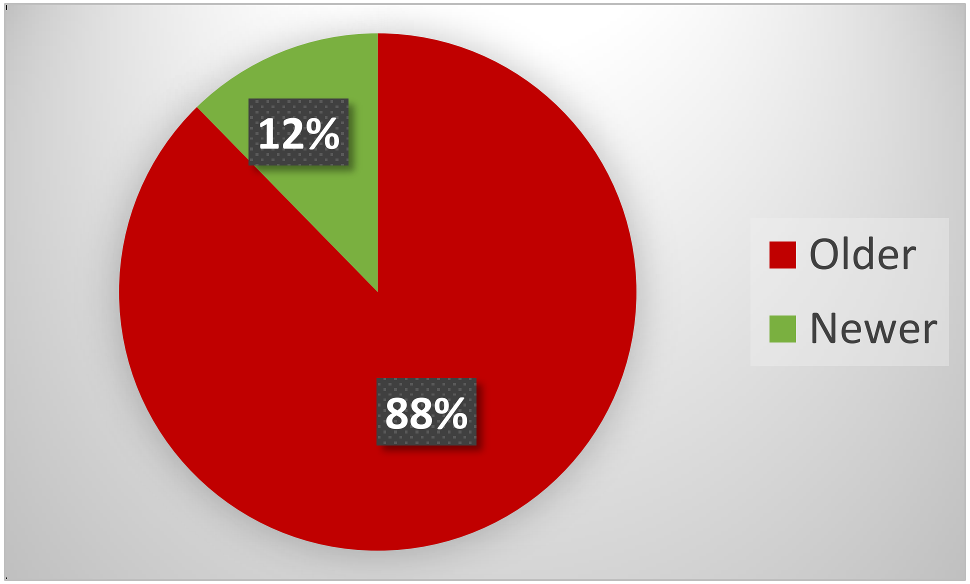
- CVE-2014-3153 (Towelroot)
  - Advisory/Patch Publication Date: Jun. 3rd, 2014
  - Device distribution with kernel build date older/newer than the date:



Devices with Unpatched Kernels

(Stats from devices with Baidu apps installed, May 2016)

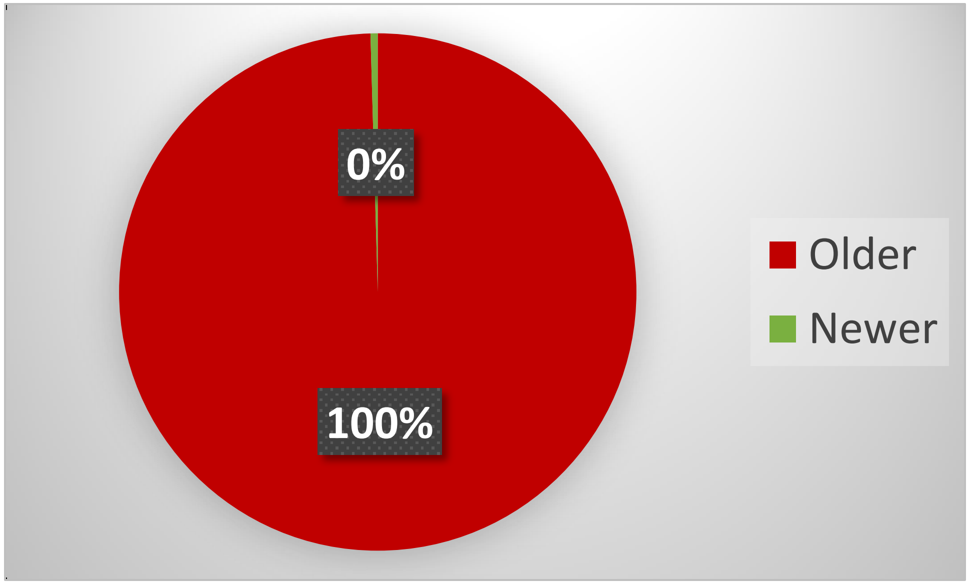
- CVE-2015-3636 (PingPong Root)
  - Advisory/Patch Publication Date: Sep. 9th, 2015
  - Device distribution with kernel build date older/newer than the date:



Devices with Unpatched Kernels

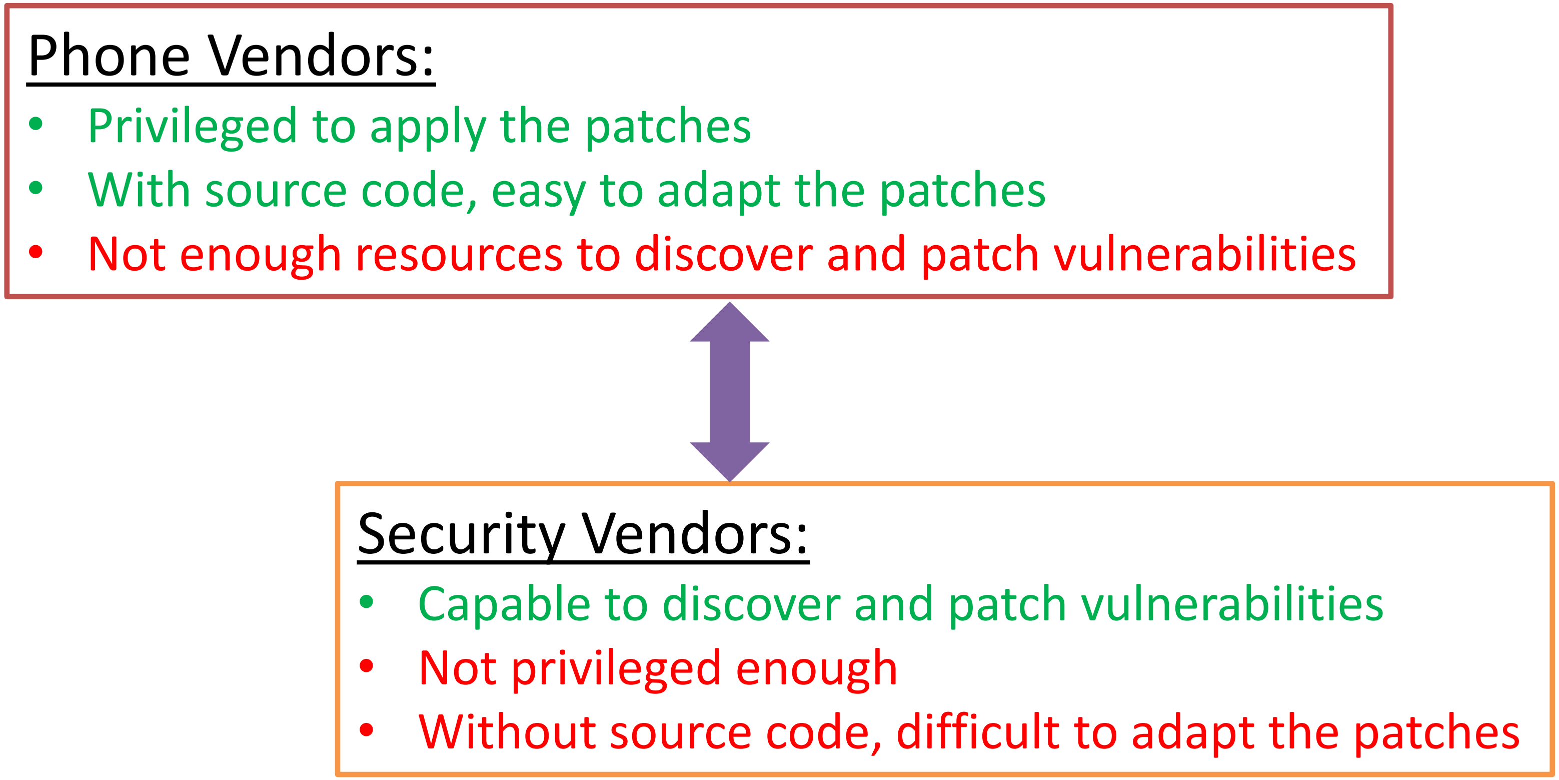
(Stats from devices with Baidu apps installed, May 2016)

- CVE-2015-1805 (used in KingRoot)
  - Advisory/Patch Publication Date: Mar. 18th, 2016
  - Device distribution with kernel build date older/newer than the date:



Devices Unpatched Forever/for A Long Period

- Cause B: Fragmentation & Capability Miss-matching



Phone Vendors:

- Privileged to apply the patches
- With source code, easy to adapt the patches
- Not enough resources to discover and patch vulnerabilities

Security Vendors:

- Capable to discover and patch vulnerabilities
- Not privileged enough
- Without source code, difficult to adapt the patches

Phone Vendors

My first priority is not on vulnerability discovery and real-world exploits…

Security Vendors

So challenging to protect the world…

Google

I've tried my best…

Image sources:
http://conservativetribune.com/wp-content/uploads/2015/12/Donald-Trump-Sad-2.jpg
https://d.gr-assets.com/hostedimages/1417789603ra/12537314.gif
http://1.bp.blogspot.com/-lnMpoEJ4zgk/TknyHEBtD4I/AAAAAAAACRY/6ogSBlPJFWI/s1600/obama%2Bsweats.jpg

How/Who to Secure Them???

