FLASHBACK OS X MALWARE

Broderick Ian Aquilino
F-Secure Corporation, Tammasaarenkatu 7, 00181, Helsinki, Finland

Email broderick.aquilino@f-secure.com

ABSTRACT

Windows has been the target of malware for decades. Over time, the sustained pressure of being an attack target has forced its evolution into a more hardened system, as well as increasing user awareness of computer security. On the other hand, *OS X* has not needed to go through all the troubles of crime fighting until recently. Now, with its growing market share and lower user awareness, it is clear that *OS X* is becoming more and more attractive to malware authors.

In 2011, we saw *OS X* come under siege by several malware families. Towards the end of the year, we saw new families or variants appear almost every week, where each was more sophisticated than the last. At the forefront of these developments was the Flashback malware.

Flashback is the most advanced *OS X* malware we've ever seen. It boasts a series of firsts for its kind. It was both the first to be *VMware*-aware and the first to disable XProtect, *OS X*'s built-in malware protection program. Both these features were removed from later variants (the former presumably to avoid heuristic detections, and the latter presumably once the authors realized it was unnecessary, as XProtect was not designed to protect against non-quarantine files). Their removal indicates that Flashback is actively being reviewed and improved by its authors.

Another interesting first is Flashback's exploitation of an unpatched vulnerability in the Java distribution of *OS X*, which allowed it to infect more than 650,000 Macs around the world [1]. This made Flashback roughly as common for Macs as Conficker was for *Windows* [2]. This means Flashback is not only the most advanced, but also the most successful *OS X* malware we've seen so far.

Flashback's infection strategy is explicitly designed to select unprotected systems and will not infect a machine if certain security software or analysis tools are found. This implies that Flashback's authors are targeting less security-conscious users, at the expense of the total number of potential targets. This turns out to be an effective strategy, as security researchers had difficulties getting sufficient samples from users. It took a mistake on the part of Flashback's author to alert users to the presence of an infection and subsequently, to lead to the mass discovery of the malware.

So what is Flashback? Basically, it is a piece of malware that modifies targeted web pages displayed in web browsers. In older variants, the targeted web pages and modification were based on the configurations retrieved from a remote server during installation. In newer variants, the target and modification have been hard-coded into the binaries.

Flashback's development indicates that its intended purpose was to redirect *Google* results to third-party advertisers, for the authors' profit. Flashback's intended purpose and the social engineering tactics used in its distribution are exactly the same as those of the Mac QHost malware. Mac QHost redirected infected users to a fake *Google* website that served search results belonging to third-party advertisers. It also posed as a *Flash Player* installer to spread [3]. These similarities suggest that Flashback may be the next evolution of Mac QHost.

This paper focuses on the technical analysis of the binaries of new Flashback variants, with brief comparisons against the older variants. Actual network data used by Flashback is beyond the scope of this paper; examples used here are for illustration purposes only and do not reflect actual data.

DISTRIBUTION

Flashback is distributed via malicious websites, usually hosted in the '.rr.nu' domains. Users infected by Flashback didn't intend to visit such websites, but were redirected to them on visiting compromised legitimate websites. In the cases I encountered, the hacked websites were running *WordPress*.

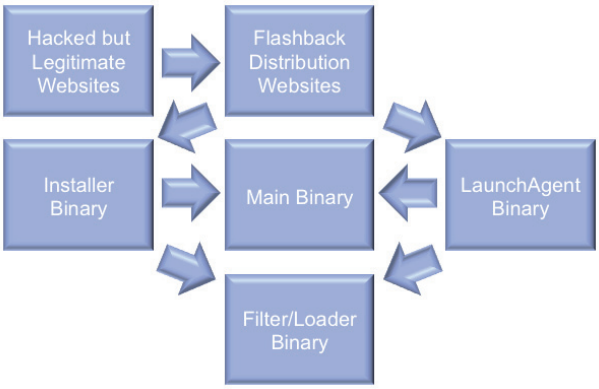

Figure 1: Flashback infection summary.

Early on in Flashback's history, visitors to the malicious websites would have seen an image – not a real event, just an image – of a *Flash Player* crash. This is how Flashback got its name.

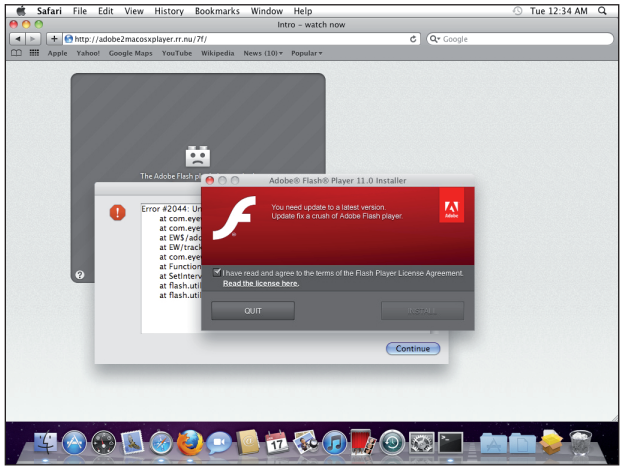

Figure 2: Fake Flash Player crash image.

Like most social engineering tactics, the image was meant to fool users into believing the player had crashed, and that they needed to download an 'update'. In the background, a .pkg package was downloaded.

Figure 3: Fake Flash Player installer.

Although this social engineering tactic has proven fairly effective, alert users are still able to detect the trickery involved. At this point in time, Flashback still needed to be manually installed before it could infect the system, offering an escape point for users. This weakness was effectively erased in February 2012, when Flashback started using drive-by downloads to infect users [4].

In this revision, the malware attempted to exploit the following vulnerabilities in Java when a user visited a Flashback distribution website:

- CVE-2008-5353
- CVE-2011-3544

Unfortunately for Flashback, these vulnerabilities had already been patched by *Apple* in November 2011. In case a user is already patched, Flashback simply reverts back to using social engineering tactics. This time the malware used a self-signed applet pretending to be from *Apple*.

Figure 4: Self-signed Java applet claiming to be from Apple.

The next month, however [5], Flashback started to exploit an (at the time) unpatched vulnerability in Java: CVE-2012-0507. *Oracle*, Java's developer, had already patched this vulnerability in the previous month but *Apple* had not yet released the patch for the Java distribution of *OS X*. This left *OS X* users with Java installed on their systems vulnerable to infection if they simply happened to visit the wrong site at the wrong time.

