Security in Mobile Devices

Hacking Mobiles for Fun and Profit

Tobias Mueller

Universität Hamburg

&

Dublin City University

2010-12-16
1. Hardware Security
2. Platform Security
3. Hacking
4. Q&A
About me

Contact

<table>
<thead>
<tr>
<th>Jabber</th>
<th><a href="mailto:muelli@jabber.ccc.de">muelli@jabber.ccc.de</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACF0 F5EC E9DC 1BDC F09D</td>
</tr>
<tr>
<td></td>
<td>B992 4147 7261 7CB6 4CEF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mail</th>
<th><a href="mailto:muelli@cryptobitch.de">muelli@cryptobitch.de</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CF3E D935 AE6B DE0A D508</td>
</tr>
<tr>
<td></td>
<td>AF86 3EE0 57FF AA20 8D9E</td>
</tr>
</tbody>
</table>

👏 Talk ~ 40 mins
👏 Ask immediately
👏 Q&A afterwards
Motivation
Why the heck?

- Show underlying Technology
- Show Security Frameworks
- Show Exploits in the Wild
- Maybe get you started hacking
- Making you feel responsible

- No Policies
- Not showing anything very new
- No cr4ckz for ur appz
- Explore not exploit
# Why mobile?

<table>
<thead>
<tr>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFi</td>
</tr>
<tr>
<td>Bluetooth</td>
</tr>
<tr>
<td>Email</td>
</tr>
<tr>
<td>Web</td>
</tr>
<tr>
<td>Video (Podcasts?)</td>
</tr>
<tr>
<td>GSM (Calls, Texts)</td>
</tr>
</tbody>
</table>
Why mobile? (cont.)

More than a PC

- Personal Data
- GPS
- Cellular
- Financial Gain/Loss
- Always on
- Infection Not Obvious
- pwn 1 pwn many (cloud syndrome)
Why mobile? (cont.)

However...

- Few publicly known vulnerabilities
- Just PoCs, nobody really exploiting... orly?
Virus Infects 1 Million Cell Phones in China

A virus infecting more than 1 million cell phone users in China is costing users a combined 2 million yuan ($300,000) per day, according to InformationWeek.

ShanghaiDaily.com explains how the virus works:

The 'zombie' virus, hidden in a bogus anti-virus application, can send the phone user's SIM card information to hackers, who then remotely control the phone to send URL links, usually pay-per-click ads, in text messages to contacts in the user's address book.
Outline

1. Hardware Security
   - Complexity
   - Buffer Overflow
     - Function Calls
     - Overwrite Ret Addr
   - Shellcode
   - Protection

2. Platform Security

3. Hacking

4. Q&A
x86 vs. ARM
What's different then?

Classic Vulnerabilities/Architecture revisited:
- Opcodes
- Buffer Overflows
- Endianness
- Format Strings
ARM is much less complex

<table>
<thead>
<tr>
<th>Opcodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage: N900: Cortex A8, N800: ARM 9E</td>
</tr>
<tr>
<td>ARM, MIPS, SPARC: 4 bytes, “NOP”: 4 bytes</td>
</tr>
<tr>
<td>(ARM with THUMBS: 2 bytes)</td>
</tr>
<tr>
<td>x86: omgwtf NOP: 1 byte</td>
</tr>
</tbody>
</table>
Remember f0 0f c7 c8?

Admittedly, it’s old: 1997, but still interesting

```
lock cmpxchg8b eax
```

*Using the LOCK prefix on this form of CMPXCHG8B is illegal in and of itself. LOCK prefixes are only allowed on memory-based read-modify-write instructions. Hence a LOCK prefix on the register-based CMPXCHG8B EAX instruction should also generate an invalid opcode exception.*
function calls

call label
next instruction

...  
label:
push %ebp
mov %esp, %ebp
sub $0x08,%esp
do something interesting
mov %ebp, %esp
pop %ebp
ret

the stack

\[ \begin{array}{c|c}
0xFF & \ldots \\
\hline
\ldots & \ldots \\
\hline
\ldots & \text{return address} \\
\hline
%ebp & %ebp \\
\hline
bytebuffer & \\
\hline
0x00 & \\
\end{array} \]

\[ \leftarrow %ebp \]

\[ \leftarrow %esp \]
function calls

- call label
- next instruction
- ...
- label:
  - push %ebp
- mov %esp, %ebp
- sub $0x08,%esp
- do something interesting
- mov %ebp, %esp
- pop %ebp
- ret
function calls

call label

next instruction

... 

