Made In America: Green Lambert for OS X

by: Runa Sandvik / September 30, 2021

This guest blog post, was written by **Runa Sandvik**, a noted security researcher who works on digital security for journalists and other high-risk people.

Mahalo for sharing Runa!

Background

In March 2017, WikiLeaks began publishing thousands of files detailing the CIA's spying operations and hacking tools. The leak, known as **Vault 7**, was the largest disclosure of classified information in the agency's history. In April, Symantec **publicly linked** Vault 7 to an advanced threat actor named Longhorn. Kaspersky then announced it tracks the same actor as **The Lamberts**, and revealed the existence of an OS X implant called *Green Lambert*.

Kaspersky's research showed that **The Lamberts' toolkit** includes "network-driven backdoors, several generations of modular backdoors, harvesting tools, and wipers." A **timeline of actvitiy** for tools used by The Lamberts shows that "Green Lambert is the oldest and longest-running in the family." Green Lambert is described as an "active implant" and "the only one where non-Windows variants have been found."

This blog post, along with the [Made in America](**https://objectivebythesea.com/v4/talks.html#Made** In America) talk at **Objective By The Sea v.4.0**, provides a comprehensive analysis of Green Lambert for OS X. I'll share how I approached the research, the tools I used, the things I figured out, and the things I didn't. I'll also look at whether the developers followed the agency's guidelines for **development tradecraft**. Some might ask why I'd look at an implant this old? Doing so helps us better understand the capabilities of its sophisticated creator, past and present. And, if we're being honest: I could, so I did.

Victimology

We don't know how this implant makes it into a target system; the type of system it's used on; or the geographical location of a typical target. **Symantec** said that the actor has infiltrated governments, "in addition to targets in the financial, telecoms, energy, aerospace, information technology, education, and natural resources sectors." **QI-ANXIN** said the actor has previously "targeted personnel and institutions in China."

A version of Green Lambert for OS X was first uploaded to **VirusTotal**, from Russia, in September 2014. Kaspersky **marked it** as malicious in October 2016. AegisLab, a security firm based in Taiwan, **followed** a couple of weeks later. VirusTotal identified that the implant calls itself *GrowlHelper*, possibly referencing the popular **Growl** notification system for OS X from 2004.

Triage

Using static analysis methods, we can triage the implant without running it. For example, we can determine that GrowlHelper is a small, unsigned Mach-O executable.

```
$ file GrowlHelper
GrowlHelper: Mach-O executable i386

$ codesign -dvv GrowlHelper
GrowlHelper: code object is not signed at all

$ du -h GrowlHelper
208K
```

We can use otool -L to print a list of linked libraries. This can sometimes provide insight into the capabilities of the malware, but doesn't appear to be particularly helpful here. Note the few dependencies in the list below.

```
$ otool -L GrowlHelper
/System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation
/System/Library/Frameworks/CoreServices.framework/Versions/A/CoreServices
/System/Library/Frameworks/Security.framework/Versions/A/Security
/System/Library/Frameworks/SystemConfiguration.framework/Versions/A/SystemConfiguration
/usr/lib/libSystem.B.dylib
/usr/lib/libgcc_s.1.dylib
```

What's more interesting is the output of strings -. This tool can also provide insight into the capabilities of the malware.

```
$ strings - GrowlHelper
LoginItem
LaunchAgent
/Library/LaunchDaemons

www.google.com
Error from libevent when adding event for DNS server
1.3a

_SecKeychainFindInternetPassword
_SecKeychainItemCopyAttributesAndData
_kSCPropNetProxiesHTTPProxy
_kSCPropNetProxiesProxyAutoConfigEnable
_kSCPropNetProxiesProxyAutoConfigURLString
```

The references to LoginItem, LaunchAgent and LaunchDaemons suggest this implant has different options for gaining persistence on a system. In other words: how the implant ensures it's executed again if the system is rebooted. Check out **this post** by **Phil Stokes** at SentinelOne for an overview of malware persistence techniques seen in the wild.

The following three lines appear to be related to **libevent**, the same event notification library that is used by **Tor**. The open-source library is very popular now, but was perhaps less known back when this implant was created. The reference to 1.3a may shed some light on the development timeline for this implant: version 1.3a of libevent **was released** in February 2007.

The references to Keychain, Proxies and AutoConfig suggest this implant determines proxy settings on the target system. According to **this post**, kSCPropNetProxiesProxyAutoConfigEnable and kSCPropNetProxiesProxyAutoConfigURLString were added in Xcode version 2.2. This version **was released** in November 2005. Could be another clue about the development timeline.

OS X Version

The static analysis methods we used were helpful, but we're going to want to see how the implant behaves on a system. For that, we'll turn to dynamic analysis in a virtual machine. But which version of OS X does the implant need? We know that it's a 32-bit executable, and the latest macOS is 64-bit only. We can narrow this down further by looking at symbols using nm.

```
$ nm GrowlHelper
         U _CFArrayAppendValue
         U _CFArrayCreateMutable
         U _CFArrayCreateMutableCopy
         U _CFArrayGetCount
         U _CFArrayGetValueAtIndex
         U _CFArrayRemoveValueAtIndex
         U _CFDictionaryCreate
         U _CFDictionaryGetValue
         U _CFGetTypeID
         U _CFNumberGetTypeID
         ...
```

The next step is a bit tedious, but does provide helpful information. To better understand what these symbols represent, we can look them up in Apple's **Developer Documentation**. Not only will we be able to learn how and where a given symbol is used, but we can also see when it was deprecated. With that information, we can determine which version of OS X the implant will run on.

