Breaking Security Systems:
From Vulnerability Discovery to Disclosure

Charlie Miller | Independent Security Evaluators | 04/08/08 | Session Code: HT2-106

RSA CONFERENCE 2008

Agenda

- Why break systems?
- Disclosure options
- Hurdles to disclosure
- Case studies
  - Technical details
  - Method and result of disclosure

My One Slide Resume

- Background
  - PhD from University of Notre Dame
  - 5 years at the NSA
- Principal Analyst at Independent Security Evaluators
- Author
  - *"Fuzzing For Software Security Testing and Quality Assurance"*, with Jared DeMott and Ari Takanen, due out in May
- Security Researcher
  - Have discovered bugs, written papers, given talks, etc.

Why Break Systems?

- It's fun
- It's interesting and challenging
- Learn how to build systems more securely
- Learn to think like an attacker
- Raise awareness of defective products
  - e.g. an exploding coffee maker
  - There is no cyber consumer reports
- Keep vendors honest
  - E.g. Microsoft
  - More likely to try to get security right the first time

Reasons You Break Things

- You are responsible for the system's security
- Someone hired you to test the security of a system
- You are a researcher
  - Proving utility of new analyses technique
  - Raising your profile
- Someone says their product is unbreakable
- You have nothing better to do

What's Unique About Security Research?

- Controversial
  - Radio program example
  - Paper pre-release dance
- Real world impact
  - Public policy, e.g. IP protection
  - Congressional hearings
  - Mainstream press coverage
    - Constantly speaking with non-techies
  - Help drafting legislation

Involving Lawyers

- Before you break something, consider:
  - Why are you breaking this?
  - Who will be upset if you succeed?
  - How will you disclose this?
  - DMCA
    - Does the system protect copyrighted material?
    - Do you qualify for any of the exemptions?
    - Must consider criminal as well as civil liability
  - Trade secret
    - Federal laws & state
  - EULA
  - How likely are you to be sued?
    - Who will defend you
    - How much will it cost?
- Call the EFF!!!

DMCA: Samuelson example

- There are legitimate reasons to circumvent a copyrighted content protection scheme.
- Imagine you are handed an encrypted object
  - Reason to believe it contains a virus/worm
  - This virus/worm is wreaking havoc
  - You are a security expert asked to circumvent the encryption
- Attempting to extract virus/worm would be a DMCA violation
  - Criminal offense
  - Civil offense
- Encryption research exemption has too many caveats, might not apply
  - E.g. if you were paid to do the analysis

Disclosure

- First step!
  - Get your own organization behind you
    - Your management
    - Your legal department
    - Your subordinates
  - This is critical

Disclosure (Continued)

- When not a good idea
  - Public harm is possible
    - Examples
      - How to shut down power grid
      - How to steal money from online bank
      - National security risks
- What to disclose
  - Proof of vulnerability
  - Not recipe for attack
- Consider timing of your disclosure
  - E.g. voting machine vulnerability & next election

Ground Rules

- Do not break the law
  - No active attacks against systems you don't own
  - No physical trespassing
- Don't cause damage that results in loss to others
  - Second Life example
- Get legal advice
  - Read the EULA, consider DMCA
  - Talk to a lawyer
- Never forget why you are doing this

Disclosure options

- No disclosure
- Full disclosure
- Responsible disclosure

No Disclosure

- Can mean a few things
  - Don't tell anyone
  - Just tell your friends
  - Sell information to interested third party
- Pros
  - Little chance of legal action
  - Requires little work - easy
  - Possible financial gain
- Cons
  - Consumers may not be protected

Full Disclosure

- Reveal information without previously contacting vendor
- Pros
  - Vendors tend to react quickly to this information
  - Trivial to do
  - Can raise your profile - for good or bad
- Cons
  - Puts consumers at risk until patch is developed
  - Can provide recipe for bad guys

Responsible Disclosure

- Contact vendor with vulnerability details
- Wait for vendor to develop patch, fix, or new version of product
- Coordinate release of vulnerability information
- Pros:
  - Consumers are protected at all times
  - Can develop good relationship with vendor
- Cons:
  - Vendor may not be cooperative
  - Vendor may not understand the severity of the vulnerability
  - Vendor may not develop patch in a timely manner
    - Oracle has taken over two years to patch

