Security in Mobile Devices

Hacking Mobiles for Fun and Profit

Tobias Mueller

Universität Hamburg
&
Dublin City University

2010-12-16

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 $\sim$ 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?

Interfaces

- WiFi
- Bluetooth
- Email
- Web
- Video (Podcasts?)
- GSM (Calls, Texts)

Why mobile? (cont.)

More than a PC

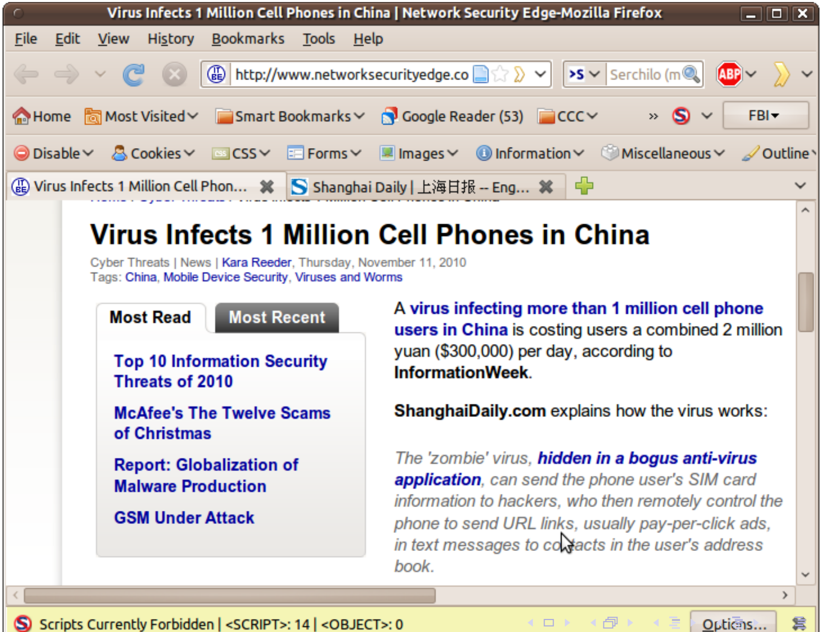
- Personal Data
- GPS
- Cellular
- Financial Gain/Loss
- Always on
- Infection Not Obvious
- pwn 1 pwn many (cloud syndrome)

Why mobile? (cont.)

However...

- few publicly known vulnerabilities
- just PoCs, nobody really exploiting... orly?

In the news


Virus Infects 1 Million Cell Phones in China | Network Security Edge-Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.networksecurityedge.co
Home Most Visited Smart Bookmarks Google Reader (53) CCC FBI
Disable Cookies CSS Forms Images Information Miscellaneous Outline
Virus Infects 1 Million Cell Phon...
Shanghai Daily | 上海日报 -- Eng...
Scripts Currently Forbidden | <SCRIPT>: 14 | <OBJECT>: 0
Options...

Virus Infects 1 Million Cell Phones in China

Cyber Threats | News | Kara Reeder, Thursday, November 11, 2010
Tags: China, Mobile Device Security, Viruses and Worms

Most Read | Most Recent

- Top 10 Information Security Threats of 2010
- McAfee's The Twelve Scams of Christmas
- Report: Globalization of Malware Production
- GSM Under Attack

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

x86 vs. ARM

What's different then?

Classic Vulnerabilites/Architecture revisited:

- Opcodes
- Buffer Overflows
- Endianness
- Format Strings

Complexity

ARM is much less complex

Opcodes

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

Complexity (cont.)

Remember `f0 0f c7 c8`?