When successfully exploited or when users choose to run the signed applet, a binary is dropped to the '/tmp' folder of the system and executed. This binary is the main installer and is the equivalent binary found inside .pkg packages of earlier variants.

INSTALLATION

Upon execution, the installer checks a list of paths that belong to security software and analysis tools. The latest list consists of the following:

- /Library/Little Snitch
- /Developer/Applications/Xcode.app/Contents/MacOS/Xcode
- /Applications/VirusBarrier X6.app
- /Applications/iAntiVirus/iAntiVirus.app
- /Applications/avast!.app
- /Applications/ClamXav.app
- /Applications/HTTPScoop.app
- /Applications/Packet Peeper.app

If any of the listed paths are found, the installer skips the rest of the routines and deletes itself. However, the latest variant of the installer has a bug where it didn't check the existence of the paths during enumeration. This caused Flashback to be installed regardless.

If the installer is cleared to proceed, it connects back to the distribution website to obtain the installation data:

- http://%distribution_website%/counter/%Base64_encoded_data%

The encoded data contains information about the target system. The actual data differs between variants but the latest form consists of the following:

- PlatformUUID
- hw.machine
- kern.osrelease
- '0'
- proc_cputype of the installer
- utsname.machine
- '1' if installer is running as first user of the target system or daemon otherwise '0'.

The Base64 decoded data would look something like this:

```
00000000-0000-0000-0000-000000000000|i386|10.8.0|0|x86_64|i386|1
```

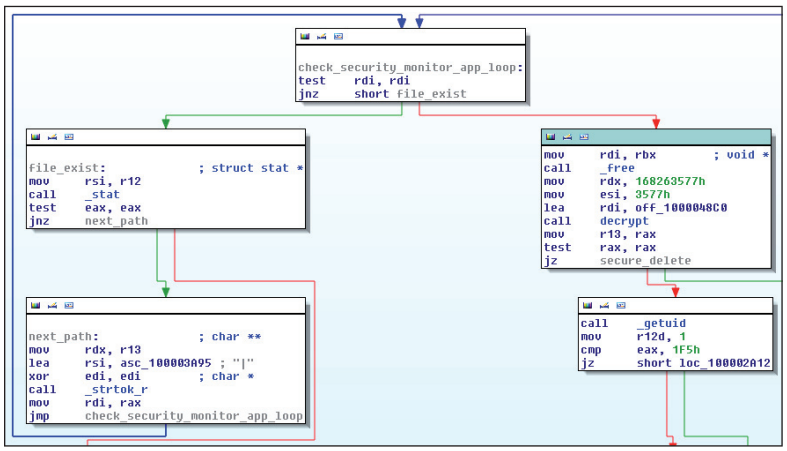

Figure 5(a): Product check loop in earlier variants.

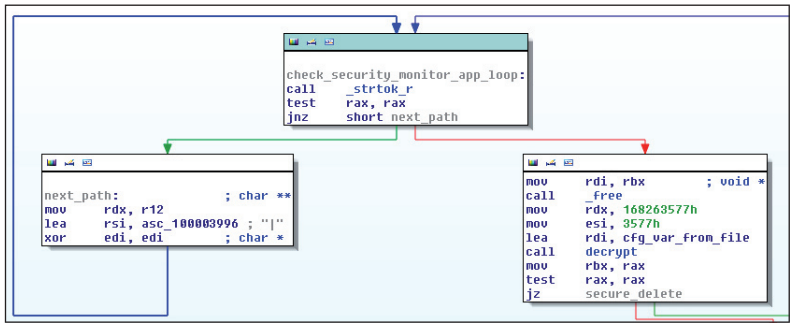

Figure 5(b): Product check loop in the latest variant.

The installer expects the response to be in the following format:

- Layer 1 – RC4 encrypted with the MD5 hash of the target system's PlatformUUID as the key
- Layer 2 – zlib packed
- Layer 3 – installation data delimited by 'l' where each item is Base64 encoded.

The actual installation data differs between variants but the latest form consists of the following:

- Filename
- Main binary
- Configuration data
- Filter/loader binary
- Icon used in authentication dialog box (Figure 6).

There are generally two types of Flashback infection, with the first occurring if the installer has administrative privilege, and the second occurring if it does not. In some earlier variants of the installer, Flashback asks users for their admin password and will only infect the system when administrative privilege is obtained. Other earlier variants immediately infect the system without trying to obtain administrative privilege. In the latest variant of the installer, Flashback asks users for their admin password but infects the system whether users input their passwords or not. How the users respond only determines the subsequent infection type.

Figure 6: Authentication dialog box in recent variants of the installer.

INFECTION TYPE 1: ADMINISTRATIVE PRIVILEGE

Flashback installers that have obtained administrative privilege drop the binaries returned by the distribution website to the 'Contents/Resources' folder of the targeted browsers. Earlier variants targeted *Firefox* and *Safari*, while recent ones target only *Safari*.

The dropped files use the following name format in recent variants of the installer:

- Main binary – /Applications/Safari.app/Contents/Resources/.%filename%.png
- Filter/loader binary – /Applications/Safari.app/Contents/Resources/.%filename%.xsl

where %filename% is a variable returned by the distribution website. In the samples I encountered, the malware used filenames such as BananaSplittervxall, ALOPowerAudio and ACDExpressv, which resemble real audio-video application names and were probably chosen to maintain the illusion that the installer is some kind of multimedia application.

The installer then creates a launch point by adding a DYLD_INSERT_LIBRARIES value to the LSEnvironment dictionary in the property list file of the targeted browsers. The value is the filter/loader binary's path. For variants that do not have this component, the value is the main binary's path.

In this infection type, Flashback is loaded by infected browsers launched by any system users. As such, this infection type is Flashback's preferred method.

The installer inserts the main binary's final path into the main binary's own body where it is used by the binary to determine its own location. The path is inserted after the 'slfp' marker.

```
000216F8:  73 6C 66 70-2F 41 70 70-6C 69 63 61-74 69 6F 6E  slfp/Application
00021708:  73 2F 53 61-66 61 72 69-2E 61 70 70-2F 43 6F 6E  s/Safari.app/Con
00021718:  74 65 6E 74-73 2F 52 65-73 6F 75 72-63 65 73 2F  tents/Resources/
00021728:  2E 42 61 6E-61 6E 61 53-70 6C 69 74-74 65 72 76  .BananaSplitterv
00021738:  78 61 6C 6C-2E 70 6E 67-00 00 00 00-00 00 00 00  xall.png
```

Figure 7: Main binary's path in its own body.

