Gatekeeper Exposed

come, see, conquer!
“leverages the best combination of humans and technology to discover security vulnerabilities in our customers’ web apps, mobile apps, IoT devices and infrastructure endpoints”
SYNACK & THE SYNACK RED TEAM (SRT)
join, find bugs, profit!

signup → pass → find bugs → get paid!

why Synack?

smaller 'crowd' + larger customers = more, higher, faster, payouts
OUTLINE
all aspects of gatekeeper

Gatekeeper

understanding  bypassing  fixing
...os x trojans everywhere? everywhere!

countless OS X users infected

LIFE BEFORE GATEKEEPER

jahlav-a
rkosx-a
hovdy-a

devilrobbie
opinionspy
revir
boonana
qhost

leap-a
rsplug
macsweeper
iwork-a
pinhead
macdefender

gatekeeper

2006 2007 2008 2009 2010 2011 2012
Gatekeeper aims to protect as there is no patch for human stupidity ;)

Gatekeeper is a built-in anti-malware feature of OS X (10.7+): "If a [downloaded] app was developed by an unknown developer—one with no Developer ID—or tampered with, Gatekeeper can block the app from being installed" -apple.com

TL;DR block unauthorized code from the internet
Gatekeeper Protect Users
...from low-tech adversaries

Gatekeeper Slams the Door on Mac Malware Epidemics - tidbits.com

- Rogue "AV" products
- Fake codecs
- Fake installers/updates
- Infected torrents
- Poor naive users!
GATEKEEPER PROTECTS USERS
from high-tech adversaries

Q1 2015: all security software, I downloaded -> served over HTTP :(

mitM + infect insecure downloads

my dock

LittleSnitch
Avira
Kaspersky
intego
Bitdefender
ClamXav
Sophos
Norton
Trend Micro
**How Gatekeeper Works**

**An overview**

//attributes

$ xattr -l ~/Downloads/malware.app
com.apple.quarantine:0001;534e3038;
Safari; B8E3DA59-32F6-4580-8AB3...

**Quarantine attributes**

**Gatekeeper in action**

**Gatekeeper settings**

iff quarantine attribute is set!
# Extended File Attributes

## Simply Put; File Metadata

<table>
<thead>
<tr>
<th>attribute</th>
<th>brief details</th>
</tr>
</thead>
<tbody>
<tr>
<td>extended attr. (com.apple.*)</td>
<td><img src="https://example.com/table.png" alt="Table" /></td>
</tr>
<tr>
<td>FinderInfo</td>
<td>information for <strong>Finder.app</strong> (such as folder colors)</td>
</tr>
<tr>
<td>metadata</td>
<td>Spotlight data, such as download location &amp; version info</td>
</tr>
<tr>
<td>quarantine</td>
<td>indicates that file is from an 'untrusted' source (internet)</td>
</tr>
</tbody>
</table>

## Dumping Quarantine Attributes

```
$ xattr -l ~/Downloads/eicar.com.txt
```

```
com.apple.metadata:kMDItemWhereFroms:
00000000 62 70 6C 69 73 74 30 30 A2 01 02 5F 10 2B 68 74 |bplist00..._.+ht|
00000010 74 70 3A 2F 2F 77 77 77 2E 65 69 63 61 72 2E 6F |tp://www.eicar.o|
00000020 72 6F 72 6F 72 67 2F 64 6F 77 6E 6C 6F 61 64 2F |rg/download/eica|
00000030 65 69 63 61 72 2E 63 6F 6D 2E 74 78 74 5F 10 27 68 74 74 70 |r.com.txt......|

com.apple.quarantine: 0001;55ef7b62;Google Chrome.app;3F2688DE-C34D-4953-8AF1-4F8741FC1326
```

## Dump W/ `xattr` Command

![Dumping quarantine attributes](https://example.com/dump.png)
'FILE QUARANTINE'
realized by the `com.apple.quarantine` file attribute

added in Leopard
"file from internet"

```swift
//dictionary for quarantine attributes
NSDictionary* quarantineAttributes = nil;

//get attributes
[fileURL getResourceValue:quarantineAttributes
forKey:NSURLQuarantinePropertiesKey error:NULL];
```

dumping a file's `com.apple.quarantine` attribute

code to get attributes

```bash
$ dumpAttrs ~/Downloads/eicar.com.txt
LSQuarantineAgentBundleIdentifier = "com.google.Chrome";
LSQuarantineAgentName = "Google Chrome.app";
LSQuarantineDataURL = "http://www.eicar.org/download/eicar.com.txt";
LSQuarantineEventIdentifier = "3F2688DE-C34D-4953-8AF1-4F8741FC1326";
LSQuarantineOriginURL = "http://www.eicar.org/85-0-Download.html";
LSQuarantineTimeStamp = "2015-09-09 00:20:50 +0000";
LSQuarantineType = LSQuarantineTypeWebDownload;
```
**SETTING THE QUARANTINE ATTRIBUTE**

who done it!?  