When Disclosure is a Bad Idea

- You want the vendor to work quickly

ZDI-CAN-226	Symantec	High	2007-09-14, 195 days ago
ZDI-CAN-211	Microsoft	High	2007-07-20, 251 days ago
ZDI-CAN-206	Hewlett-Packard	High	2007-07-17, 254 days ago
ZDI-CAN-224	Oracle / PeopleSoft	High	2007-07-13, 258 days ago
ZDI-CAN-222	Motorola	High	2007-07-10, 261 days ago
ZDI-CAN-200	IBM	High	2007-05-22, 310 days ago
ZDI-CAN-185	Hewlett-Packard	High	2007-05-22, 310 days ago
ZDI-CAN-174	Symantec	High	2007-05-22, 310 days ago
ZDI-CAN-186	Microsoft	High	2007-03-29, 364 days ago
ZDI-CAN-177	Hewlett-Packard	High	2007-03-19, 374 days ago
ZDI-CAN-175	Microsoft	High	2007-03-19, 374 days ago
ZDI-CAN-165	Novell	High	2007-03-09, 384 days ago
ZDI-CAN-160	Oracle / PeopleSoft	High	2007-01-29, 423 days ago
ZDI-CAN-105	Hewlett-Packard	High	2006-10-10, 534 days ago
ZDI-CAN-103	Microsoft	High	2006-09-14, 560 days ago
ZDI-CAN-088	Computer Associates	High	2006-09-12, 562 days ago
ZDI-CAN-063	Computer Associates	High	2006-09-12, 562 days ago

- You fear legal prosecution
  - http://www.securityfocus.com/columnists/466/4

Case Study 1: Apple's iPhone

Background

- Jake Honoroff and I had discovered a Safari (Webkit) bug in January 2007
  - Crash only - didn't seem exploitable
  - Didn't disclose
- Apple released the iPhone on June 19, 2007
- My boss was showing it off to me a few weeks later
  - Tried out the WebKit POC on iPhone - crashed
- Decided it would be interesting to evaluate the security mechanisms of the iPhone

Finding the Bug

- Started with WebKit
  - Most Apple Internet applications share this same code
  - WebKit is an open source library
- Got the source code and did some code coverage analysis
- From the development site:

The JavaScriptCore Tests

If you are making changes to JavaScriptCore, there is an additional test suite you must run before landing changes. This is the Mozilla JavaScript test suite.

Some Code Needs More Attention

LTP GCOV extension - code coverage report

Current view: directory
Test: testsuite.info
Date: 2007-06-01
Code covered: 59.3 %
Instrumented lines: 13622
Executed lines: 8073

Directory name	Coverage		
/System/Library/Frameworks/CoreFoundation.framework/Headers		100.0 %	1 / 1 lines
/System/Library/Frameworks/JavaVM.framework/Headers		0.0 %	0 / 53 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/API		0.0 %	0 / 474 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/bindings		0.0 %	0 / 530 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/bindings/c		0.0 %	0 / 190 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/bindings/jni		0.0 %	0 / 890 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/bindings/objc		0.0 %	0 / 476 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/kjs		79.3 %	5723 / 7219 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/pcre		54.7 %	1338 / 2445 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/wtf		0.0 %	0 / 56 lines
/usr/include		100.0 %	2 / 2 lines
/usr/include/architecture/i386		100.0 %	3 / 3 lines
/usr/include/c++/4.0.0/bits		50.0 %	4 / 8 lines
/usr/share		89.7 %	96 / 107 lines
JavaScriptCore/kjs		84.8 %	357 / 421 lines
kjs		0.0 %	0 / 39 lines
wtf		76.9 %	528 / 687 lines
wtf/unicode/icu		100.0 %	21 / 21 lines

Generated by: LTP GCOV extension version 1.5

Results Indicated

- 59.3% of 13622 lines in JavaScriptCore were covered
  - The main engine (53% of the overall code) had 79.3% of its lines covered
  - Perl Compatible Regular Expression (PCRE) library (17% of the overall code) had 54.7% of its lines covered
- We decided to investigate PCRE further

Let the Fuzzing Begin

- Fuzzing is a technique for finding bugs - especially security related
- Send many malformed inputs into a system and monitor it for problems
  - Inputs shouldn't be too random, or system may quickly reject it
  - Inputs shouldn't be too well behaved, or system may not exhibit problem
- Best case scenario: program crashes

And So

- Wrote a PCRE fuzzer (20 lines of perl)
- Ran it on a standalone PCRE parser (pcredemo from the PCRE library)
- We started getting errors like:

```
PCRE compilation failed at offset 6: internal error: code overflow.
```

- This was good

Now We Need a POC

- Problem:
  - At the time, the iPhone didn't have terminal access (when running), a compiler, or debugger
- However:
  - It did have CrashReporter and we figured out how to make it dump core
- So:
  - We (joined by Josh Mason) fuzzed the iPhone with various Javascript regular expressions containing "[[**]]"
  - Sorted through the crash reports until we found a good one

Eventually Found One

```
Thread 2 crashed with ARM Thread State:
r0: 0x00065000    r1: 0x0084f800    r2: 0x00000017    r3: 0x15621561
r4: 0x00000018    r5: 0x0084ee00    r6: 0x00065000    r7: 0x005523ac
r8: 0x0000afaf    r9: 0x00817a00   r10: 0x00ff8000   r11: 0x00000005
ip: 0x15641563    sp: 0x00552358    lr: 0x30003d70    pc: 0x3008cbc4
cpsr: 0x20000010 instr: 0xe583c004
```

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

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

Almost Done

- Gave us the ability to write four bytes anywhere
- Decided to overwrite a return address on the stack
  - found via brute forcing
- Overwrote with the address of a buffer we controlled
- When function returns, we get control

Payload

- iPhone is very stripped down
  - No /bin/sh even!
- iPhone runs on an ARM processor
- Had to write shellcode from scratch
- At least the browser (and hence our shellcode) runs as root!

