Mac Hackin' 2: Snow Leopard Boogaloo

Charlie Miller

Independent Security Evaluators

cmiller@securityevaluators.com

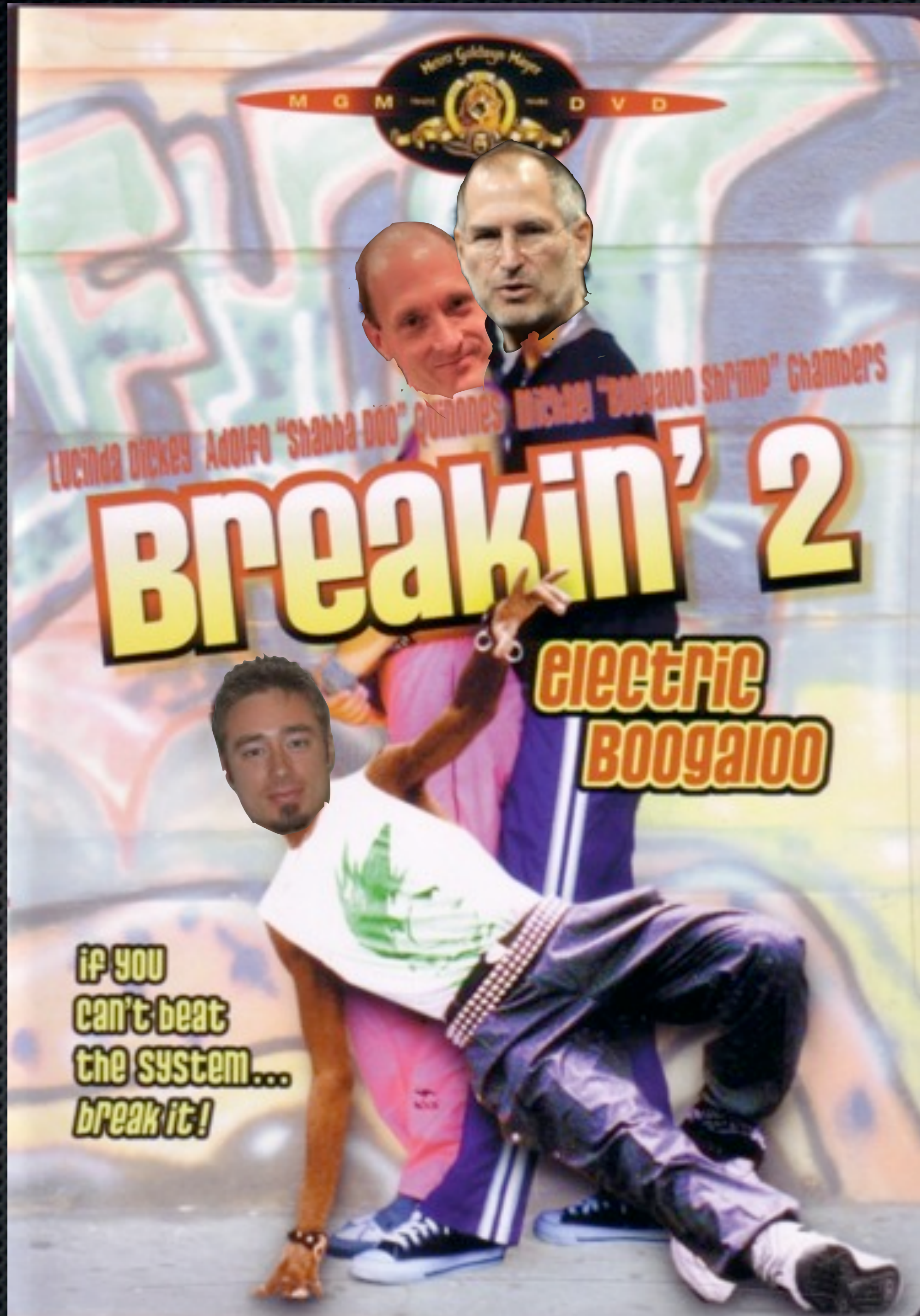
@0xcharlie

Dino A. Dai Zovi

Trail of Bits

ddz@theta44.org

@dinodaizovi


M G M
D V D
Lucinda Dickey Adolfo "Shabba Doo"
Breakin' 2
electric
Boogaloo
if you
can't beat
the system...
break it!

About Us

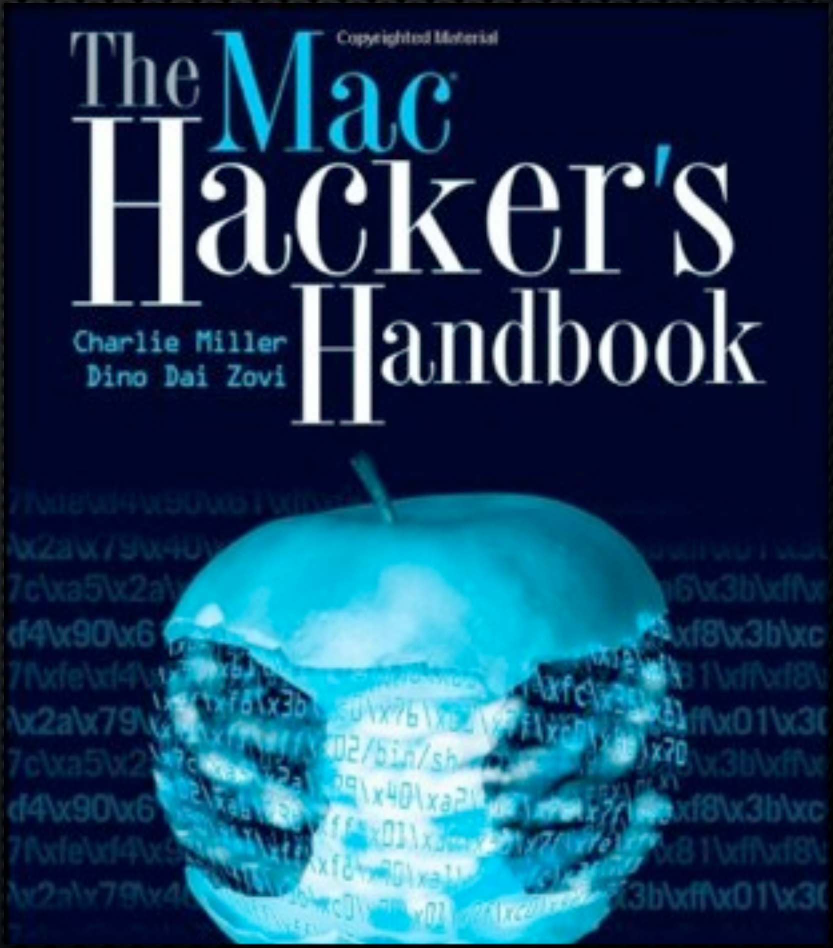
- We hack Macs
    - Every year at PWN2OWN (Dino: 2007, Charlie: 2008-2010)
    - We've probably hacked yours also (look for an extra thread in launchd)
    - Wrote the book on it