![Custom Downloader](http://www.eicar.org/download/eicar.com.txt)

//button handler: download file
- (IBAction)download:(id)sender {
  //url
  NSURL *remoteFile = [NSURL URLWithString:self.textField.stringValue];

  //local file
  NSString* localFile = [NSString stringWithFormat:@"/tmp/%@", [remoteFile lastPathComponent]];  

  //download & save to file
  [[NSData dataWithContentsOfURL:remoteFile] writeToFile:localFile atomically:NO];
  return;
}

custom downloader's source code

any extended attributes?  

```
$ xattr -l ~/Downloads/eicar.com.txt
$ dumpAttrs ~/Downloads/eicar.com.txt
```

none; huh?
**Setting The Quarantine Attribute**

Apps can manually add it

```c
-(void)setQAttr:(NSString*)localFile
{
    // quarantine attributes dictionary
    NSMutableDictionary* quarantineAttributes = [NSMutableDictionary dictionary];

    // add agent bundle id
    quarantineAttributes[kLSQuarantineAgentBundleIdentifierKey] = [[NSBundle mainBundle] bundleIdentifier];

    // add agent name
    quarantineAttributes[kLSQuarantineAgentNameKey] = [[NSBundle mainBundle] infoDictionary objectForKey:kCFBundleNameKey];

    // manually add quarantine attributes to file
    [[NSURL fileURLWithPath:localFile] setResourceValues:@{NSURLQuarantinePropertiesKey: quarantineAttributes} error:NULL];
    return;
}
```

Code to set a file's quarantine attribute

```
$ xattr -l ~/Downloads/eicar.com.txt
com.apple.quarantine: 0000;55efddeb;downloader;ED9BFEA8-10B1-48BA-87AF-623EA7599481

$ dumpAttrs ~/Downloads/eicar.com.txt
    LSQuarantineAgentBundleIdentifier = "com.synack.downloader";
    LSQuarantineAgentName = downloader;
    LSQuarantineDataURL = "http://www.eicar.org/download/eicar.com.txt";
    LSQuarantineEventIdentifier = "ED9BFEA8-10B1-48BA-87AF-623EA7599481";
    LSQuarantineTimeStamp = "2015-09-09 07:21:15 +0000";
    LSQuarantineType = LSQuarantineTypeWebDownload;
```
Setting the Quarantine Attribute
or, apps can generically tell the OS to add it

Info.plist keys: LSFileQuarantineEnabled
"When the value of this key is true, all files created by the application process will be quarantined by OS X" -apple.com

app's Info.plist file updated (LSFileQuarantineEnabled)

```
$ grep -A 1 LSFileQuarantineEnabled Info.plist
<key>LSFileQuarantineEnabled</key>
<true/>
```

```
$ xattr -l ~/Downloads/eicar.com.txt
com.apple.quarantine: 0000;55f139c4;downloader.app;
$ dumpAttrs ~/Downloads/eicar.com.txt
LSQuarantineAgentName = "downloader.app";
LSQuarantineTimeStamp = "2015-09-10 08:05:24 +0000";
```

automatically (OS) set, quarantine attribute
**Gatekeeper in Action**

**an overview**

1. Finder.app
2. XPC request
3. LaunchServices framework
4. CoreServicesUIAgent
5. Quarantine.kext

**XPC request**

**Launchd**

**CoreServicesUIAgent**

**XProtect framework**

**Malware.app** can't be opened because it is from an unidentified developer.

Your security preferences allow installation of only apps from the Mac App Store.

Something downloaded this file on an unknown date.
LAUNCHING THE BINARY/APP
handled by the launchservices framework

Finder.app
LaunchServices framework

XPC request

libxpc.dylib:_spawn_via_launchd
LaunchServices`LaunchApplicationWithSpawnViaLaunchD
LaunchServices`_LSLaunchApplication
LaunchServices`_LSLaunch
LaunchServices`_LSOpenApp
LaunchServices`_LSOpenStuffCallLocal
LaunchServices`_LSOpenStuff
LaunchServices`_LSOpenURLsWithRole_Common
LaunchServices`_LSOpenURLsWithRole

call stack

pid_t _spawn_via_launchd(
    const char *label,
    const char *const *argv,
    const struct spawn_via_launchd_attr *spawn_attrs,
    int struct_version
);

_launch_priv.h

(lldb) x/s $rdi
"[0x0-0xb92b92].com.nsa.malware"

(lldb) print *(char**)$rsi
"~/Downloads/Malware.app/Contents/MacOS/Malware"

(lldb) print *(struct spawn_via_launchd_attr*)$rdx
{
    spawn_flags = SPAWN_VIA_LAUNCHD_STOPPED
    ...
}