label:
push %ebp

mov %esp, %ebp

sub $0x08,%esp

do something interesting

mov %ebp, %esp

pop %ebp

ret
function calls

- call label
- next instruction
- ...
- label:
  - push %ebp
- mov %esp, %ebp
- sub $0x08,%esp
- do something interesting
- mov %ebp, %esp
- pop %ebp
- ret

```
label:
  push %ebp
  mov %esp, %ebp
  sub $0x08,%esp
  ; do something interesting
  mov %ebp, %esp
  pop %ebp
ret
```

the stack

```
↓ 0xFF
  ... ...
  ... return address
  %ebp %ebp %esp
  ... bytebuffer

↑ 0x00
```
function calls

call label

next instruction

... 

label:
push %ebp

mov %esp, %ebp

sub $0x08,%esp

do something interesting

mov %ebp, %esp

pop %ebp

ret

the stack

\[ \begin{array}{c}
\downarrow 0xFF \\
\vdots \\
\vdots \\
\vdots \\
\downarrow 0x00 \\
\end{array} \]

\[ \begin{array}{c}
\text{return address} \\
%ebp \\
\text{bytebuffer} \\
\end{array} \]

\[ \leftarrow %ebp \]

\[ \leftarrow %esp \]
function calls

call label

next instruction

... 

label:
push %ebp

mov %esp, %ebp

sub $0x08,%esp

do something interesting

mov %ebp, %esp

pop %ebp

ret

the stack

\[
\begin{array}{|c|}
\hline
\text{0xFF} & \text{...} \\
\hline
\hline
\hline
\hline
\hline
\hline
\hline
\hline
\hline
\text{return address} \\
\hline
\hline
\text{%ebp} \\
\hline
\hline
\text{bytebuffer} \\
\hline
\hline
\text{\text{0x00}} \\
\hline
\end{array}
\]

← %ebp

← %esp
function calls

- call label
- next instruction
- ...
- label:
  - push %ebp
- mov %esp, %ebp
- sub $0x08,%esp
- do something interesting
- mov %ebp, %esp
- pop %ebp
- ret

the stack

\[
\begin{array}{c|c|c}
\textit{0x00} & \textit{0xFF} & \ldots \\
\hline
& & \ldots \\
& & \ldots \\
& & \ldots \\
& & \text{return address} \\
& & %ebp \\
& & \\
& & \text{bytebuffer} \\
& & \hline
0x00 & & \\
\end{array}
\]

\[\leftarrow %ebp \leftarrow %esp\]
function calls

- call label
- next instruction
- ...
- label:
  - push %ebp
- mov %esp, %ebp
- sub $0x08,%esp
- do something interesting
- mov %ebp, %esp
- pop %ebp
- ret

the stack

```
↓ 0xFF    ...    ← %ebp
    ...       ...
    ...       ...
    return address  ← %esp
    %ebp
    bytebuffer
↑ 0x00
```
function calls

- call label
- next instruction
- ...
- label:
  - push %ebp
- mov %esp, %ebp
- sub $0x08,%esp
- do something interesting
- mov %ebp, %esp
- pop %ebp
- ret = pop %eip

The stack:

\[
\begin{array}{c|c|c}
\downarrow 0xFF & \ldots & \leftarrow %ebp \\
& \ldots & \leftarrow %esp \\
& \ldots & \\
& \text{return address} & \\
& \%ebp & \\
& \text{bytebuffer} & \\
\uparrow 0x00 & & \\
\end{array}
\]
function calls

call label

next instruction

... 

label:
push %ebp

mov %esp, %ebp

sub $0x08,%esp

do something interesting

mov %ebp, %esp

pop %ebp

ret

the stack

↓ 0xFF

... 

↓ return address

%ebp 

↑ bytebuffer

↑ 0x00

← %esp 

← %ebp
#include <stdio.h>
#include <string.h>

void vulnerable(char *source)
{
    char destination[80];
    strcpy(destination, source);
}

void main(int argc, char **argv)
{
    vulnerable(argv[1]);
}
"push *source" #1st arg

call vulnerableFunction

next instruction

... vulnerableFunction:
  pushl %ebp
  movl %esp, %ebp
  subl $80, %esp
  leal -80(%ebp), %eax
  pushl 8(%ebp) # source
  pushl %eax
  call strcpy
  mov %ebp, %esp
  pop %ebp
  ret
Overwrite Return Address

```
“push *source”  #1st arg

#call vulnerableFunction

next instruction

. . .

vulnerableFunction:
pushl %ebp

movl %esp, %ebp

subl $80, %esp

leal -80(%ebp), %eax

pushl 8(%ebp)  # source

pushl %eax

call strcpy

call strpy

movl %esp, %ebp

pop %ebp

ret
```