About Us

- We hack Macs
  - Every year at PWN2OWN (Dino: 2007, Charlie: 2008-2010)
  - We've probably hacked yours also (look for an extra thread in launchd)
  - Wrote the book on it

About Us

- We hack Macs
  - Every year at PWN2OWN (Dino: 2007, Charlie: 2008-2010)
  - We've probably hacked yours also (look for an extra thread in launchd)
  - Wrote the book on it



About this talk

- The Mac Hackers Handbook came out in March 2009 and covered Tiger and Leopard
- That summer Snow Leopard came out with many runtime security improvements (and broke the book's example code)
- This talk will discuss just how much real protection these improvements provide and how they make exploitation impossible

About this talk

- The Mac Hackers Handbook came out in March 2009 and covered Tiger and Leopard
- That summer Snow Leopard came out with many runtime security improvements (and broke the book's example code)
- This talk will discuss just how much real protection these improvements provide and how they make exploitation more fun!

Overview

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

Overview

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

Overview

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

Overview

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

Overview

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

Overview

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

Overview

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

64-Bit

64-Bit

The 64-bit applications in Snow Leopard are even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. Learn more about 64-bit ▸

64-bit in Mac OS X 10.6

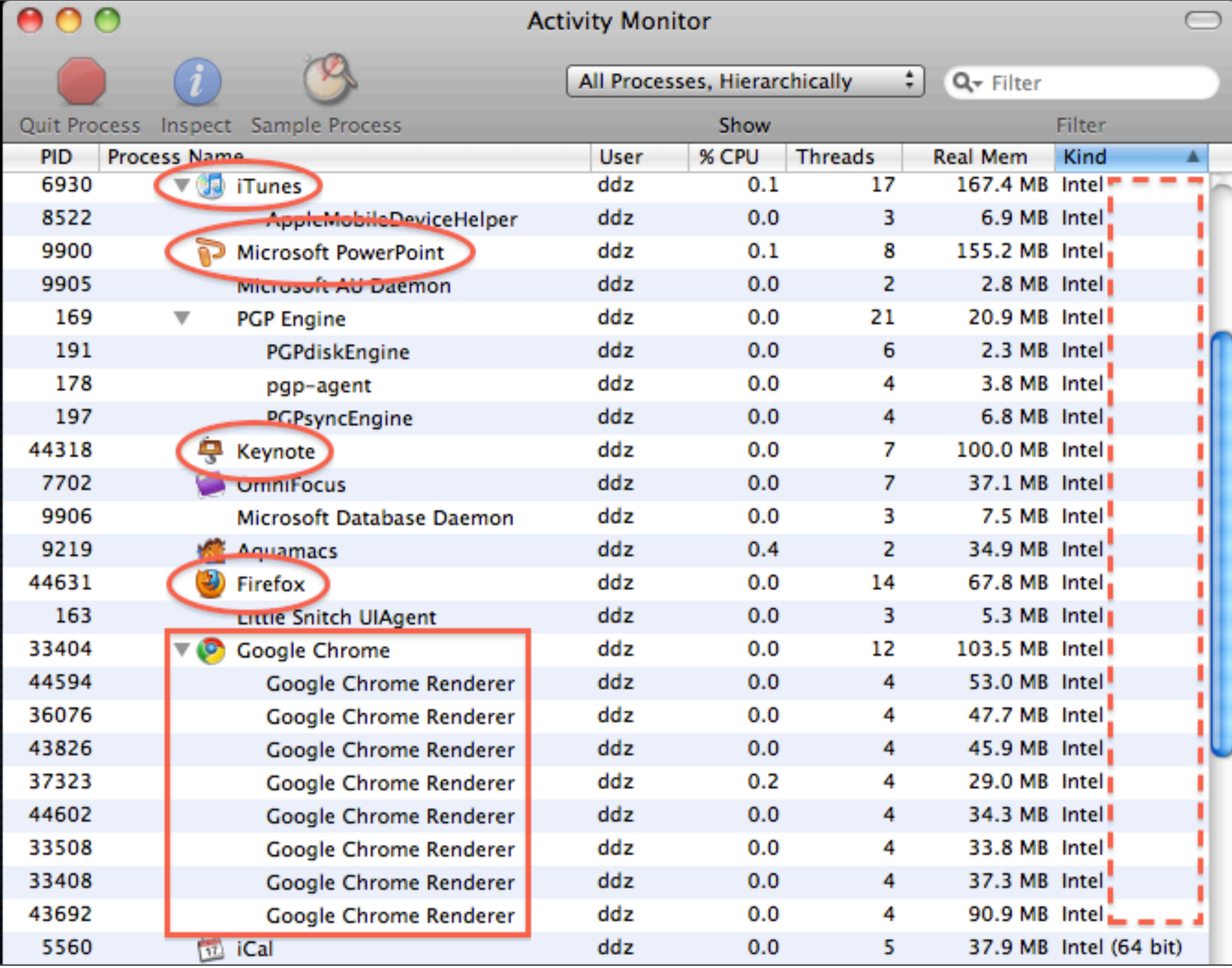
- Snow Leopard's increased use of 64-bit was touted as one of its key features
  - Primarily for making more memory available to applications
  - But Apple even touts 64-bit applications as a security feature
  - It offers some security benefits, but not as much as you would hope

Technically, That is True

More secure than ever.