Shellcode

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

Got a Working POC, Now What?

- Got our management behind us
- My boss, Avi Rubin contacted Jason Schultz at the EFF
  - Advised us that this was probably legal and Apple didn't have a history of suing researchers
    - Makes you feel all warm and cozy inside...

Disclosure Time

- The day we finished our POC we contacted Apple (July 17)
  - Used their recommended approach: product-security@apple.com
  - Informed them of this and one other bug
  - Supplied the POC
  - Supplied the line of source code in WebKit where the bug occurs
  - Supplied a source code patch for the bug
  - Informed them we would release the information at BlackHat USA on August 2 (in 17 days)
  - Asked for them to coordinate release of the patch for that day

Initial Response

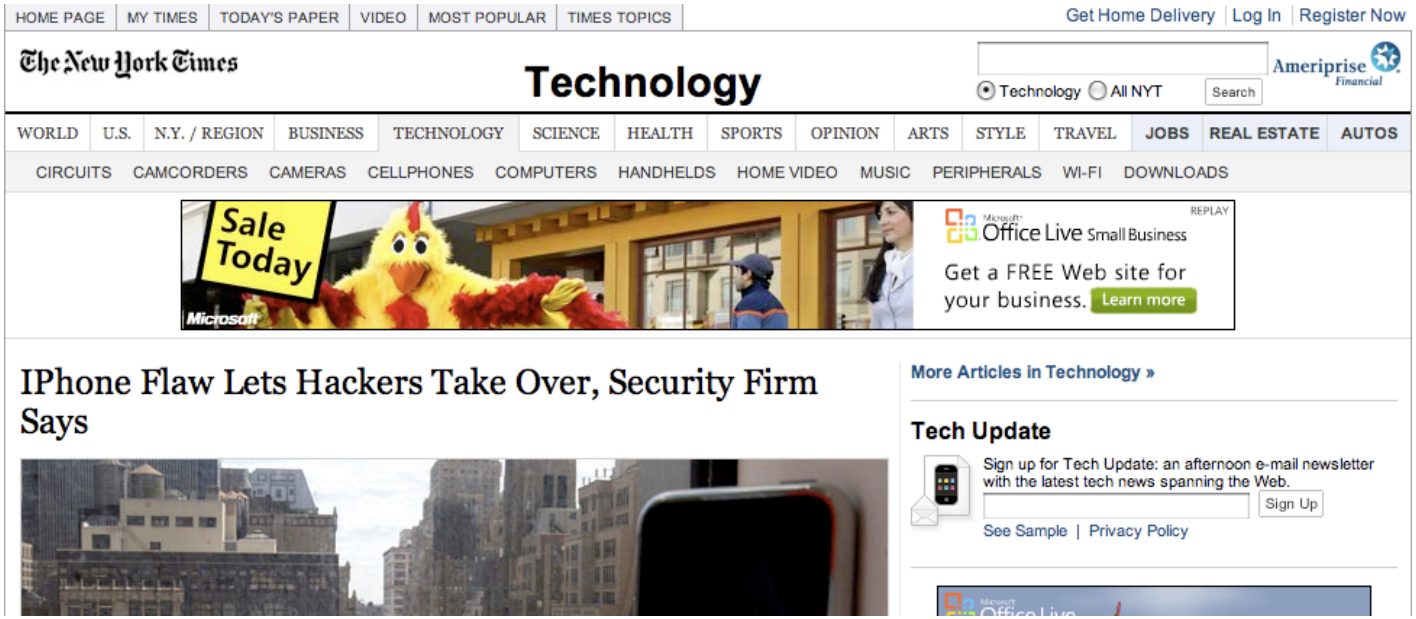
- “Thank you for forwarding this issue to us. We take any report of a potential security issue very seriously”
- ....
- “Would you be willing to keep this information private until we are able to address it on our affected platforms?”