'spawn' attributes, etc.
Policy Enforcement with Quarantine.kext

Kernel-mode Mac component

XPC request → Quarantine.kext

Launchd

Quarantine policy

hook_vnode_check_exec

//bail if sandbox'ing not enforced
cmp cs:_sandbox_enforce, 0
jz leaveFunction

//bail if file previously approved
call _quarantine_get_flags
and eax, 40h
jnz leaveFunction

//bail if file is on read-only file system
call _vfs_flags ; mnt flags
test al, MNT_RDONLY
jnz leaveFunction

hook_vnode_check_exec

(lldb) print *(struct mac_policy_conf*)0xFFFFF7F8B447110
mpc_name = 0xffffffff7f8b446c3a "Quarantine"
mpc_fullname = 0xffffffff7f8b446cb0 "Quarantine policy"
...
first, the xpc request

LaunchServices framework

getting XPC connection to CoreServicesUIAgent

pseudo code

void ____LSAgentGetConnection_block_invoke(void * _block)
{
    rax = xpc_connection_create_mach_service("com.apple.coreservices.quarantine-resolver",
        dispatch_get_global_queue(0x0, 0x0), 0x0);

    xpc_connection_set_event_handler(rax, void ^(void * _block, void * arg1)
    {
        return;
    });

    xpc_connection_resume(rax);
    return;
}

(lldb) po $rax
{
    LSQAllowUnsigned = 0;
    LSQAppPSN = 3621748;
    LSQAppPath = "/Users/patrick/Downloads/Malware.app";
    LSQAuthorization = <bed76627 c7cc0ae4 a6860100 00000000 ... 
    LSRiskCategory = LSRiskCategoryUnsafeExecutable;
}
4 User Interaction via CoreServicesUIAgent

then, analysis via xprotect

CoreServicesUIAgent

XProtect framework

program control flow

<table>
<thead>
<tr>
<th>XProtectMalwareType</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2</td>
<td>unsigned</td>
</tr>
<tr>
<td>0x3</td>
<td>modified bundle</td>
</tr>
<tr>
<td>0x5</td>
<td>signed app</td>
</tr>
<tr>
<td>0x7</td>
<td>modified app</td>
</tr>
</tbody>
</table>
finally, display the alert

CoreServicesUIAgent

U I NTERACTION VIA CoreSERVICESUIAgent

CoreServicesUIAgent

"malware.app" can't be opened because it is from an unidentified developer.
Your security preferences allow installation of only apps from the Mac App Store.

alert customization

alert strings (QuarantineHeadlines.strings)

$ less QuarantineHeadlines.strings

$ key Q_HEADLINE_CASPIAN_BAD_DISTRIBUTOR
<string>"%@" can't be opened because it is from an unidentified developer.</string>

$ key Q_HEADLINE_CASPIAN_BLOCKED
<string>"%@" can't be opened because it was not downloaded from the Mac App Store.</string>

application termination

gatekeeper alert

Synack
What if the App Conforms & is Allowed by the User?

Quarantine attributes updated, then application resumed

```
-GK Quarantine Resolver
approveUpdatingQuarantineTarget: recursively: volume:
```

```
call __qtn_file_get_flags
or eax, 40h
mov rdi, [rbp+var_B8]
mov esi, eax
call __qtn_file_set_flags
```

```
mov rsi, [r13+r14+0]
mov rax, __kLSApplicationInStoppedStateKey_ptr
mov rdx, [rax]
mov edi, 0xFFFFFFFF
xor r8d, r8d
mov rcx, rbx
call __LSSetApplicationInformationItem
```

; on error
```
lea rsi, "Unable to continue stopped application"
mov edi, 4
xor eax, eax
mov edx, ecx
call logError
```

```
$ xattr -l ~/Downloads/KnockKnock.app/Contents/MacOS/KnockKnock
com.apple.quarantine: 0001;55f3313d;Google\x20Chrome.app;FBF45932...
```

```
$ xattr -l ~/Downloads/KnockKnock.app/Contents/MacOS/KnockKnock
com.apple.quarantine: 0041;55f3313d;Google\x20Chrome.app;FBF45932...
```

Quarantine alert

Updating quarantine attributes

Resuming application

Before & After
BYPASSING GATEKEEPER
unsigned code allowed!?
RECALL; GATEKEEPER AIDS TO PROTECT
...unauthorized code should be blocked!

block unauthorized code from the internet
“malware that comes onto the system through vulnerabilities...bypass quarantine entirely. The infamous Flashback malware, for example, used Java vulnerabilities to copy executable files into the system. Since this was done behind the scenes, out of view of quarantine, those executables were able to run without any user interactions” -www.thesafemac.com
Gatekeeper Shortcomings