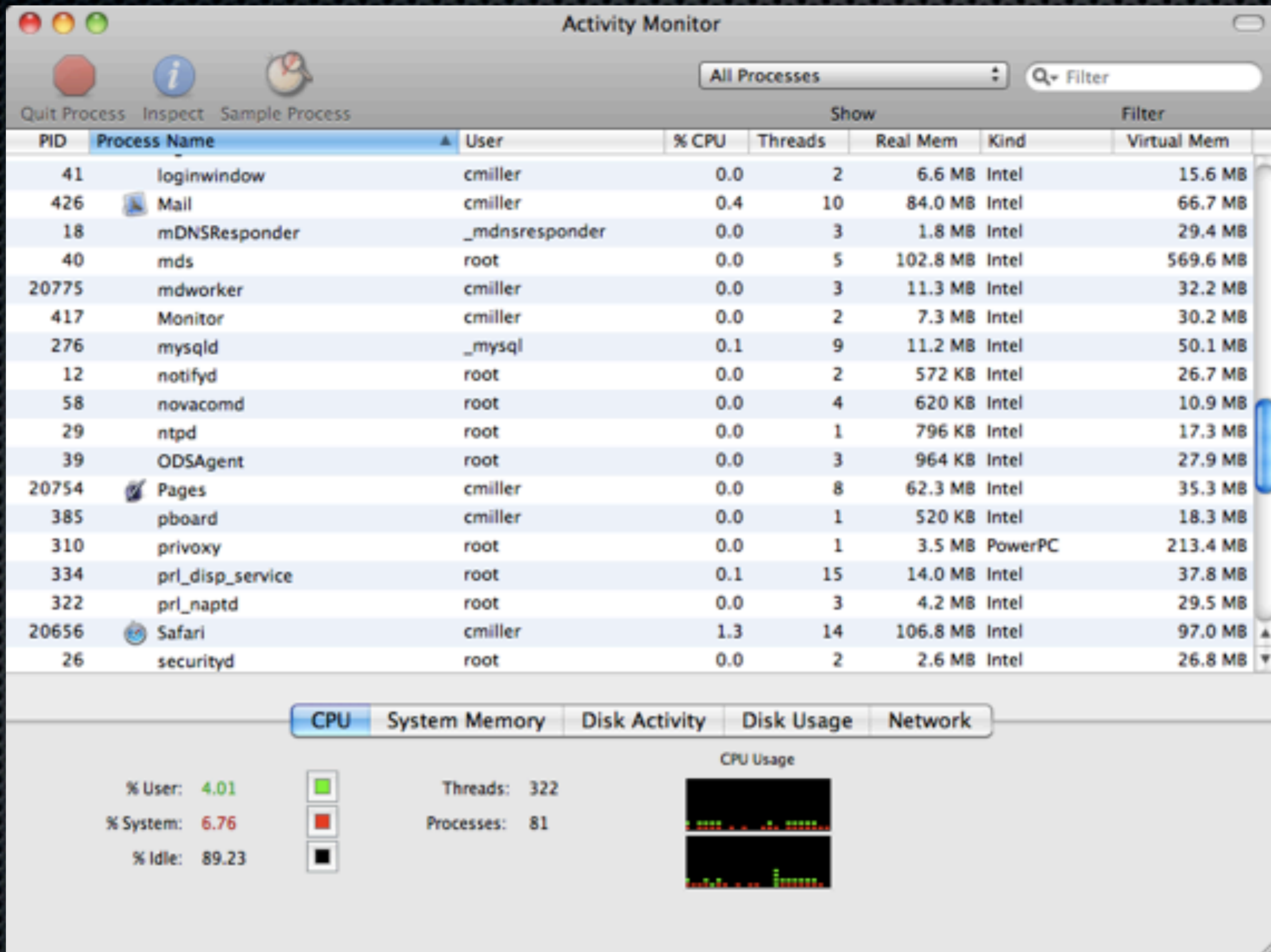
Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

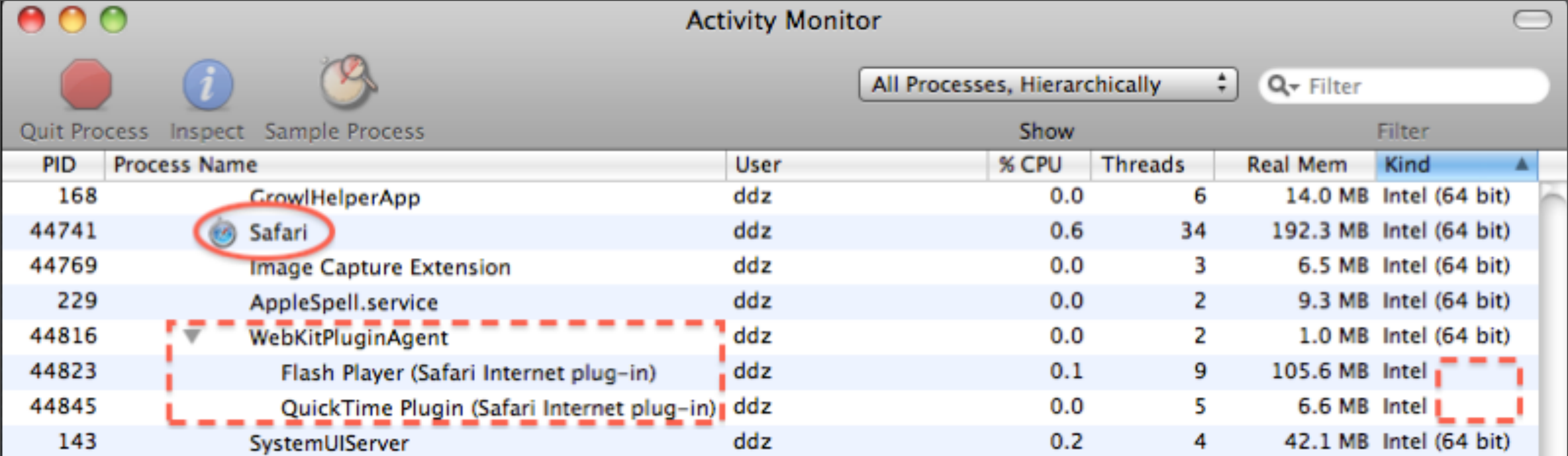
- Function arguments are no longer stored on the stack
- Hardware-supported non-executable heap memory
- Heap block header metadata checksums (also in 32-bit procs)



PID	Process Name	User	% CPU	Threads	Real Mem	Kind
6930	iTunes	ddz	0.1	17	167.4 MB	Intel
8522	AppleMobileDeviceHelper	ddz	0.0	3	6.9 MB	Intel
9900	Microsoft PowerPoint	ddz	0.1	8	155.2 MB	Intel
9905	Microsoft AU Daemon	ddz	0.0	2	2.8 MB	Intel
169	PGP Engine	ddz	0.0	21	20.9 MB	Intel
191	PGPdiskEngine	ddz	0.0	6	2.3 MB	Intel
178	pgp-agent	ddz	0.0	4	3.8 MB	Intel
197	PGPsyncEngine	ddz	0.0	4	6.8 MB	Intel
44318	Keynote	ddz	0.0	7	100.0 MB	Intel
7702	OmniFocus	ddz	0.0	7	37.1 MB	Intel
9906	Microsoft Database Daemon	ddz	0.0	3	7.5 MB	Intel
9219	Aquamacs	ddz	0.4	2	34.9 MB	Intel
44631	Firefox	ddz	0.0	14	67.8 MB	Intel
163	Little Snitch UIAgent	ddz	0.0	3	5.3 MB	Intel
33404	Google Chrome	ddz	0.0	12	103.5 MB	Intel
44594	Google Chrome Renderer	ddz	0.0	4	53.0 MB	Intel
36076	Google Chrome Renderer	ddz	0.0	4	47.7 MB	Intel
43826	Google Chrome Renderer	ddz	0.0	4	45.9 MB	Intel
37323	Google Chrome Renderer	ddz	0.2	4	29.0 MB	Intel
44602	Google Chrome Renderer	ddz	0.0	4	34.3 MB	Intel
33508	Google Chrome Renderer	ddz	0.0	4	33.8 MB	Intel
33408	Google Chrome Renderer	ddz	0.0	4	37.3 MB	Intel
43692	Google Chrome Renderer	ddz	0.0	4	90.9 MB	Intel
5560	iCal	ddz	0.0	5	37.9 MB	Intel (64 bit)