Never Let Them See You Sweat

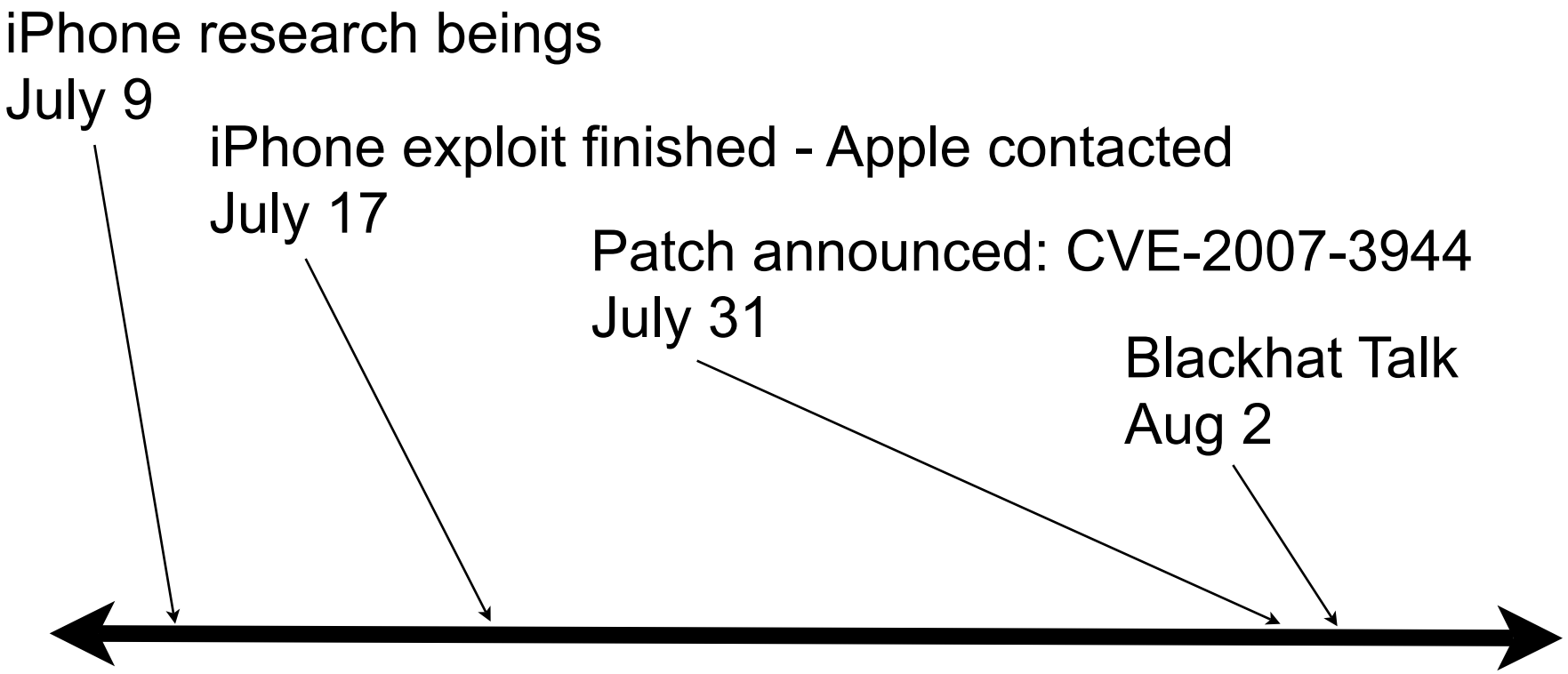
- No, I'm scheduled to give a talk that day and it can not be rescheduled
  - Is this a reasonable reaction on my part?
    - "Given the platforms affected, the testing requirements, and our release cycles, we may not have a patch available by the time of your talk. We appreciate that you will not disclose the details of the vulnerabilities."
- I appreciate your problems but I can not change the date, I urge to you coordinate the patch release for that date
  - Again, is this reasonable?

Publicity

- Contacted a reporter at the NY Times two days later.
- Article ran July 23rd
- No technical details given



iPhone Timeline



iPhone Summary

- Patch was released just prior to technical details being revealed
- Patch was released in 14 days - very quick
- Apple was very polite - although not always “verbose”
- What would have happened if August 2nd came and there was no patch?
- Overall, very good experience
  - However, was very high profile...

Case Study 2: Samba

- Samba is a common open source server which implements the Server Message Block (SMB) protocol
- Found on many Linux and Mac OS X computers - especially in Windows environments
- Allows Windows-style file sharing
- Has a history of bugs...

Bug Discovery

- I discovered a remote, pre-authentication exploit in Samba sometime in the Summer of 2005
- Found it the old fashioned way: reading the source code
  - Not Hollywood exciting - but effective

The Vulnerability

```
static BOOL lsa_io_trans_names(const char *desc, LSA_TRANS_NAME_ENUM2 *trn, prs_struct
*ps, int depth)
{
...
        if(!prs_uint32("num_entries    ", ps, depth, &trn->num_entries))
...
        if (trn->ptr_trans_names != 0) {
                if(!prs_uint32("num_entries2   ", ps, depth, &trn->num_entries2))
                        return False;
....
                if (UNMARSHALLING(ps)) {
                        if ((trn->name = PRS_ALLOC_MEM(ps, LSA_TRANS_NAME2, trn-
>num_entries)) == NULL) {
                                return False;
...
                }
                for (i = 0; i < trn->num_entries2; i++) {
...
                        if(!lsa_io_trans_name2(t, &trn->name[i], ps, depth))
```