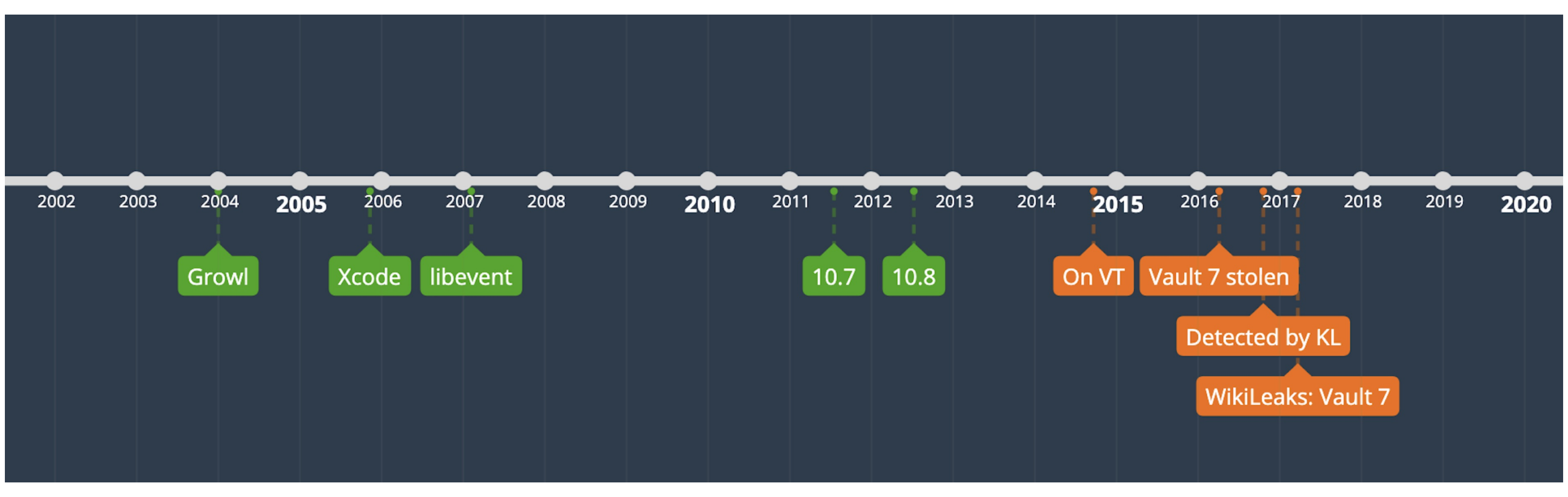
- FSGetCatalogInfo is **available** in macOS 10.0 - 10.8
- FSPathMakeRef is **available** in macOS 10.0 - 10.8
- FSSetCatalogInfo is **available** in macOS 10.0 - 10.8
- SecKeychainSearchCopyNext is **available** in macOS 10.0 - 10.7
- SecKeychainSearchCreateFromAttributes is **available** in macOS 10.0 - 10.7
- SecKeychainSetUserInteractionAllowed is **available** in macOS 10.2 - 12.0

This means that the implant will run on (at least) 10.7: OS X Lion.

Note: I confirmed the implant runs on 10.8. It probably runs on any OS X that supports 32-bit executables.

Development / Use Timeline

Let's look at a potential timeline for the development and use of this implant.



Growl was released in 2004 and retired in 2020. Xcode version 2.2 was released in November 2005, while libevent 1.3a was released in February 2007. OS X 10.7 was released in 2011, and 10.8 in 2012. The implant first appeared on VirusTotal in late 2014. Court records **show** Vault 7 was stolen sometime in early 2016 and published by WikiLeaks a year later. Based on these datapoints, it's likely the implant was created and used between 2007 and (at least) 2013.

Setting Up a Virtual Machine

As of June 2021, OS X 10.7 is **available** for free from Apple. You can also do what I did: buy an old MacBook on eBay for $95.

You may have to unpack the .dmg you get from Apple to get a file that'll work with your virtual machine software. If so, try:

```
$ hdiutil attach InstallMacOSX.dmg
```

Click on *Install Mac OS X* on the Desktop and use **The Unarchiver** (or another tool) to extract InstallMacOSX.pkg to a temporary folder. Go into this folder, click on the new copy of InstallMacOSX.pkg and select *Show Package Contents*. Copy InstallESD.dmg to where you keep your virtual machine images, and use that instead.

We're going to use **lldb**, the default debugger, to execute the implant, modify registers, and examine memory contents. OS X 10.7 doesn't include Xcode by default, but a quick Google search suggests we need version 4.6.3 and can get it from Apple's **Developer Downloads** page. After installing Xcode and confirming that lldb is working, we isolate the machine and create a clean snapshot.

Persistence

Phil Stokes at SentinelOne **wrote** that "the most common way malware persists on macOS is via a LaunchAgent. Each user on a Mac can have a LaunchAgents folder in their own Library folder to specify code that should be run every time that user logs in." We can confirm this is the case with Green Lambert by running the implant, then checking the user's LaunchAgents folder.

```
$ ls ~/Library/LaunchAgents
com.apple.GrowlHelper.plist
```

Once installed, it'll delete the original GrowlHelper file from the system. This is where our VM snapshot comes in handy.

From Phil's post, we know that "LaunchAgents take the form of property list files, which can either specify a file to execute or can contain their own commands to execute directly." We can confirm this by looking at com.apple.GrowlHelper.plist.

```
$ cat ~/Library/LaunchAgents/com.apple.GrowlHelper.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-
1.0.dtd">
<plist version="1.0">
<dict>
 <key>Label</key>
```

```
  <string>com.apple.GrowlHelper</string>
  <key>ProgramArguments</key>
  <array>
   <string>/Users/user/Library/Caches/com.apple.Growl.GrowlHelper/5d0d/GrowlHelper</string>
   <string>-f</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>OnDemand</key>
  <false/>
 </dict>
 </plist>
```

The ProgramArguments tell us where GrowlHelper is installed and that it takes at least one command line argument (-f). The RunAtLoad key confirms the implant will run every time the user logs in. To get an overview of the installation process, we can monitor file system activity for GrowlHelper events.

```
$ sudo fs_usage -w -f filesys > filesys.out
$ sudo grep GrowlHelper filesys.out
execve      /Users/user/GrowlHelper     0.015273 W bash.2848
execve      /Users/user/GrowlHelper     0.000383   GrowlHelper.2851

open        /Users/user/.profile        0.000018   GrowlHelper.2851
open        /Users/user/.bash_profile   0.000015   GrowlHelper.2851
open        /Users/user/.bash_login     0.000015   GrowlHelper.2851
open        /Users/user/.bashrc         0.000014   GrowlHelper.2851
open        /Users/user/.cshrc          0.000014   GrowlHelper.2851
open        /Users/user/.login          0.000014   GrowlHelper.2851
open        /Users/user/.tcshrc         0.000014   GrowlHelper.2851
open        /Users/user/.xsession       0.000007   GrowlHelper.2851
open        /Users/user/.xinitrc        0.000006   GrowlHelper.2851
```

We see that GrowlHelper has a handful of options for maintaining persistence, in case the LaunchAgent is removed. In one case, the implant uses a .profile file to ensure it's launched whenever the user opens the Terminal. (Path to GrowlHelper was lightly edited due to space constraints.)

```
$ cat ~/.profile
GrowlHelper=`/path/to/com.apple.Growl.GrowlHelper/5d0d/GrowlHelper 2>&1` # Automatic GrowlHelper. Do not remove
```

Self-Update

We can compare how GrowlHelper behaves when the system is offline v. online. Here are the files it created in an isolated VM.

```
$ file /Users/offline/Library/Caches/com.apple.Growl.GrowlHelper/5d0d/*
GrowlHelper: Mach-O executable i386
db:      Berkeley DB 1.85 (Hash, version 2, native byte-order)
fifo:     socket
queue:    directory
```

And here are the files GrowlHelper created on that old MacBook I got from eBay.

```
$ file /Users/online/Library/Caches/com.apple.Growl.GrowlHelper/5d0d/*
GrowlHelper:      Mach-O executable i386
Software Update Check:  Mach-O executable i386
db:            Berkeley DB 1.85 (Hash, version 2, native byte-order)
fifo:          socket
queue:         directory
```