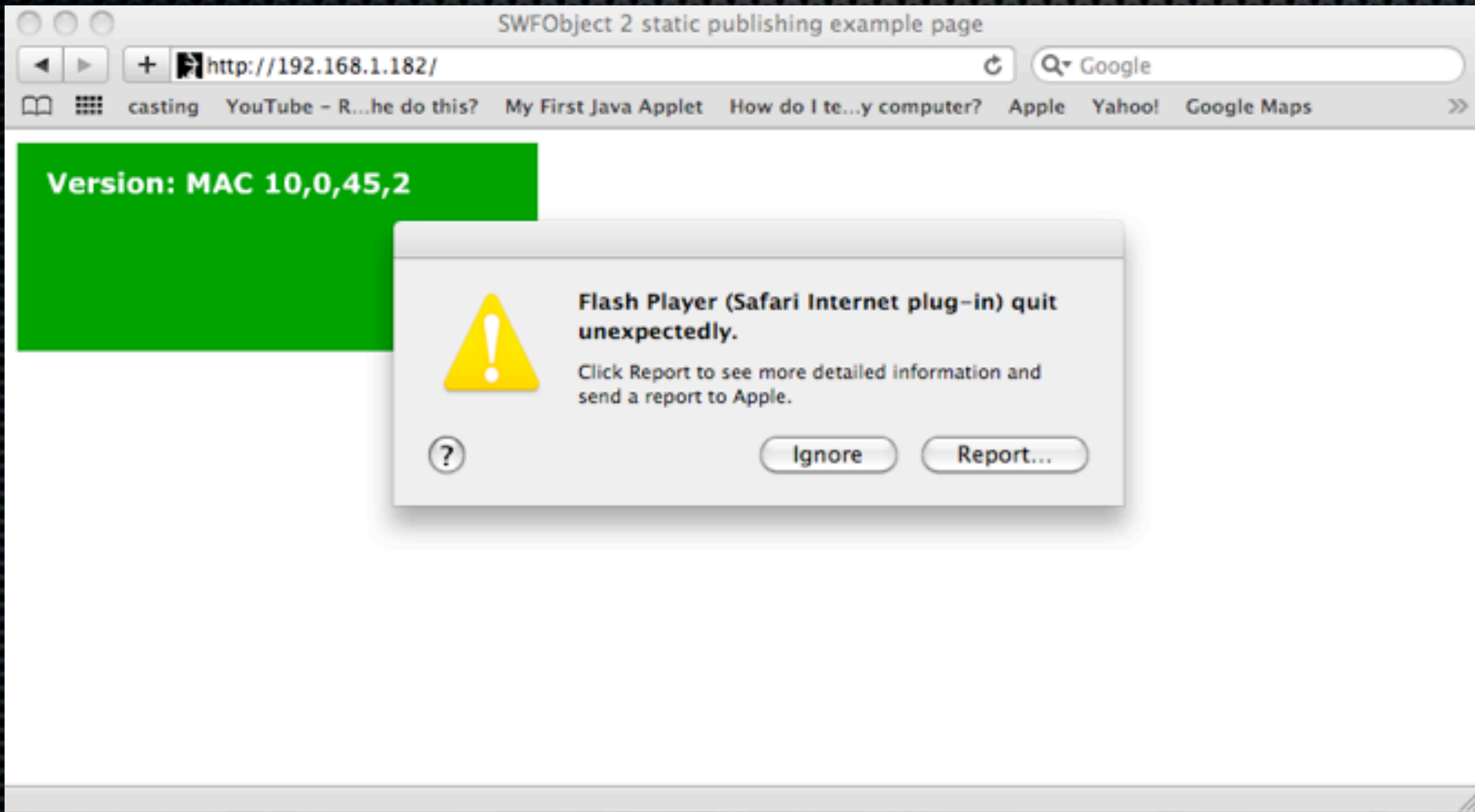
Older macs - all 32 bit





- The Safari browser itself is 64-bit
- Safari runs 32-bit plugins out-of-process
  - Flash Player is 32-bit
  - QuickTime Plugin is 32-bit
  - Some plugins are 64-bit and run in-process (Java)
- WebKitPluginAgent (64-bit) and WebKitPluginHost (32-bit) communicate over Mach IPC
- Avoids requiring a 32-bit Safari to watch YouTube

“Crash resiliency”

