Security in Mobile Devices

Hacking Mobiles for Fun and Profit

Tobias Mueller

Universität Hamburg
&
Dublin City University

2010-03-08
1. Hardware Security
2. Platform Security
3. Hacking
4. Q&A
About me

Contact

Jabber muelli@jabber.ccc.de
ACF0 F5EC E9DC 1BDC F09D
B992 4147 7261 7CB6 4CEF

Mail muelli@cryptobitch.de
CF3E D935 AE6B DE0A D508
AF86 3EE0 57FF AA20 8D9E

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
Security

What are we talking about anyway?

Customer expects

- not to leak/expose data  → Confidentiality
- data not being corrupted (i.e. by other application)  → Integrity
- apps with no malicious features  → Platform Security
- apps not exploitable to do malicious stuff  → Software Security
Security
What are we talking about anyway?

Customer expects

- not to leak/expose data  → **Confidentiality**
- data not being corrupted (i.e. by other application)  → **Integrity**
- apps with no malicious features  → **Platform Security**
- apps not exploitable to do malicious stuff  → **Software Security**
Security
What are we talking about anyway?

Customer expects

- not to leak/expose data → **Confidentiality**
- data not being corrupted (i.e. by other application) → **Integrity**
- apps with no malicious features → **Platform Security**
- apps not exploitable to do malicious stuff → **Software Security**
Security
What are we talking about anyway?

Customer expects

- not to leak/expose data → Confidentiality
- data not being corrupted (i.e. by other application) → Integrity
- apps with no malicious features → Platform Security
- apps not exploitable to do malicious stuff → Software Security
Security

What are we talking about anyway?

Customer expects

♀ not to leak/expose data → **Confidentiality**
♀ data not being corrupted (i.e. by other application) → **Integrity**
♀ apps with no malicious features → **Platform Security**
♀ apps not exploitable to do malicious stuff → **Software Security**
### 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?
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

**Opcodes**

- Usage: N900: Cortex A8, N800: ARM 9E
- ARM, MIPS, SPARC: 4 bytes, “NOP”: 4 bytes
- (ARM: Except THUMBS: 2 bytes)
- x86: omgwtf NOP: 1 byte
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

debugger instructions:
function call
chain of instructions
the stack

 push %ebp
 mov %esp, %ebp
 sub $0x08,%esp

return address
%ebp

bytebuffer

return

%esp

%ebp

↑ 0x00

↓ 0xFF
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

<table>
<thead>
<tr>
<th></th>
<th>← %ebp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>%esp</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>return address</td>
<td>%ebp</td>
</tr>
<tr>
<td>%ebp</td>
<td>bytebuffer</td>
</tr>
<tr>
<td>↑ 0x00</td>
<td></td>
</tr>
</tbody>
</table>

0x00 ← %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

[Diagram showing the stack with labels for push, mov, and pop operations, along with the return address and byte buffer positions.]
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
```
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

![The Stack Diagram]

- 0xFF
- ...
- ...
- ...
- return address
- %ebp
- bytecode
- ← %ebp
- ← %esp
- 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

\[
\begin{array}{c|c}
\text{\textbackslash wnFF} & \text{\ldots} \\
\text{\ldots} & \text{\ldots} \\
\text{\ldots} & \text{\ldots} \\
\text{\textbackslash nreturn address} & \text{\ldots} \\
\%ebp & \text{\ldots} \\
\text{bytebuffer} & \leftarrow %ebp \\
\text{\textbackslash u00} & \leftarrow %esp \\
\end{array}
\]
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
```

• %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 = pop %eip

#### the stack

<table>
<thead>
<tr>
<th>%esp</th>
<th>%ebp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bytebuffer</td>
</tr>
<tr>
<td></td>
<td>return address</td>
</tr>
<tr>
<td>0xFF</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

![](image)
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

%ebp

bytebuffer

↑ 0x00

← %ebp

← %esp
#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
“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
<table>
<thead>
<tr>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>*source</td>
</tr>
<tr>
<td>return address</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
call strcpy
movl %ebp, %esp
pop %ebp
ret
```

The stack