- downloading app, must 'support' quarantine attribute

Gatekeeper Shortcomings - Gatekeeper's shortcomings include the reliance on apps to support quarantine attributes. Not all apps do so, which can result in files not being quarantined.

```
xattr -p com.apple.quarantine Adobe\ Photoshop\ CC\ 2014.dmg
xattr: Adobe Photoshop CC 2014.dmg: No such xattr: com.apple.quarantine
```

```
no quarantine attribute :(>
```

"the quarantine system relies on the app being used for downloading doing things properly. Not all do, and this can result in the quarantine flag not being set on downloaded files" - www.thesafemac.com
**Gatekeeper Bypasses**
allowing unsigned code to execute

- **2014**
  - CVE 2014-8826 (patched)
  - Malicious jar file

- **2015**
  - CVE 2015-3715 (patched)
  - Dylib hijacking

- "Runtime shenanigans"

Required Java

Default OS install

CVE-2015-7024 (patched)*
**Gatekeeper Bypass 0x1 (CVE 2015-3715)**

(dylib) hijacking external content

Gatekeeper **only** verified the app bundle!

1. Find an signed app that contains an **external, relative dependency** to a hijackable dylib

2. Create a .dmg/.zip with the necessary folder structure (i.e. placing the malicious dylib in the **externally** referenced location)

3. Host online or inject

   → **Virus Bulletin** white paper
   
   www.virusbtn.com/dylib
Gatekeeper Bypass Ox1 (CVE 2015-3715)

- a signed app that contains an external dependency to hijackable dylib

`spectl` tells you if gatekeeper will accept the app

```
$ spectl -vat execute /Applications/Xcode.app/Contents/Applications/Instruments.app
Instruments.app: accepted
source=Apple System
```

```
$ otool -l Instruments.app/Contents/MacOS/Instruments

Load command 16
  cmd LC_LOAD_WEAK_DYLIB
  name @rpath/CoreSimulator.framework/Versions/A/CoreSimulator

Load command 30
  cmd LC_RPATH
  path @executable_path/../../../../SharedFrameworks

Instruments.app - it's the bill
```
create a .dmg with the necessary layout

**Gatekeeper Bypass 0x1 (CVE 2015-3715)**

1. **required directory structure**

2. 'clean up' the .dmg
   - hide files/folder
   - set top-level alias to app
   - change icon & background
   - make read-only

(deployable) malicious .dmg
**Gatekeeper Bypass Ox1 (CVE 2015-3715)**

Host online or inject into downloads

Allow apps downloaded from:
- Mac App Store
- Mac App Store and identified developers
- Anywhere

Gatekeeper setting's (maximum)

Quarantine popup (anything downloaded)

Quarantine alert

Unsigned (non-Mac App Store) code execution!!

Gatekeeper bypass :)
Gatekeeper Bypass Ox2 (CVE 2015-7024)

Runtime Shenanigans

Gatekeeper only **statically** verifies the app bundle!

1. Find any signed app that **at runtime**, loads or executes a 'relatively external' binary

2. Create a .dmg/.zip with the necessary folder structure (i.e. placing the malicious binary in the externally referenced location)

3. Host online/inject into insecure downloads
GATEKEEPER BYPASS 0x2 (CVE 2015-7024)

example 1: Adobe (Photoshop, etc)

Q: Can I add/modify files in my signed (app) bundle?
A: "This is no longer allowed. If you must modify your bundle, do it before signing. If you modify a signed bundle, you must re-sign it afterwards. Write data into files outside the bundle" -apple.com

---

**Plugin Loading Pseudo Code**

```objc
NSString* pluginDir = APPS_DIR + @"../Plug-ins";
for (NSString* plugins in pluginDir) {
    //load plugin dylib
    // ->not validated, can unsigned
}
```
Gatekeeper Bypass 0x2 (CVE 2015-7024)

element 2: Apple (ictool)

```c
//execute ibtoold
void IBExecDirectly()
{
    //build path to ibtool
    ibToolPath = IBCopyServerExecutablePath()
    
    //exec ibtoold
    execv(ibToolPath, ....)
}

//build path to ibtool
char* IBCopyServerExecutablePath()
{
    //get full path to self (ictool)
    icToolPath = IBCopyExecutablePath()
    
    //remove file component
    icToolDir = IBCreateDirectoryFromPath(exePath)
    
    //add 'ibtool'
    ibToolPath = IBCreatePathByAppendingPathComponent(icToolDir, "ibtoold")
    return ibToolPath
}
```

Gatekeeper, happy with ictool

```bash
$ spctl -vat execute Xcode.app/Contents/Developer/usr/bin/ictool
Xcode.app/Contents/Developer/usr/bin/ictool: accepted
source=Apple System
```

```bash
$ xattr -l *
ibtoold: com.apple.quarantine: 0001;55ee3be6;Google\x20Chrome.app
ictool: com.apple.quarantine: 0001;55ee3be6;Google\x20Chrome.app
```

```bash
$ codesign -dvv ibtoold
ibtoold: code object is not signed at all
```

...but ibtoold is unsigned

Ictool's pseudo code
GATEKEEPER BYPASS 0x2 (CVE 2015-7024)

element 2: Apple (ictool)

Gatekeeper setting's (max.)