---

### the stack

<table>
<thead>
<tr>
<th>↓ 0xFF</th>
<th>. . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>*source</td>
<td>. . .</td>
</tr>
<tr>
<td>return address</td>
<td>. . .</td>
</tr>
</tbody>
</table>

---

| ↑ 0x00 | . . . |
Overwrite Return Address

“push *source” #1st arg
call vulnerableFunction
next instruction
.
.
vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp) # source
pushl %eax
call strcpy
mov %ebp, %esp
pop %ebp
ret

the stack

↓ 0xFF

... *

source

return address

%ebp

↑ 0x00

...
Overwrite Return Address

“push *source”  #1st arg
call vulnerableFunction
next instruction
...
vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp) # source
pushl %eax
call strcpy
mov %ebp, %esp
call strcpy
mov %ebp, %esp
pop %ebp
ret

the stack

\[\begin{array}{c|c}
\text{↓ 0xFF} & \ldots \\
\text{*source} & \\
\text{return address} & \\
%ebp & \\
\text{\uparrow 0x00} & \ldots \\
\end{array}\]
“push *source”    #1st arg
call vulnerableFunction
next instruction
...
vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp)  # source
pushl %eax
call strcpy
mov %ebp, %esp
pop %ebp
ret
Overwrite Return Address

- "push *source"  #1st arg
- call vulnerableFunction
- next instruction
- ...
- vulnerableFunction:
  pushl %ebp
  movl %esp, %ebp
  subl $80, %esp
  leal -80(%ebp), %eax
  pushl 8(%ebp) # source
  pushl %eax
  call strcat
  mov %ebp, %esp
  pop %ebp
  ret

```
the stack
```

```
  ↓ 0xFF
  *source
  return address
  %ebp
  buffer[76-79]
  ↓ ... 0x00
  buffer[0-3]
  ↓ ...
```

↑ 0x00
Overwrite Return Address

“push *source” #1st arg
call vulnerableFunction
next instruction...
vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp) # source
pushl %eax
call strcpy
mov %ebp, %esp
pop %ebp
ret

<table>
<thead>
<tr>
<th>Stack Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0xFF )</td>
</tr>
<tr>
<td>*source</td>
</tr>
<tr>
<td>%ebp</td>
</tr>
<tr>
<td>buffer[76-79]</td>
</tr>
<tr>
<td>( \ldots )</td>
</tr>
</tbody>
</table>
"push *source"  #1st arg

call vulnerableFunction

next instruction

... 
vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp)  # source
pushl %eax
call strcpy
movl %esp, %ebp
pop %ebp
ret

<table>
<thead>
<tr>
<th>Stack Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ 0xFF</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>↑ 0x00</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Overwrite Return Address

“push *source” #1st arg
call vulnerableFunction
next instruction
... vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp) # source
pushl %eax
call strcpy
mov %ebp, %esp
pop %ebp
ret

the stack

*source
return address
%ebp
buffer[76-79]
... buffer[0-3]
*source
*destination
...
"push *source"  #1st arg
call vulnerableFunction
next instruction
...
vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp) # source
pushl %eax
call strcpy
mov %ebp, %esp
pop %ebp
ret

---

**the stack**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ 0xFF</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>return address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%ebp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>buffer[76-79]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>buffer[0-3]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*destination</td>
<td></td>
</tr>
<tr>
<td>↑ 0x00</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
“push *source”  #1st arg

call vulnerableFunction

next instruction

...  

vulnerableFunction:
pushl %ebp

movl %esp, %ebp

subl $80, %esp

leal -80(%ebp), %eax

pushl 8(%ebp) # source

pushl %eax

call strcpy

mov %ebp, %esp

pop %ebp

ret
“push *source” #1st arg
call vulnerableFunction
next instruction
... 
vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp) # source
pushl %eax
call strcpy
mov %ebp, %esp
pop %ebp
ret = pop %eip
“push *source”  #1st arg

call vulnerableFunction

next instruction

....