It looks like GrowlHelper creates an executable named Software Update Check when it thinks it's online. I was pretty excited when I first found this, but quickly realized it just drops a copy of itself with a different name.

```
3fcdbd3c5fa34fb8e8d58038fa1d1f13d37e8a4b GrowlHelper
3fcdbd3c5fa34fb8e8d58038fa1d1f13d37e8a4b Software Update Check
```

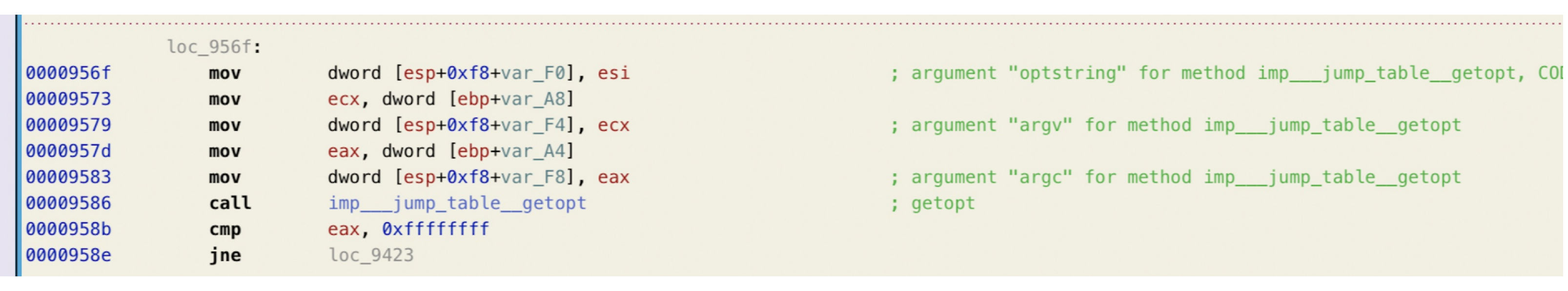
It's possible that Software Update Check is used to update GrowlHelper.

Command Line Arguments

We know where GrowlHelper is installed and that it takes at least one command line argument (-f). With this information, we can identify other arguments by manually looping through options a - z and A - Z on the command line. The output below is the result of doing this try/fail experiment in a VM.

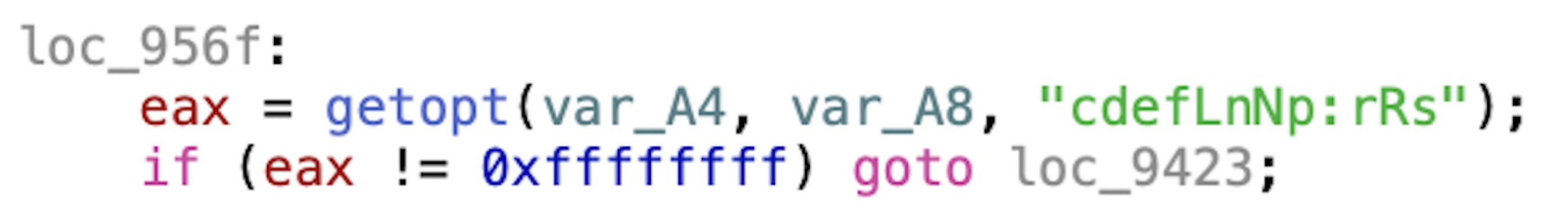
Args	Meaning	Action
c	??	Prints: ** Commands will be processed immediately **
d	??	If GrowlHelper is installed, drops Software Update Check
f	Default	Persists as LaunchAgent, creates: GrowlHelper, db, fifo, queue
p:	??	Prints: GrowlHelper: option requires an argument – p
s	??	Runs without persisting, creates: db, fifo, queue
L	??	Runs without persisting, does not create files
N	??	Persists as LaunchAgent, creates: GrowlHelper, Software Update Check, db

Hopper Disassembler is a tool that helps you disassemble, decompile and debug malware. There's a free version, and you can get a personal license for $99. Using Hopper, we can confirm the arguments we found by looking for argc, argv, and getopt.



```
                loc_956f:
0000956f        mov        dword [esp+0xf8+var_F0], esi                     ; argument "optstring" for method imp___jump_table__getopt, CO
00009573        mov        ecx, dword [ebp+var_A8]
00009579        mov        dword [esp+0xf8+var_F4], ecx                     ; argument "argv" for method imp___jump_table__getopt
0000957d        mov        eax, dword [ebp+var_A4]
00009583        mov        dword [esp+0xf8+var_F8], eax                     ; argument "argc" for method imp___jump_table__getopt
00009586        call       imp___jump_table__getopt                         ; getopt
0000958b        cmp        eax, 0xffffffff
0000958e        jne        loc_9423
```

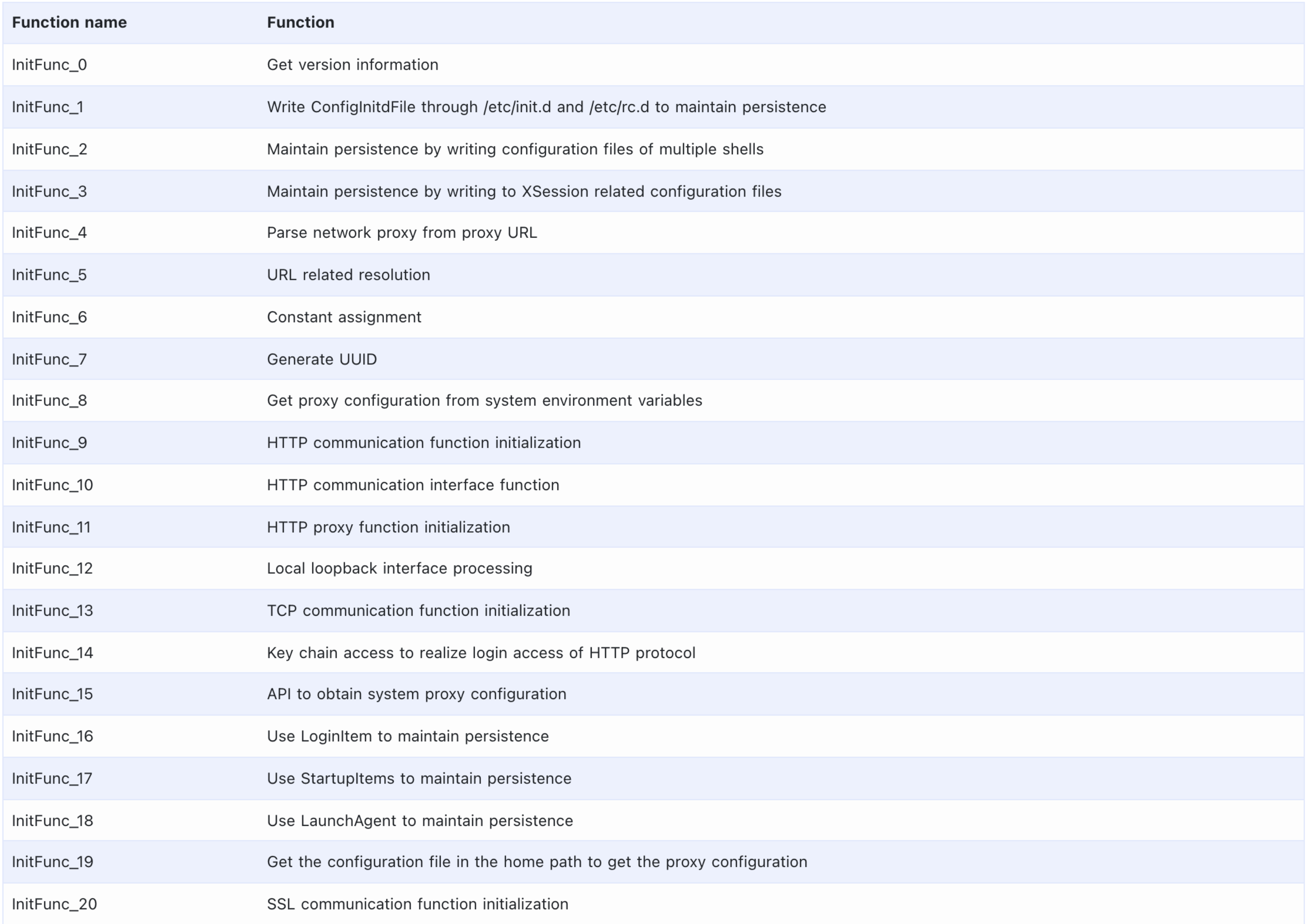
By using Hopper's pseudo-code mode, we can see the full set of possible command line arguments.



```
loc_956f:
    eax = getopt(var_A4, var_A8, "cdefLnNp:rRs");
    if (eax != 0xffffffff) goto loc_9423;
```