The Exploit

- Heap overflow
- Can control the size of the malloc (mostly)
- Can overwrite with an arbitrary amount of data
- Data has no restrictions on byte values

Heap Overflows in Linux Can Be Hard

- Luckily, Samba implements it's own heap management
  - Use of this is controlled by a #ifdef
    - Present in SuSE distro, not some others

```
struct talloc_chunk {
        struct talloc_chunk *next, *prev;
        struct talloc_chunk *parent, *child;
        struct talloc_reference_handle *refs;
        talloc_destructor_t destructor;
        const char *name;
        size_t size;
        unsigned flags;
};
```

Samba Exploit

```
int talloc_free(void *ptr)
...
  tc = talloc_chunk_from_ptr(ptr);
...
        if (tc->destructor) {
                talloc_destructor_t d = tc->destructor;
...
                if (d(ptr) == -1) {
...
```

- Overflow the destructor function pointer with the address of system().
- Put command to run at the begining of the buffer being free'd
- Get system(command). Nice!

Ok, Got a POC - Now What?

- I was not employed by a company that particularly cared about vulnerability disclosure
- This was a very serious vulnerability
  - Send some packets - get remote root
- Took significant time and effort to find this bug
  - Had been in the code for years
- Thought I'd see about the no-disclosure route
  - i.e. selling it

How Exactly Does One Sell an Exploit?

- Good question!
- Can try sending out emails to companies that buy/sell these things
  - Immunity
  - Netragard
  - Argeniss
- Can use an established company
  - iDefense, TippingPoint
- Now you could use an auction site
  - Wabisabilabi

Another Option

- Contacted a friend who had friends
- Negotiated a deal with a large company for $80,000 for exclusive knowledge of it
  - No guarantee they would disclose it (or not)
    - They didn't afaik
  - Had to work against a particular flavor of Linux
  - My friend would get a 10% commision
- They tried to get it to work for this other Linux...
- Eventually I took $50,000 for it as is

Proof?

```
echo "Charlie Miller found a vulnerability in Samba in the function
lsa_io_trans_names where trn->num_entries and trn->num_entries2 are
of different sizes." | md5sum
 e9a4f234e0f5d3e587c3d27e709b7eda  -
```

[Full-disclosure] Security researcher

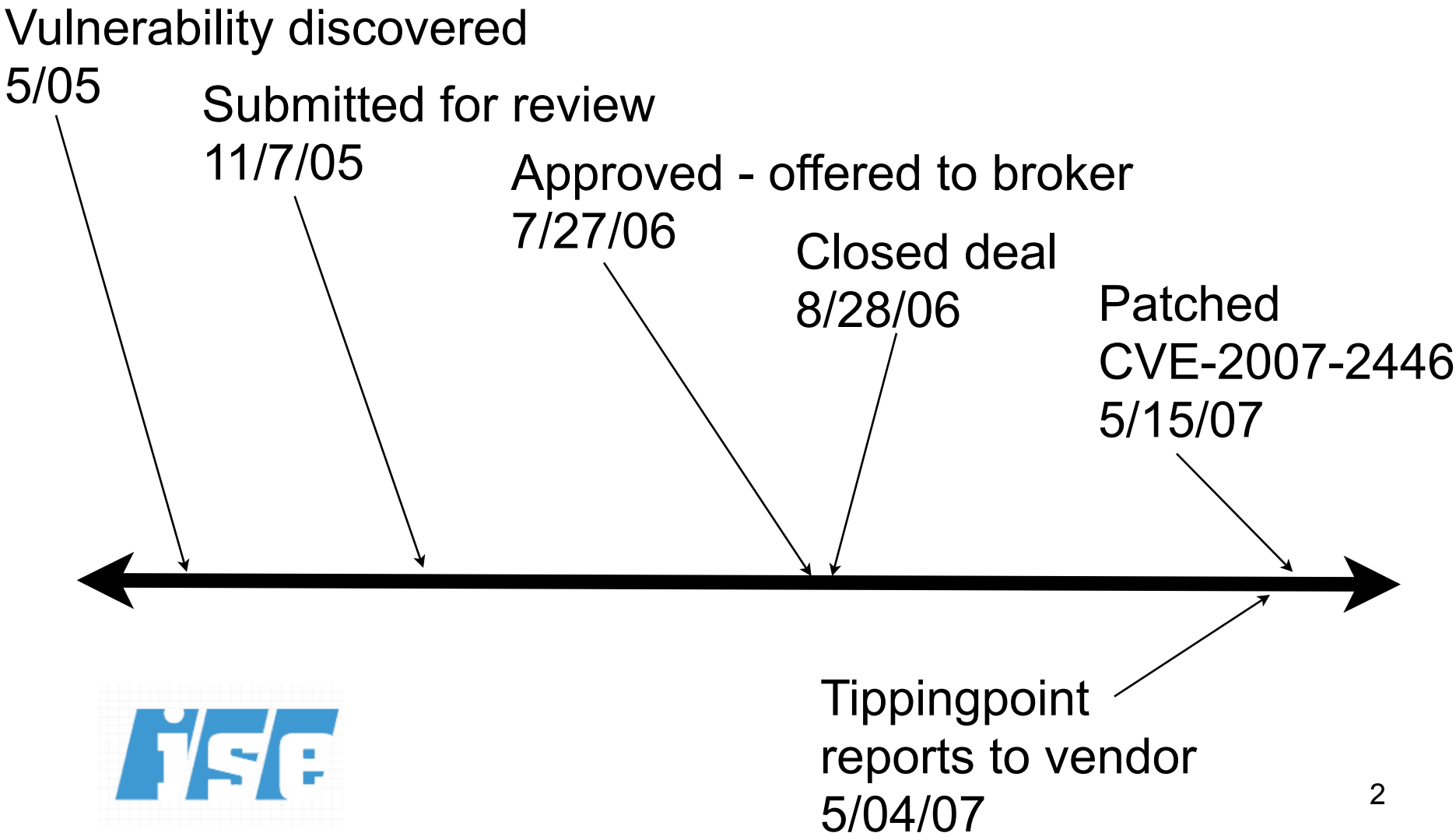
From: asdfasf (*zerodayinithotmail.com*)
Date: Fri Aug 25 2006 - 09:01:39 CDT
Messages sorted by: [date] [thread] [subject] [author]