vulnerableFunction:
pushl %ebp

movl %esp, %ebp

subl $80, %esp

leal -80(%ebp), %eax

pushl 8(%ebp) # source

pushl %eax

call strcpy

mov %ebp, %esp

call strcpy

mov %ebp, %esp

pop %ebp

ret
Overwrite Return Address

“push *source” #1st arg
call vulnerableFunction
next instruction
...
vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp) # source
pushl %eax
call strcpy
mov %ebp, %esp
pop %ebp
ret

the stack

↓ 0xFF

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*source</td>
<td></td>
</tr>
<tr>
<td>return address</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td></td>
</tr>
<tr>
<td>buffer[76-79]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>buffer[0-3]</td>
<td></td>
</tr>
<tr>
<td>*source</td>
<td></td>
</tr>
<tr>
<td>*destination</td>
<td></td>
</tr>
</tbody>
</table>

↑ 0x00

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overwrite Return Address

“push *source” #1st arg
call vulnerableFunction
next instruction
...
vulnerableFunction:
pushl %ebp
movl %esp, %ebp
subl $80, %esp
leal -80(%ebp), %eax
pushl 8(%ebp) # source
pushl %eax
call strcpy
mov %ebp, %esp
pop %ebp
ret

<table>
<thead>
<tr>
<th>the stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ 0xFF</td>
</tr>
<tr>
<td>*source</td>
</tr>
<tr>
<td>buffer</td>
</tr>
<tr>
<td>buffer</td>
</tr>
<tr>
<td>buffer[76-79]</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>buffer[0-3]</td>
</tr>
<tr>
<td>*source</td>
</tr>
<tr>
<td>*destination</td>
</tr>
<tr>
<td>↑ 0x00</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
“push *source”  #1st arg

call vulnerableFunction

next instruction

... 

vulnerableFunction:
pushl %ebp

movl %esp, %ebp

subl $80, %esp

leal -80(%ebp), %eax

pushl 8(%ebp)  # source

pushl %eax

call strcpy

mov %ebp, %esp

pop %ebp

ret
"push *source"  #1st arg
call vulnerableFunction
next instruction...
vulnerableFunction:
  pushl %ebp
  movl %esp, %ebp
  subl $80, %esp
  leal -80(%ebp), %eax
  pushl 8(%ebp)  # source
  pushl %eax
call strcpy
  mov %ebp, %esp
  pop %ebp
  ret = pop %eip
BOOOOM!!!11oneone
Buffer Overflow

**BOF on x86 :-)**

- How it generally works
- Why it works so well

**BOF on ARM :-(**

- 1 level of nesting
- overwrite a lot of bytes to hit saved return address
- Jumping to NOP Slide hard, b/c alignment (Format Strings)
- Off by one: Endianess issues

But possible and doable
Shellcode

Symbian uses UCS-2 encoded strings
Shellcode Linux (x86): 10 lines
Shellcode Symbian (ARM): 500 lines (WTF!?)
Protection / Mitigation

- Write proper code (haha)
- Compile properly
- ASLR
- W^X
- Canaries
Outline

1 Hardware Security

2 Platform Security
   - Symbian
   - iPhone
   - Maemo
     - Maemo 6
   - Android

3 Hacking

4 Q&A
What security does the Platform give the user (and developer) give?

- (Symbian)
- iPhone
- Maemo
- Android

Lacking Time/Interest:

- Windows
- WebOS
- Blackberry
- ...
“Symbian is THE MOST developer hostile system I have ever worked with.”

**Packages**

- Symbian installs signed packages only
- Concept of (not very fine grained) Capabilities (∴ Do well in Maemo 6)
- Caps can be claimed during installation
- Caps depend on who signed the certificate (Nokia vs. Homebrew)
- However, a malicious program (Sexy View) was built, signed and distributed
Kernel

- Microkernel with client-server architecture
- Filesystems, Drivers, etc. as processes
- Single User: No Admin, No Users, No Login/Logout

Memory Protection

- ARMv5: None, ARMv6: W^X
Exploits in the Wild

- Many lame approaches (CommWarrior, Sexy View, ...)
- All require user interaction
- Not exciting research field
- Not really clear where to report to
- Curse of Silence (Video)
uname -a
Darwin my-iPhone 10.0.0d3 Darwin Kernel Version
10.0.0d3: Fri Sep 25 23:35:35 PDT 2009;
root:xnu-1357.5.30 3/RELEASE ARM S5L8920X iPhone2,1
arm N88AP Darwin
## iPhone (cont.)