Entry Points

When you open GrowlHelper in Hopper, you'll see that it has multiple entry points: EntryPoint_1 through EntryPoint_21. These entry points are called when GrowlHelper starts executing, before the main entry point at 0x2cd8. GrowlHelper will use these entry points to initialize certain functionality. QI-ANXIN detailed these entry points in **this** post / this screenshot below.



Function name	Function
InitFunc_0	Get version information
InitFunc_1	Write ConfigInitdFile through /etc/init.d and /etc/rc.d to maintain persistence
InitFunc_2	Maintain persistence by writing configuration files of multiple shells
InitFunc_3	Maintain persistence by writing to XSession related configuration files
InitFunc_4	Parse network proxy from proxy URL
InitFunc_5	URL related resolution
InitFunc_6	Constant assignment
InitFunc_7	Generate UUID
InitFunc_8	Get proxy configuration from system environment variables
InitFunc_9	HTTP communication function initialization
InitFunc_10	HTTP communication interface function
InitFunc_11	HTTP proxy function initialization
InitFunc_12	Local loopback interface processing
InitFunc_13	TCP communication function initialization
InitFunc_14	Key chain access to realize login access of HTTP protocol
InitFunc_15	API to obtain system proxy configuration
InitFunc_16	Use LoginItem to maintain persistence
InitFunc_17	Use StartupItems to maintain persistence
InitFunc_18	Use LaunchAgent to maintain persistence
InitFunc_19	Get the configuration file in the home path to get the proxy configuration
InitFunc_20	SSL communication function initialization

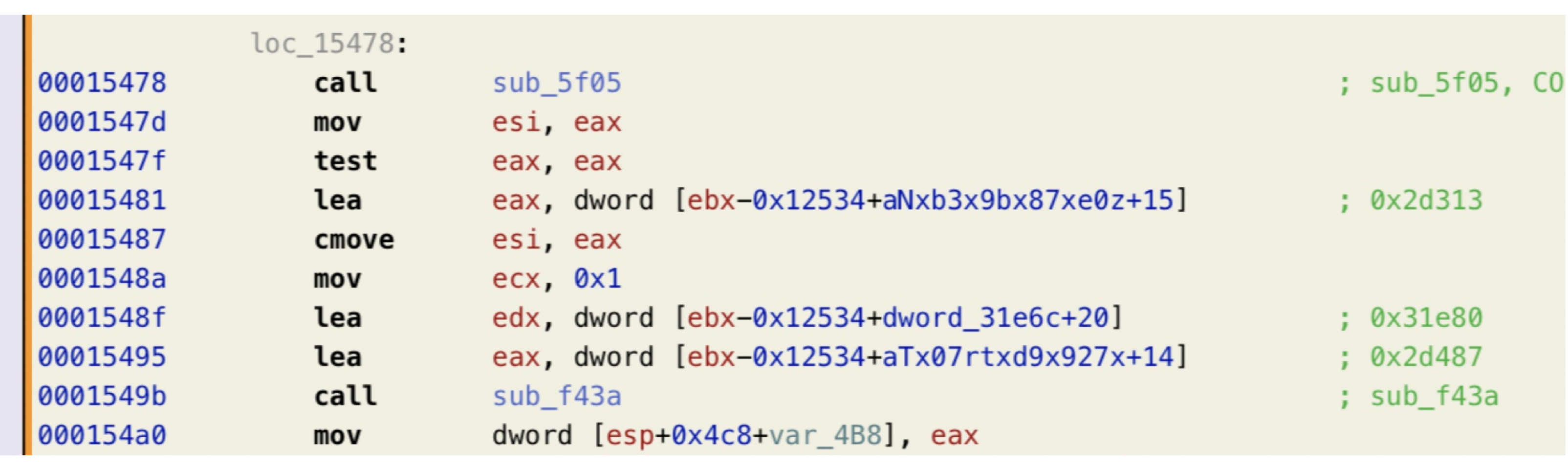
It appears GrowlHelper has a preflight checklist of sorts: it initializes functionality, figures out what it needs, deletes the rest.

```
$ sudo grep GrowlHelper filesys.out
mkdir      /Users/user/.DS_Info              0.000083   GrowlHelper.2851
mkdir    /Users/user/.DS_Info/5d0d           0.000044   GrowlHelper.2851
mkdir    /Users/User/Library/Caches/com.apple.advanced  0.000066   GrowlHelper.2851

rmdir    /Users/user/.DS_Info/5d0d           0.000109 W GrowlHelper.2851
rmdir    /Users/user/.DS_Info              0.000240 W GrowlHelper.2851
rmdir    /Users/User/Library/Caches/com.apple.advanced  0.000068   GrowlHelper.2851
```

Decrypting a String

Given the author, it's no surprise that most strings in this implant are encrypted. The implant appears to handle encrypted strings in a bunch of different ways, which makes it challenging to automate decryption. Hopper has done some of the analysis work for us, allowing us to at least manually decrypt strings with lldb. Here's one example.