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

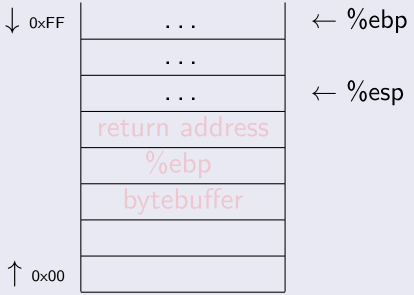
```
lock cmpxchg8b eax
```

> *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

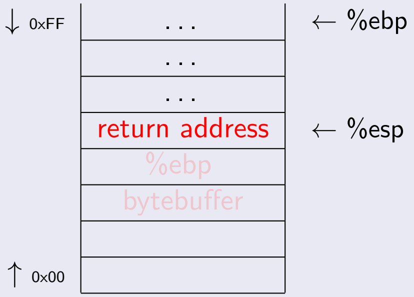
the stack



function calls

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

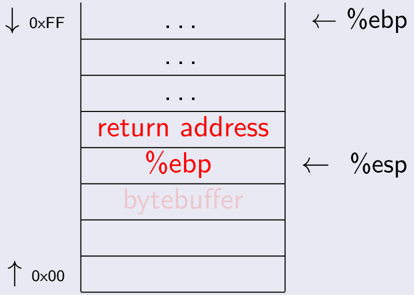
the stack



function calls

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

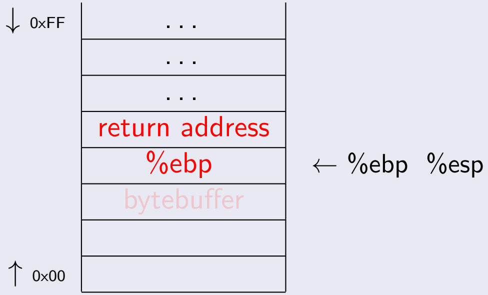
the stack



function calls

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

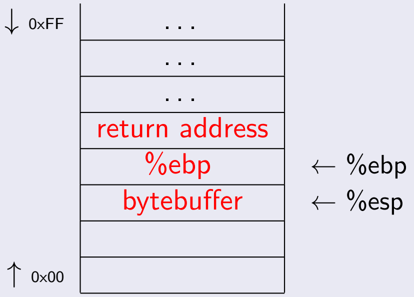
the stack



function calls

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

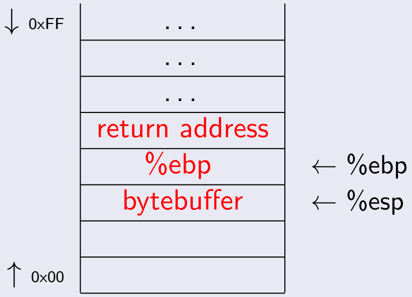
the stack



function calls

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

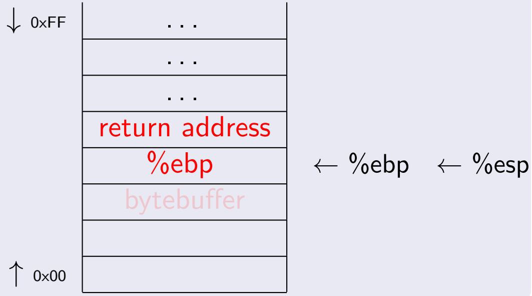
the stack



function calls

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

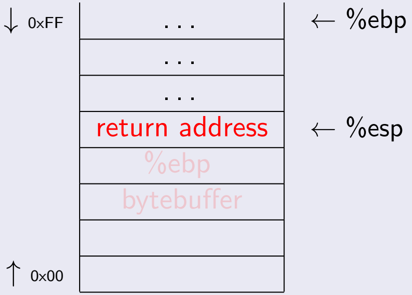
the stack



function calls

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

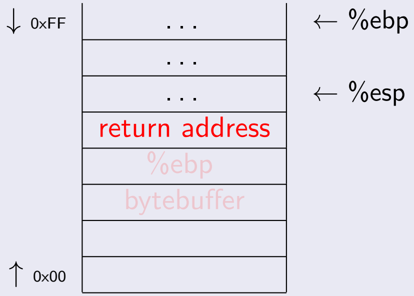
the stack



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

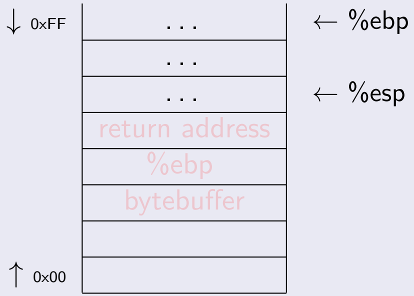
the stack



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



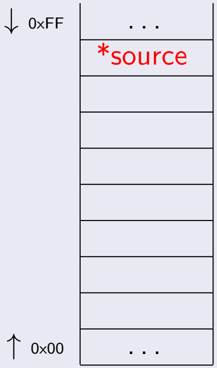
Example: vulnerable.c

```c
#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]);
}
```