I'm looking for a security researcher named "Gobbles". If anyone could send me his contact information I would appreciate it.
sadf
e9a4f234e0f5d3e587c3d27e709b7eda

_____

Full-Disclosure - We believe in it.
Charter: http://lists.grok.org.uk/full-disclosure-charter.html
Hosted and sponsored by Secunia - http://secunia.com/

Samba Timeline



Samba Wrap-Up

- The government is slow
- Didn't disclose vulnerability
- Vulnerabilities can live in products for years
- Samba users at risk approximately 9 (additional) months
- Was this unethical, wrong, evil?
  - How many people would honestly trade $50k for their name in a changelog and a warm fuzzy?
- What other mechanisms could we use to ensure this practice doesn't occur?

Case Study 3: QuickTime

- QuickTime is Apple's multimedia player
- Available for Mac OS X and Windows
- Can play a variety of formats
  - 3g2, aac, amc, avi, caf, gif, jpg, mp3, etc.
- Has a history of bugs
  - 34 security bugs fixed in 2007 alone
- Ripe for the picking!

Fuzz It

- I chose the .avi file format to fuzz.
- I used a mutation based approach
  - Obtain valid .avi file
  - Change random bytes of the file
  - Launch QuickTime with these mutated files
  - Wait for it to crash
  - Sort through the crashes to find the vulnerabilities

fuzz.c

- Written by Josh Mason
- Very simple C program
- Changes from 1-16 bytes in a file
- Replaces those bytes with a random byte
- Sad that this program can find an endless stream of bugs in QuickTime
  - Are you listening Apple?

Creating the Mutated Files

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#define NUM_FILES 8192
#define SIZE 6250577

int main(void)
{
        FILE *in, *out, *lout;
        unsigned int n, i, j;
        char *buf = malloc(SIZE);
        char *backup = malloc(SIZE);
        char outfile[1024];
        int rn, rn2, rn3, rn4;
        int rbyte;
        int numwrites;

        char tmp, tmp2, tmp3, tmp4;
        char cmd[4096];
        char crap[1024];
        int pid;
        char deathcmd[4096];
        in = fopen("good.avi", "r");
        n = read(fileno(in), buf, SIZE);
        memcpy(backup, buf, n);

        lout=fopen("list", "w");

        srand(time(NULL));
```

```
for (i=0;i<NUM_FILES;i++)
{
        // seek and write
        numwrites=rand() % 16;
        numwrites++;

        printf("[+] Writing %d bytes\n", numwrites);
        for (j=0;j<numwrites;j++)
        {
                rbyte = rand() % 257;
                if (rbyte == 256)
                        rbyte = -1;
                rn = rand() % n - 1;
                printf("[+] buf[%d] = %d\n", rn, rbyte);
                buf[rn] = rbyte;
        }