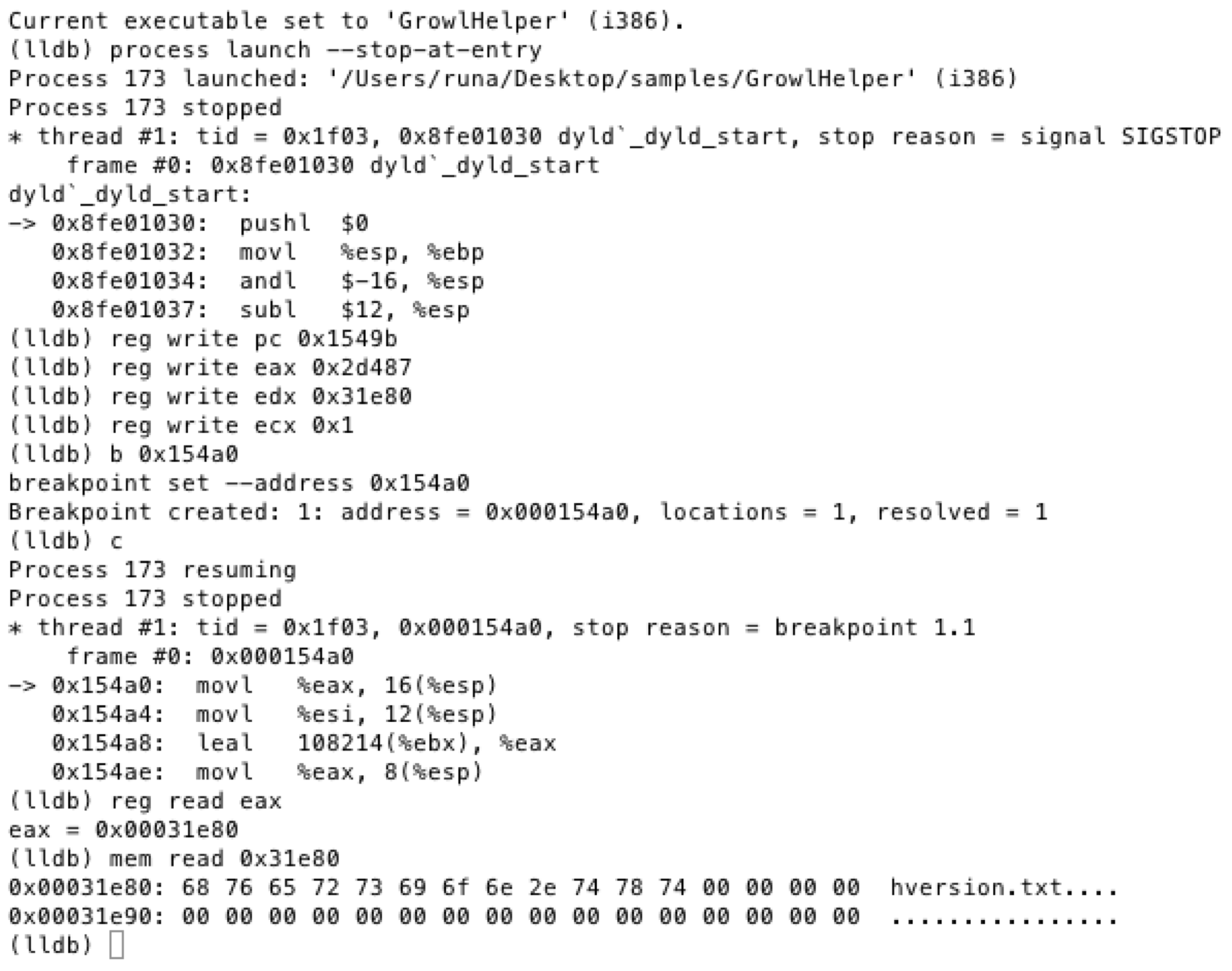


```
                loc_15478:
00015478        call       sub_5f05                                          ; sub_5f05, CO
0001547d        mov        esi, eax
0001547f        test       eax, eax
00015481        lea        eax, dword [ebx-0x12534+aNxb3x9bx87xe0z+15]       ; 0x2d313
00015487        cmove      esi, eax
0001548a        mov        ecx, 0x1
0001548f        lea        edx, dword [ebx-0x12534+dword_31e6c+20]           ; 0x31e80
00015495        lea        eax, dword [ebx-0x12534+aTx07rtxd9x927x+14]       ; 0x2d487
0001549b        call       sub_f43a                                          ; sub_f43a
000154a0        mov        dword [esp+0x4c8+var_4B8], eax
```

In the screenshot above, we have:

- The address for the program counter / call to the decryption routine (0x1549b)
- The values for ecx (0x01), edx (0x31e80), eax (0x2d487)
- The address after the decryption routine, which we'll use as a breakpoint for lldb (0x154a0)

We load the implant into the debugger using lldb GrowlHelper, and decrypt the string:



```
Current executable set to 'GrowlHelper' (i386).
(lldb) process launch --stop-at-entry
Process 173 launched: '/Users/runa/Desktop/samples/GrowlHelper' (i386)
Process 173 stopped
* thread #1: tid = 0x1f03, 0x8fe01030 dyld`_dyld_start, stop reason = signal SIGSTOP
    frame #0: 0x8fe01030 dyld`_dyld_start
dyld`_dyld_start:
-> 0x8fe01030:  pushl  $0
   0x8fe01032:  movl   %esp, %ebp
   0x8fe01034:  andl   $-16, %esp
   0x8fe01037:  subl   $12, %esp
(lldb) reg write pc 0x1549b
(lldb) reg write eax 0x2d487
(lldb) reg write edx 0x31e80
(lldb) reg write ecx 0x1
(lldb) b 0x154a0
breakpoint set --address 0x154a0
Breakpoint created: 1: address = 0x000154a0, locations = 1, resolved = 1
(lldb) c
Process 173 resuming
Process 173 stopped
* thread #1: tid = 0x1f03, 0x000154a0, stop reason = breakpoint 1.1
    frame #0: 0x000154a0
-> 0x154a0:  movl   %eax, 16(%esp)
   0x154a4:  movl   %esi, 12(%esp)
   0x154a8:  leal   108214(%ebx), %eax
   0x154ae:  movl   %eax, 8(%esp)
(lldb) reg read eax
eax = 0x00031e80
(lldb) mem read 0x31e80
0x00031e80: 68 76 65 72 73 69 6f 6e 2e 74 78 74 00 00 00 00  hversion.txt....
0x00031e90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
(lldb) 
```

Decrypting More Strings

Manually decrypting strings turned into a rabbit hole for me, but that’s OK. I’m sure there are ways to do this faster, but I have to admit I really enjoyed the process of learning to do this manually. Here are the strings I’ve decrypted so far, minus duplicates.