PayPal

VISA MasterCard

DISCOVER NETWORK

Google Wallet

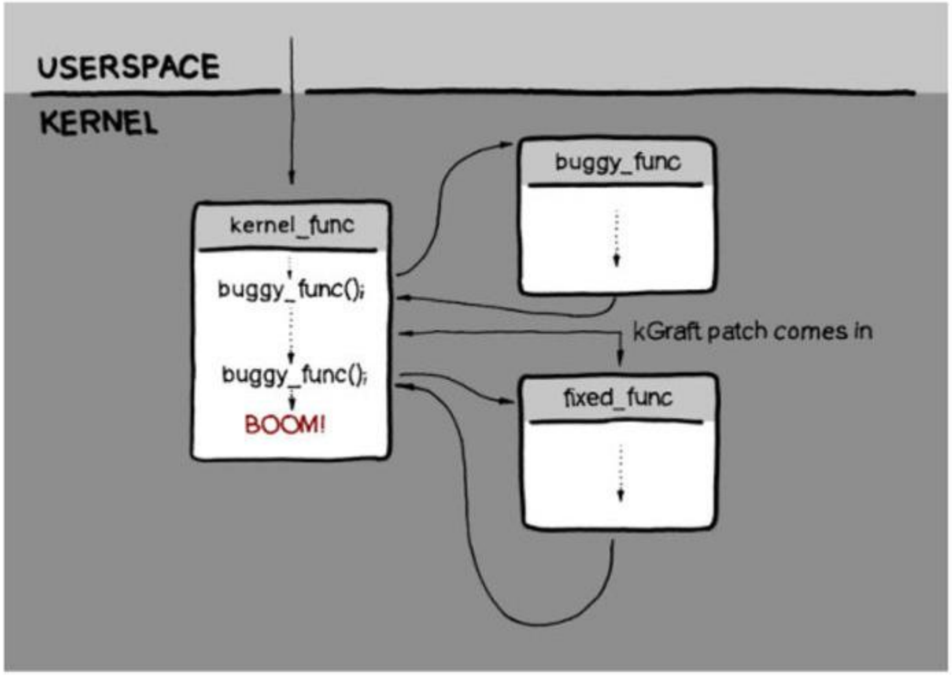
STOCK $

SAMSUNG pay

Outline

- Android Kernel Vulnerability Landscape
- The Problem:
  - Devices Unpatched Forever/for A Long Period
  - Difficult to Patch due to Fragmentation
- The Solution: Adaptive Kernel Live Patching
- Establishing the Ecosystem

Kernel Live Patching



kGraft as an example

Kernel Live Patching

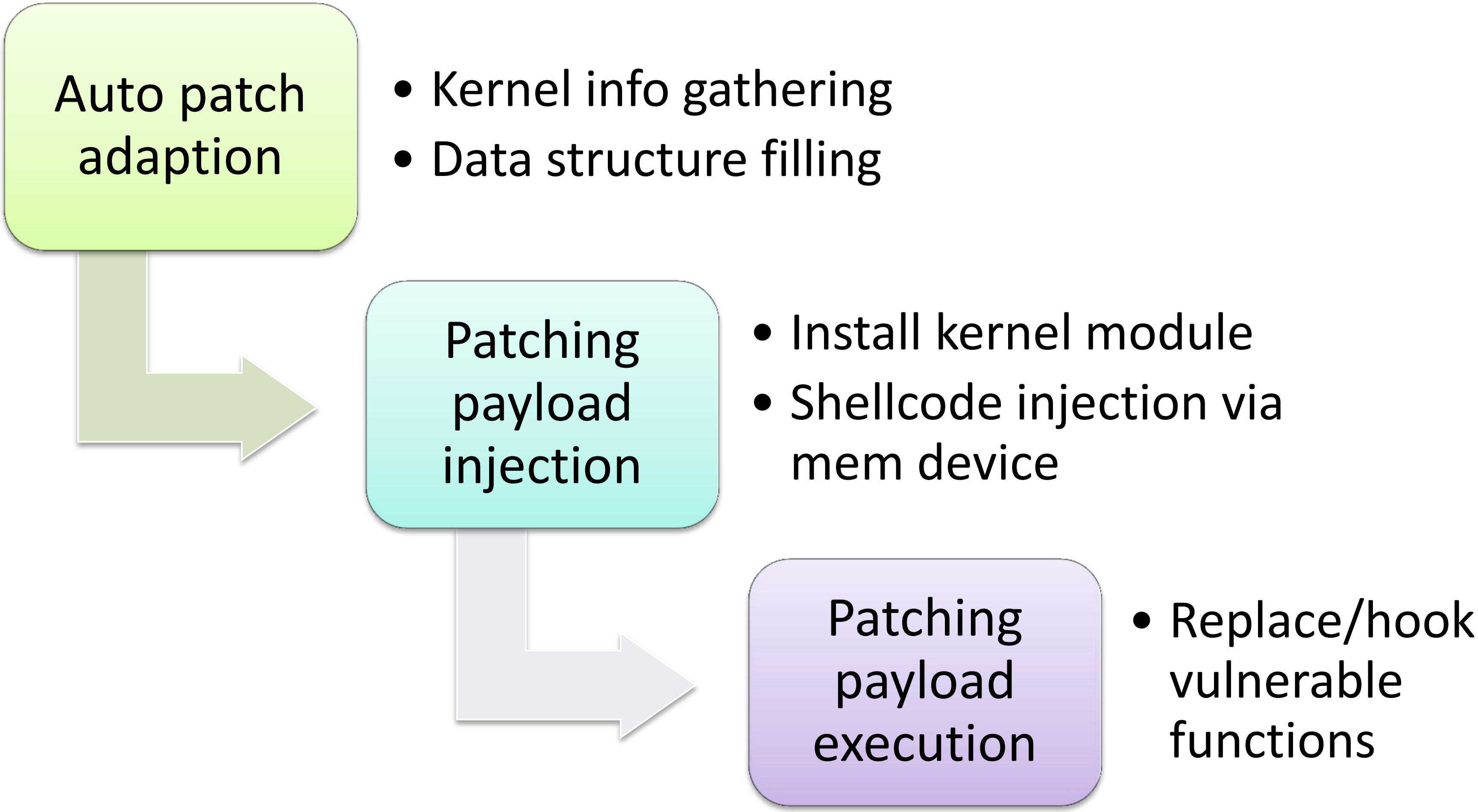
- Load new functions into memory
- Link new functions into kernel
  - Allows access to unexported kernel symbols
- Activeness safety check
  - Prevent old & new functions from running at same time
  - stop_machine() + stack backtrace checks
- Patch it!
  - Uses ftrace etc.

https://events.linuxfoundation.org/sites/events/files/slides/kpatch-linuxcon_3.pdf

Challenges for Third Party

- Most existing work requires source code
  - Phone vendor is the only guy that can generate the live patches
- Unable to directly apply patches to other kernel builds
  - Load code into kernel adaptively

