“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
all aspects of gatekeeper

Gatekeeper

understanding

bypassing

fixing
UNDERSTANDING GATEKEEPER
...under the hood
os x trojans everywhere? everywhere!

countless OS X users infected

Life Before Gatekeeper

leap-a 2006
rsplug 2007
macsweeper 2008
iworks-a 2009
pinhead 2010
macdefender 2011
Gatekeeper 2012

jahlav-a 2006
rkosx-a 2008
hovdy-a 2008

opinionspy
boonana

Revir
PDF

Devilrobber

Qhost

PDF

Qhost

OpinionSpy

YouTube

Gatekeeper

Synack
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

fake codecs

rogue "AV" products

fake installers/updates

poor naive users!

infected torrents
Gatekeeper Protects Users
...from high-tech adversaries

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

MitM + infect insecure downloads

my dock
HOW GATEKEEPER WORKS
an overview

quarantine attribute added

quarantine attributes

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

gatekeeper settings

iff quarantine attribute is set!

gatekeeper in action

“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.
## Extended File Attributes

Simply put; file metadata

### Extended attr. (com.apple.*)

<table>
<thead>
<tr>
<th>Brief Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>FinderInfo</td>
</tr>
<tr>
<td>Metadata</td>
</tr>
<tr>
<td>Quarantine</td>
</tr>
</tbody>
</table>

### Dump w/ xattr command

```bash
$ 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 37 70 3A 2F 2F 77 77 77 65 69 63 61 72 2E 6F
00000010 72 67 2F 64 6F 77 6E 61 64 2F 65 69 63 61 72
00000020 6E 6C 69 6E 74 5F 10 27 68 74 74 70 5F 10 27 68 74 74 70 5F 10 27 68 74 74 70
```

**Dumping quarantine attributes**

---

"Mac OS X & iOS Internals"
Jonathan Levin
'File Quarantine'
realized by the `com.apple.quarantine` file attribute

added in Leopard  "file from internet"

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

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

code to get attributes

$ 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;

file quarantine in action

dumping a file's `com.apple.quarantine` attribute note; not gatekeeper
**SETTING THE QUARANTINE ATTRIBUTE**

who done it!?

---

**custom downloader**

http://www.eicar.org/download/eicar.com.txt

---

any extended attributes?

none; huh?

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

---

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

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

    //download & save to file
    [[NSData dataWithContentsOfURL:remoteFile] writeToFile:localFile atomically:NO];
    return;
}
```
Setting The Quarantine Attribute

Apps can manually add it

```objective-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;55efdddeb;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)

$ 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

Finder.app

LaunchServices framework

Launchd

CoreServicesUIAgent

XProtect framework

Quarantine.kext

"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

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

call stack

XPC request

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

`_spawn_via_launchd()`

```
(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

Launchd

Quarantine.kext

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
**USER INTERACTION VIA CoreServicesUIAgent**

First, the XPC request

```c
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;
}
```

Getting XPC connection to CoreServicesUIAgent

XPC message contents

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

then, analysis via xprotect

---

```plaintext
-CoreUIController handleIncomingXPCMessage:clientConnection:]
-[GKQuarantineResolver resolve]
-[GKQuarantineResolver malwareChecksBegin]
-[GKQuarantineResolver malwareCheckNextItem]
    mov rdi, cs:classRef_XProtectAnalysis
    mov rsi, cs:selRef_alloc
    call r15 ; _objc_msgSend
    mov rdi, rax
    mov rsi, cs:selRef_initWithURL_
    mov rdx, r14 ; path to app
    call r15 ; _objc_msgSend

-XProtectAnalysis
    beginAnalysisWithDelegate:didEndSelector:contextInfo:
]+[WorkerThreadClass threadEntry:
    mov rdi, [rbp+staticCodeRef]
    lea rdx, [rbp+signingInfo]
    xor esi, esi ; flags
    call _SecCodeCopySigningInformation
```

**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>
**USER INTERACTION VIA CORESERVICESUIAGENT**

finally, display the alert

```
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.
```

CoreServicesUIAgent generates alerts for quarantine violations.

```
alert customization

$ less QuarantineHeadlines.strings
<key>Q_HEADLINE_CASPIAN_BAD_DISTRIBUTOR</key>
<string>%@ can't be opened because it is from an unidentified developer.</string>
<key>Q_HEADLINE_CASPIAN_BLOCKED</key>
<string>%@ can't be opened because it was not downloaded from the Mac App Store.</string>
```

Alert customization via QuarantineHeadlines.strings provides customizable alert messages.