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

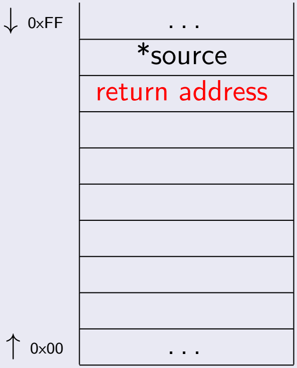
the stack



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

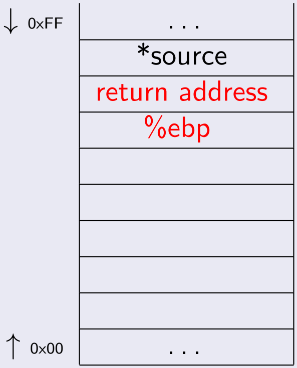
the stack



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

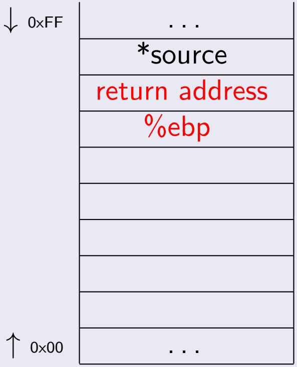
the stack



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

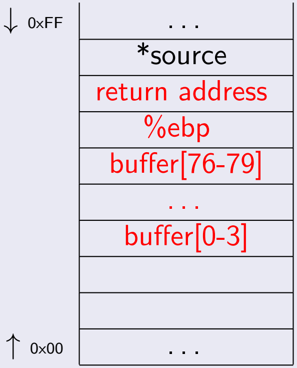
the stack



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

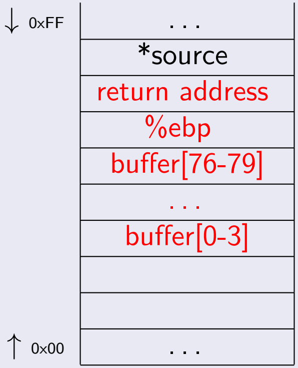
the stack



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

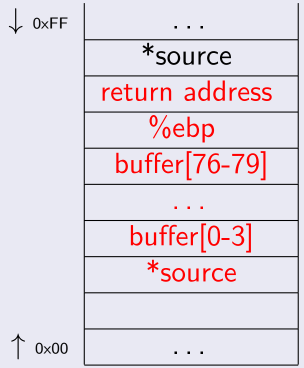
the stack



Overwrite Return Address

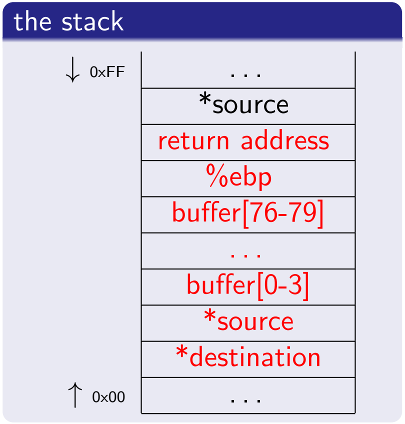
- "push *source" #1st arg
- call vulnerableFunction
- next instruction
- ...
- vulnerableFunction:
  pushl %ebp
- movl %esp, %ebp
- subl $80, %esp
- leal -80(%ebp), %eax
- pushl 8(%ebp) # source
- pushl %eax
- call strcpy
- mov %ebp, %esp
- pop %ebp
- ret

the stack



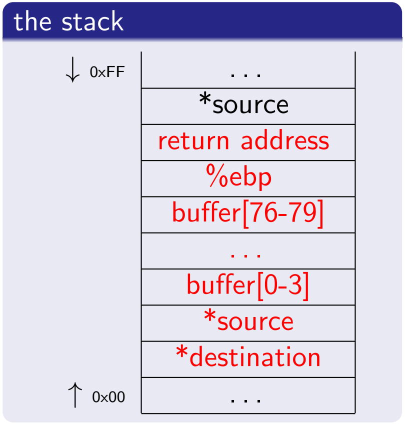
Overwrite Return Address

- “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



Overwrite Return Address

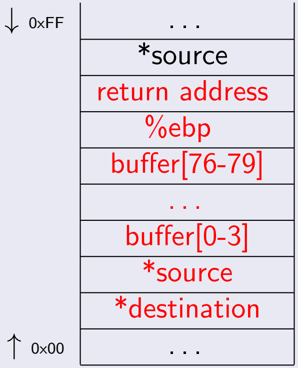
- “push *source”   #1st arg
- call vulnerableFunction
- next instruction
- . . .
- vulnerableFunction:
  pushl %ebp
- movl %esp, %ebp
- subl $80, %esp
- leal -80(%ebp), %eax
- pushl 8(%ebp) # source
- pushl %eax
- call strcpy
- mov %ebp, %esp
- pop %ebp
- ret