Our Solution - Adaptive Live Patching



Auto patch adaption

- Kernel info gathering
- Data structure filling

Patching payload injection

- Install kernel module
- Shellcode injection via mem device

Patching payload execution

- Replace/hook vulnerable functions

Kernel Info Collection

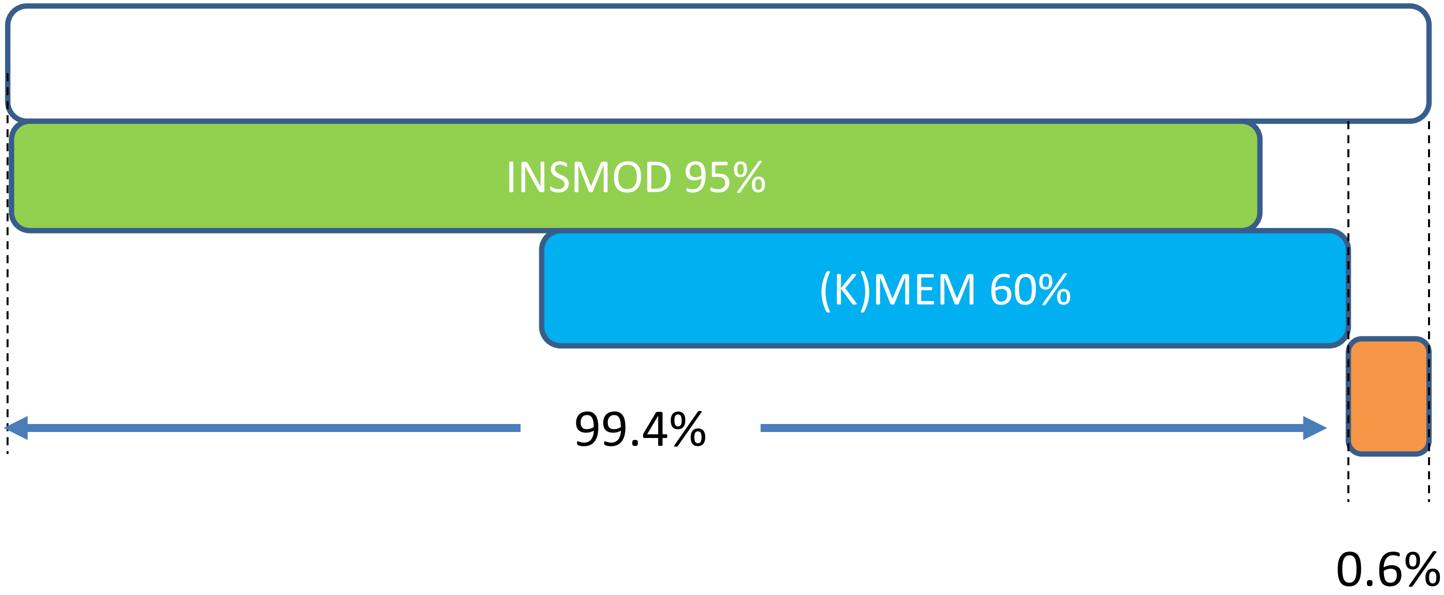
- Kernel version
  - /proc/version
  - vermagic

- Symbol addresses/CRC
  - /proc/kallsyms (/proc/sys/kernel/kptr_restrict)

- Other kernel modules
  - Symbol CRC/module init offset

- Boot image
  - decompress gzip/bzip/lzma/lzo/xz/lz4
  - some are raw code or even ELF file

Patching payload injection Device Coverage

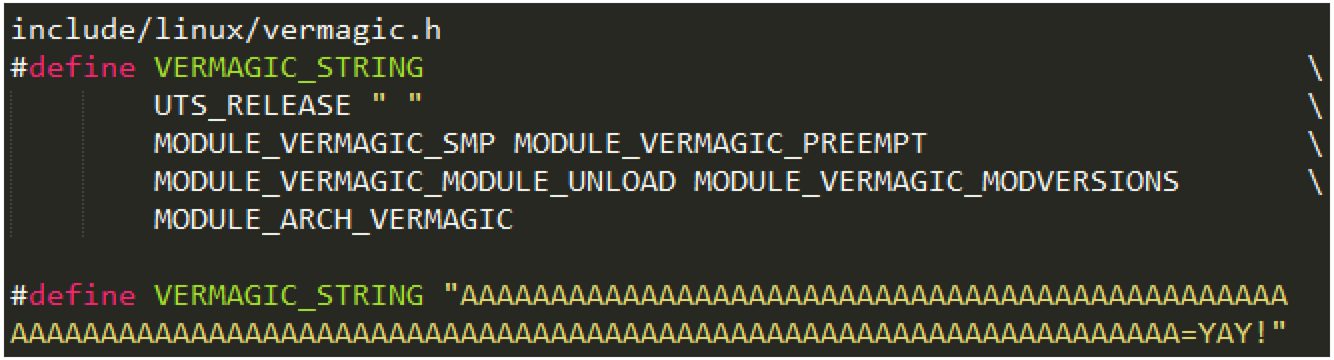


Method A: Kernel Module Injection

- init_module
  - CONFIG_MODVERSIONS
  - CONFIG_MODULE_FORCE_LOAD
- finit_module
  - Linux 3.8+
  - MODULE_INIT_IGNORE_MODVERSIONS
  - MODULE_INIT_IGNORE_VERMAGIC
- restrictions
  - vermagic check
  - symbol CRC check
  - module structure check
  - vendor’s specific check
    - Samsung lkmauth

Bypass vermagic/symbol CRC

- Big enough vermagic buffer
- Copy kernel vermagic string to module
- Copy kernel symbol CRCs to module

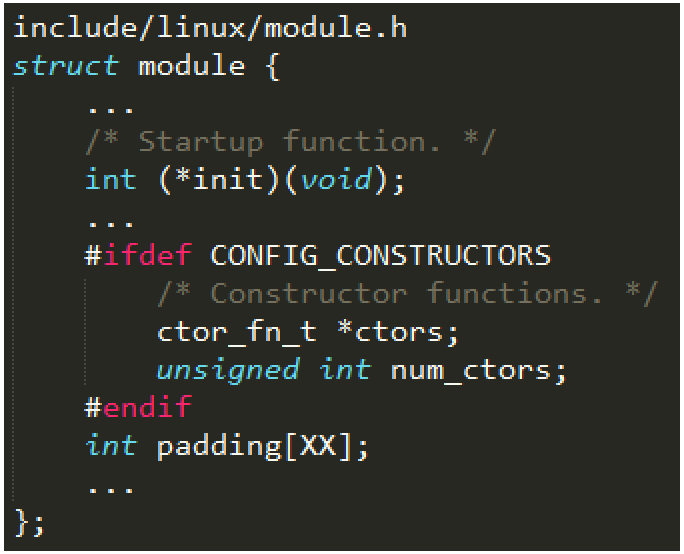


```
include/linux/vermagic.h
#define VERMAGIC_STRING                                                  \
        UTS_RELEASE " "                                                  \
        MODULE_VERMAGIC_SMP MODULE_VERMAGIC_PREEMPT                      \
        MODULE_VERMAGIC_MODULE_UNLOAD MODULE_VERMAGIC_MODVERSIONS        \
        MODULE_ARCH_VERMAGIC

#define VERMAGIC_STRING "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=YAY!"
```