pc	String
0xe8a0	/tmp
0xe9ba	upload_dir
0xe9e2	upload_key
0xea23	upload_header
0xed50	52
0x185ef	download
0x187d7	?
0x18eae	InternetOpen
0x19121	** Commands will be processed immediately **
0x191f6	login.php
0x19216	getconf.php
0x19236	s|%s|%s|%s upload.gethostname
0x195be	show.php
0xa2f6	ConfigInitdFile
0x2ce6f	/etc/init.d
0xa762	/etc/rc.d.File
0xaccc	.xinitrc
0xae0b	ConfigPersistXsessionFile
0xae23	ConfigPersistXSession
0xaec9	.xsession
0xaf39	ConfigPersistXinitRCFile
0xaf51	ConfigPersistXInitRC
0xc8f0	proxy_type
0xc916	proxy_url
0xc948	Could not set proxy
0xca62	**http://www.google.com**
0xce05	no proxy_url
0x11309	index.html
0x11816	hps.txt
0x11d35	NODELETE
0x11d64	DELETE
0x11d93	SECDELETE
0x1218d	NOWAIT
0x121c0	WAIT
0x121f1	WAIT_FOREVER
0x1225a	/bin/sh -c
0x132b1	Version
0x13c1e	Service
0x147f8	Proxy
0x14b1e	ProxyUser
0x1549b	hversion.txt
0x15c12	HHLogEntry
0x15c5b	HHLogHead
0x15e2d	HHLogTail
0x1a427	hh_last_attempt
0x1a530	localhost_sock_create(pipe)
0x1a8ab	hh_last_attempt
0x649e	No LP configured
0x6a66	16

Listening Post

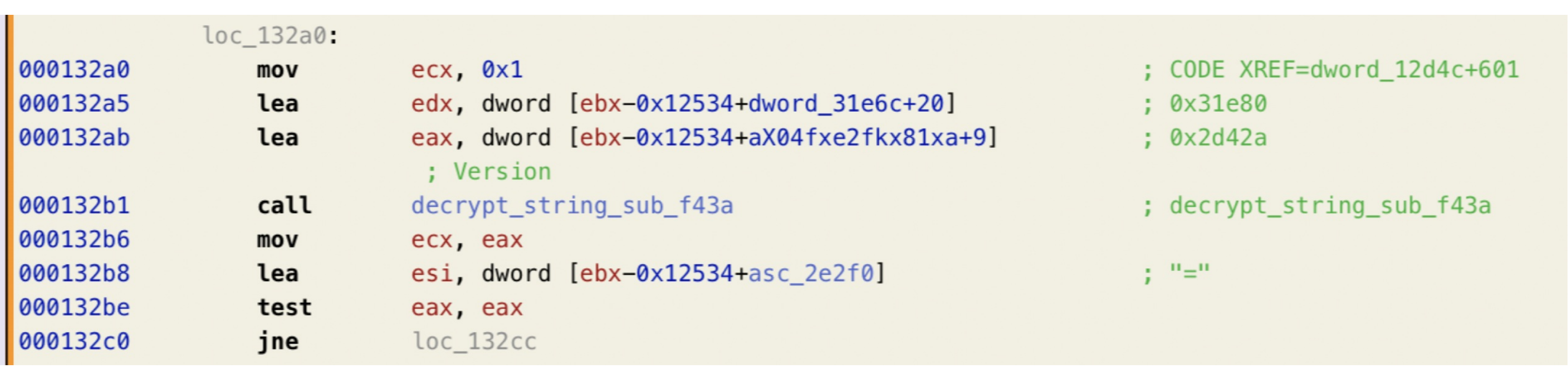
One of the decrypted strings is No LP configured. LP likely stands for *Listening Post*, a military term used in the context of signals intelligence and reconnaissance. Where other types of malware would likely use the terms *C2* or *Command & Control*, the CIA and the NSA use LP. One Vault 7 document is titled **Listening Post (LP) Creation**, and another details **requirements** for a Listening Post.

Configuration Files

Some of the decrypted strings refer to .html, .php, and .txt files, but I’m unable to access them. But we know that Kaspersky found “a hostname and an IP address” hardcoded in the implant. And researchers at QI-ANXIN determined the implant talks to the Listening Post through login.php and getconf.php, and downloads follow-up code through getfile.php.

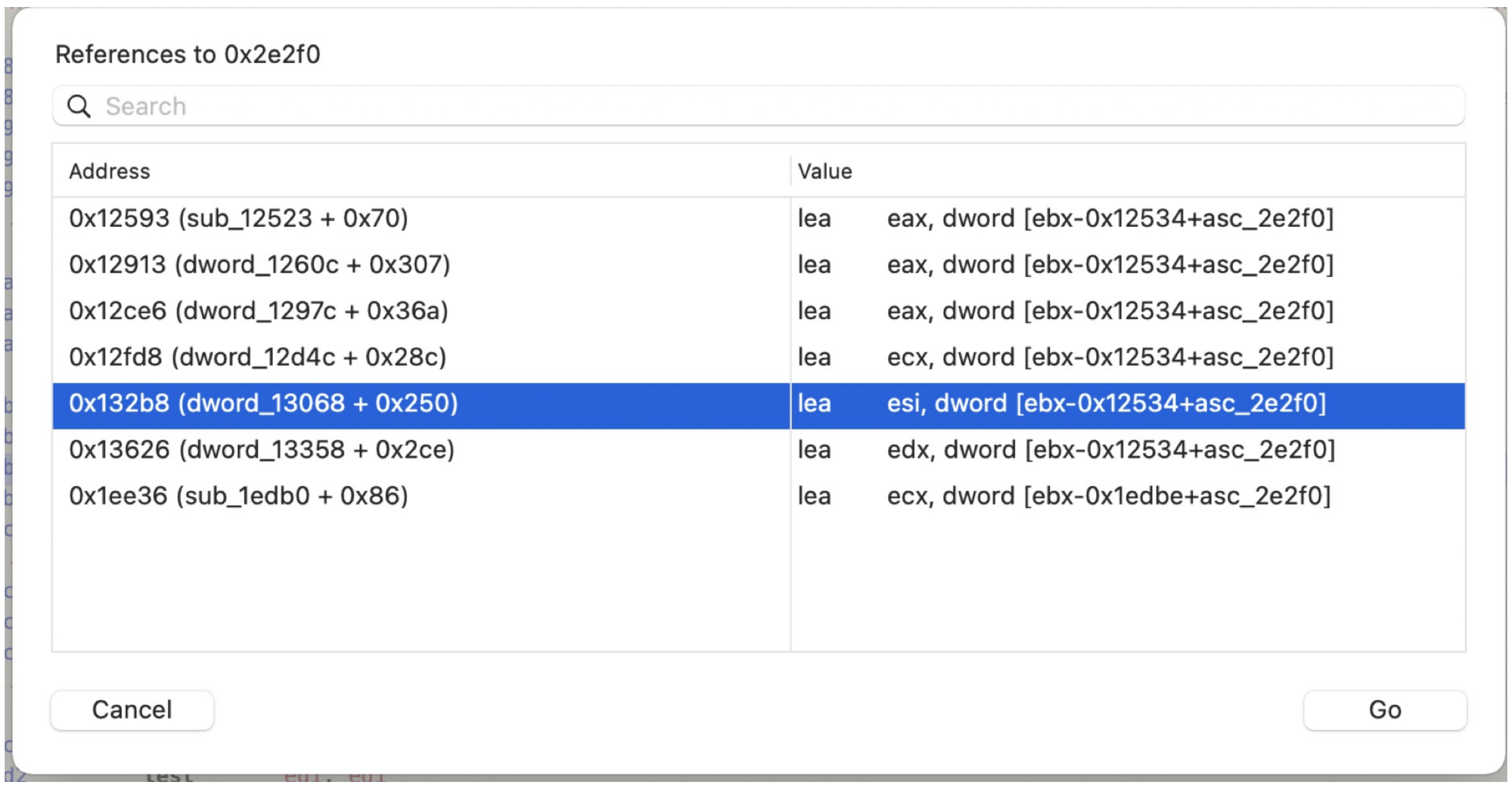
Configuration? Survey?