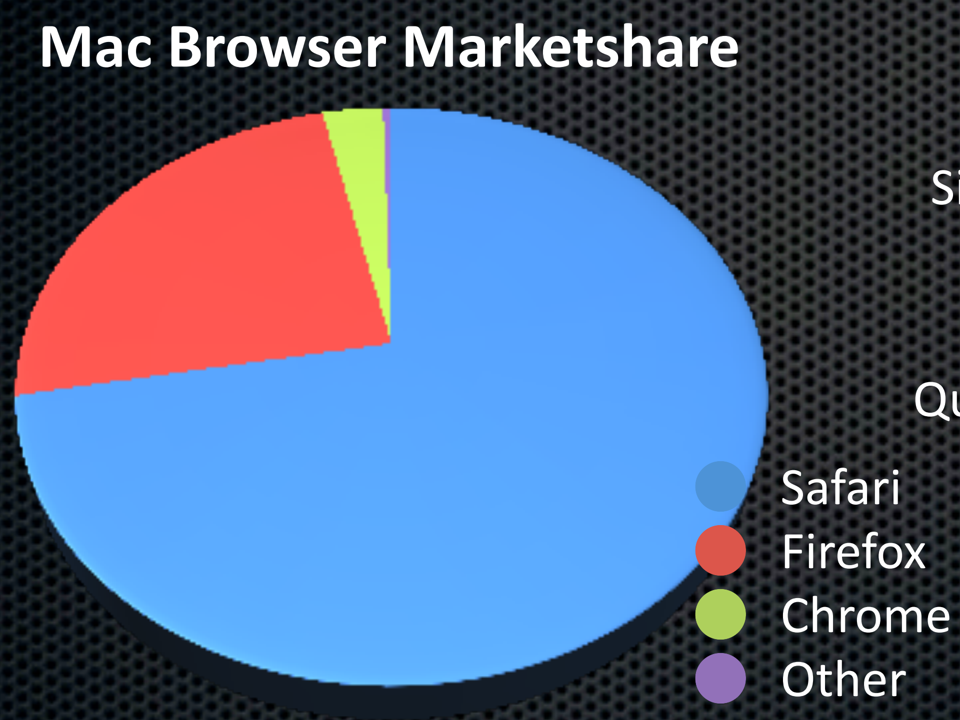


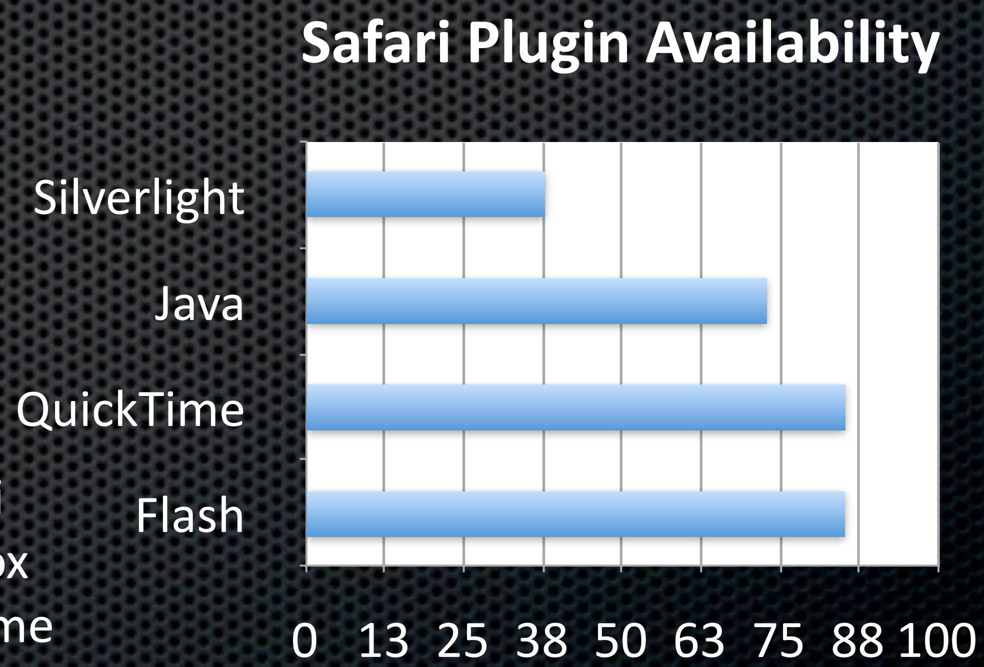
Older macs

- ...and users who launch Safari under 32 bit
- Plugins run within Safari's (32-bit) address space

```
$ vmmap PID
__TEXT                 00001000-0052b000 [ 5288K] r-x/rwx SM=COW  /
Applications/Safari.app/Contents/MacOS/Safari
...
__TEXT                 19dcb000-1a50b000 [ 7424K] r-x/rwx SM=COW  /
Users/cmiller/Library/Internet Plug-Ins/Flash Player.plugin/Contents/
MacOS/Flash Player
```

TargetShare (TM)





Statistics for June 2010, StatOwl.com

64 is 32 More Than I Need to Pwn

- 27% of Mac users use a 32-bit web browser
  - The "more secure" Firefox and Chrome browsers
- 85% of Mac Safari users have 32-bit plugins available
  - Flash Player or QuickTime Plugin
  - Both have a long history of security vulnerabilities
- Most key client-side applications are still 32-bit
  - Office, iWork, iTunes, iLife, etc.
- But Adobe CS5 is 64-bit
  - Don't have to worry about getting owned by a PSD

64-Bits Are Hard, Bro

- 64-bit exploitation has various complications
  - NULLs in every memory address
  - Subroutines take arguments in registers, not stack
  - All data memory regions are non-executable
  - No more RWX __IMPORT regions
- 64-bit exploitation techniques are not yet really needed on Mac OS X, especially for targeting client-side applications
  - Server-side attack surface is minimal and not critical

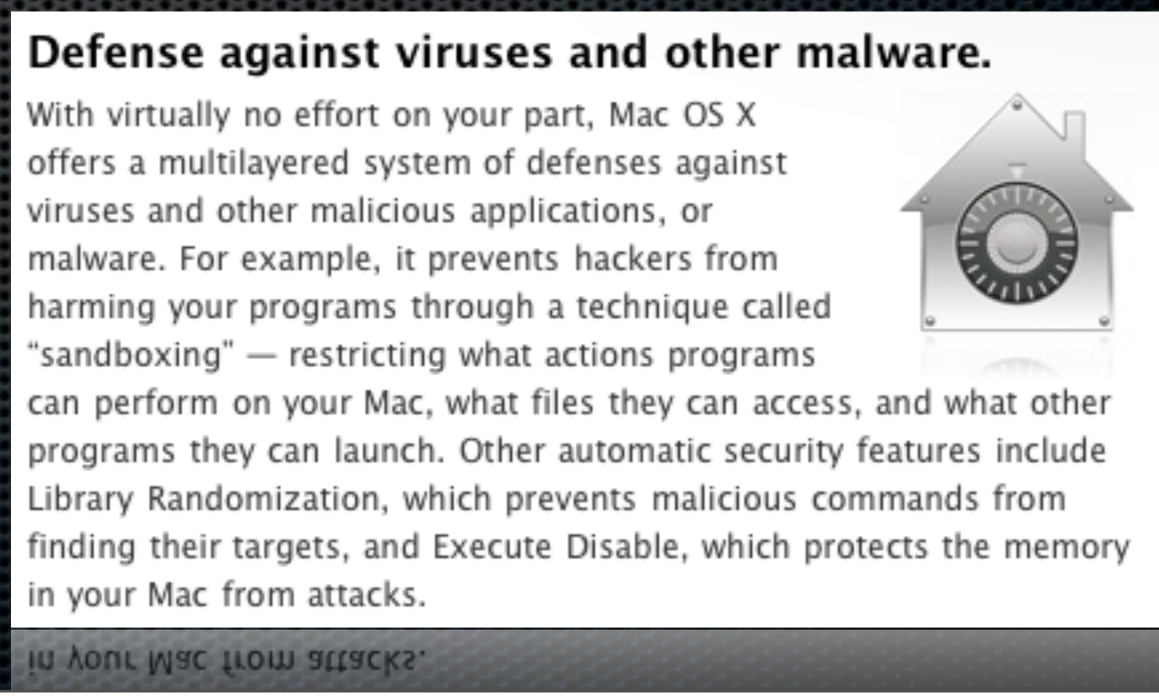
Shellcode

- x86 shellcode doesn't typically work
  - For example, no metasploit Mac OS X shellcode works on x86_64
- First public x86_64 OS X shellcode was from @fjserna
  - Connect() shellcode, contains NULL's
  - See Charlie's POC 2010 presentation for cleaner and smaller version (120 bytes vs. 165)

Tools

- Some tools won't work on 64-bit
  - pydbg
  - valgrind
- GDB still works fine

Sandboxing



Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

Sandboxing

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

Sandboxing

- Implements fine-grained access controls
  - Accessing resources (sockets, files, shared mem)
  - Sending/receiving Mach messages
  - Sending/receiving BSD signals
- Started via sandbox_init() call (or sandbox_exec)

Mac OS X sandboxing architecture

- User process calls sandbox_init() (in libSystem)
- libSystem dynamically loads libSandbox for support functions
- Initiates action in the kernel via SYS__mac_syscall system call
- Sandbox.kext kernel module hooks syscalls via TrustedBSD interface and limits access as defined in policy

Snow Leopard sandboxing

- No client-side applications are sandboxed, including
    - Safari
    - Mail
    - iTunes
    - Plugins including Flash and QuickTime

Heap

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

Heap

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

Heap

More secure than ever.