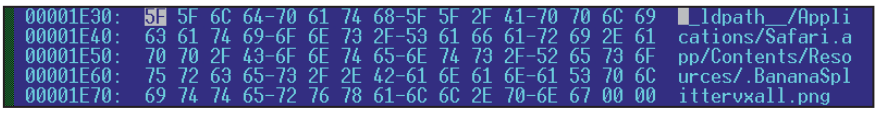

Figure 8: Main binary's path in filter/loader binary.

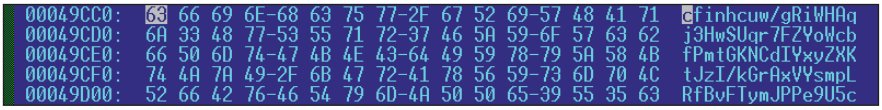

Figure 9: Configuration data in main binary.

For variants that have the filter/loader component, the main binary's path is also inserted into the filter/loader's body where it is used to determine which component to load. The path is inserted after the '__ldpath__' marker.

Finally, the installer inserts the configuration data returned by the distribution website into the body of the main binary. The binary's dependence on this data will become obvious in later sections of this paper. The data is inserted after the 'cfinh' marker.

In earlier variants, the encoded configuration data is inserted as is. In the latest variant, the data is decoded and encrypted first, and then re-encoded again before being inserted into the main binary's body. The RC4 algorithm is used in the encryption with the target system's PlatformUUID as the key. This in effect locks the main binary to the target system. This prevents the sample from being analysed by researchers who have got hold of the sample but do not know where it originally came from, or by automated systems to which the user has submitted the sample for analysis.

The latest variant of the installer also reports the result of the infection attempt back to the distribution website:

- Successful – http://%distribution_website%/stat_d/
- Failed – http://%distribution_website%/stat_n/

INFECTION TYPE 2: USER PRIVILEGE

Flashback installers that didn't get administrative privilege simply drop the binaries to a location where the user who executed the installer does have permission, such as the '/Users/Shared' folder [6].

The dropped files use the following name format in recent variants of the installer:

- Main binary – ~/Library/Application Support/.%filename%.tmp
- Filter/loader binary – /Users/Shared/.libgmalloc.dylib

where %filename% is a variable returned by the distribution website.

The installer then creates a launch point by adding a DYLD_INSERT_LIBRARIES value to the local property list file of the user running the installer. As in Type 1 infection, this value is the filter/loader binary's path. In variants without this component, it is the main binary's path.

In this infection type, Flashback is loaded by any application launched in the infected user profile. This indiscriminate launch behaviour makes it more likely that the malware will be loaded by an incompatible application, and it may be the reason why the filter/loader binary was introduced in new variants.

All the filter/loader binary does is ensure the main binary is loaded only by the correct processes; however even then, there's an issue – the filter binary only runs on *Intel*-based Macs. To avoid dealing with *PowerPC*-based applications running on top of *Rosetta*, the latest variant of the installer refers to a list of popular *PowerPC*-based or related applications and will not infect a system if any of the following are found:

- /Applications/Microsoft Word.app
- /Applications/Microsoft Office 2008
- /Applications/Microsoft Office 2011
- /Applications/Skype.app

Much like Type 1 infections, the main binary's final path and configuration data are inserted into the binary's body. In variants with a filter/loader component, the main binary's path is also inserted into the component's body.

The latest variant of the installer also reports the result of Type 2 infection attempts back to the distribution website, though only a successful infection is reported:

- http://%distribution_website%/stat_u/

After successfully infecting a system, the installer kills the targeted browsers to force users to restart them and activate the malware.

Finally, regardless of whether or not infection was successful, Flashback securely deletes itself (by writing zeros to itself) before removing itself from the file system.

CONFIGURATION DATA

The main binary's behaviour is mostly defined by the configuration data returned by the distribution website during installation. In the latest version, the configuration data is in the following format:

- Layer 1 – Base64 encoded
- Layer 2 – RC4 encrypted with the infected system's PlatformUUID as the key
- Layer 3 – RC4 encrypted with a static key found in the body of the main binary
- Layer 4 – zlib packed
- Layer 5 – configuration data in plain text.

The plain text configuration data is a series of configuration entries arranged in the following format:

- [0,{%config_entry1%},{%config_entry2%},…]

The configuration entries have the following format:

- {%entry_id%:%entry_type%:%value%}

Entry type	Value
1	Integer (in ASCII)
2	Boolean (in string)
3	Base64 encoded string
4	Base64 encoded configuration data (used in nested configuration data)
7	Array of Base64 encoded strings delimited by '|'

As discussed earlier, the configuration data is embedded in the main binary's body; in later variants, however, this can be overridden by an external configuration file specified in entry ID 1007785508 of the embedded configuration data. In my reference sample, the value is '/Users/Shared/.sbl'. If the data in the external configuration file is newer than in the embedded one – as determined by entry ID 3702222652, which contains the time a configuration data is created – the embedded data is overridden. If entry ID 3702222652 is not present in the configuration data, it gets the default value of 0. The introduction of an external configuration file makes it possible for C&Cs to update the malware without needing to modify the binaries.

UPDATE SERVERS

Flashback uses three groups of C&C servers. The first group communicates with a VM-like module in the main binary that interprets a series of instructions returned by the C&C. In this section, we'll discuss the selection process of the C&C servers in this group.

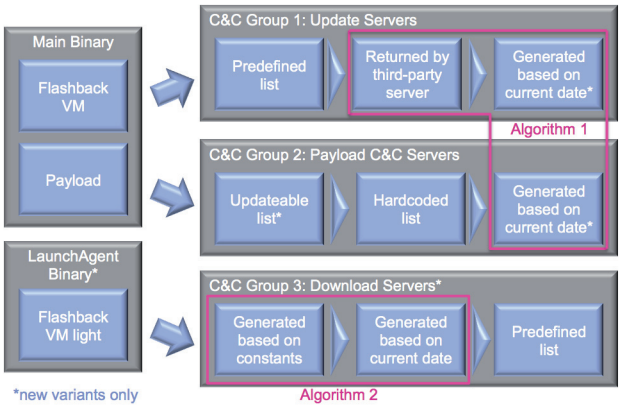

Figure 10: Flashback C&C server groups.