If you dig around in Hopper and use pseudo-code mode from time to time, you’ll likely find some interesting bits of information. When I stumbled upon the string Version=1.2.0, I decided to see where else = was referenced. To do that, highlight 0x132b8 as shown below and hit x.



```
                loc_132a0:
000132a0            mov        ecx, 0x1                                          ; CODE XREF=dword_12d4c+601
000132a5            lea        edx, dword [ebx-0x12534+dword_31e6c+20]           ; 0x31e80
000132ab            lea        eax, dword [ebx-0x12534+aX04fxe2fkx81xa+9]        ; 0x2d42a
                                ; Version
000132b1            call       decrypt_string_sub_f43a                           ; decrypt_string_sub_f43a
000132b6            mov        ecx, eax
000132b8            lea        esi, dword [ebx-0x12534+asc_2e2f0]                ; "="
000132be            test       eax, eax
000132c0            jne        loc_132cc
```

The list of references looks like this, with the current one selected.



References to 0x2e2f0

Search

Address	Value
0x12593 (sub_12523 + 0x70)	lea eax, dword [ebx-0x12534+asc_2e2f0]
0x12913 (dword_1260c + 0x307)	lea eax, dword [ebx-0x12534+asc_2e2f0]
0x12ce6 (dword_1297c + 0x36a)	lea eax, dword [ebx-0x12534+asc_2e2f0]
0x12fd8 (dword_12d4c + 0x28c)	lea ecx, dword [ebx-0x12534+asc_2e2f0]
0x132b8 (dword_13068 + 0x250)	lea esi, dword [ebx-0x12534+asc_2e2f0]
0x13626 (dword_13358 + 0x2ce)	lea edx, dword [ebx-0x12534+asc_2e2f0]
0x1ee36 (sub_1edb0 + 0x86)	lea ecx, dword [ebx-0x1edbe+asc_2e2f0]

We can then go through all these references, decrypt the strings, and get an output that looks like this.

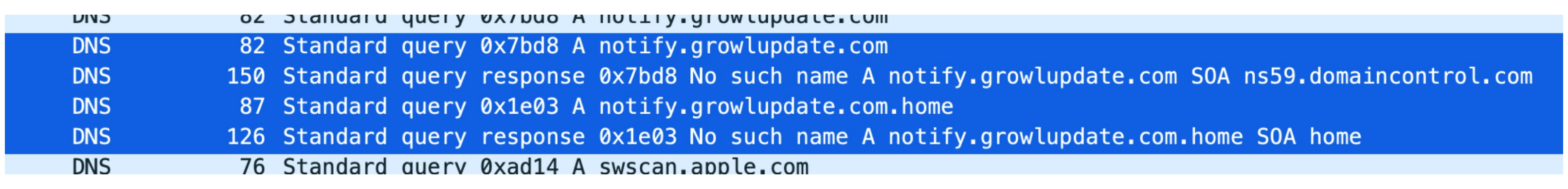
```
uname=
Time=%Y\%m\%d %H:%M:%S Z
Uptime=
Version=1.2.0
PID=
```

The output lists information from the target system (e.g. uname) and information from the implant (e.g. Version). This could be a combination of a configuration file and system survey.

Network Traffic

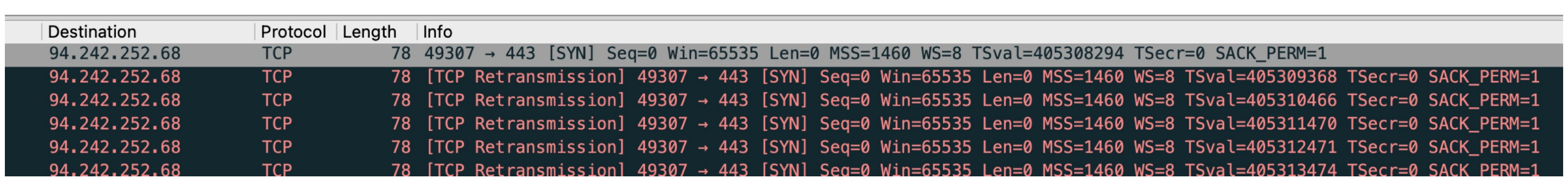
We can monitor the network traffic on our OS X 10.7 system using tcpdump and then view the output in Wireshark.

This gives us the hardcoded hostname notify[.]growlupdate[.]com. Very clever given the name of the executable.



```
DNS         82 Standard query 0x7bd8 A notify.growlupdate.com
DNS         82 Standard query 0x7bd8 A notify.growlupdate.com
DNS        150 Standard query response 0x7bd8 No such name A notify.growlupdate.com SOA ns59.domaincontrol.com
DNS         87 Standard query 0x1e03 A notify.growlupdate.com.home
DNS        126 Standard query response 0x1e03 No such name A notify.growlupdate.com.home SOA home
DNS         76 Standard query 0xad14 A swscan.apple.com
```