Another benefit of the 64-bit applications in Snow Leopard is that they're even more secure from hackers and malware than the 32-bit versions. That's because 64-bit applications can use more advanced security techniques to fend off malicious code. First, 64-bit applications can keep their data out of harm's way thanks to a more secure function argument-passing mechanism and the use of hardware-based execute disable for heap memory. In addition, memory on the system heap is marked using strengthened checksums, helping to prevent attacks that rely on corrupting memory.

Welcome to the Heap of Pain

- Some significant improvements were made in the heap implementation in Snow Leopard compared to Leopard
- Check out Libc source code from opensource.apple.com
- Change from scalable_malloc.c to magazine_malloc.c

10.5 Heap Pointer Checksums

- Free list pointer checksums detect accidental overwrites, not intentional ones
  - cksum(ptr) = (ptr >> 2) | 0xC0000003
  - verify(h) = ((h->next & h->prev & 0xC0000003) == 0xC0000003)
  - uncksum(ptr) = (ptr << 2) & 0x3FFFFFFC
- Overwriting the next/prev pointers in a free block allows an attacker to write a chosen value to a chosen location when that block is removed from the free list

Heap metadata overwrites

- In Leopard it is trivial to overwrite heap metadata to get arbitrary 4-byte writes (see MHH)
  - You know how to fake the checksums
- In Snow Leopard, this can't easily be done due to security cookie

Snow Leopard

- In Snow Leopard, random security cookie used

```
...
    szone->cookie = arc4random();
...
static INLINE uintptr_t
free_list_checksum_ptr(szone_t *szone, void *ptr)
{
    uintptr_t p = (uintptr_t)ptr;
    return p | free_list_gen_checksum(p ^ szone->cookie);
}

static INLINE uintptr_t free_list_gen_checksum(uintptr_t ptr)
{
    uint8_t chk;

    chk  = (unsigned char)(ptr >>  0);
    chk += (unsigned char)(ptr >>  8);
    chk += (unsigned char)(ptr >> 16);
    chk += (unsigned char)(ptr >> 24);

    return chk & (uintptr_t)0xF;
}
```

Application data overflows

```
#include <iostream>
using namespace std;

class Base
{
public:
     virtual void function1() {};
     virtual void function2() {};
};

int main()
{
     int *buf = (int *)malloc(4*sizeof(int));;
     memset(buf, 0x41, 4*sizeof(int));

     Base *pClass = new Base();
     buf[4] = (int) buf;  // overflow into pClass on heap

     pClass->function1();
}
```

```
(gdb) r
Starting program: /Users/cmiller/test2
Reading symbols for shared libraries ++. done

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x41414141
0x41414141 in ?? ()
```

10.5 Zones, Regions, and Allocations

- A process has a number of malloc zones
- Each malloc zone manages allocations in tiny, small, large, and huge size ranges
- Tiny and small range allocations are managed in regions
- Huge and large allocations are managed directly
- A region contains allocations of sizes bounded by the region type

10.6 Magazine Malloc

- Each zone has a magazine per CPU-core (or virtual CPU-core if hyperthreading is used)
- Regions are now specific to the magazine of the CPU core that created them
- Allocations are stored in the region specific to the CPU core running the thread that allocated them

Exploiting Magazine Malloc (10.6)

- Free block free list pointer checksums are now XOR'd with a randomly generated security cookie
  - Effectively defeats heap block metadata overwrites
- Per-CPU regions complicate reliable exploitation if overflown or freed object is "tiny" or "small"
  - Non-deterministic use of different regions complicates heap manipulation
  - Reliability may become dependent on number of CPU cores on target (i.e. new MBP has 8 b/c of HyperThreading)

ASLR

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called “sandboxing” — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

Library Randomization

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

Library Randomization

Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

Library Randomization

- No significant change from Leopard
- Library load locations are randomized per machine periodically when new apps or updates are installed
  - Not per application or application launch
  - See /var/db/dyld/
- dyld, application binary, heap, stack are not randomized
- 64-bit memory space allows for “more” randomization

Fixed RX areas (ROP targets)

- dyld: 0x7fff5fc00000
- binary: 0x100000000
- commpage 64-bit: 0x7fffffe00000

Fun with wild writes

- Many times with exploitation, the “primitive” is to be able to write a DWORD to memory
- This write should eventually lead to getting control of $pc

32-bit processes

- Still use lazy symbol binding
- At fixed, predictable location in memory
- Is writable

```
__la_symbol_ptr:0000201C ; ===========================================================================
__la_symbol_ptr:0000201C
__la_symbol_ptr:0000201C ; Segment type: Pure data
__la_symbol_ptr:0000201C __la_symbol_ptr segment dword public 'DATA' use32
__la_symbol_ptr:0000201C                 assume cs:__la_symbol_ptr
__la_symbol_ptr:0000201C                 ;org 201Ch
__la_symbol_ptr:0000201C _exit_ptr       dd offset __imp__exit   ; DATA XREF: _exit↑r
__la_symbol_ptr:0000201C __la_symbol_ptr ends
__la_symbol_ptr:0000201C
```

32-bit example

```
int main(){
     int *p = 0x201c;
     *p = 0xdeadbeef;
}

$ gcc -g -m32 -o test test.c


Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0xdeadbeef
0xdeadbeef in ?? ()
```

64-bit

- No easy function pointers like in 32-bit (no __IMPORT)
- However, the heap is not randomized
- szone pointers are available starting at predictable address following main executable's __DATA segment
  - Memory management pointers
- In particular szone_malloc()