1. alias to 'update.app' (ictool)
   ...name & icon attacker controlled

2. apple-signed 'update.app' (ictool)
   .app extension prevents Terminal.app popup

hide

3. unsigned `ibtool`
   command-line executable

4. unsigned application

.dmg setup

only visible item

unsigned code execution

Synack
FIXING GATEKEEPER
'patches' & runtime validation
**Patches CVE 2015-3715/2015-7024**
both bypasses now "patched"

- **CVE 2015-3715**
  - Patched in OS X 10.10.4
  - "I'm all safe now, right? ...right?"

- **CVE 2015-7024**
  - Patched in OS X 10.11.1

---

- **dylib hijack bypass**
- **runtime bypass**

---

[Attribution]
Anyone reporting a vulnerability to Apple, is doing a disservice to the rest of the world. You are feeding a company that doesn't care.
Patching CVE 2015-3715
external dylibs; verified

“Instruments.app” can’t be opened because the identity of the developer cannot be confirmed.

Gatekeeper in action

Debug messages in syslog

External dylibs, now verified

Malicious .dmg/.zip layout

(signed) Apple application
**Patch for CVE 2015-3715**

**What is this 'dylib check'?**

```assembly
    mov    rsi, cs:selRef_performDylibBundleCheck_
    mov    rbx, [rbp+WorkerThreadClass]
    mov    rdi, rbx
    mov    rdx, r14 ;path to app
    call   cs:_objc_msgSend_ptr
    test   al, al
    jz     checkFailed
    checkFailed:
        lea    rdi, "Fails dylib check"
        xor    eax, eax
        call   _NSLog
```

**Error msg in XProtectFramework**

```objective-c
if(![WorkerThreadClass performDylibBundleCheck:app]) {
    NSLog(@"Fails dylib check");
}
```

Translated to C

---

**'Enable' debugging OS X 10.11**

1. boot into recovery mode via cmd+r
2. csrutil disable (from Terminal.app)
3. reboot

---

Debugging with LLDB:

```
(lldb) br s -a 0x00007FFF9A12AA22
Breakpoint 1: where = XprotectFramework`+[WorkerThreadClass threadEntry:] + 4845, address = 0x00007fff9a12aa22

Process 381 stopped
XprotectFramework`+[WorkerThreadClass threadEntry:] + 4845: -> 0x7fff9a12aa22: callq *%r13
```

```
(lldb) po $rdi
WorkerThreadClass

(lldb) x/s $rsi
0x7fff9a12cb84: "performDylibBundleCheck:"

(lldb) po $rdx
file:///Volumes/unsafe/Applications/Instruments.app/
```
Patch for CVE 2015-3715

Overview of `performDylibBundleCheck`:

```swift
@interface XProtectDylibCheck : NSObject
{
    NSString *_absolutePath;
    NSMutableArray *_rPaths;
    NSMutableArray *_loadCommands;
    unsigned long long _numCommands;
    NSURL *_executablePath;
    NSURL *_loaderPath;
    BOOL _isExecutable;
    NSMutableDictionary *_scannedLibraries;
    ....
}
```

+ (BOOL)path:(id)arg1 isInsideBundle:(id)arg2;
+ (BOOL)path:(id)arg1 isSafeWithBundle:(id)arg2;
+ (id)allowedLibraryPaths;
- (BOOL)parseMacho;
- (id)parseExecutableAndLoaderPaths:(id)arg1;
- (BOOL)parseLoadCommands;
- (id)substituteRpath:(id)arg1;
- (BOOL)checkCommandsWithBundleURL:(id)arg1;
....

XProtectDylibCheck class

```shell
$ classdump XprotectFramework
```
**Patch for CVE 2015-3715**

**dylib location verification(s)**

- `[XProtectDylibCheck checkCommandsWithBundleURL:]`

invoking `path:isSafeWithBundle:`

1. allows if dylib falls in an 'allowLibraryPath'

- `invoking path:isSafeWithBundle:`

- `mov rdi, r12`
- `mov rsi, cs:selRef_path_isSafeWithBundle_`
- `mov rdx, r15`
- `mov rcx, rax`
- `call rbx`
- `test al, al`
- `jz unsafeDylib`

2. allows if dylib falls within the (verified) application bundle

```swift
if (YES != [dylib hasPrefix:appBundle])
{
    //NOT SAFE!
}
```

 allows if dylib falls in an 'allowLibraryPath'