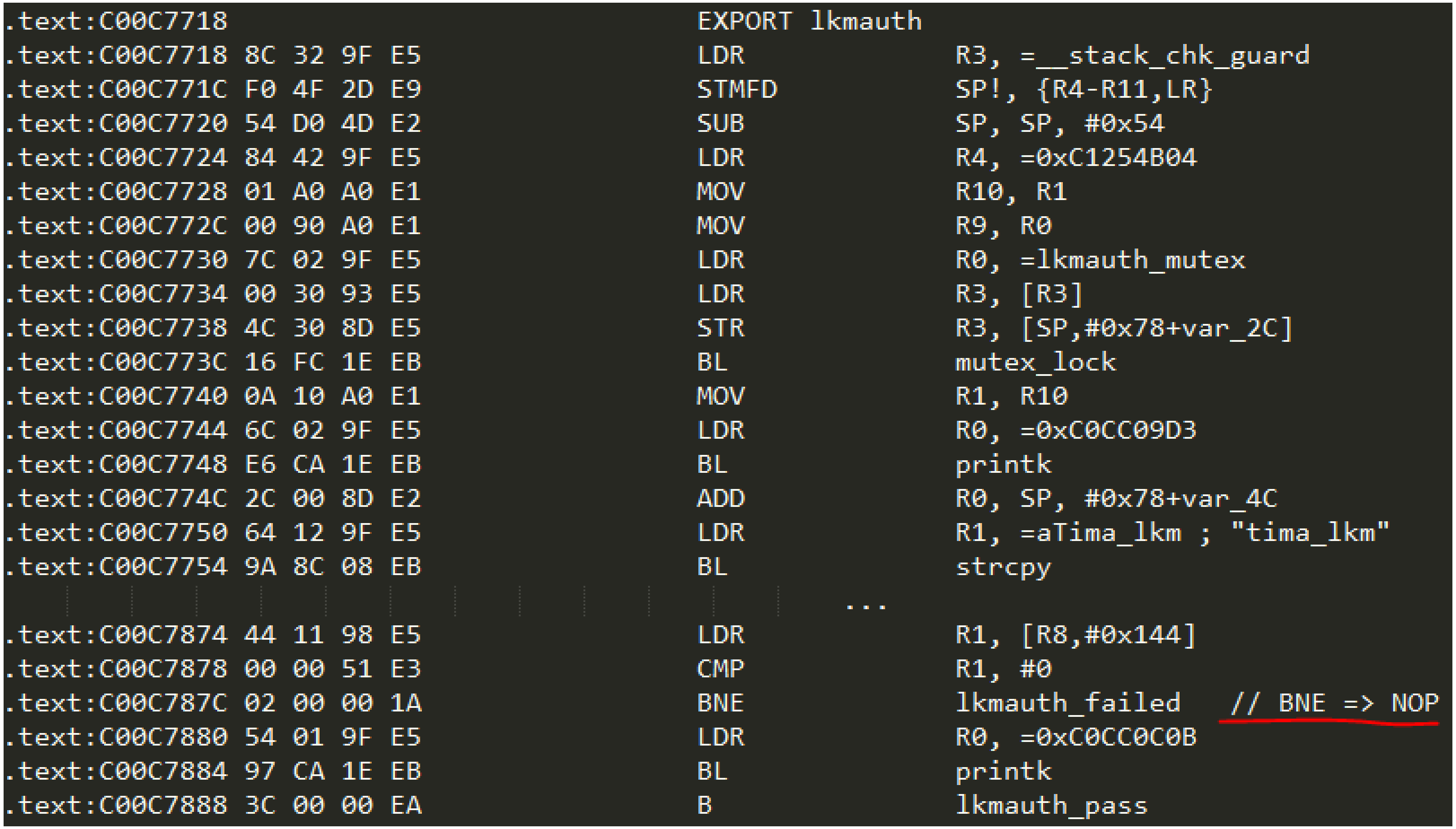
Bypass module structure

- offsetof(init) difference
- Big enough struct module



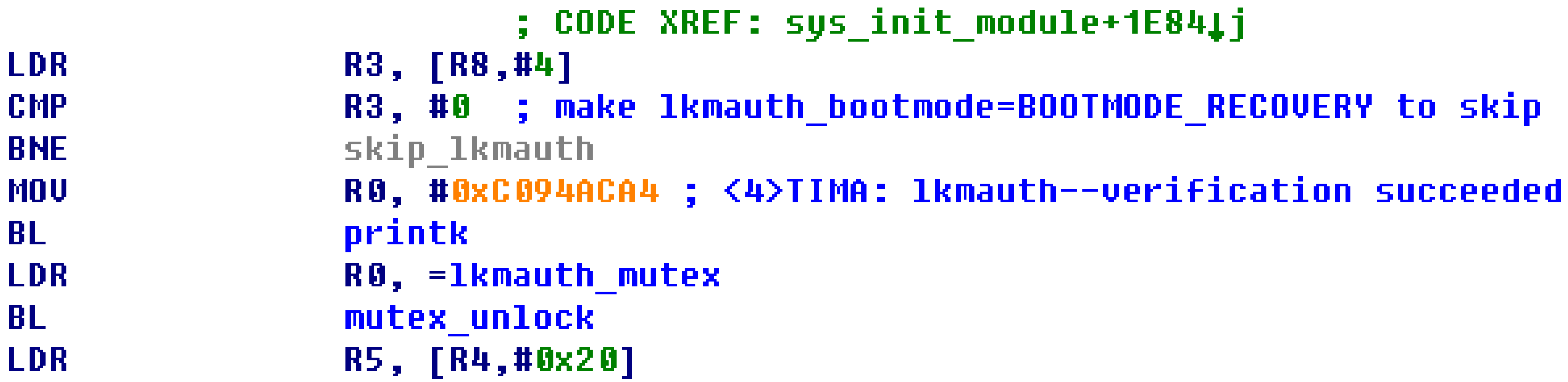
```
include/linux/module.h
struct module {
    ...
    /* Startup function. */
    int (*init)(void);
    ...
    #ifdef CONFIG_CONSTRUCTORS
        /* Constructor functions. */
        ctor_fn_t *ctors;
        unsigned int num_ctors;
    #endif
    int padding[XX];
    ...
};
```

Bypass Samsung lkmauth1

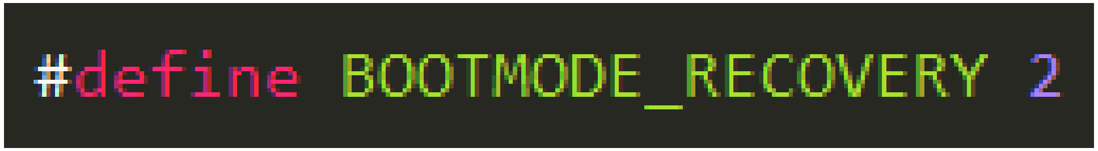


```
.text:C00C7718                          EXPORT lkmauth
.text:C00C7718 8C 32 9F E5              LDR             R3, =__stack_chk_guard
.text:C00C771C F0 4F 2D E9              STMFD           SP!, {R4-R11,LR}
.text:C00C7720 54 D0 4D E2              SUB             SP, SP, #0x54
.text:C00C7724 84 42 9F E5              LDR             R4, =0xC1254B04
.text:C00C7728 01 A0 A0 E1              MOV             R10, R1
.text:C00C772C 00 90 A0 E1              MOV             R9, R0
.text:C00C7730 7C 02 9F E5              LDR             R0, =lkmauth_mutex
.text:C00C7734 00 30 93 E5              LDR             R3, [R3]
.text:C00C7738 4C 30 8D E5              STR             R3, [SP,#0x78+var_2C]
.text:C00C773C 16 FC 1E EB              BL              mutex_lock
.text:C00C7740 0A 10 A0 E1              MOV             R1, R10
.text:C00C7744 6C 02 9F E5              LDR             R0, =0xC0CC09D3
.text:C00C7748 E6 CA 1E EB              BL              printk
.text:C00C774C 2C 00 8D E2              ADD             R0, SP, #0x78+var_4C
.text:C00C7750 64 12 9F E5              LDR             R1, =aTima_lkm ; "tima_lkm"
.text:C00C7754 9A 8C 08 EB              BL              strcpy
                                              ...
.text:C00C7874 44 11 98 E5              LDR             R1, [R8,#0x144]
.text:C00C7878 00 00 51 E3              CMP             R1, #0
.text:C00C787C 02 00 00 1A              BNE             lkmauth_failed   // BNE => NOP
.text:C00C7880 54 01 9F E5              LDR             R0, =0xC0CC0C0B
.text:C00C7884 97 CA 1E EB              BL              printk
.text:C00C7888 3C 00 00 EA              B               lkmauth_pass
```

Bypass Samsung lkmauth2



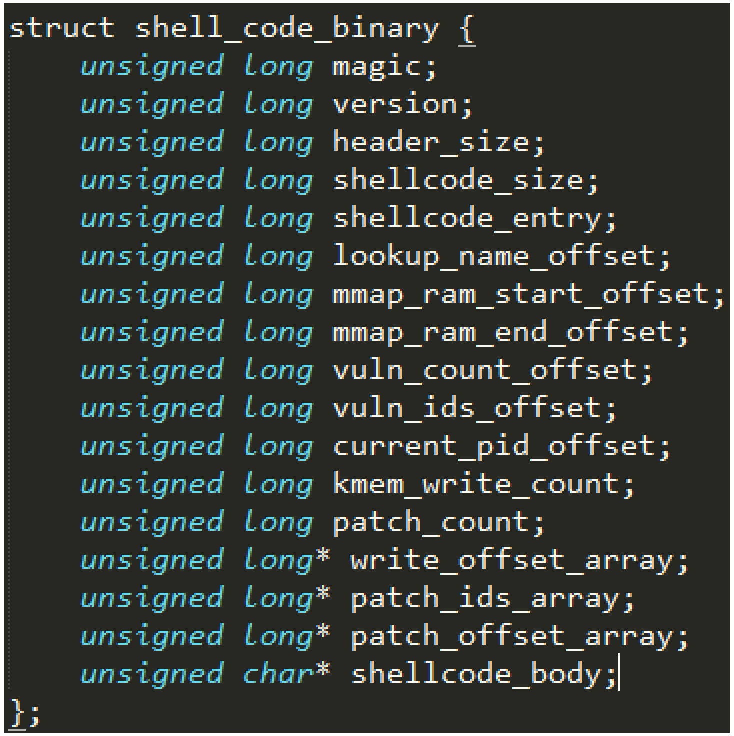
```
                                 ; CODE XREF: sys_init_module+1E84↓j
LDR             R3, [R8,#4]
CMP             R3, #0   ; make lkmauth_bootmode=BOOTMODE_RECOVERY to skip
BNE             skip_lkmauth
MOV             R0, #0xC094ACA4 ; <4>TIMA: lkmauth--verification succeeded
BL              printk
LDR             R0, =lkmauth_mutex
BL              mutex_unlock
LDR             R5, [R4,#0x20]
```



```
#define BOOTMODE_RECOVERY 2
```

