Banking Trojans Under Development

By deugenio

Published: 2018-06-06 · Archived: 2026-04-13 02:11:17 UTC

Although banks themselves have taken measures to strengthen the security of their authentication processes, Banker Trojans, however, are still a popular tool for stealing user's financial details and draining bank accounts.

The Check Point Research team recently came across one such banking Trojan under development and already being distributed through the RIG Exploit Kit. Dubbed 'Karius', the Trojan aims to carry out web injects to add additional fields into a bank's legitimate login page and send the inputted information to the attacker.

While Karius is not yet in full infection mode, initial tests have already been made and our research below shows the evolution of how such malware takes place. Our analysis also shows how banking trojans such as Karius are put together and makes use of code from other well-known bankers such as Ramnit, Vawtrak and TrickBot.

Malware Analysis

Karius works in a rather traditional fashion to other banking malware and consists of three components:

- injector32\64.exe
- proxy32\64.dll
- mod32\64.dll

These components essentially work together to deploy webinjects in several browsers, thus intercepting the user's communication to the internet.

Normally, the content sought in the intercepted traffic would be a specific banking website that the victim has visited. This site would then be modified so as to trick the user to enter his credentials which are then sent to the attacker. However, it seems that this malware is still in a development and possibly in testing stages, which is why the webinjects don't yet target any financial institutions in particular.

The initial component that will be executed is **injector32.exe** (or injector64.exe, its equivalent for 64 bit systems). This component is responsible for several actions, namely:

- Modifying several registry keys, for the purpose of disabling protection mechanisms in Internet Explorer.These include:

1. HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\2500 = DWORD:3
2. HKCU \Software\Microsoft\Internet Explorer\Main\TabProcGrowth = DWORD:0
3. HKCU \Software\Microsoft\Internet Explorer\Main\NoProtectedModeBanner = DWORD:1

- Copying itself to the startup directory with the name TEMP APPLICATION.lnk as a means of persistence, as well as creating a duplicate under the name %USERPROFILE%/music/temp.exe.

- Writing a batch script to disk which will delete the original file.
- Contacting the Command and Control (C&C) server for registering the bot using a message with the following json structure:

{“id”: “<computer_name_hash><digital_product_id_hash>”

“action”: “register”

“os”: “<os_version>”

“lang”: “<lang>”

“integrity”: “None”

“admin”: “admin”

“group”: “group”

}

After contacting the C&C server, the malware would then send a message with a similar id and a “ping” action, which is then responded to by the server with one of two commands –

Update – sends a binary with a new version of the malware.

Config – issues a webinjects configuration payload, saved to a file named “temp.bin”.

All the traffic to the C&C server will then be encrypted with RC4 using a key hardcoded in a configuration struct that resides within the .cfg section of this module’s binary.

cfg section:

Figure 1: *The .cfg section of the configuration module's library.*

– Creating the named pipe "\\.\pipe\form" in order to communicate with other modules, which in turn use it to forward stolen credentials to it. This data will then be sent to the C&C periodically (upon an interval predefined in the internal configuration section), using the following message format:

{"id": "<computer_name_hash><digital_product_id_hash>"

"action": "report"

"url": "<url_of_stolen_credentials>"

"browser": "UNK"

"Data": "<data_of_stolen_credentials>"

}

– Decompressing the subsequent components (proxy32.dll and proxy64.dll) using aPLib and injecting the one corresponding to the system's architecture to the "explorer.exe" process.

The dll component injected to "explorer.exe" is used to hook the CreateProcessInternalW API function, which is invoked upon initiation of processes under Explorer. The hook function, in this case, will look for one of several browsers and would inject it with yet another decompressed DLL named mod32.dll (or mod64.dll for 64 bit systems). These browsers include: Internet Explorer, Chrome, Firefox and Edge.

This final DLL payload, which will execute in the context of one or more of the aforementioned browsers, is the component that will implement the webinjects mechanism by applying some hooks, as done by many other banking malwares. The hooked functions can be found below.

Figure 2: *Table of hooked functions per browser.*

As mentioned previously, the webinjects reside within the “temp.bin” file, and have the following format:

host: <url_of_host>

path: <path_in_url>

data:

inject: <data_to_inject>

before: <what is before the injected data>

after: <what is after the injected data>

The credentials stolen by applying the webinjects will then be sent using the named pipe to the injector component, and subsequently forwarded to the C&C.

A figure that illustrates all aforementioned actions can be seen below:

Figure 3: *Diagram of Karius’ processes.*

Similarity to Known Banking Trojans

As seen in the previous section, the malware's internal workings are quite standard for the Banking Trojans. What's more interesting, though, is the fact that major parts of the malware are heavily based on well-known bankers that can be found in the wild. For instance, the usage of aPLib compression to store both 32 and 64 bit modules one after the other in the file was used previously in Vawtrak.

Also, in one of the malware's inspected samples we were able to track hardcoded webinjects, which to our surprise were used previously by Ramnit.

Webinjects similarity:

Figure 4: *Comparison of code used in Karius and Ramnit's webinjects.*

But probably the most significant resemblance we were able to spot is to the widely distributed TrickBot banker. In particular, we saw a very high similarity in binary code between the two malwares. The functionalities of injection and hooking methods showed very similar code elements.

It might be not so surprising for the injection functionality, since both used the reflective loading injection method, which has an online open source implementation, found here.

However, it is not clear whether the inline hooking implementation was taken from any particular known source code.

Figure 5: *The Hooking_Method function.*

Figure 6: *The InternetWriteFile_Hook_Function.*

It’s interesting to note that in one of the similar hooking functions observed in Karius, we could spot a call to the OutputDebugString function, which contains a string that hints on the function’s name. This may suggest that the author of Karius was in possession of the source code used to implement the hooking mechanism of TrickBot, and added this line for debugging purposes. It also hints that this malware is still undergoing some development and that more versions of it are yet to be released.

HttpSendRequestA_Hook_Function

Figure 7: *The HttpSendRequestA_Hook function.*

An Evolving Malware

Throughout the research of this malware, we observed two main versions which outline a certain evolution in its development. The first version seemed to be very buggy, and possibly used only for testing purposes, while the second, while more functional, is still not fully complete. This can be seen in the presence of various "placeholder" strings such as the one below:

Figure 8: *An incomplete "placeholder" string.*

The major changes between the two versions are outlined in the following table:

Figure 9: *Table of development changes between the two versions of the malware.*

IOCs:

Md5 hashes:

728911a915d9ec3b6defa430d24bc0d5

857430b8c9dc78ce4eabbe57cb3ae134

Domains:

http://proxyservice.site/updates/gateway.php