```
↓ 0xFF  
    ...  
    *source
    return address
    %ebp
      ↓ 0x00  
      ...  
```
“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 strcpy
mov %ebp, %esp
pop %ebp
ret

the stack

\[ \begin{array}{c|c}
\downarrow 0xFF & \ldots \\
*source & return address \\
%ebp & \\
\uparrow 0x00 & \ldots
\end{array} \]
"push *source"    #1st arg

```
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
```
"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

\[\begin{array}{c|c}
\text{0xFF} & \ldots \\
\hline
*source & \text{return address} \\
%ebp & \text{buffer[79]}
\hline
\text{buffer[0]} & \ldots \\
\end{array}\]
"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

the stack

↓ 0xFF

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

```
/e call vulnerableFunction
/e next instruction
/e ...
/e 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

<table>
<thead>
<tr>
<th>0xFF</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*source</td>
</tr>
<tr>
<td></td>
<td>return address</td>
</tr>
<tr>
<td></td>
<td>%ebp</td>
</tr>
<tr>
<td></td>
<td>buffer[79]</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x00</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>*source</td>
</tr>
<tr>
<td></td>
<td>*destination</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

the stack

| 0xFF | ... |
|      | *source |
|      | return address |
|      | %ebp |
|      | buffer[79] |
|      | ... |
|      | buffer[0] |
|      | *source |
| 0x00 | ... |
|      | *destination |
"push *source"  #1st arg

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
buffer[79]

... buffer[0]

*source
*destination

0x00

...
"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

\[
\begin{array}{c|c|c|c}
\hline
\text{↓ 0xFF} & \ldots & \text{\texttt{*source}} & \text{\texttt{return address}} \\
\hline
\text{\texttt{ebp}} & \text{\texttt{ebp}} & \text{\texttt{return address}} & \text{\texttt{ebp}} \\
\hline
\text{\texttt{ebp}} & \text{\texttt{ebp}} & \text{\texttt{return address}} & \text{\texttt{ebp}} \\
\hline
\text{\texttt{ebp}} & \text{\texttt{ebp}} & \text{\texttt{return address}} & \text{\texttt{ebp}} \\
\hline
\text{\texttt{ebp}} & \text{\texttt{ebp}} & \text{\texttt{return address}} & \text{\texttt{ebp}} \\
\hline
\end{array}
\]
“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 = pop %esi
“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
“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
buffer[79]

↓ 0x00

*source
*destination

↑ 0x00

...
“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>buffer</td>
<td>buffer</td>
</tr>
<tr>
<td></td>
<td>buffer</td>
<td>buffer[79]</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>buffer[0]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>*source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*destination</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

the stack

↓ 0xFF
*source
buffer
buffer

↑ 0x00
...
“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 %ebp, %esp
pop %ebp
ret = pop %esi
0wned

BOOOOOM!!!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.)

<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
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 Nokia-N900-02-8 2.6.28-omap1 #1 PREEMPT Thu Dec 17 09:40:52 EET 2009 armv7l unknown
Hey Linux..?

```
ps aux

<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>haldaemo</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.)

<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.appsuploader</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>
Android (cont.)

Memory Protection

```
$ 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]
```
### 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, …?
DIY

🧶 Buffer Overflow: Simple Sample Code
🧶 Play around with mprotect
🧶 ASLR: Memory Maps
/* 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 ./%sAAAAAAAAAAABCD\n", argv[0]);
printf("Or ./%sAAAAAAAADCBA\n", argv[0]);

printf("Attempting to self exploit\n");
strcpy(buf, "AAAAAAAAAAABCD"); /* Use this to
printf("Cookie now is %08x\n", cookie);
strcpy(buf, "AAAAAAAAAACDAB"); /* Use this to
printf("Cookie now is %08x\n", cookie);
strcpy(buf, "AAAAAAAAAADCBA"); /* 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 -o 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

🔗 Remember the shellcode?!  

But it’s not only Symbian that crashes
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
- Todays 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

What do you want anyway?!
Q&A
Who dares to have a question?!

Muito Obrigado!

Questions?! (Feedback)