When an infected browser is launched, the malware creates a thread that connects to one of the C&C servers found in its configuration data every 3,670 seconds:

Entry ID	Description
1543152164	Total servers (n)
4294901760 (up to 4294901760 + n - 1)	Server
201539444	URI scheme (new variants only) In my reference samples, this was 'http://'
2907319225	Resource path In my reference samples, this was '/auupdate/'

In older variants, the URI scheme was provided by a static string in the main binary. The switch to using configuration data for the URI scheme in later variants may be a sign that the authors intended to support different protocols in the future.

The final URL of a C&C server would look like this:

- http://updateserver1.com/auupdate/

The user-agent in the request is a Base64 encoded string containing details of the infected system. Specific details differ between variants, but in the latest variants, the following data was gathered:

- Malware version (specified in entry ID 2279540384)
- hw.machine
- kern.osrelease
- PlatformUUID
- SHA1 of the main binary
- A series of '1' or '0', each representing a browser: '1' if found, '0' if not
- '999'
- '1' if the host process (which is the infected browser) is running as root or daemon, otherwise '0'.

The following entry IDs are used to determine which browsers are installed:

Entry ID	Description
1829190487	Total paths (n)
3735941120 (up to 3735941120 + n - 1)	Path of browser

My reference samples have the following configuration data entries:

Entry ID	Sample entry data
1829190487	3
3735941120	/Applications/Google Chrome.app/Contents/MacOS/Google Chrome
3735941121	/Applications/Opera.app/Contents/MacOS/Opera
3735941122	/Applications/Firefox.app/Contents/MacOS/firefox

Taking the above entries as an example, if only *Firefox* is installed, the Base64 decoded user-agent would look something like the following:

```
38|i386|10.8.0|00000000-0000-0000-0000-000000000000|d7
8b0b8e15d6cde73ff23c5f58513524f6772bf1|001|999|0
```

After the request is sent to the C&C server, Flashback performs a verification check on the response from the server. It expects a two-part response in the following format:

%marker1%%part1%%marker2%%part2%%marker3%

Where the various elements are:

- Markers: Special characters to parse out the relevant contents
- Part 1: The Base64 encoded update instructions in the form of a Flashback VM program
- Part 2: The Base64 encoded RSA signature of part 1.

The markers are determined by configuration data entry IDs 24453559, 3138449062 and 2843125468, which in my reference samples are '**', '%%' and '^^', respectively.

Part 1, the Flashback VM program, is discussed in the next section.

Part 2, the RSA signature, uses MD5 hashing and is verified using a static public key found in the main binary. This prevents the malware from being hijacked if an unknown party (i.e. a security researcher) gains access to the C&C server, as they still need the corresponding private key to sign the instructions. If the RSA signature returned by a C&C server is valid, the malware executes the Flashback VM program.

If the RSA signature is invalid, the malware tries the next C&C server listed in the configuration data. If none of the servers returned a valid RSA signature, Flashback connects to a third-party server to obtain the URL of the next C&C server.

The third-party server location is partly determined by subentry ID 3793486453 of entry ID 1131202474, which in my reference samples is 'http://mobile.twitter.com/searches?q=%23'. A string derived from the current date is then appended to this to form the actual URL. This string is calculated using a simple algorithm based on the current date, where the values are IDs to the subentries of entry ID 1131202474.

As an example, for the date 27 September 2012:

Third-party Server Location		
Entry ID 1131202474, subentry ID 3793486453		http://mobile.twitter.com/searches?q=%23
Date-based algorithm	**Subentry ID of Entry ID 1131202474**	**Sample Entry Data**
Current day	27	rdel
Current month – 1	8	d2ir
Current year – 2000	12	xloa

The full URL for the third-party server would then be:

- http://mobile.twitter.com/searches?q=%23rdeld2irxloa

Once the URL is generated and contacted, the malware parses the response from the third-party server to obtain the URL of the next C&C server it will contact:

- %marker1%%server_URL%%marker2%

The markers are specified in subentry IDs 2680482723 and 3938521920 of entry ID 1131202474, which in my reference samples were 'bumpbegin', and 'endbump' respectively. However, Flashback may fail to get the intended URL due to a bug in its parsing routine: it subtracted the length of marker 2 instead of marker 1 while computing the length of the server URL.

```
mov     rax, [rbp+var_40_marker2]
sub     rcx, rbx        ; rcx = position of marker2,  rbx = position of marker1
lea     r13, [rbp+var_B0_url]
mov     rsi, r14
mov     rdi, r13
sub     rcx, [rax-18h]
mov     rax, [rbp+var_80_marker1]
add     rbx, [rax-18h]
mov     rdx, rbx
call    __ZNKSs6substrEmm ; std::string::substr(ulong,ulong)
```

Figure 11: Bug in the parsing response of the third-party server.

Taking the response 'bumpbeginhttp://updateserver2.com/auupdate/endbump' as an example, the malware would connect to the URL 'http://updateserver2.com/auupdate/en' to obtain the update program.

In new variants, an additional set of C&C servers are generated in case none of the C&C servers returned a valid signature. The domain names on this list are generated based on the current date and the same algorithm used to create the appended string for the third-party server URL. This could be a sign that the authors intend to eliminate the need for a third-party server in the future.

Once the domain names are generated, they are appended with a set of TLD names specified in entry ID 3078701270, which in my reference samples were: '.org', '.com', '.co.uk', '.cn' and '.in'. As before, Flashback uses entry IDs 201539444 and 2907319225 for the URI scheme and resource path to form the server URLs.

Using the previous example, the malware would connect to the following URLs on 27 September 2012:

- http://rdeld2irxloa.org/auupdate/
- http://rdeld2irxloa.com/auupdate/
- http://rdeld2irxloa.co.uk/auupdate/
- http://rdeld2irxloa.cn/auupdate/
- http://rdeld2irxloa.in/auupdate/

FLASHBACK VM

If a valid RSA signature is returned, the malware executes the Flashback VM program if it has not been previously executed on the system. To track this, Flashback logs the SHA1s of executed programs to a file specified in configuration data entry ID 3982392222, which in my reference samples was either '{HOME}/Library/Logs/swlog' or '{HOME}/Library/Logs/vmLog'.

The Flashback VM program is RC4 encrypted configuration data similar to Flashback's configuration data. The RC4 key is

specified in entry ID 445920421. Decrypted, it looks something like this:

- [0,{3356105434:1:38},{432209166:1:40},{248949197:1:2},{18446744073691529216:4:%Base64_encoded_instruction1%},{18446744073691529217:4:%Base64_encoded_instruction2%}]

The Flashback VM program contains the malware's minimum and maximum supported version and a series of instruction entries.