```
application termination

mov rsi, cs:selRef_deny
mov rdi, r14
call cs:_objc_msgSend_ptr

-GKQuarantineResolver showGKAlertForPath:
-GKQuarantineResolver alertForPath:malwareInfo:

mov rax, __OBJC_IVAR_$_GKQuarantineResolver__appASN
mov rsi, [rbp+GKQuarantineResolver]
cmp byte ptr [rcx+rax], 0
lea rdi, cfstr_Q_headline_cas ; "Q_HEADLINE_CASPIAN_BAD_DISTRIBUTOR"
mov rdi, cs:classRef_NSAlert
mov rsi, cs:selRef_alloc
call r12 ; _objc_msgSend
```

Application termination is handled through specific procedures and calls.
**What if the App Conforms & Is Allowed by the User?**

Quarantine attributes updated, then application resumed.

![Quarantine alert]

```assembly
; approve UpdatingQuarantineTarget: recursively: volume:
```

```assembly
mov rdi, [rbp+var_B8]
mov esi, eax
```

```assembly
mov rsi, [r13+r14+0]
mov rdx, [rax]
```

```assembly
mov edi, 0FFFFFFFFFh
xor r8d, r8d
mov rcx, rbx
```

```assembly
call __LSSetApplicationInformationItem
```

```assembly
; on error
lea rsi, "Unable to continue stopped application"
```

```assembly
mov edi, 4
xor eax, eax
mov edx, ecx
```

```assembly
call logError
```

![Updating quarantine attributes]

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

![Before & after]

```bash
$ xattr -l ~/Downloads/KnockKnock.app/Contents/MacOS/KnockKnock
com.apple.quarantine: 0041;55f3313d;Google\x20Chrome.app;FBF45932...
```
BYPASSING GATEKEEPER
unsigned code allowed!?
RECALL; GATEKEEPER AIMS TO PROTECT

...unauthorized code should be blocked!

XcodeGhost

gatekeeper in action

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

attribute added?

uTorrent

iWorm infected applications

Xattr:

Adobe Photoshop CC 2014.dmg: No such xattr: com.apple.quarantine

"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

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.

Gatekeeper only verified the app bundle! (signed) application
<external>.dylib
.dmg/.zip layout

Wasn't verified!
Verified, so can't modify

White paper: www.virusbtn.com/dylib
GATEKEEPER BYPASS 0x1 (CVE 2015-3715)

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

$ spctl -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 - fit's the bill
GATEKEEPER BYPASS 0x1 (CVE 2015-3715)

1. Create a .dmg with the necessary layout

- 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 0x1 (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 0x2 (CVE-2015-7024)

Runtime shenanigans

Gatekeeper only statically verifies the app bundle!

1. Find any signed app that at runtime, 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

---

Adobe Photoshop

App bundle validates!

Plugin loading pseudo code

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

example 2: Apple (ictool)

```c
//execute ibtool
void IBExecDirectly()
{
    //build path to ibtool
    ibToolPath = IBCopyServerExecutablePath()
    ibToolPath = IBCreatePathByAppendingPathComponent(ibToolPath, "ibtool")

    //exec ibtool
    execv(ibToolPath, ....)
}

//build path to ibtool
char* IBCopyServerExecutablePath()
{
    //get full path to self (ictool)
    icToolPath = IBCopyExecutablePath()
    icToolPath = IBCreateDirectoryFromPath(exePath)

    //remove file component
    icToolDir = IBCreateDirectoryFromPath(exePath)

    //add 'ibtool'
    ibToolPath = IBCreatePathByAppendingPathComponent(icToolDir, "ibtool")

    return ibToolPath
}
```

`ictool's pseudo code`

---

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

gatekeeper, happy with ictool
```