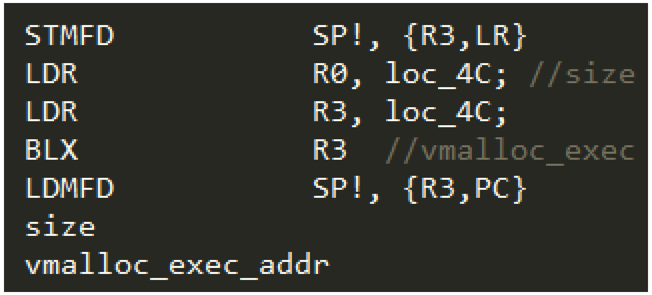
Method B: Shellcode Injection

- Symbol addresses
  - vmalloc_exec
  - module_alloc
- Structured shellcode
- Allocate/reuse memory
- Write into memory
- Trigger the running

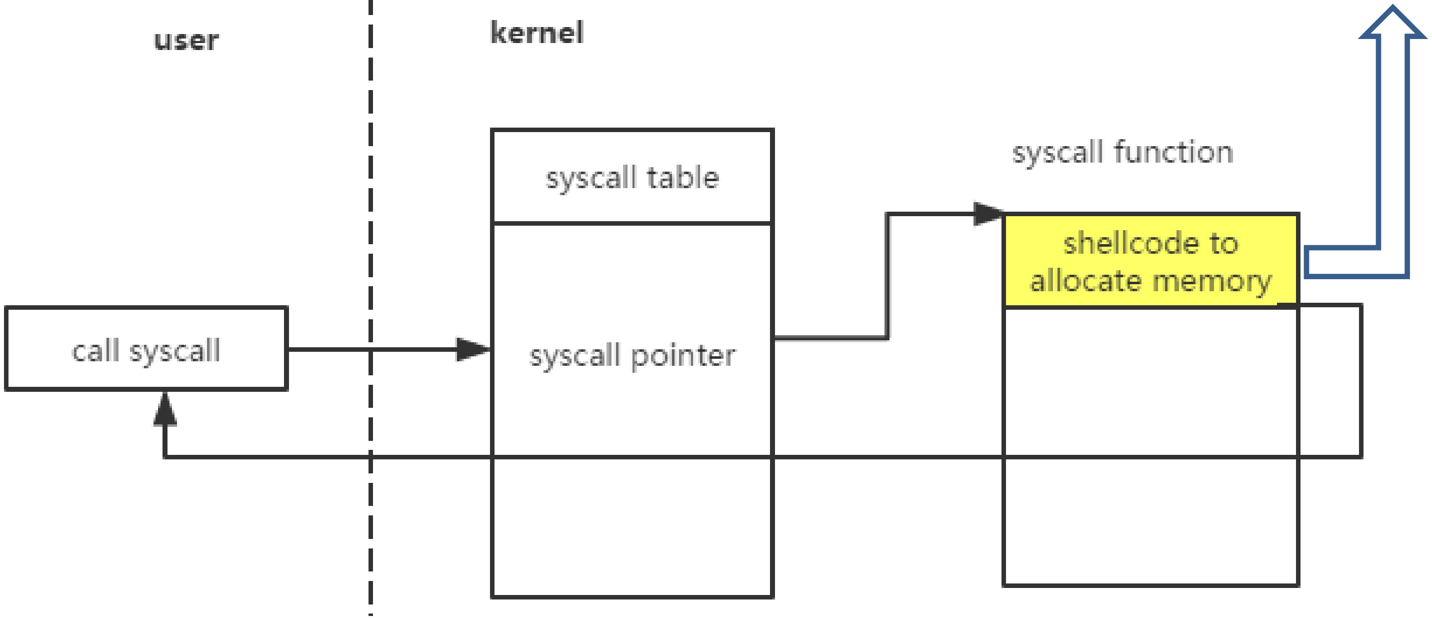


```
struct shell_code_binary {
    unsigned long magic;
    unsigned long version;
    unsigned long header_size;
    unsigned long shellcode_size;
    unsigned long shellcode_entry;
    unsigned long lookup_name_offset;
    unsigned long mmap_ram_start_offset;
    unsigned long mmap_ram_end_offset;
    unsigned long vuln_count_offset;
    unsigned long vuln_ids_offset;
    unsigned long current_pid_offset;
    unsigned long kmem_write_count;
    unsigned long patch_count;
    unsigned long* write_offset_array;
    unsigned long* patch_ids_array;
    unsigned long* patch_offset_array;
    unsigned char* shellcode_body;
};
```

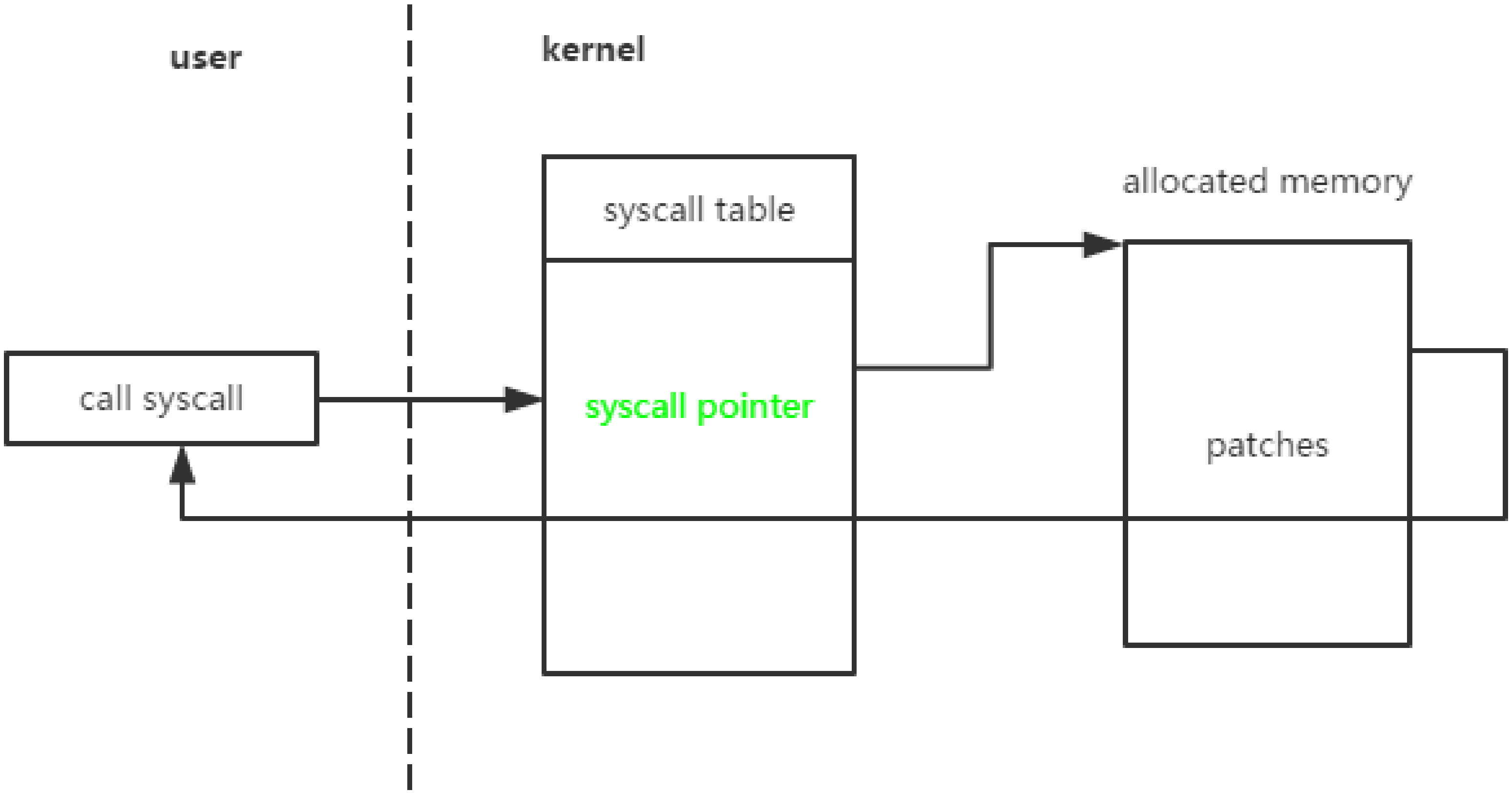
Memory Allocation



```
STMFD           SP!, {R3,LR}
LDR             R0, loc_4C; //size
LDR             R3, loc_4C;
BLX             R3  //vmalloc_exec
LDMFD           SP!, {R3,PC}
size
vmalloc_exec_addr
```