Entry ID	Description
3356105434	minimum malware version supported (must be <= entry ID 2279540384)
432209166	maximum malware version supported (must be >= entry ID 2279540384)
248949197	number of instructions (n)
18446744073691529216 (up to 18446744073691529216 + n – 1)	instructions

Unlike the configuration data, the program's instruction entries start with the opcode instead of the static value '0', followed by entries specifying the parameters for the instruction:

- [%opcode%,{%parameter1%},{%parameter2%},…]

The following is a summary of the Flashback VM instruction set:

Opcode	Action	Parameters
10	Drop file(s)	{3326721549:3:%filename_format%} {1327789618:2:%overwrite_existing%} {3155285172:3:%content%} {1843261979:1:%permission%} {1096798383:2:%expand_variables%}
11	Change file permission	{3326721549:3:%filename%} {1843261979:1:%permission%}
12	Update file	{3326721549:3:%filename%} {3448248571:3:%strings_to_remove%} {3736136079:3:%strings_to_add%}
13	Does not do anything	{1610807374:3:%value%} {3155285172:3:%value%}
14	Secure delete file(s)	{3326721549:3:%filename_format%}
15	Make directory(s)	{3143747264:3:%directory_name_format%} {1843261979:1:%permission%}
16	Execute shell command	{513266406:3:%command%} {1096798383:2:%expand_variables%}
17	Search and replace file content	{3326721549:3:%filename%} {2410934700:3:%string_to_search%} {2142339923:3:%search_padding%} {289096310:3:%string_to_replace%} {1096798383:2:%expand_variables%}
18	Terminate host process	N/A
19	Execute shell command if a process is found	{2410934700:3:%process_name%} {513266406:3:%command%}
20*	Add payload C&C server(s)	{3013044947:3:%list_of_servers%}

**new variants only.*

Though the Flashback VM may be used for other purposes, it is most likely intended for updating the malware because:

- Programs are executed in a scheduled manner and each program is only executed once
- Malware information (e.g. version, hash) is collected by C&C before a program is issued
- The resource path name '/auupdate/' suggests it is meant for automatic updates.

PAYLOAD C&C SERVERS

The second group of C&C servers is used by Flashback to execute its payload. Each time an infected browser is launched, the malware selects a server in a process similar to the way it contacts its update server.

Old variants use configuration data entry IDs 2413278617 and 2718965927 for the server list and resource path respectively when forming the URL of the server. Taking the following entries as an example:

Entry ID	Sample entry data
2718965927	/owncheck/
2413278617	oldserver1.com|oldserver2.com

The malware connects to the following URLs:

- http://oldserver1.com/owncheck/
- http://oldserver2.com/owncheck/

New variants use two different server lists plus server names generated based on the current date using the same algorithm discussed in the 'Update servers' section:

Entry ID	Description
2718965927	Resource path
3035856777	Updateable list
2522550406	Hard-coded list
3078701270	TLD list
1131202474	Sub-configuration data used for generating server name based on current date

Taking the following entries as an example:

Entry ID	Sample entry data	
2718965927	/owncheck/	
3035856777	priorityserver.com	
2522550406	payloadserver1.com|payloadserver2.com	
3078701270	.orgl.coml.co.ukl.cnl.in	
1131202474	**Subentry ID**	**Sample entry data**
	8	d2ir
	12	xloa
	27	rdel

The malware connects to the following URLs on 27 September 2012 (in the presented order):

- http://priorityserver.com/owncheck/
- http://payloadserver1.com/owncheck/
- http://payloadserver2.com/owncheck/
- http://rdeld2irxloa.org/owncheck/
- http://rdeld2irxloa.com/owncheck/
- http://rdeld2irxloa.co.uk/owncheck/
- http://rdeld2irxloa.cn/owncheck/
- http://rdeld2irxloa.in/owncheck/

Much as in the verification check done for the update servers, the response from Flashback's payload servers is expected to contain two parts, though this time the encoded data are delimited by 'l'.

Part 1 contains the Base64 encoded SHA1 of the server name and part 2 contains the Base64 encoded RSA signature of part 1. The RSA signature again uses MD5 hashing and is verified using a static public key found in the main binary. Flashback selects the first server that returns a valid response.

If none of the servers returns a valid response, a default server is selected. Old variants use the second server found in entry ID 2413278617. New variants use the first server found in entry ID 2522550406, or 'localhost' if the entry does not exist. The selected server is refreshed every 3,670 seconds.

PAYLOAD

Before executing the payload, Flashback first confirms it is loaded by a correct process, to ensure it doesn't cause an application crash that may alert the user to its presence. It also rechecks if any security applications or analysis tools have been installed since infection took place, using the following configuration data entries:

Entry ID	Description
3604130400	List of *Safari* processes
2368804422	List of *Firefox* processes
36406636	Total blacklisted application paths (n)
3740205056 (up to 3740205056 + n - 1)	Blacklisted application path

Flashback's payload involves modifying the content of targeted web pages displayed by the affected browsers. In practical terms, Flashback redirects visitors from *Google*-related web pages to third-party advertisers.

To do this, the malware creates interpose functions to hijack APIs used by the browser to handle web traffic. Though the specific details of the modifications have evolved significantly during Flashback's development, the end result remains the same.

OLD PAYLOAD

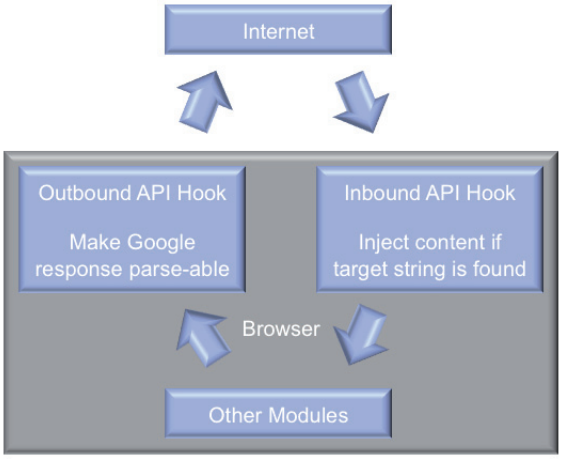

Figure 12: Payload in old variants.