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

$ codesign -dvv ibtool
ibtool: code object is not signed at all
...but ibtool is unsigned
```
Gatekeeper Bypass 0x2 (CVE 2015-7024)

Example 2: Apple (ictool)

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

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

3. Unsigned ibtool
   Command-line executable

4. Unsigned application

Gatekeeper setting's (max.)

.dmg setup

Only visible item

Hide

Unsigned code execution
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

- **Security**
  - 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: A malicious application may be able to bypass code signing checks
  - Description: An issue existed where code signing did not verify libraries loaded outside the application bundle. This issue was addressed with improved bundle verification.

**CVE-ID**
CVE-2015-3715 : Patrick Wardle of Synack

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

- **Security**
  - Available for: OS X Mavericks v10.9.5, OS X Yosemite v10.10.5, and OS X El Capitan 10.11
  - Impact: An Apple-signed binary could be used to load arbitrary files
  - Description: Certain Apple-signed executables loaded applications from relative locations. This was addressed through additional checks in Gatekeeper.

**CVE-ID**
CVE-2015-7024 : Patrick Wardle of Synack
Patching CVE 2015-3715
external dylibs; verified

gatekeeper in action

deploy messages in syslog
external dylibs, now verified

malicious .dmg/.zip layout

<external>.dylib
(signed) 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
```

---

**Translated to C**

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

---

**Debugging with LLDB**

```bash
(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
tilde://Volumes/unsafe/Applications/Instruments.app/
```

---

**Instructions**

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

**'Enable' debugging OS X 10.11**
PATCH FOR CVE 2015-3715

overview of `performDylibBundleCheck`:

```objective-c
@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

```
+[WorkerThreadClass performDylibBundleCheck:]
+[XProtectDylibCheck alloc]
-[XProtectDylibCheck parseMacho]
-[XProtectDylibCheck checkCommandsWithBundleURL:]
```

$ classdump XprotectFramework

@interface XProtectDylibCheck : NSObject
{...}
**Patch for CVE 2015-3715**

dylib location verification(s)

- [XProtectDylibCheck checkCommandsWithBundleURL:]
  mov rdi, r12
  mov rsi, cs:selRef_path_isSafeWithBundle_
  mov rdx, r15
  mov rcx, rax
  call rbx
  test al, al
  jz unsafeDylib

invoking `path:isSafeWithBundle`:

1. allows if dylib falls in an 'allowLibraryPath'

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

if(YES != [dylib hasPrefix:appBundle])
{
    // NOT SAFE!
}
PATCHING CVE 2015-7024
external binaries; verified?

gatekeeper in action

double messages in syslog

external binaries, now verified

<external> binary

(signed) apple application

.dmg/.zip layout
**Patch for CVE 2015-7024**

What is this 'Failed GK check'?

```objective-c
if(![WorkerThreadClass performBlockListCheck:app blockDict:blockedSigs]) {
    NSLog(@"Failed GK check");
}
```

Translated to C:

```c
if(!WorkerThreadClass->performBlockListCheck:app blockDict:blockedSigs)) {
    NSLog("Failed GK check");
}
```

Error msg in `XProtectFramework`

```c
if(!WorkerThreadClass->performBlockListCheck:blockDict:blockedSigs)) {
    NSLog(@"Failed GK check");
}
```

Debugging with LLDB:

```
Process 381 stopped
XprotectFramework`+[WorkerThreadClass threadEntry:] + 4809: -> 0x00007fff9a12a9fe: callq *%r13
```

```
0x7fff9a12cb63: "performBlockListCheck:blockDict:"
```

```
(Nlldb) po $rdi
WorkerThreadClass
```

```
(Nlldb) po $rcx
file:///Volumes/unsafe/update.app
```

```
CodeSignatureIDs = (
    "com.apple.a2p",
    "com.apple.ibtool",
    "com.apple.pythonw",
    "com.apple.python",
    "com.apple.ictool"
);
```
Patch for **CVE 2015-7024**

`performBlockListCheck: blockDict:`

- Takes app & black-listed IDs

```objective-c
+[WorkerThreadClass performBlockListCheck: blockDict:]
```

- `SecStaticCodeCreateWithPath()` & `SecCodeCopySigningInformation()`

```objective-c
appid = //get app's id from code signing blob
```

- Check if app's ID matches any black listed ones

```objective-c
for(blackListedID in blockDict)
```

- If YES == `[blackListedID isEqualToString:appID]`

```objective-c
//black-listed! GTFO
```

In pseudo code, takes app & black-listed IDs.
### Patch(s) Summary

**Are OS X users now protected?** Hint: **NO**

<table>
<thead>
<tr>
<th>Patch ID</th>
<th>Description</th>
<th>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><strong>✅</strong> effective patch</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>❌</strong> only blocks specific vector</td>
</tr>
<tr>
<td>2015-7024</td>
<td>(run-time exec's)</td>
<td>blacklist binaries</td>
<td><strong>❌</strong> ineffective patch</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></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)
BYPASSING CVE 2015–7024
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:
```
$ 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`
```
$ 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/ibtoold
```