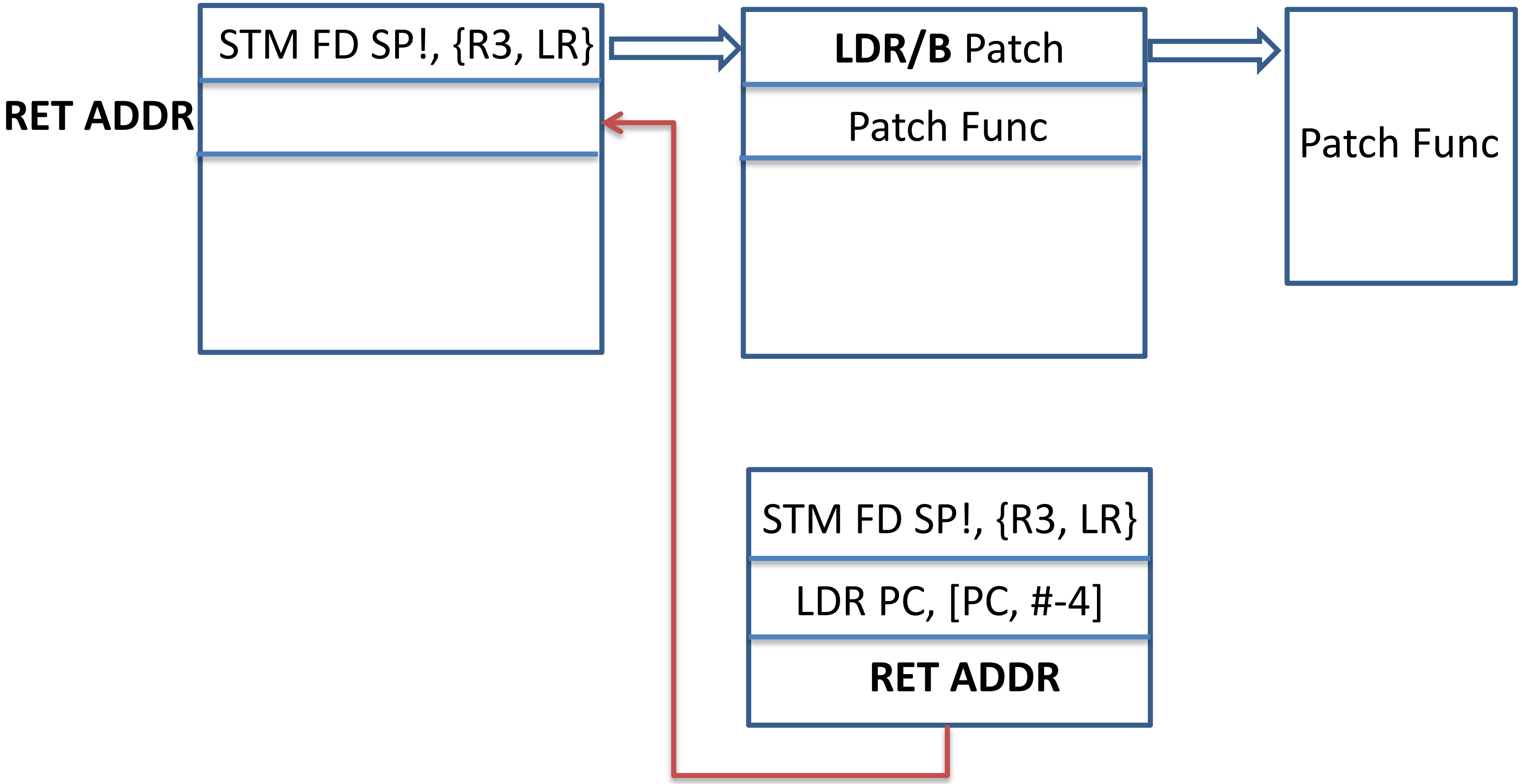
Shellcode Execution



Patching Payload Execution

- Overwrite the function pointer
  - with our own implementation
- Overwrite with patch code directly
  - Need permission, CP15 to help
- Inline hook
  - Atomic with best effort
  - Hook from prolog
  - Hook from middle of the function
    - Need save some context

Vulnerable Function Hook

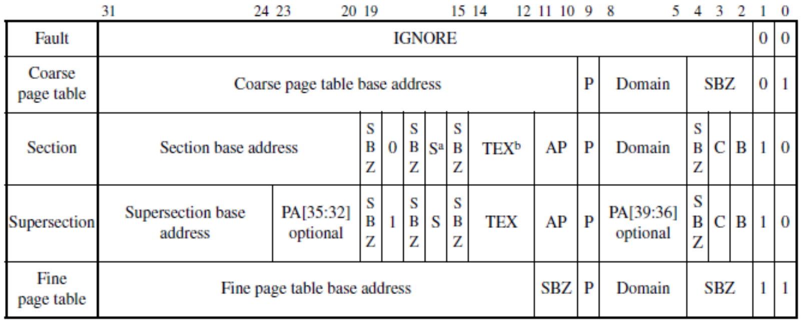


Vulnerable Function Hook(cont.)

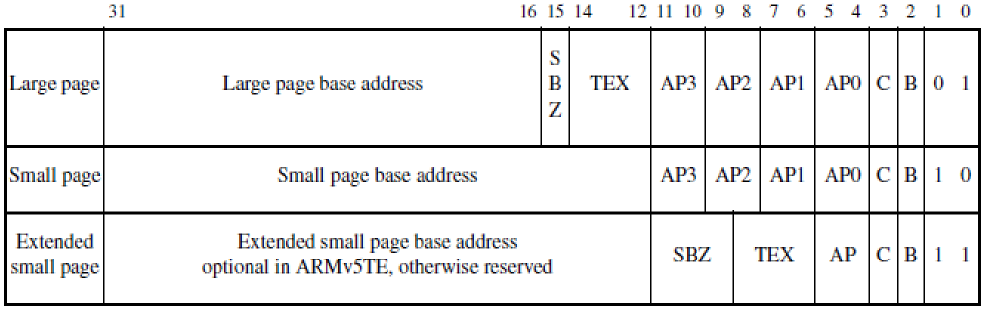
- The patch has the option to execute the original function or just do not
- No option if patch hook from the middle of the vulnerable function
- Painful in 64bit, no explicit operation on PC

Optimizations

- Utilizing kallsyms_lookup_name
  - minimize the symbols imported
- Utilizing existing kernel mem write functions
  - mem_text_write_kernel_word
  - set_memory_rw
- CP15 to change permission