Early Flashback variants hooked the CFWriteStreamWrite() and send() APIs to intercept outbound traffic from the infected browser and modify HTTP requests for *Google* before sending. The modifications simply ask *Google* to reply in a format that is parseable by the malware.

Flashback then hooks the CFReadStreamRead() and recv()APIs to intercept inbound traffic and check if the HTTP response is from a targeted web page by checking if it contains strings specified in configuration data entry IDs 986378308 and 3539672383. In my reference sample, these data entries are 'google.time|Google' and '/title>' respectively, which means the malware checks if the response is from *Google*.

If the response is from a targeted web page, Flashback injects the content specified in entry ID 2260730474 before handing the response over to the browser. In addition, any '{DOMAIN}' strings found in the injected content are replaced with the selected payload C&C server, while any '{BSRQ}' strings are replaced with a Base64 encoded string containing the injected content version (specified in entry ID 2588545561) and the infected host's PlatformUUID.

In my reference sample, the injected content is a JavaScript that will fetch and execute another JavaScript from the payload C&C server.

There are a lot of possibilities that can be derived from the different combinations of the configuration data and the remote JavaScript. One combination would be an entry ID 986378308 containing a bank's name and a JavaScript that renders a phishing HTML form. This possibility may be the basis for reports that Flashback is designed to steal passwords [7]. Though possible, this usage appears to be less likely, as a sample targeting non-*Google* web pages has yet to be seen. Changes in the payload of new variants also serve as further proof that the malware is only interested in the search giant.

NEW PAYLOAD

In new variants, the payload routines have been modified to such an extent that they show little resemblance to earlier variants. The targeted web page and injected content are no longer controlled by the configuration data. They have been changed to static routines specifically designed for *Google*. Duplicate hooks have also been removed and only CFWriteStreamWrite() and CFReadStreamRead() are hooked.

In the new payload, most of the processes have been moved to the infected host, with the payload C&C servers becoming more 'interactive' rather than simply hosting JavaScript. This may be why new variants use the list of servers in configuration data entry IDs 3035856777 and 2522550406 rather than 2413278617.

The payload starts with the CFWriteStreamWrite() hook waiting for a request containing 'google.', and 'GET /search?' followed by 'q=' (i.e. a *Google* search).

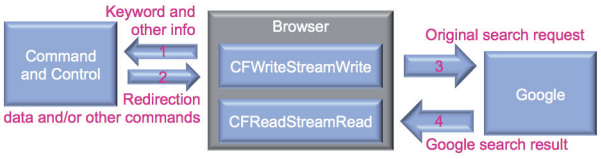

Figure 13: Hooks and Google search.

The value of the 'q=' parameter (the search keyword) is parsed out and Flashback sends this and other details to the payload server:

- http://%payload_C&C_server%/search?q=%Base64_encoded_keyword%&ua=%Base64_encoded_browser_useragent%&al=%Base64_encoded_browser_language%&cv=%Base64_malware_version%

The response from the server will be as follows:

- Layer 1 – Base64 encoded
- Layer 2 – RC4 encrypted with the MD5 hash of the infected system's PlatformUUID as the key
- Layer 3 – list of commands delimited by 'l' where each item is Base64 encoded.

The decoded commands can be any of the following:

Command	Action
BIDOK|%destination_url%|%tracking_value%|%referrer%	Store or update redirection data of keyword in cache.
BIDFAIL	Does not do anything.
H_SETUP|%value1%|%value2%|%value3%|%value4%	Update redirect values related to %tracking_value% returned by a 'BIDOK' command, which determine when redirection should occur. Initial values are loaded from entry IDs 2651194085, 2053826074, 131556884 and 1948502003 of the configuration data.
ADD_S|%server%	Add a payload C&C server.
MU|%Base64_encoded_Flashback_VM_program%	Run a Flashback VM program.
SK|%unused_value%	Uninstall Flashback. Basically just run the commands specified in entry IDs 1987052121, 344144970, 1759733748, and 2264415946 of the configuration data then securely delete the main binary.

The rest of the payload is dependent on the 'BIDOK' and 'H_SETUP' commands returned by the server. The malware performs the returned commands and then sends the original request to *Google* to get the actual results.

Now the CFWriteStreamWrite() hook waits for the user to click a link in the *Google* result by monitoring for a request containing the following:

- 'GET /url?', 'q=', and 'google'
- Followed by one of: 'sa=', 'ved=', or 'usg='

Once a link in the *Google* result is clicked, Flashback parses out the value of the 'q=' and 'url=' parameters, which are the keyword and original destination of the *Google* result respectively, and uses them to check (1) whether there is corresponding redirection data for the keyword and (2) if the

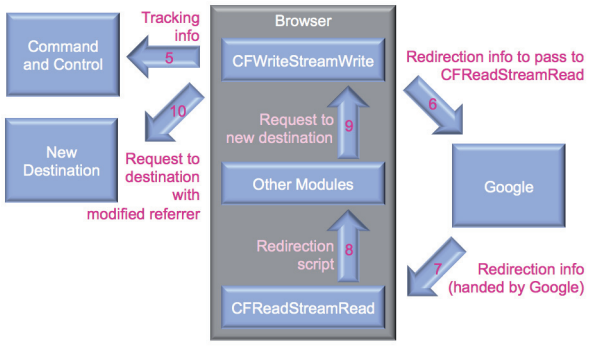

Figure 14: Hooks and Google result click.

original destination is not whitelisted (determined by configuration data entry ID 4072365446).

If neither condition is satisfied, Flashback does nothing; otherwise it checks the corresponding tracking value (see 'BIDOK' command) of the keyword in the cache against the redirect values (see 'H_SETUP' command) to determine whether to proceed with the payload. Should the payload proceed, Flashback reports to the following URL:

- http://%payload_C&C_server%/click?data=%Base64_encoded_data%

The decoded data contains the following information:

- %keyword%|%tracking_value%|%original_destination%

The tracking value is the price of each visit to the new destination URL [8] and is most likely reported to help Flashback's authors track how much they should be earning.

At the same time, the malware creates a specially crafted request for *Google*:

- http://google.com/%marker1%%Base64_encoded_data%%marker1%

The CFWriteStreamWrite() hook uses this request to communicate with the CFReadStreamRead() hook. Flashback expects *Google*'s response to contain the request it sends.