### ps aux

<table>
<thead>
<tr>
<th>USER</th>
<th>PID</th>
<th>%CPU</th>
<th>%MEM</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>mobile</td>
<td>32</td>
<td>8.6</td>
<td>22.7</td>
<td>/System/L</td>
</tr>
<tr>
<td>root</td>
<td>1079</td>
<td>0.0</td>
<td>0.4</td>
<td>-sh</td>
</tr>
<tr>
<td>root</td>
<td>1076</td>
<td>0.0</td>
<td>0.5</td>
<td>/usr/sbin</td>
</tr>
<tr>
<td>mobile</td>
<td>1073</td>
<td>0.0</td>
<td>10.2</td>
<td>/Applicat</td>
</tr>
<tr>
<td>root</td>
<td>1049</td>
<td>0.0</td>
<td>0.2</td>
<td>login -fp</td>
</tr>
<tr>
<td>mobile</td>
<td>1040</td>
<td>0.0</td>
<td>0.4</td>
<td>-sh</td>
</tr>
</tbody>
</table>

...
Observations

- no ALSR, GCC but no SSP (i.e. canaries)
- Arrived in 20th century: $W^X$
- 2 (in words two) users

Wild Exploits

- Website Calling Home (Video)
- SMS Fuzzing
Chat - Jon Reece

Jon Reece: Hi, do you want to come over and see a movie on Friday? The whole crew is coming over for a get together!

You: Hey there! Sure! What time should I be there?

Jon Reece: How does 7 sound? 😊 We've got about 5 people coming so far. If you want to bring a friend along, feel free!

You: Oh great! I was going to ask if I can bring someone. Ashley wants to tag along. 😊 Need us to bring anything?

Jon Reece: Popcorn! 😊
N900
Hey Linux..?

uname -a
Linux muelli-N900 2.6.28-omap1 #1 PREEMPT Fri Aug 6 11:50:00 EEST 2010 armv7l unknown
**N900 (cont.)**

Hey Linux..?

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>VSZ</th>
<th>STAT</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>root</td>
<td>1844</td>
<td>S</td>
<td>/sbin/init</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>745</td>
<td>avahi</td>
<td>2804</td>
<td>S</td>
<td>avahi-daemon: running...</td>
</tr>
<tr>
<td>755</td>
<td>root</td>
<td>3288</td>
<td>S</td>
<td>/usr/sbin/csd -m -p c...</td>
</tr>
<tr>
<td>764</td>
<td>pulse</td>
<td>83028</td>
<td>S</td>
<td>/usr/bin/pulseaudio -...</td>
</tr>
<tr>
<td>825</td>
<td>haldaemon</td>
<td>3088</td>
<td>S</td>
<td>hald-addon-mmc: liste...</td>
</tr>
<tr>
<td>919</td>
<td>user</td>
<td>3332</td>
<td>S</td>
<td>/usr/bin/dbus-daemon ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
N900 (cont.)
Hey Linux..?

Memory Protection

$ cat /proc/$$/maps | egrep 'stack|heap|wx'
00067000-0008a000 rw-p 00067000 00:00 0 [heap]
be959000-be96e000 rw-p befeb000 00:00 0 [stack]

Observations

↩️ W^X *yay*
↩️ But neither ASLR nor SSP
↩️ 2.5 users
Maemo 6
They’ll fix it, right?

- IPC Sec
- App Credentials
- Crypto

- TPM to store keys and sign/verify
- Load signed Kernel (Integrity)
- Load signed binaries
- But some TPMs have been broken
- Thus don’t wait for 100% security
Android
uname -a

Linux localhost 2.6.29.6-cm42 #1 PREEMPT Sun Jan 31 15:10:14 EST 2010 armv6l GNU/Linux
## Android (cont.)

```bash
ps aux
```

<table>
<thead>
<tr>
<th>PID</th>
<th>UID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>149</td>
<td>radio</td>
<td>com.android.phone</td>
</tr>
<tr>
<td>151</td>
<td>app_12</td>
<td>android.process.acore</td>
</tr>
<tr>
<td>166</td>
<td>app_5</td>
<td>com.android.setupwizard</td>
</tr>
<tr>
<td>183</td>
<td>app_22</td>
<td>com.android.mms</td>
</tr>
<tr>
<td>211</td>
<td>app_6</td>
<td>com.google.android.apps.uploader</td>
</tr>
<tr>
<td>214</td>
<td>app_23</td>
<td>android.process.media</td>
</tr>
<tr>
<td>231</td>
<td>app_8</td>
<td>com.google.android.apps.maps:FriendService</td>
</tr>
<tr>
<td>241</td>
<td>root</td>
<td>audmgr_rpc</td>
</tr>
<tr>
<td>244</td>
<td>app_10</td>
<td>com.amazon.mp3</td>
</tr>
<tr>
<td>254</td>
<td>app_11</td>
<td>com.android.voicedialer</td>
</tr>
</tbody>
</table>
$ cat /proc/\`pidof mediaserver\`/maps |
    egrep 'stack|heap|wx' | wc -l
81