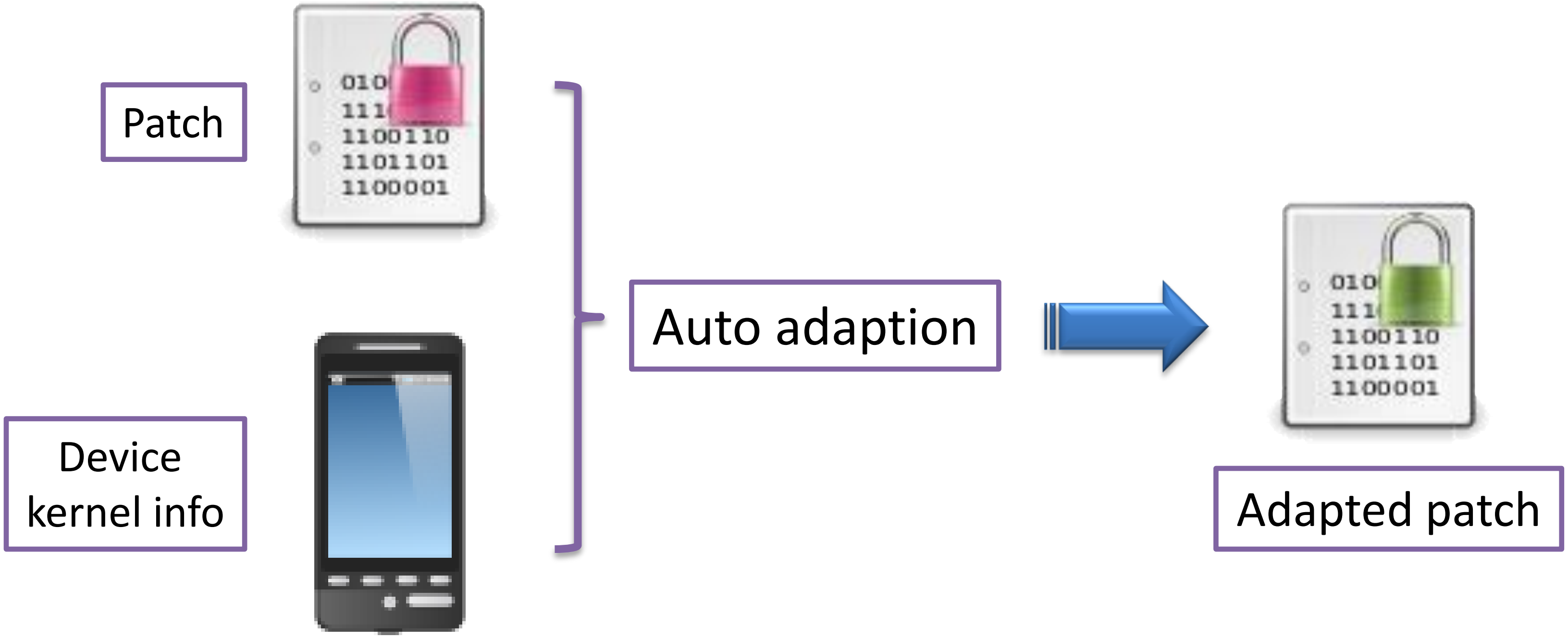
<table>
<tr><th></th><th>31 24</th><th>23 20</th><th>19</th><th></th><th></th><th></th><th>15</th><th>14 12</th><th>11 10</th><th>9</th><th>8 5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>Fault</td><td colspan="14">IGNORE</td><td>0</td><td>0</td></tr>
<tr><td>Coarse page table</td><td colspan="9">Coarse page table base address</td><td>P</td><td>Domain</td><td colspan="3">SBZ</td><td>0</td><td>1</td></tr>
<tr><td>Section</td><td colspan="2">Section base address</td><td>SBZ</td><td>0</td><td>SBZ</td><td>S[a]</td><td>SBZ</td><td>TEX[b]</td><td>AP</td><td>P</td><td>Domain</td><td>SBZ</td><td>C</td><td>B</td><td>1</td><td>0</td></tr>
<tr><td>Supersection</td><td>Supersection base address</td><td>PA[35:32] optional</td><td>SBZ</td><td>1</td><td>SBZ</td><td>S</td><td>SBZ</td><td>TEX</td><td>AP</td><td>P</td><td>PA[39:36] optional</td><td>SBZ</td><td>C</td><td>B</td><td>1</td><td>0</td></tr>
<tr><td>Fine page table</td><td colspan="7">Fine page table base address</td><td>SBZ</td><td>P</td><td>Domain</td><td colspan="3">SBZ</td><td>1</td><td>1</td></tr>
</table>



<table>
<tr><th></th><th>31 16</th><th>15</th><th>14 12</th><th>11 10</th><th>9</th><th>8</th><th>7 6</th><th>5 4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>Large page</td><td>Large page base address</td><td>SBZ</td><td>TEX</td><td>AP3</td><td colspan="2">AP2</td><td>AP1</td><td>AP0</td><td>C</td><td>B</td><td>0</td><td>1</td></tr>
<tr><td>Small page</td><td colspan="3">Small page base address</td><td>AP3</td><td colspan="2">AP2</td><td>AP1</td><td>AP0</td><td>C</td><td>B</td><td>1</td><td>0</td></tr>
<tr><td>Extended small page</td><td colspan="3">Extended small page base address
optional in ARMv5TE, otherwise reserved</td><td colspan="2">SBZ</td><td colspan="2">TEX</td><td>AP</td><td>C</td><td>B</td><td>1</td><td>1</td></tr>
</table>

Challenges Solved

- No source code & fragmentation problem solved
  - ➢Patch automatic adaption



Patch

Device kernel info

Auto adaption

Adapted patch

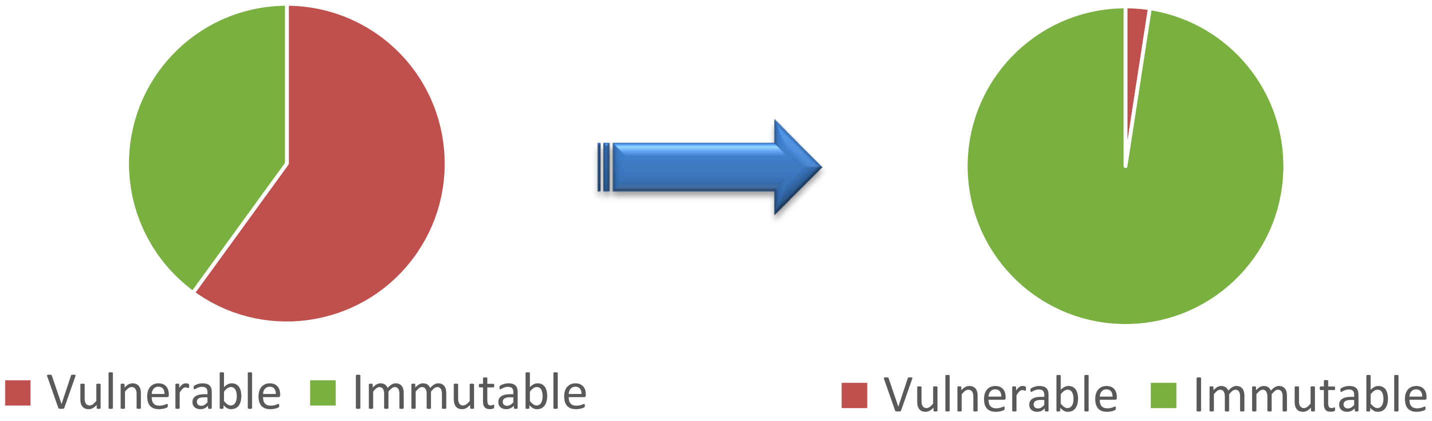
Challenges Solved

✓ Most existing work requires source code

– Phone vendor is the only guy that can generate the live patches

✓ Unable to directly apply patches to other kernel builds

– Load code into kernel adaptively



Successfully Evaluated CVEs

- mmap CVEs (Framaroot)
- CVE-2014-3153 (Towelroot)
- CVE-2015-0569
- CVE-2015-1805
- CVE-2015-3636 (PingPong Root)
- CVE-2015-6640
- CVE-2016-0728
- CVE-2016-0805
- CVE-2016-0819
- CVE-2016-0844
- ……