- `lldb` po $rax

 `<__NSArrayI 0x7f89ca4ed960>(/usr/, /opt, /System/, /Library/, /Network/, /AppleInternal/, /Developer, /build)`

allowed library paths

- dylib inside app bundle?
Patching **CVE 2015-7024**

*external binaries; verified?*

---

**gatekeeper in action**

---

**debug messages in syslog**

---

*external binaries, now verified*
Patch for CVE 2015-7024

What is this 'Failed GK check'?"
Patch for CVE 2015-7024

performBlockListCheck: blockDict:

`+[WorkerThreadClass performBlockListCheck: blockDict:]`

- `[blacklistedID isEqualToString:appID]`

SecStaticCodeCreateWithPath() & SecCodeCopySigningInformation()

(debugging)

```_objc
(app's code signing info) {
    "digest-algorithm" = 1;
    identifier = "com.apple.ictool";
    "main-executable" =
        "file://Volumes/unsafe/update.app";
    ... 
}
```  

(lldb) po <block dictionary>

CodeSignatureIDs = (
    "com.apple.a2p",
    "com.apple.ibtool",
    "com.apple.pythonw",
    "com.apple.python",
    "com.apple.ictool"
);

(app's id from code signing blob)

```
//check if app's ID matches any black listed ones
for(blacklistedID in blockDict)
{
    if(YES == [blacklistedID isEqualToString:appID])
        //black-listed! GTFO
}
```  

(Pseudo code) takes app & blacklisted IDs
### Patch(s) Summary

<table>
<thead>
<tr>
<th>Patch ID</th>
<th>Description</th>
<th>Protection Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-3715</td>
<td>External dylib hijack</td>
<td>Scan for external dylibs</td>
<td>Effective patch</td>
</tr>
<tr>
<td></td>
<td>(run-time exec's)</td>
<td></td>
<td>Only blocks specific vector</td>
</tr>
<tr>
<td>2015-7024</td>
<td>Blacklist binaries</td>
<td></td>
<td>Ineffective patch</td>
</tr>
</tbody>
</table>

Neither, generically blocks the execution of unsigned internet code.
BYPASSING CVE 2015-7024
...with ease

appID = //get app's id from code signing blob
//check if app's ID matches any black listed ones
for(blackListedID in blockDict)
{
    if(YES == [blackListedID isEqualToString:appID])
        //black-listed! GTFO
}

"patch"

1. apple-signed binaries +
2. that calls `execv` +
3. on a 'relative' binary

"Wardle said he suspects there are other Apple-trusted binaries...that will also allow attackers to bypass Gatekeeper." (summer 2015)
Finding moar binaries to abuse

```python
def scan(rootDir):
    #dbg msg
    print 'scanning %s' % rootDir
    # enum bins
    # -> signed by apple proper
    appleBins = enumBinaries(rootDir)
    #check imports
    # -> looking for execv/etc
    candidateBins = checkImports(appleBins)
    #dbg
    print candidateBins
```

Scanner output

```bash
$ python scan.py /Applications/Xcode.app/Contents/Developer/usr/bin/
scanning /Applications/Xcode.app/Contents/Developer/usr/bin/
['/Applications/Xcode.app/Contents/Developer/usr/bin/actool',
 '/Applications/Xcode.app/Contents/Developer/usr/bin/atos', ...
```

File i/o for `actool`

```bash
$ sudo fs_usage -w -f filesystem | grep -i actool
getattrlist /Applications/Xcode.app/Contents/Developer/usr/bin/actool
stat64 /Applications/Xcode.app/Contents/Developer/usr/bin/ibtoolld
```

'Process monitoring' `actool`
CVE 2015-7024 Bypass

- Replace `ictool` with `actool`.
- Gatekeeper setting's (max.) alias to `'update.app'` (ictool).
- Apple-signed `'update.app'` (ictool).
- .app extension prevents `Terminal.app` popup.
- Unsigned `ibtool` command-line executable.
- Unsigned application.
- .dmg setup.
- Only visible item.
- Unsigned code execution.
- Hide unsigned application.
CVE 2015-7024 Bypass

OS X El Capitan
Version 10.11.2

Security & Privacy

- General
- Finder
- Firewall
- Privacy

A login password has been set for this user. Change Password...

- Require password: immediately
- Show a message when the screen is locked: Set Lock Message...
- Disable automatic login

Allow apps downloaded from:
- Mac App Store
- Mac App Store and identified developers
- Anywhere

Click the lock to make changes.

Authorization Plugins
registered current authorization bundles

Browser Extensions
plugins/extensions hosted in the browser

Cron Jobs
current users cron jobs

Kernel Extensions
installed modules, possibly kernel loaded

Launch Items
deemons and agents loaded by Launch

Library Inserts
devices and agents loaded by Launch

Login Items
items started when the user logs in

Login/Logout Hooks
items executed upon login or logout

Spotlight Importers
services loaded by Spotlight (indexer)

Bypass only 4 launch items
no 'java' processes
fully patched OS X
Gatekeeper enabled
Leveraging OS-Level Mitigations?