The CFReadStreamRead() hook constantly checks if an HTTP response contains the strings 'google.com' and %marker1%, the latter of which the malware generates randomly when the browser is launched. If these strings are not found, the hook hands over the original HTTP response to the browser. If found,

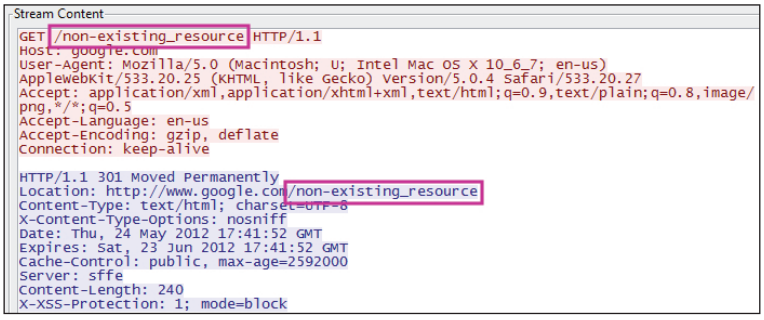

Figure 15: Google response containing the request.

Flashback uses the marker to locate the encoded data sent by the CFWriteStreamWrite() hook. The data would be as follows, with sample values:

Decoded data	%keyword%|%new_destination%&%marker2%%ID%&%marker2%
Sample data	viagra|http://www.destination.com/page?p=v&VXAuoc5lKM8&VXAu

Where 'VXAu' is %marker2% and 'oc5lKM8' is the %ID%.

Flashback then returns the following content to the browser:

```
HTTP/1.0 200 OK
Content-Type: text/html;
Content-Length: %content_length%

<script>window.googleJavaScriptRedirect=1</script>
<script>
   var a=parent,b=parent.google,c=location;
   if(a!=window&&b)
   {
     if(b.r)
     {
       b.r=0;
         a.location.href="%url%";
         c.replace("about:blank");
     }
   }
   else
   {
     c.replace("%url%");
   };
</script>
<noscript><META http-equiv="refresh" content="0;URL='%url%'"></noscript>
```

Where:

- %content_length% is replaced with the actual length
- %url% is replaced with %new_destination%&%marker2%%ID%&%marker2%, for example: 'http://www.destination.com/page?p=v&VXAuoc5lKM8&VXAu'.

The end result is that the browser will be redirected to the new destination. The HTTP request to the new destination will, however, contain a referrer value from *Google*, as the browser still thinks the response returned by the CFReadStreamRead() hook is from *Google*. To make it look more legitimate, Flashback replaces the referrer with the corresponding value returned by the 'BIDOK' command. This happens when the browser's HTTP request to the new destination is caught by the CFWriteStreamWrite() hook, the malware uses %marker2% to parse out the %ID% in the request and uses it to locate the corresponding referrer value returned by the 'BIDOK' command in the cache.

FILTER/LOADER BINARY

We mentioned earlier that some Flashback variants include a filter/loader binary. The only purpose of this binary is to ensure that the main binary is

loaded only by the correct processes. Though the main binary already includes the necessary checks, Flashback thoughtfully includes redundancy. After all, it is always better to be careful, right?

Actually, this time it appears this malware has been too careful for its own good. In the samples I encountered, it appears the authors made a typo in the routine checking if the host process is 'WebProcess'.

```
cmp     al, 57h ; 'W'
jnz     short righprocess_check
cmp     byte ptr [rbx+1], 65h ; 'e'
jnz     short righprocess_check
cmp     byte ptr [rbx+2], 62h ; 'b'
jnz     short righprocess_check
cmp     byte ptr [rbx+3], 50h ; 'P'
jnz     short righprocess_check
cmp     byte ptr [rbx+4], 6Fh ; 'o'
jnz     short righprocess_check
mov     cs:_rightProcess, 0FFDAFEh
jmp     short loc_BBB
```

Figure 16: Typo in 'WebProcess' process check.

This resulted in the malware only being loaded by the 'Safari' process and not by the 'WebProcess' process, which is the one actually responsible for handling the page load [9]. Because of this, some Flashback variants may not be able to execute their payload. This is good news for Mac users. Even in these cases however, the Flashback VM is still able to communicate with the update server.

LAUNCHAGENT BINARY

In the latest Flashback variant, the installer is replaced with a binary that functions like a stand-alone light version of the Flashback VM. Initially, the binary was thought to be just an updater when it was distributed together with the installer [10].

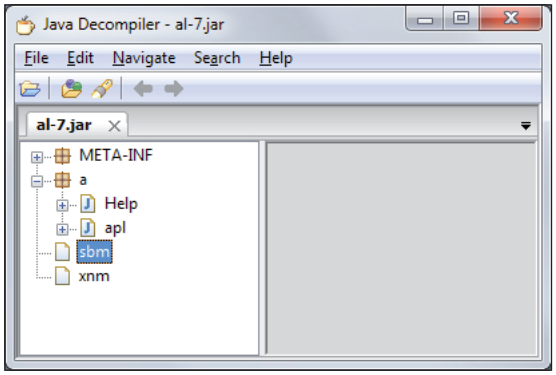

Figure 17: .jar containing both installer and LaunchAgent binary.

However, the binary has taken complete responsibility for installing Flashback in the latest variant [11]. This looks similar to a trend happening in *Windows* malware where new variants start with something small instead of deploying the full payload immediately. This may be done to avoid AV signatures, as it is easier to morph a simple downloader than a full-blown malware.

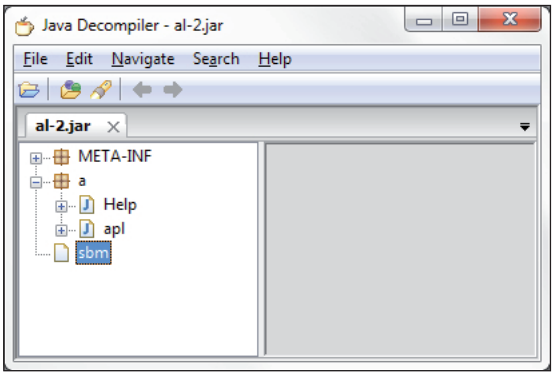

Figure 18: .jar containing LaunchAgent binary only.

The binary is installed via drive-by downloads by malicious Java applets hosted on Flashback distribution websites and is dropped to the user's home folder. The launch point is a property list file created in the '~/Library/LaunchAgents/' folder, which will launch the binary every 4,212 seconds. The actual filenames given by the distribution sites may vary between cases.