Successfully Evaluated on Most Popular Phones

GT-I8552

GT-S7572

S4

A7

SM-G5308W

Grand 2

Note 4

SAMSUNG

Successfully Evaluated on Most Popular Phones

C8813

P6-U06

Hornor

U8825D

HUAWEI

Successfully Evaluated on Most Popular Phones

M7

M8Sw

S720e

T528d

htc

Successfully Evaluated on Most Popular Phones

A630t

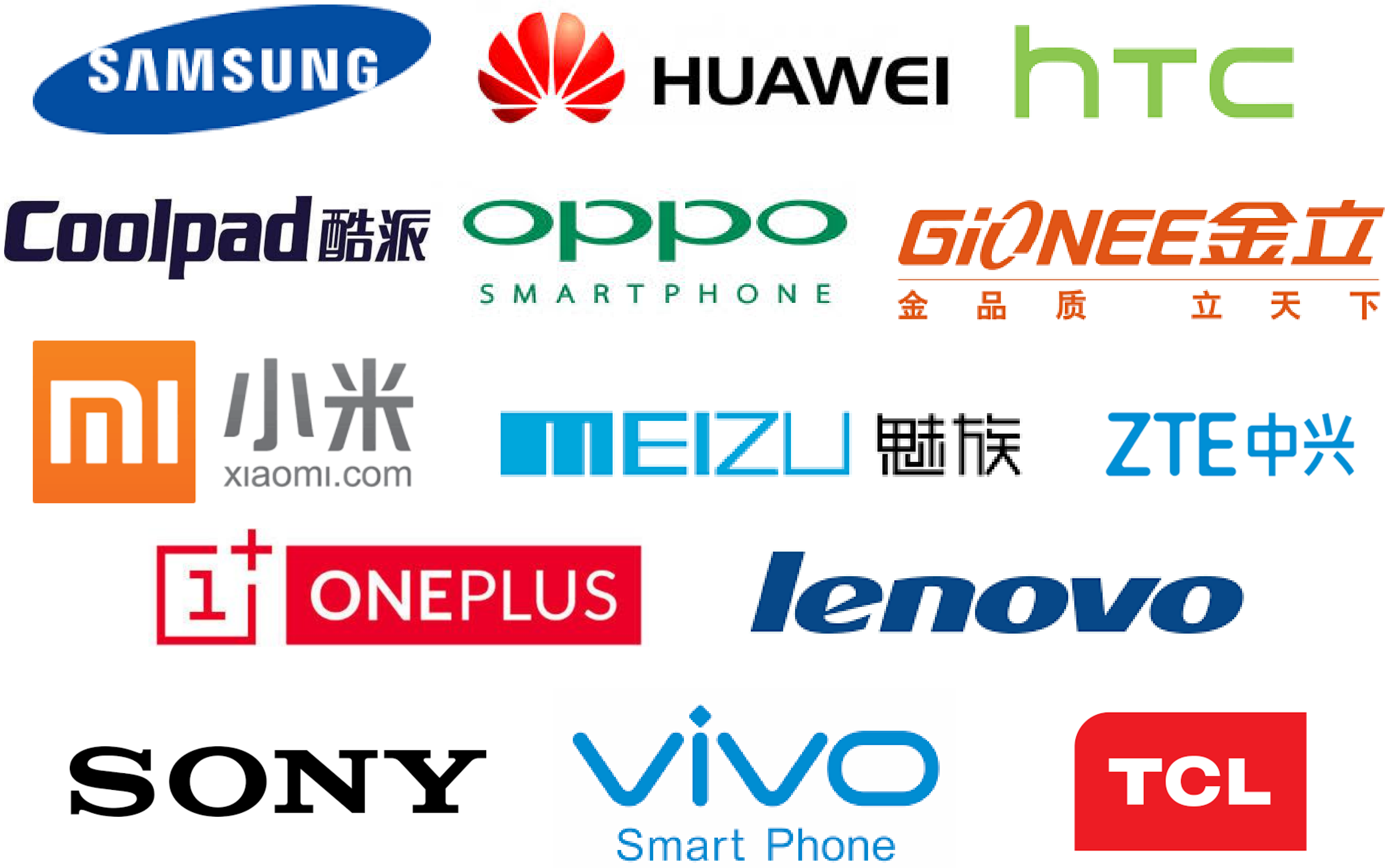
A788t

A938t

K30-T

lenovo

Successfully Evaluated on Most Popular Phones



Demo

Samsung S4

Before Patch: **PingPong** Root succeed

After Patch: **PingPong** Root fail

Outline

- Android Kernel Vulnerability Landscape
- The Problem
  - Devices Unpatched Forever/for A Long Period
  - Difficult to Patch due to Fragmentation
- The Solution: Adaptive Kernel Live Patching
- Establishing the Ecosystem

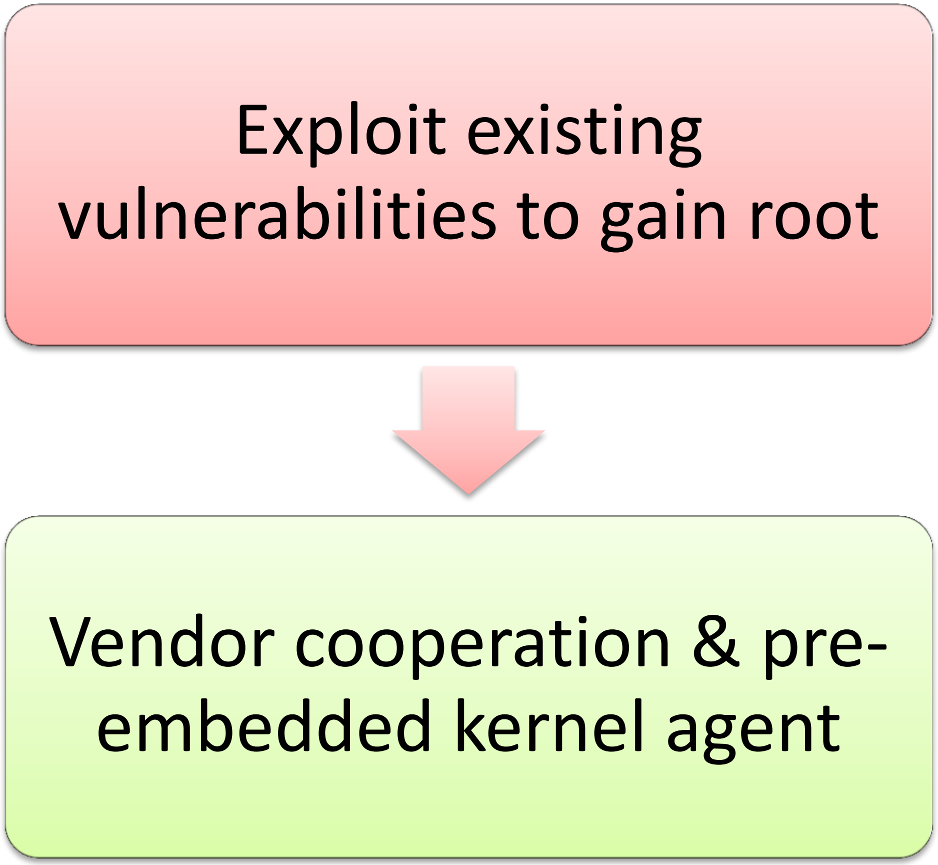
Recall the Two Problems

- The long patching chain
  - Solved by adaptive live patching
- Capability miss-matching
  - To be solved by a joint-effort

Incentives

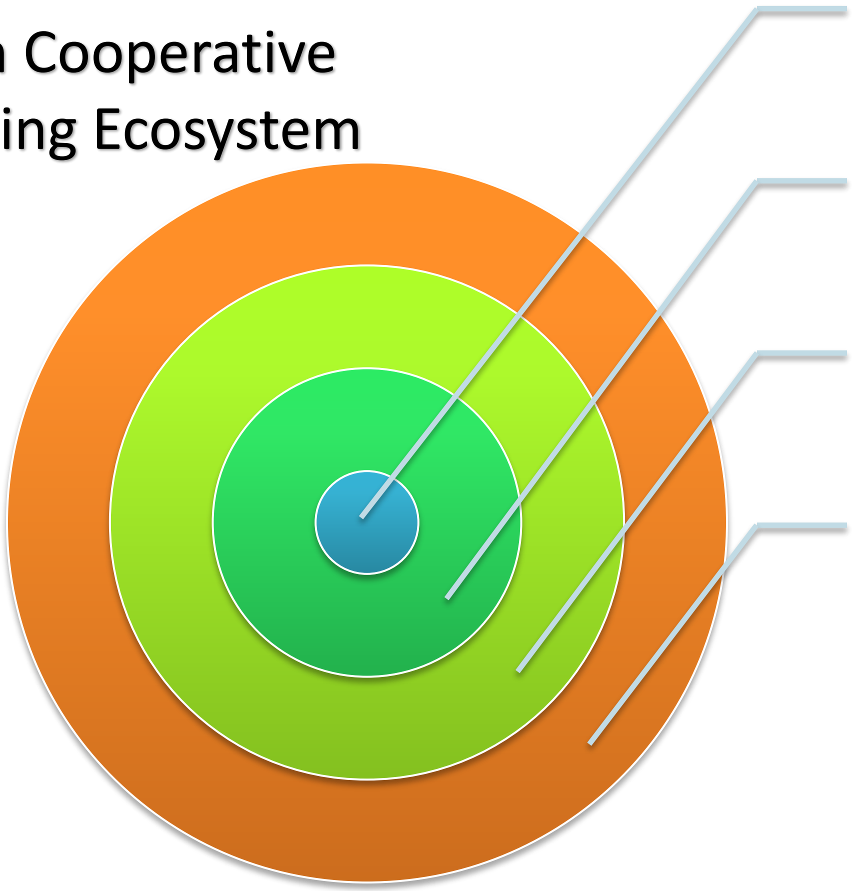
- Vendors
  - More secure products
  - More users & sales
- Security Providers
  - Reputation
  - profits

Transition to Cooperative Patching


Exploit existing vulnerabilities to gain root
Vendor cooperation & pre-embedded kernel agent

Establishing the Ecosystem

Open Cooperative Patching Ecosystem



Vendor qualification

Signature based patch distribution

Security vetting procedure

Reputation ranking

To Be Announced

- Ecosystem alliance
- Flexible & easy-to-review patching mechanism

Thanks!

Tim Xia, Longri Zheng, Yongqiang Lu,
Chenfu Bao, Yulong Zhang, Lenx Wei
Baidu X-Lab
May 2016