        sprintf(outfile, "bad-%d.avi", i);
        out = fopen(outfile, "w");
        write(fileno(out), buf, n);
        fclose(out);
        fprintf(lout, "%s\n", outfile);
        memcpy(buf, backup, n);
}
```

Launch the Fuzzed Files

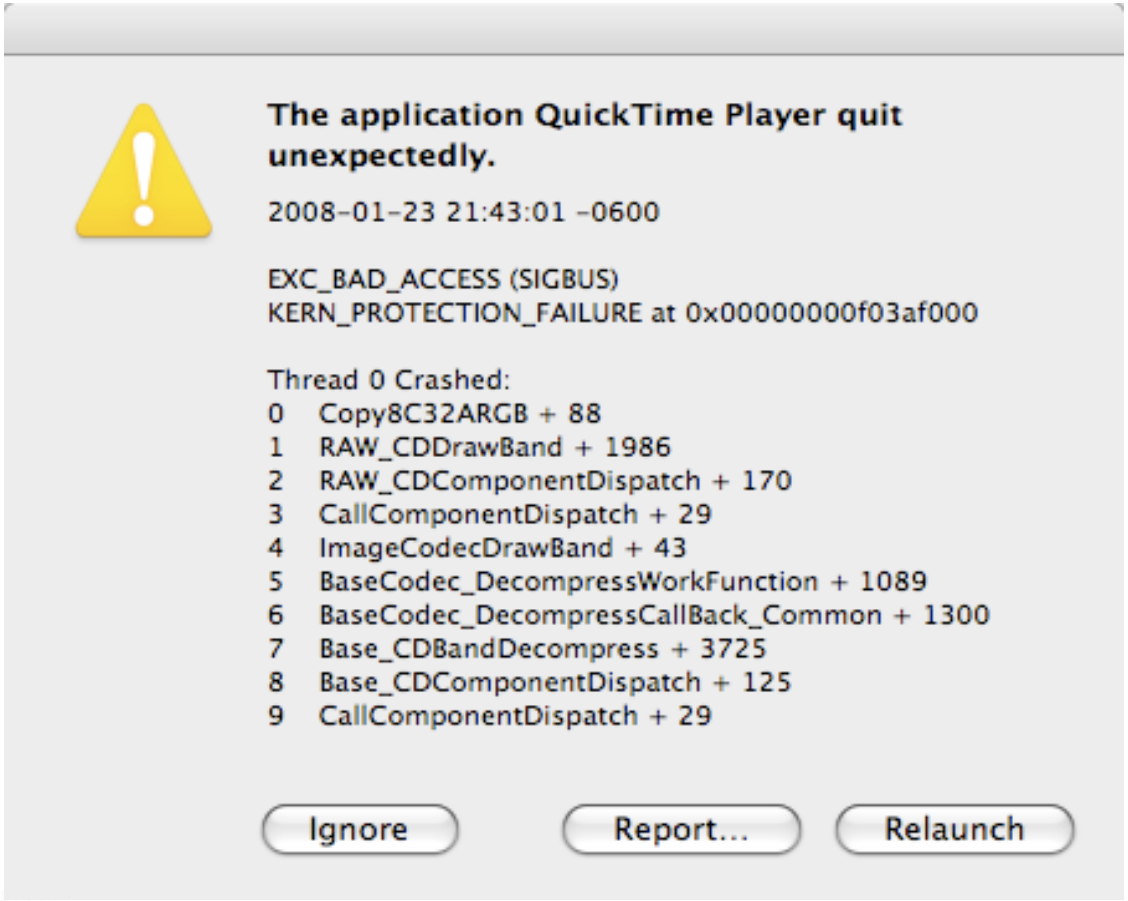
```
#!/bin/bash
VAR=0;
X=0;
Y=`wc -l /var/log/crashreporter.log | awk '{print $1}'`;
for i in `cat list`;
        do
                echo $i;
               DYLD_INSERT_LIBRARIES=/usr/lib/libgmalloc.dylib /Applications/QuickTime\
Player.app/Contents/MacOS/QuickTime\ Player $i &

                VAR=`expr $VAR + 1`;

                if [ $VAR == 10 ]; then
                        sleep 60;
                        X=`wc -l /var/log/crashreporter.log | awk '{print $1}'`;
                        echo $X;
                        echo $Y;