The behaviour of the binary is mostly defined by a configuration block found in its body. The block is a series of entries that have the structure shown in Figure 19.

Upon execution, the binary goes through a list of paths that belong to security software and analysis tools (entry ID 0x92FA). As with the original installer, the LaunchAgent binary has a bug where it didn't check the existence of the paths during enumeration. This resulted in reports of suspicious network activity warnings from *Little Snitch* filling up discussions in *Apple Support Communities* just moments after its release [12]. This ultimately led to mass discovery of Flashback and security researchers finally obtaining samples of the malware.

On first execution, the LaunchAgent binary reports back to the following URL:

- http://%distribution_website%/stat_svc/

It then locks itself to the infected system by encrypting the configuration block in its body using the RC4 algorithm, with the infected system's PlatformUUID as the key.

On successive executions, the binary contacts a server, which belongs to the last group of Flashback C&C servers. For this group of servers, the binary generates a name using a custom algorithm based on the current day, month and year. Then it generates five more names using the same algorithm but this time using three constants found in its configuration block (entry ID 0x6192).

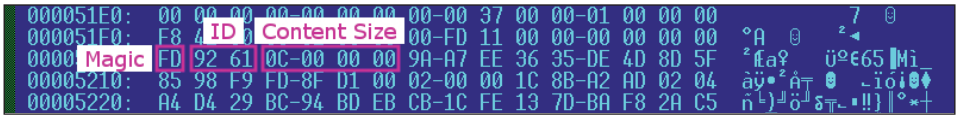

Figure 19: LaunchAgent binary configuration block format.

The same algorithm is able to generate five different names because the algorithm involves modifying the constants in the process. Therefore, this is like running the algorithm with a new set of constants on each run. However, the final generated list of names will always be the same since they started with the same set of constants. For details of the actual algorithm, *Scrammed!* blog has a very good C language representation [13].

The values of the constants differ between variants, making the generated names variant-specific, while the name generated using the current date values is common to all variants. Each name is appended with the TLDs (entry ID 0x1F91) '.kz', '.in', '.info', '.net' and '.com' to form the final list of servers. In addition to the generated list of servers, the binary also keeps a pre-defined list of servers in its configuration block (entry ID 0x92BE), which differs between variants.

When selecting a server, the binary starts with the variant-specific list, then the date-specific list, and finally the pre-defined list. The binary runs the installation/update program returned by the first server that has a valid response, and then exits; the same routine is repeated whenever the binary is executed by launchd again.

The server verification process is similar to those from previous C&C groups. The binary expects the response to contain information in two parts: part 1 contains the Base64 encoded installation/update program and part 2 contains the corresponding Base64 encoded RSA signature. The binary uses markers (entry ID 0x4280) to parse out the relevant information from the response, then verifies the RSA signature using a public key found in the configuration block (entry IDs 0x0FA7 and 0xD18F). Unlike the verification process used in previous C&C groups, the RSA signature uses the SHA1 hashing method.

The installation/update program is actually just another configuration block similar to the one found in the body of the binary. Here, the entry IDs determine the action to be taken with the entry content. The binary will execute entries with IDs 0x2020 to 0x2028 and 0x1010 to 0x1019.

Entry ID	Action
0x2020 - 0x2028	Drop the entry content as '/tmp/.%random%' then execute it using 'nohup' (entry ID 0x4280 of the configuration block in the body of the binary)
0x1010 - 0x1019	Execute the shell command specified in the entry content

To ensure previous installations/update programs are not executed again, the binary creates the file '/tmp/.%CRC32_of_program%' containing the CRC32 of the program. If the file already exists, the binary skips the rest of its routine and exits. It is not clear why the '/tmp/' folder is used. Unlike the main binary's Flashback VM, it seems that the authors do not mind the same program being executed again when the folder is cleared. Perhaps the authors just wanted to reduce the load of the C&C servers. This approach also allows the malware to reinfect the system if only the main binary is removed during disinfection.

CONCLUSION

While malware on the *OS X* operating system is not yet as sophisticated as that found on *Windows*, it is catching up quickly and Flashback isn't far behind. It has advanced features such as infected host locking and domain name generation, which are established techniques in *Windows* malware. Malware authors appear simply to be migrating the toolset they developed over the years from one platform to the other. Some aspects don't even require much adjustment, as the Java exploits used in Flashback's distribution scheme are cross-platform and work out of the box.

Though Flashback may be the only real outbreak we've seen so far on *OS X*, it is unsettling to consider that its success may inspire other malware authors to start migrating to *OS X* as well. All they need is a period to adjust.

APPENDIX

MD5 of reference samples:

- FC25B95BB01EA1049D64358198D8B585 – installer binary
- 434C675B67AB088C87C27C7B0BC8ECC2 – main binary (old)
- A455A86A888E4D2FB1F94BBCF6B6C850 – main binary (new)
- 42A1D6C009EB3F003ACA344E821AFDD1 – filter/loader binary
- 919146E4D79A3C38F3D600355FED2120 – LaunchAgent binary

REFERENCES

[1] Dr.Web blog. http://news.drweb.com/show/?i=2353&lng=en&c=9.

[2] Hyppönen, M. http://twitter.com/mikko/statuses/187886540486230017.

[3] F-Secure blog. http://www.f-secure.com/weblog/archives/00002206.html.

[4] Intego blog. http://www.intego.com/mac-security-blog/new-flashback-trojan-horse-variant-uses-novel-delivery-method-to-infect-macs/.

[5] Sorokin, I. http://twitter.com/hexminer/status/186807718596706306.

[6] Intego blog. http://www.intego.com/mac-security-blog/flashback-mac-trojan-horse-infections-increasing-with-new-variant/.

[7] Computerworld. http://www.computerworld.com/s/article/9224651/New_Mac_malware_exploits_Java_bugs_steals_passwords.

[8] Symantec blog. http://www.symantec.com/connect/blogs/osxflashbackk-motivation-behind-malware.

[9] Apple Support Communities. https://discussions.apple.com/thread/3216267?start=0&tstart=0.

[10] F-Secure blog. https://www.f-secure.com/weblog/archives/00002341.html.

[11] Symantec blog. http://www.symantec.com/connect/blogs/osxflashbackk-overview-and-its-inner-workings.

[12] Apple Support Communities. https://discussions.apple.com/thread/3844172?start=0&tstart=0.

[13] Scrammed! blog. http://scrammed.blogspot.it/2012/04/flashback-trojan-domain-generator.html.