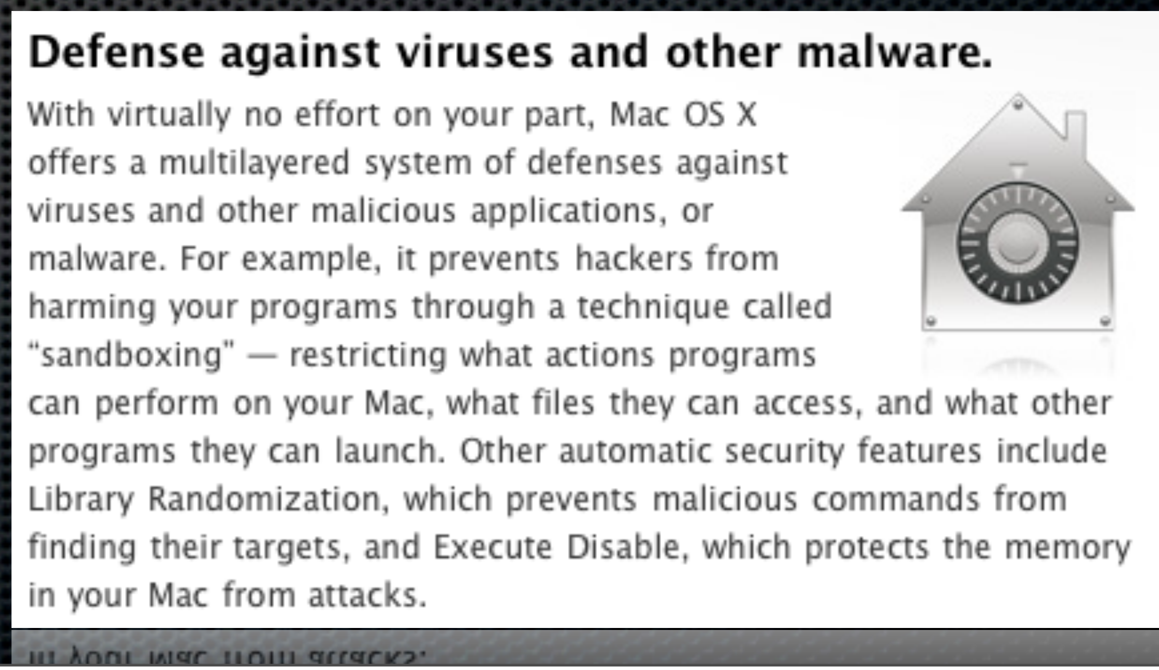
64-bit example

```
int main(){
        long int *p = 0x100004018;
        *p = 0xdeadbeefbabecafe;
        malloc(16);
}

gcc -g -o test test.c

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: 13 at address: 0x0000000000000000
0x00007fff821ddf06 in malloc_zone_malloc ()
(gdb) x/i $pc
0x7fff821ddf06 <malloc_zone_malloc+78>:    call   QWORD PTR [r13+0x18]
(gdb) x/4wx $r13+0x18
0x100004018: 0xbabecafe   0xdeadbeef   0x821e01da   0x00007fff
```

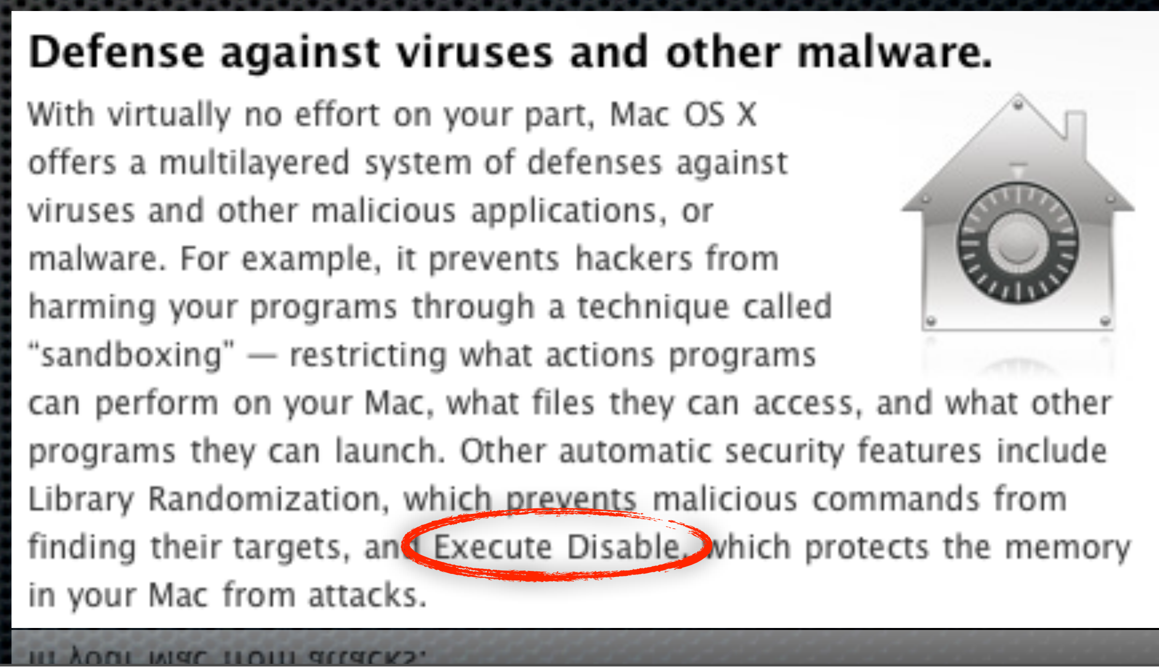
Execute Disable



Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

Execute Disable



Defense against viruses and other malware.

With virtually no effort on your part, Mac OS X offers a multilayered system of defenses against viruses and other malicious applications, or malware. For example, it prevents hackers from harming your programs through a technique called "sandboxing" — restricting what actions programs can perform on your Mac, what files they can access, and what other programs they can launch. Other automatic security features include Library Randomization, which prevents malicious commands from finding their targets, and Execute Disable, which protects the memory in your Mac from attacks.

XD, DEP, PAGEEXEC

- On 32-bit x86, OS memory page permissions are a lie
  - Operating System shows R/W/X permissions
  - Hardware only supports RO or RW page protections
  - XD (Intel) and NX (AMD) added extra page protection bit for eXecute
- PaX PAGEEXEC feature used ITLB/DTLB desync to simulate support of R/W/X memory page permissions

XD < DEP < PAGEEXEC

- Mac OS X "Execute Disable" on 32-bit x86 procs
  - Only thread stacks are non-executable
- Windows DEP
  - Covers stack/heap/data unless a DLL opts out
  - Can be disabled with one function call
- PaX PAGEEXEC
  - The original implementation and most thorough

Leopard

- Stacks were non-executable
- Heap was executable, even though page permissions indicated it was not
- Heap could always be executed, even if explicitly set to not allow execution
- Same for data pages and new pages allocated from the operating system via mmap(), vm_allocate(), etc.

64-bit Non-Executable Memory

- Support for non-executable page protections are required as part of the 64-bit ABI
- 64-bit processes under Snow Leopard are good about keeping memory RW, RX, or RO
  - no RWX except for JIT

Snow Leopard

- Stack and heap are protected (64-bit processes)
  - This is the biggest security difference between Leopard and Snow Leopard
- 32 bit processes (i.e. Flash and QT plugin) have executable heap
  - Exploiting QT or Flash is very easy!
- 32 bit processes (old macs) have executable heaps

What about a Flash JIT spray?