$ egrep 'stack|heap' /proc/\`pidof mediaserver\`/maps
0000a000-0003c000 rwxp 0000a000 00:00 0 [heap]
beaf3000-beb08000 rwxp befeb000 00:00 0 [stack]
Android (cont.)

Observations

- many users *yay*
- Weird ASLR
- Java needs wx on stack & heap *sigh*
- Flashback: ASLR since Linux 2.6.12, but neither Maemo nor Android use it (WTF?!)
- Question: WebOS, Windows, ...?
Outline

1. Hardware Security
2. Platform Security

3. Hacking
   - Exploitability
   - Bluetooth
   - WLAN
   - HTML
   - GSM
   - NFC

4. Q&A
Buffer Overflow: Simple Sample Code
Play around with mprotect
ASLR: Memory Maps
Example: overflow.c

/* specially crafted to feed your brain by gera */

int main(int argc, char* argv[]) {
    int cookie;
    char buf[8];

    printf("buf:%p cookie:%p\n", &buf, &cookie);
    if (&cookie < &buf)
        printf("Not exploitable: The compiler aligned\n");

    if (argc > 1)
        strcpy(buf, argv[1]); /* Yes it *is* insecure*/

    printf("cookie:%08x\n", cookie);

    if (cookie == 0x41424344) {

```
printf("you \_win!\n");
}

else {
    printf("Try \_/%sAAAAAAAAABCD\n", argv[0]);
    printf("Or \_/%sAAAAAAAADCBA\n", argv[0]);

    printf("Attempting to self exploit \n");
    strcpy(buf, "AAAAAAAAAABCD"); // Use this to
    printf("Cookie now is %08x\n", cookie);
    strcpy(buf, "AAAAAAAAACDAB"); // Use this to
    printf("Cookie now is %08x\n", cookie);
    strcpy(buf, "AAAAAAAADCBA"); // Use this to
    printf("Cookie now is %08x\n", cookie);
}
}
Bluetooth
Oh look, Symbian crashes

- Set name to: F00 0x09 0x2E 0x0A
- Vulnerability found in 2005 (sic!)
- No backtraces, no wild exploits
- Not really harmful: Phone reboots
WLAN
Oh look, another Symbian crasher

- WLAN Stack
- `./aireplay-ng -x 1024 -0 230 -a $ap -c $target $iface`
- Phone reboots
HTML and the Browsers
It's Symbian again

Browser crashes on

```html
<input type='checkbox' id='c'>
<script>
  r=document.getElementById('c');
  a=r.setAttributeNode();
</script>
```

- No publicly known exploit
- Hard to get traces
- let alone symbols
HTML and the Browsers (cont.)

It’s Symbian again

 Tattoo Remember the shellcode?!

But it’s not only Symbian that crashes
GSM

- It’s now possible to run your own network cheaply
- Send weirdly formatted packages
- Beer Fuzzing: Signal Calls and SMS
Curse of Silence

- Video
- No 3rd party application
- No way of deactivating the service
- No way of mitigating by, i.e. install different SMS stack
- Eventually Nokia provided a tool (not a fix!) to get rid of malicious SMS
MITM GSM Modem

*Very* awesome

Pretend to be the modem (runs on 2nd CPU anyway)

Inject anything into the OS

SMS: unsolicited message

Back to the 90s: No user interaction, no firewalling

Credits to Collin Mulliner and Charlie Miller

Work needed for Maemo, Windows, Blackberry, ...
Near Field Communication

- Create random Tags
- URL parser crashes Symbian

btw: who’s got a spare Nokia 6313 or 6212?
Outline

1. Hardware Security
2. Platform Security
3. Hacking
4. Q&A
   - Summary
   - Q&A
Summary

What do you want anyway?!

- “Security” is a bit fuzzy
- Today’s mobile devices are more general purpose computers
- Mobile Security affects loads of people
- Understand new Threat model
- Test your stuff by trying to hack it
- Write better code
Questions?!