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

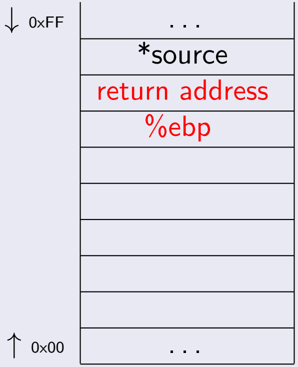
the stack



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

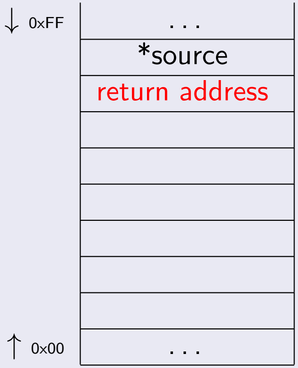
the stack



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 = pop %eip

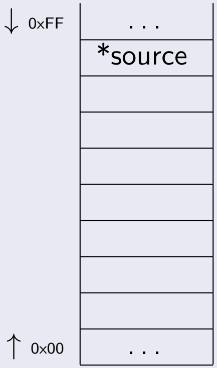
the stack



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

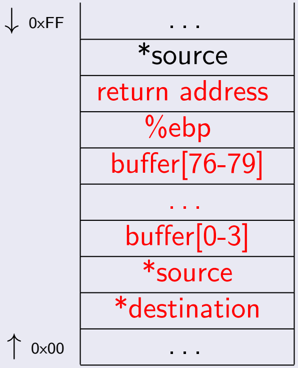
the stack



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

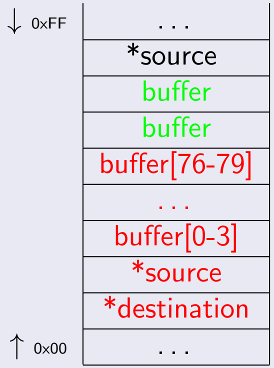
the stack



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

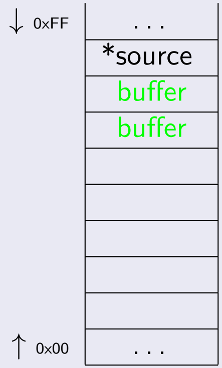
the stack



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

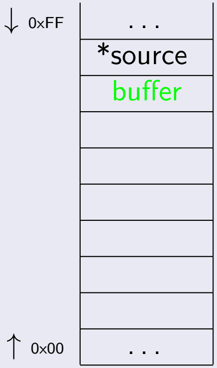
the stack



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 = pop %eip

the stack



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

What security does the Platform give the user (and developer) give?

- (Symbian)
- iPhone
- Maemo
- Android

Lacking Time/Interest:

- Windows
- WebOS
- Blackberry
- ...

symbian OS

"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 ($\rightarrow$ 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)

iPhone

AT&T 3G
9:42 AM
Tuesday
9
SMS
Text
Calendar
Photos
Camera
280
73°
YouTube
Stocks
Maps
Weather
Clock
Calculator
Notes
Settings
iTunes
App Store
Phone
Mail
Safari
iPod

iPhone

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

```
USER        PID %CPU %MEM   COMMAND
mobile       32   8.6 22.7  /System/L
root       1079   0.0  0.4  -sh
root       1076   0.0  0.5  /usr/sbin
mobile     1073   0.0 10.2  /Applicat
root       1049   0.0  0.2  login -fp
mobile     1040   0.0  0.4  -sh
...
```

iPhone (cont.)

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

N900

NOKIA
N900
4:00
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! 15:14
You: Hey there! Sure! What time should I be there? 15:18
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! 15:22
You: Oh great! I was going to ask if I can bring someone. Ashley wants to tag along. Need us to bring anything? 15:28
Jon Reece: Popcorn! 15:31

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..?

ps aux

```
PID USER       VSZ STAT COMMAND
  1 root      1844 S    /sbin/init
...
745 avahi     2804 S    avahi-daemon: running...
755 root      3288 S    /usr/sbin/csd -m -p c...
764 pulse    83028 S <  /usr/bin/pulseaudio -...
825 haldaemo  3088 S    hald-addon-mmc: liste...
919 user      3332 S <  /usr/bin/dbus-daemon ...
...
```

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

htc
12:37 PM
www.google.com: Google
Home Mail Calendar Reader More
Google
Web Images Local News
Google Search
Settings Terms
View Google in: Mobile | Classic
Google.com in English
©2008 Google
MENU

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.)

ps aux

```
PID  UID     Name
149  radio   com.android.phone
151  app_12  android.process.acore
166  app_5   com.android.setupwizard
183  app_22  com.android.mms
211  app_6   com.google.android.apps.uploader
214  app_23  android.process.media
231  app_8   com.google.android.apps.maps:FriendService
241  root    audmgr_rpc
244  app_10  com.amazon.mp3
254  app_11  com.android.voicedialer
```

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]
```

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

DIY

- 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 aligne

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

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

    if (cookie == 0x41424344) {
```

Example: overflow.c (cont.)

```
        printf("you␣win!\n");
    } else {
        printf("Try␣./%s␣AAAAAAAAABCD\n", argv[0]);
        printf("Or␣␣./%s␣AAAAAAAADCBA\n", argv[0]);

        printf("Attempting␣to␣self␣exploit\n");
        strcpy(buf, "AAAAAAAAABCD"); /* Use this to
        printf("Cookie␣now␣is␣%08x\n", cookie);
        strcpy(buf, "AAAAAAAACDAB"); /* 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: `FOO 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

```
<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

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

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

Q&A

Who dares to have a question?!

Questions?!