                        if [ $Y -lt $X ]
                        then
                                echo "Sometime before: $i";
                        fi
                        VAR=0;
                        Y=$X;
                        killall -9 QuickTime\ Player;
                fi
        done
```

Wait For CrashReporter



Fuzzing Results

- 20 potential files to examine
- Reveals 2 distinct files causing crash
- Each file triggers the same bug
- The bug is in a loop which eventually reads off the end of valid memory
- Exploitable?
  - Dunno. Enough free security work from me for the moment.

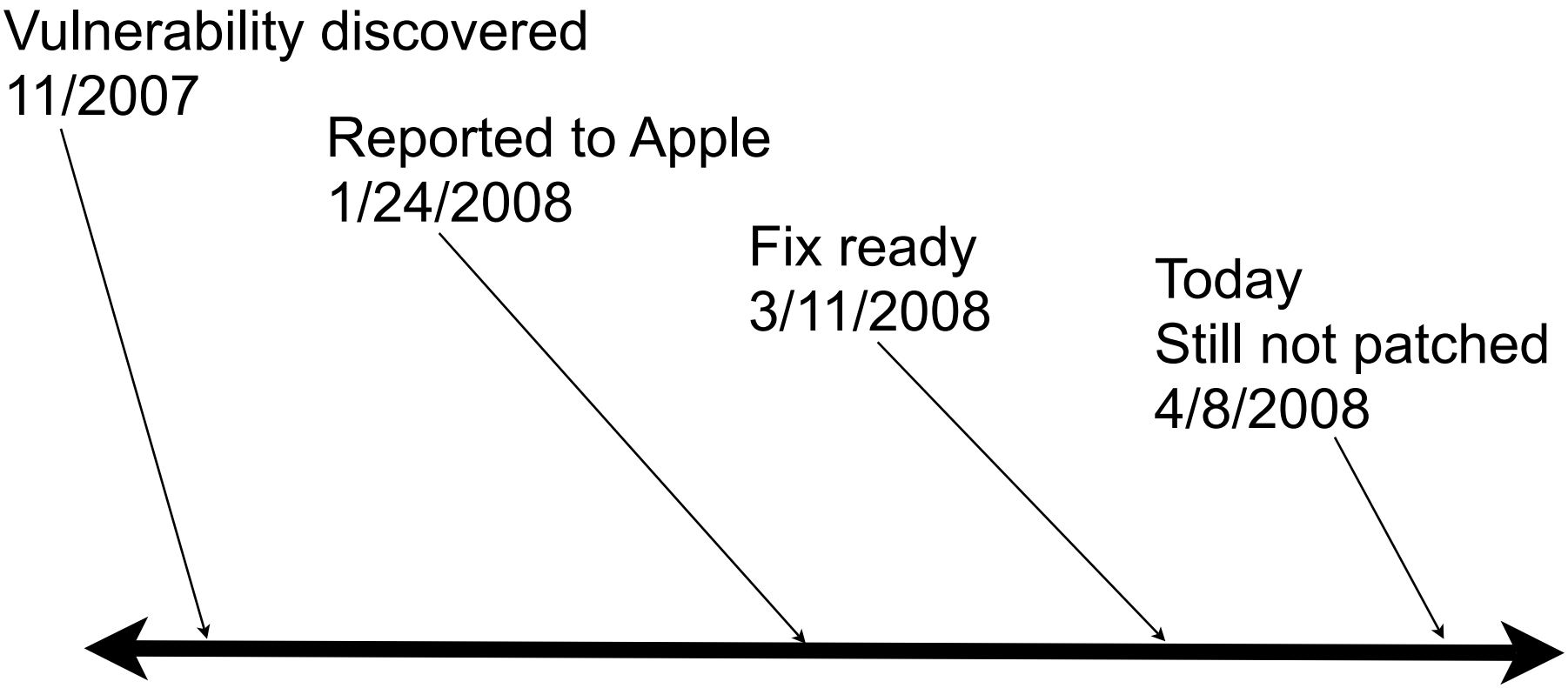
Disclosure

- Not particularly dangerous or hard to find
- Pretty easy choice, I chose responsible disclosure
- Contacted Apple January 24th, 2008
  - Provided details of the crash
  - Provided fuzzed file which crashes the program
- Received auto reply
  - "...We take any report of a potential security issue very seriously..."
- Final contact on March 11th, 2008 (1.5 months later)

Apple's Response

- "We have analyzed the events surrounding the crash you reported, and there are no side effects that weaken the security of the system.
- However, we noticed that the .avi file provided also crashes Internet Explorer on Windows XP even when QuickTime is not installed. We'll pass it on to Microsoft for them to evaluate. We'll also give them your name so they can give you credit, if that's OK with you.
- Since it doesn't appear to be a security problem for us, and to give Microsoft a chance to evaluate it, we won't mention it in a security advisory.
- We have the fix ready and it'll most likely be in the next release."

QuickTime Timeline



Summary of QuickTime Disclosure

- Vulnerability not as high profile - and potentially less dangerous - than iPhone bug
- Apple was not as quick to respond in this case
- Start to see the frustration of responsible disclosure
- Did they “patch” it but misunderstand the issue?
- I didn’t receive any credit if they did

Conclusions

- Breaking stuff is fun
- Lots of ways to find bugs
  - Source code analysis
  - Fuzzing
- Lots of ways to disclose them
  - Responsibly
  - Not so “responsibly”

The System is Broken

- Responsible disclosure
  - Get credit for your discovery (hopefully)
  - Must convince the company there is a bug
  - Typically, wait for company to fix the bug at their pace
  - Worst case, the company threatens to sue you or you could face possible criminal action
- No disclosure
  - You don't get credit for discovering it (publish a hash...)
  - You possibly get lots of money
  - You don't get sued
  - You don't have to deal with companies

Final Thoughts

- We need to make responsible disclosure easier and more pleasant
- We need to reward researchers for their work
- The community needs to make it so that responsible disclosure is the preferred method by any measurement
  - Not the method researchers have to make sacrifices to use

Final Thoughts #2

- Samba story revisited
  - If the anonymous researcher had been offered $80,000 for his Samba bug, instead of (an estimated) $5000 + disclosure, would he have taken it?
  - Researchers shouldn't be put in this position

Questions?

Contact me at:
cmiller@securityevaluators.com