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

Replace ictool with actool

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

3. Unsigned ibtool
   Command-line executable

4. Unsigned application
CVE 2015-7024 BYPASS

OS X El Capitan
Version 10.11.2

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

A login password has been set for this user. Change Password...
- Require password immediately
- Show message when the screen is locked
- Disable automatic login

Launch Items
demons and agents loaded by launchd

Login Items
- Login/logout hooks
- Spotlight importers

no 'java' processes
fully patched OS X
gatekeeper enabled
LEVERAGING OS-LEVEL MITIGATIONS?

"A kernel extension to mitigate Gatekeeper bypasses"

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

code-signing `sysctl` variables

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

enabling code-signing enforcement

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

"require enforcement"

lots of OS issues

"Code Signing-Hashed Out"

J. Levin
A kernel extension to mitigate Gatekeeper bypasses

Pedro Vilaça
@osxreverser

gatekeeper kext
(github.com/gdbinit/Gatekeeper)

blocks unsigned dylibs, but not unsigned (stand-alone) binaries from the internet
**Validate All Binaries At Runtime**

Block unsigned binaries from the internet

- Apple-signed binary
- `<some>.kext`
- Exec hook

- Does binary have quarantine attributes? (1)
- Is binary not previously user-approved? (2)
- Is binary unsigned? (3)

Does binary have quarantine attributes? ✖

Is binary not previously user-approved? ✖

Is binary is unsigned? ✖

Apple-signed binary

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

user-mode

kauth subsystem

kernel-mode

auth decision

*KAUTH_SCOPE_FILEOP, KAUTH_SCOPE_VNODE, etc
VALIDATE ALL BINARIES AT RUNTIME

step 0: register exec hook

```c
//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

```c
//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());
}
```
Validate All Binaries At Runtime

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

```c
hook_vnode_check_exec
    call _quarantine_get_flags

_quarantine_get_flags
    call quarantine_getinfo

quarantine_getinfo
    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))
{
    //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)

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

disk image: quarantine attributes

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

files: no quarantine attributes

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 is 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 ('IOHDIXHDDriveOutKernel') that has a child ('IOMedia') with matching mount point (e.g. '/dev/disk1s2')
5. Parent will have the original dmg path, in 'image_path'

With a dmg path, can access quarantine attributes if dmg is from the internet & binary is unsigned: BLOCK
**Validate All Binaries At Runtime**

Step 2: Ignore 'user-approved' binaries

```plaintext
- [GKQuarantineResolver
approveUpdatingQuarantineTarget:recursively:volume:]
call _quarantine_get_flags
or eax, 40h
mov rdi, [rbp+var_B8]
mov esi, eax
call _qtn_file_set_flags

Quarantine.kext

call _quarantine_get_flags
test eax, eax
jnz leaveFunction

mov edx, [rbp+qFlag]
mov eax, edx
and eax, 40h
jnz leaveFunction

//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;
}
```

Updating quarantine attributes

```plaintext
$ xattr -l ~/Downloads/KnockKnock.app
com.apple.quarantine: 000155f3313d;...
$ xattr -l ~/Downloads/KnockKnock.app
com.apple.quarantine: 004155f3313d;...
```

Allowing previous approved binaries

```plaintext
before & after
```
VALIDATE ALL BINARIES AT RUNTIME

step 3: ignore signed binaries

@osxreverser: how apple does it

```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;

    return platform_binary;
}
```

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);
```

```asm
_vnode_lock proc
    push    rbp
    mov     rbp, rsp
    pop     rbp
    jmp     _lck_mtx_lock

_block unsigned internet binary
```
OSTIARIUS
blocking unsigned binaries from the internet

installer

signed
open-source
protects

kext component

(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?

patch fails

highly recommend; 3rd-party security tools
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

os x malware samples

"providing visibility to the core"

TaskExplorer
KnockKnock
BlockBlock
KextViewer
Ostiarius
Hijack Scanner
QUESTIONS & ANSWERS
feel free to contact me any time!

patrick@synack.com
@patrickwardle

"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