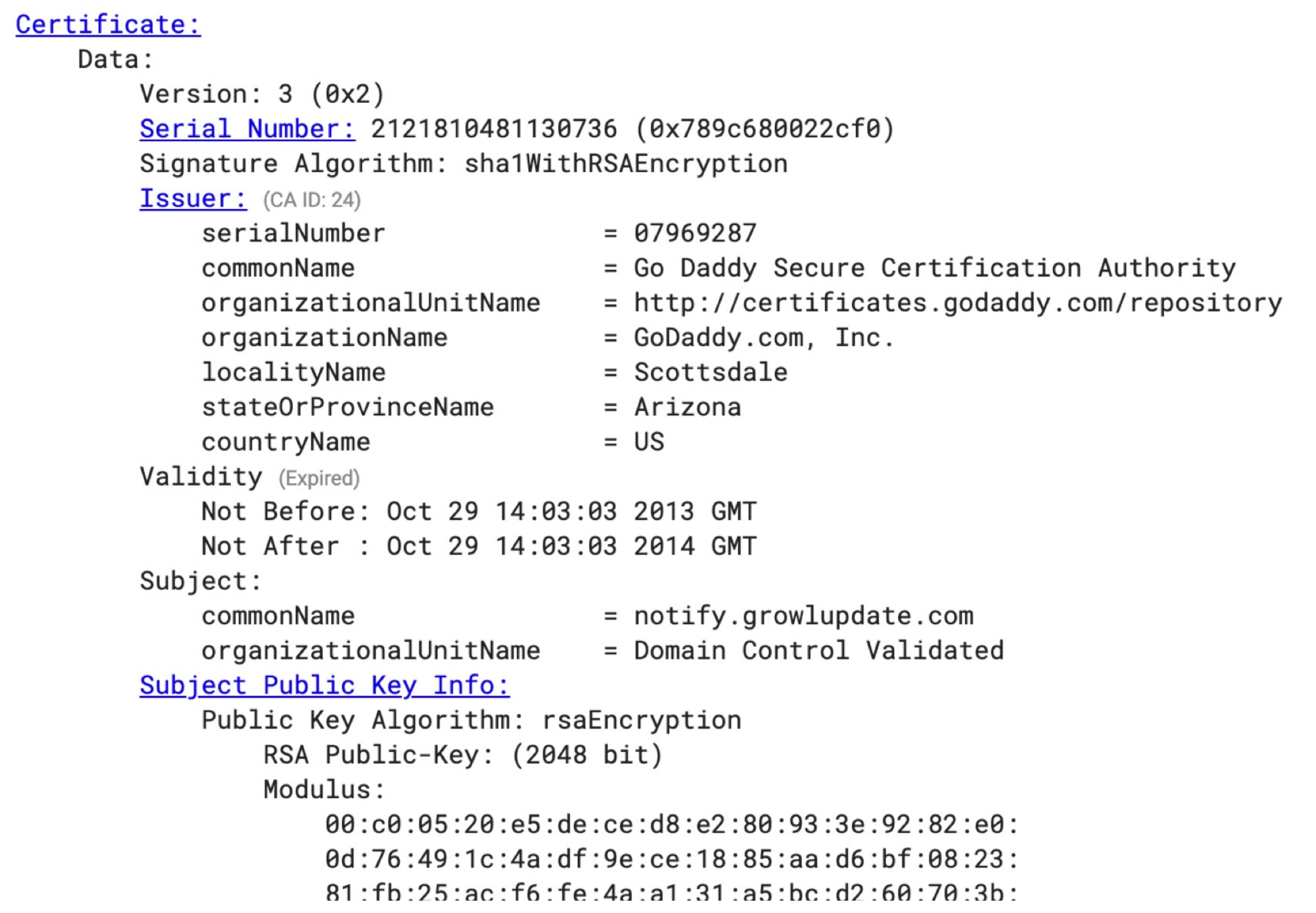
And the hardcoded IP address: 94[.]242[.]252[.]68.



```
Destination      Protocol Length Info
94.242.252.68    TCP          78 49307 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=405308294 TSecr=0 SACK_PERM=1
94.242.252.68    TCP          78 [TCP Retransmission] 49307 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=405309368 TSecr=0 SACK_PERM=1
94.242.252.68    TCP          78 [TCP Retransmission] 49307 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=405310466 TSecr=0 SACK_PERM=1
94.242.252.68    TCP          78 [TCP Retransmission] 49307 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=405311470 TSecr=0 SACK_PERM=1
94.242.252.68    TCP          78 [TCP Retransmission] 49307 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=405312471 TSecr=0 SACK_PERM=1
94.242.252.68    TCP          78 [TCP Retransmission] 49307 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=405313474 TSecr=0 SACK_PERM=1
```

Hostname

Google and the **Wayback Machine** don't have any results for the domain name. If we look it up on **VirusTotal**, we see that it was first submitted in October 2016. But if we look up the domain on **crt.sh**, we see that an SSL certificate was created on October 29, 2013. The domain may have been purchased earlier, but this at least suggests the domain was active in late 2013. This matches the timeline we created earlier, as well as **Kaspersky's timeline** of activity by The Lamberts.



```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2121810481130736 (0x789c680022cf0)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: (CA ID: 24)
            serialNumber              = 07969287
            commonName                = Go Daddy Secure Certification Authority
            organizationalUnitName    = http://certificates.godaddy.com/repository
            organizationName          = GoDaddy.com, Inc.
            localityName              = Scottsdale
            stateOrProvinceName       = Arizona
            countryName               = US
        Validity (Expired)
            Not Before: Oct 29 14:03:03 2013 GMT
            Not After : Oct 29 14:03:03 2014 GMT
        Subject:
            commonName                = notify.growlupdate.com
            organizationalUnitName    = Domain Control Validated
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c0:05:20:e5:de:ce:d8:e2:80:93:3e:92:82:e0:
                    0d:76:49:1c:4a:df:9e:ce:18:85:aa:d6:bf:08:23:
                    81:fb:25:ac:f6:fe:4a:a1:31:a5:bc:d2:60:70:3b:
```

Note: Kaspersky sinkholed the domain to 95[.]211[.]172[.]143 between October 1, 2016 and October 2, 2017.

Development Tradecraft DOs and DON'Ts

As part of Vault 7, WikiLeaks published 52 revisions of the CIA's **development tradecraft** guidelines. I mapped the revisions in a **public spreadsheet** to see how the guidance changed over time. Studying the development choices made by sophisticated actors may help us track them over time. For example, when Kaspersky identified a code overlap between **Sunburst and Kazuar**, it was because of "unusual, shared features" such as the UID generation algorithm, the sleeping algorithm, and use of the FNV-1a hash.

As Costin Raiu of Kaspersky **pointed out** on Twitter, "C2 jitter, secure erase / uninstall, SSL/TLS+extra crypto, size below 150K, encrypt logs and local collection, decrypt strings on the fly in mem… simply following these guidelines immediately makes the malware ("tools") more interesting and, recognizable by a skilled analyst." While most of these are true here as well, there are a few things that stand out.

- File size is a bit over the "ideal binary file size" for a fully featured tool (208K v. 150K)
- The references to *Listening Post / LP* may be CIA and USG specific terminology
- Use of English abbreviations for days of the week (mtwhfsu / MTWHFSU)
- Use of the libevent library back when it was perhaps less well-known

Conclusion

I've really enjoyed working on this project and certainly learned a lot along the way. I'm confident there's more to find here, and I'd love to collaborate with anyone interested in taking a closer look. As for The Lamberts? Malware from this actor keeps turning up, along with new insights. In fact, Kaspersky's **APT trends report for Q1 2021** mentions Purple Lambert, a malware "capable of providing an attacker with basic information about the infected system and executing a received payload."

Indicators of Compromise

- notify[.]growlupdate[.]com
- 94[.]242[.]252[.]68
- 3fcdbd3c5fa34fb8e8d58038fa1d1f13d37e8a4b

References

Patrick's free and open-source book on Mac malware analysis was incredibly helpful during this project. If you haven't already done so, I highly recommend checking out **The Art of Mac Malware**.