"A kernel extension to mitigate Gatekeeper bypasses"

`sysctl` variables

```
$ sysctl vm | grep cs_
vm.cs_force_kill: 0
vm.cs_force_hard: 0
vm.cs_all_vnodes: 0
vm.cs_enforcement: 0
....
```

"require enforcement"

```
$ sudo sysctl -w vm.cs_enforcement=1
vm.cs_enforcement: 0 -> 1
```

"require enforcement"

"Code Signing–Hashed Out" - apple.com

"[sysctl] allows processes with appropriate privilege to set kernel state" - apple.com

"Code Signing–Hashed Out" - J. Levin

lots of OS issues
GATEKEERPER
"A kernel extension to mitigate Gatekeeper bypasses"

int mpo_file_check_mmap_t(...)
{

    // determine if main binary is signed
    is_main_signed = ((csproc_get_teamid_t*)(void*)(cloned_csproc_get_teamid))(p);

    // determine if mapped section is signed
    is_mapped_signed = ((csfg_get_platform_binary_t*)cloned_csfg_get_platform_binary)(fg);

    // block unsigned dylibs in signed app/binary
    if(is_mapped_signed == 0 && is_main_signed == 1)
    {
        // GTFO!
    }
Validate All Binaries At Runtime

block unsigned binaries from the internet

apple-signed binary

<some>.kext

exec hook

does binary have quarantine attributes?

is binary not previously user-approved?

is binary is unsigned?

“block unsigned, non-approved internet binaries”

malicious executable
VALIDATE ALL BINARIES AT RUNTIME

step 0: register exec hook

Apple's "Kernel Authorization (KAuth) Subsystem"

"Monitoring Process Creation via the Kernel (Part II)"
objective-see.com/blog/blog_0x0A.html

kauth subsystem

scope*

listener 1

auth decision

*KAUTH_SCOPE_FILEOP, KAUTH_SCOPE_VNODE, etc
VALIDATE ALL BINARIES AT RUNTIME

step 0: register exec hook

```
//kauth listener
kauth_listener_t kauthListener = NULL;

//register listener ('KAUTH_SCOPE_FILEOP')
kauthListener = kauth_listen_scope(KAUTH_SCOPE_FILEOP, &processExec, NULL);
```

register KAUTH_SCOPE_FILEOP listener

```
//kauth callback
static int processExec(kauth_cred_t credential, void* idata, kauth_action_t action, uintptr_t arg0, uintptr_t arg1, uintptr_t arg2, uintptr_t arg3)
{
    //return var, default to defer
    int kauthResult = KAUTH_RESULT_DEFER;

    //ignore all non exec events
    if (KAUTH_FILEOP_EXEC != action)
    {
        //bail
        goto bail;
    }

    //get path
    vn_getpath((vnode_t)arg0, path, &pathLength);

    //dbg msg
    DEBUG_PRINT("OSTIARIUS: new process: %s %d\n", path, proc_selfpid());
}
```
**Validating All Binaries at Runtime**

**Step 1:** Ignore 'non-internet' binaries (NULL quarantine attributes)

- **hook_vnode_check_exec**
  - `call _quarantine_get_flags`

- **_quarantine_get_flags**
  - `call quarantine_getinfo`

- **quarantine_getinfo**

```assembly
lea rsi, "com.apple.quarantine"
lea r8, [rbp+qAttrsSize]
mov rdi, r14
mov rdx, [rbp+qAttrs]
mov rcx, r15
call _mac_vnop_getxattr
```

- //get quarantine attributes
  - // ->if this 'fails', simply means binary doesn't have quarantine attributes (i.e. not from the internet)
  - `if(0 != mac_vnop_getxattr((vnode_t)arg0, QFLAGS_STRING_ID, qAttr, QATTR_SIZE-1, &qAttrLength))`

    ```assembly
    //dbg msg
    DEBUG_PRINT(("binary has NO quarantine attributes (not from the internet), so allowing\n"));
    
    //bail
    // ->process is allowed
    goto bail;
    ```
VALIDATE ALL BINARIES AT RUNTIME

step 1: ignore 'non-internet' binaries (NULL quarantine attributes)

mounting

disk image: quarantine attributes

```
$ xattr -l ~/Downloads/malware.dmg
com.apple.quarantine: 0001...
```

files: no quarantine attributes

```
$ xattr -l /Volumes/Malware/Installer.app
```

how to tell such files are from the internet?

while disk images have quarantine attributes, their mounted (executable) files don't...
**Validate All Binaries At Runtime**

**Step 1:** Ignore 'non-internet' binaries (NULL quarantine attributes)

1. Is binary path within `/Volumes`?
2. Get mount struct for binary's `vnode`
3. Get `vfsstatfs` struct for mount and extract `f_mntfromname` value (e.g. `'/dev/disk1s2'`)
4. Iterate over the `IORegistry` to find a parent (`'IOHDIXHDDDriveOutKernel'`) that has a child (`'IOMedia'`) with matching mount point (e.g. `'/dev/disk1s2'`)

Parent will have the original dmg path, in `image_path`.