- Flash runs in a separate process, so can’t be used for JIT spray for (non-Flash) browser bugs

JIT spray within Safari

- Potential candidates are Java and Javascript

```
$ vmmap 27581 | grep 'rwx/rwx'
Java                   000000011e001000-0000000121001000 [ 48.0M] rwx/rwx SM=PRV
JS JIT generated code  0000451ca3200000-0000451cab200000 [128.0M] rwx/rwx SM=PRV
JS JIT generated code  0000451cab200000-0000451d23200000 [  1.9G] rwx/rwx SM=NUL
```

Java

- Java memory region is allocated at the “top” of the heap
- Heap is not randomized so you have a reasonable idea of where to find it
- Region is only 48mb and cannot be expanded
- Not a reliable choice for exploitation

Javascript

- Webkit JS RWX region is much larger: 1.9 gb
- However, Webkit randomizes the load address, those bastards

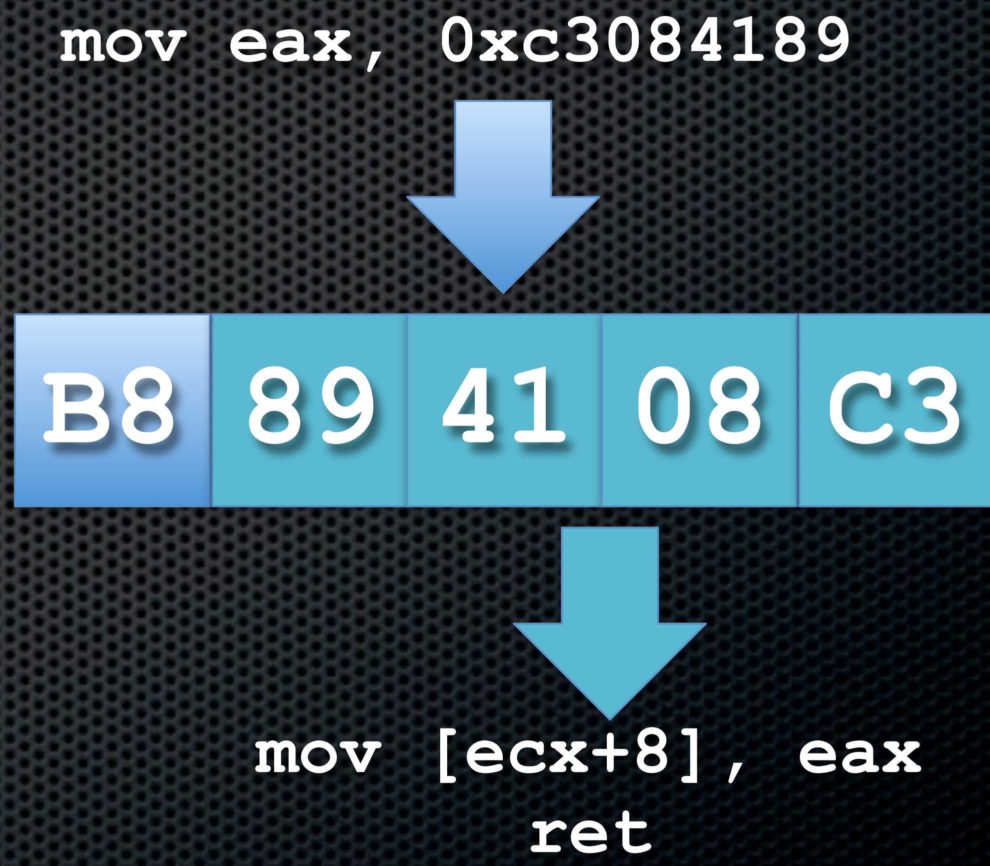
```
#define INITIAL_PROTECTION_FLAGS (PROT_READ | PROT_WRITE | PROT_EXEC)
...
        // Cook up an address to allocate at, using the following recipe:
        //   17 bits of zero, stay in userspace kids.
        //   26 bits of randomness for ASLR.
        //   21 bits of zero, at least stay aligned within one level of the pagetables.
        //
        // But! - as a temporary workaround for some plugin problems (rdar://problem/6812854),
        // for now instead of 2^26 bits of ASLR lets stick with 25 bits of randomization plus
        // 2^24, which should put up somewhere in the middle of usespace (in the address range
        // 0x200000000000 .. 0x5fffffffffff).
        intptr_t randomLocation = arc4random() & ((1 << 25) - 1);
        randomLocation += (1 << 24);
        randomLocation <<= 21;
        m_base = mmap(reinterpret_cast<void*>(randomLocation), m_totalHeapSize,
INITIAL_PROTECTION_FLAGS, MAP_PRIVATE | MAP_ANON, VM_TAG_FOR_EXECUTABLEALLOCATOR_MEMORY
, 0);
```

The good news (for us)

- The location of dyld is not randomized
- The location of the binary is not randomized
- The location of the commpage is not randomized
- We can perform Return Oriented Programming (ROP) to allocate new executable memory or change page permissions for our shellcode

Return-Oriented Programming

- Instead of returning to functions, return to instruction sequences followed by a return instruction
- Can return into middle of existing instructions to simulate different instructions
- All we need are useable byte sequences anywhere in executable memory pages



"The Geometry of Innocent Flesh on the Bone: Return-Into-Libc without Function Calls (on the x86)", Hovav Shacham (ACM CCS 2007)

Return-Oriented Programming

is a lot like a ransom note, but instead of cutting out letters from magazines, you are cutting out instructions from .text segments

Credit: Dr. Raid's Girlfriend

64-Bit Mac OS X ROP

- Passing parameters by register makes things harder than in x86, requiring more “returns”
- Code segments in dyld and commpage are not very large
- Main executable binary may be large enough, but hard to fingerprint
- Problems with rbp
- See Charlie’s presentation at POC 2010 for full ROP details for Snow Leopard x86_64

Wrapping Up

Snow Leopard vs. Leopard

- More secure than ever?
  - Yes, but not by that much since Snow Leopard still only implements a cheap imitation of ASLR
- Safe from attackers?
  - Depends on who your attackers are
  - May stay safe from mass malware, but not from targeted attacks
  - Will APT switch to Mac?

Mac OS X 10.7 “Lion”

- Will Mac OS X 10.7 Lion be the King of the Internet Jungle?
  - Yes, if they implement full ASLR and code signing enforcement for built-in and Mac App Store applications
  - Pre-releases are under NDA, so who knows?

Questions

Charlie:

cmiller@securityevaluators.com

@0xcharlie

Dino:

ddz@theta44.org

@dinodaizovi