5. With a dmg path, can access quarantine attributes if dmg is from the internet & binary is unsigned: **BLOCK**

---

*Image credits: Chris Riepe*
**Validate All Binaries At Runtime**

step 2: ignore 'user-approved' binaries

```plaintext
- [GKQuarantineResolver
   approveUpgradingQuarantineTarget:recursively:volume:]
   call quarantined_file_get_flags
   or    eax, 40h
   mov   rdi, [rbp+var_B8]
   mov   esi, eax
   call __qtn_file_set_flags
```

Updating quarantine attributes

```plaintext
$ xattr -l ~/Downloads/KnockKnock.app
com.apple.quarantine: 0001;55f3313d;...
$ xattr -l ~/Downloads/KnockKnock.app
com.apple.quarantine: 0041;55f3313d;...
```

before & after

```plaintext
//CoreServicesUIAgent sets flags to 0x40 when user allows
// -> so just allow such binaries
if(0 != (qFlags & 0x40))
{
   //dbg msg
   DEBUG_PRINT("previously approved, so allowing\n");
   //bail
   goto bail;
}
```

allowing previous approved binaries
VALIDATE ALL BINARIES AT RUNTIME
step 3: ignore signed binaries

```c
int csfg_get_platform_binary(struct fileglob *fg) {
    int platform_binary = 0;
    struct ubc_info *uip;
    vnode_t vp;

    if (FILEGLOB_DTYPE(fg) != DTYPE_VNODE)
        return 0;

    vp = (struct vnode *)fg->fg_data;
    if (vp == NULL)
        return 0;

    vnode_lock(vp);
    if (!UBCINFOEXISTS(vp))
        goto out;

    uip = vp->v_ubcinfo;
    if (uip == NULL)
        goto out;
    if (uip->cs_blobs == NULL)
        goto out;
}
```

@osxreverser; how apple does it

1. lock vnode
2. grab v_ubcinfo (ubc_info structure)
3. check if cs_blobs are NULL (i.e. binary is unsigned)

vnode_lock() is non-exported function
vnode, ubc_info, etc all private/undocumented structures
Validate All Binaries At Runtime

Step 3: Ignore signed binaries

```c
// lock vnode
lck_mtx_lock((lck_mtx_t*)arg0);

// init offset pointer
offsetPointer = (unsigned char*)(vnode_t)arg0;

// get pointer to struct ubc_info in vnode struct
// -> disasm from kernel: mov rax, [vnode+70h]
offsetPointer += 0x70;

// deref pointer to get addr of struct ubc_info
offsetPointer = (unsigned char*)(unsigned long*)(offsetPointer);

// get pointer to cs_blob struct from struct ubc_info
// -> disasm from kernel: mov rax, [ubc_info+50h]
offsetPointer += 0x50;

// null csBlogs means process is NOT SIGNED
// -> so block it
if (0 == *(unsigned long*)(offsetPointer)) {
    // kill the process
    proc_signal(pid, SIGKILL);
}

// unlock vnode
lck_mtx_unlock((lck_mtx_t*)arg0);
```

lck_mtx_lock, exported:)

 STATIC OFFSETS FOR OS X 10.11.*

block unsigned internet binary
OSTIARIUS
blocking unsigned binaries from the internet

installer

kext component

signed
open-source
protects

(debug) output

objective-see.com/products/ostiarius.html
CONCLUSIONS

wrapping it up
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

the unfortunate reality

protects naive OS X users from lame attackers & false sense of security?

highly recommend; 3rd-party security tools

patch fails
MY CONUNDRUM

...I love my mac, but it's so easy to hack

I should write some OS X security tools to protect my Mac

....and share 'em freely :)

"No one is going to provide you a quality service for nothing. If you’re not paying, you’re the product." - unnamed AV company

i humbly disagree
**OBJECTIVE-SEE(.COM)**

free security tools & malware samples

"providing visibility to the core"

- KnockKnock
- BlockBlock
- KextViewr
- Ostiarius
- Hijack Scanner

os x malware samples
"Is it crazy how saying sentences backwards creates backwards sentences saying how crazy it is?"  -Have_One, reddit.com
credits

- flaticon.com
- thezoom.com
- iconmonstr.com
- http://wirdou.com/2012/02/04/is-that-bad-doctor/

images

resources

- thesafemac.com
- "Mac OS X & iOS Internals", Jonathan Levin
- https://securosis.com/blog/os-x-10.8-gatekeeper-in-depth