Documentation for browserhooks

Introduction

Browserhooks are an extension of original apihooks plugin focused on detection of sophisticated banking malware intrusion in prevalent web browsers.. As an addition we implemented support for hook detection in 32-bit modules of WOW64 processes, which was not supported by the original apihooks plugin. Which we thought would be a great addition and greatly improve the possible hook detection on compromised systems.

Installation & Execution

Download the plugin from https://github.com/eset/volatility-browserhooks and then copy into %volatility%/plugins/malware, where %volatility% is the installation directory of the framework.

Example of the command with -D switch that stores the hooking modules in the specified directory:

```
vol.py -f “c:\data\win7.vmem” --profile Win7SP1x86 browserhooks -D
_store_mods
```

Use Cases

Almost every banking Trojan identifies a browser based on its process name, therefore we restrict the detection for these 4 processes: *chrome.exe*, *firefox.exe*, *iexplore.exe*, *microsoftedgecp.exe*. The attack support for other projects are rare (just past versions of Opera occasionally - not considered). Generally, research related to hooking techniques of contemporary banking Trojans was published in [1].

For Chromium-based projects, the crucial part in detecting hooks on the attack points is locating the virtual method table with SSL related functions, shortly denoted SSL VMT. Basically, we identified three new types of hooking:

1) Replacement of a function in SSL VMT (Win{32;64}/Spy.Ursnif-based bankers; Win{32;64}/Qadars, Win{32;64}/Trickbot, Win{32;64}/Zbot-based bankers)
2) Inline hook in SSL VMT (Win{32;64}/Dridex, Win{32;64}/Tinukebot)
3) Inline hook in a wrapper of a function in SSL VMT or another unexported function that could be misused in the same way (Win{32;64}/Qbot)

The following table sketches an overview of the patterns that help to achieve that. The string “00034300” is a shortcut for the hex string “00 00 00 03 04 03 00 00” which is a strong pattern identifying the position of the SSL VMT table in Chrome. The patterns were not optimised to catch as many release as possible, but to uniformly identify the beginning of the table. Note that the pattern changed even between the minor releases.

Chrome version	Release Date	32-bit	64-bit
61.0.3163.100 61.0.3163.91 61.0.3163.79	September 21, 2017 September 5, 2017	More of them (cf. the code)	More of them (cf. the code)
60.0.3112.113 60.0.3112.101 60.0.3112.90 60.0.3112.78	August 24, 2017 July 25, 2017	“00034300”	“00034300”
59.0.3071.115 59.0.3071.109 59.0.3071.104 59.0.3071.86	June 26, 2017 June 5, 2017	“00034300”	“00034300”
58.0.3029.110 58.0.3029.96 58.0.3029.81	May 9, 2017 April 19, 2017	“00034300”	“00034300”
57.0.2987.133 57.0.2987.110 57.0.2987.98	March 29, 2017 March 9, 2017	“00034300”	“00034300”
56.0.2924.87 56.0.2924.76	February 1, 2017 January 25, 2017	“00034300”	“00034300”
55.0.2883.87 55.0.2883.75	December 9, 2016 December 1, 2016	“00034300”	“00034300”
54.0.2840.87 54.0.2840.71 54.0.2840.59	November 1, 2016 October 12, 2016	“00034300”	“00034300”
53.0.2785.143 53.0.2785.116 53.0.2785.101	September 29, 2016 August 31, 2016	More of them (cf. the code)	More of them (cf. the code)

53.0.2785.89			
52.0.2743.116 52.0.2743.82	August 3, 2016 July 20, 2016	More of them (cf. the code)	More of them (cf. the code)
51.0.2704.106 51.0.2704.103 51.0.2704.84 51.0.2704.79 51.0.2704.63	June 23, 2016 May 25, 2016	More of them (cf. the code)	More of them (cf. the code)

We also tested the custom hooking methods of Win{32;64}/Qbot for the recent builds (v321.28) caught in-the-wild (`89E910796279F75B86399724CEAE5841FA7E34C1` (32-bit, PETS: (4.9.2017 17:47:31), `7EE4545B92BA0484C0D27FC74374CB772BDE64EB` (64-bit, PETS: 4.9.2017 17:48:12)).

	Win32/Qbot	Win64/Qbot
61.0.3163.100	-	-
60.0.3112.78	OK	OK
59.0.3071.115	-	-
58.0.3029.96	OK	-
57.0.2987.133	OK	OK
56.0.2924.87	OK	Ok
55.0.2883.75	OK	OK
54.0.2840.87	- (ssl_read only)	-
53.0.2785.116	- (ssl_write only)	OK
52.0.2743.116	OK	-
51.0.2704.84	OK	-

During the testing phase, we found some false alarms of hooks in system functions, so we added several exclusions in the whitelist.

Limitations

There were several struggles we faced limiting the potential use of the plugin.

1) Absence of the SSL VMT table in the memory dump.

After starting Chrome, only the parent *chrome.exe* has *chrome.dll* mapped in its process space and this library contains the potentially hooked SSL VMT table. However, we experienced cases when *chrome.dll* was properly loaded but the part of .data section with SSL VMT was not present in the memory dump (this was cross-checked by dumping the DLL with *dlldump* and searching the pattern unsuccessfully). We concluded that the corresponding pages had been swapped to the disk and therefore not available in the dumps.

2) Diversity

There are many variants of browsers and also many versions for Chromium-based projects with variable position of SSL VMT. There are even more different variants of banking Trojans, using various approaches that evolve in time. So it is unfortunately not possible to state that the plugin is universal.

~~3) Detecting hooks in 32-bit processes running under 64-bit Windows~~

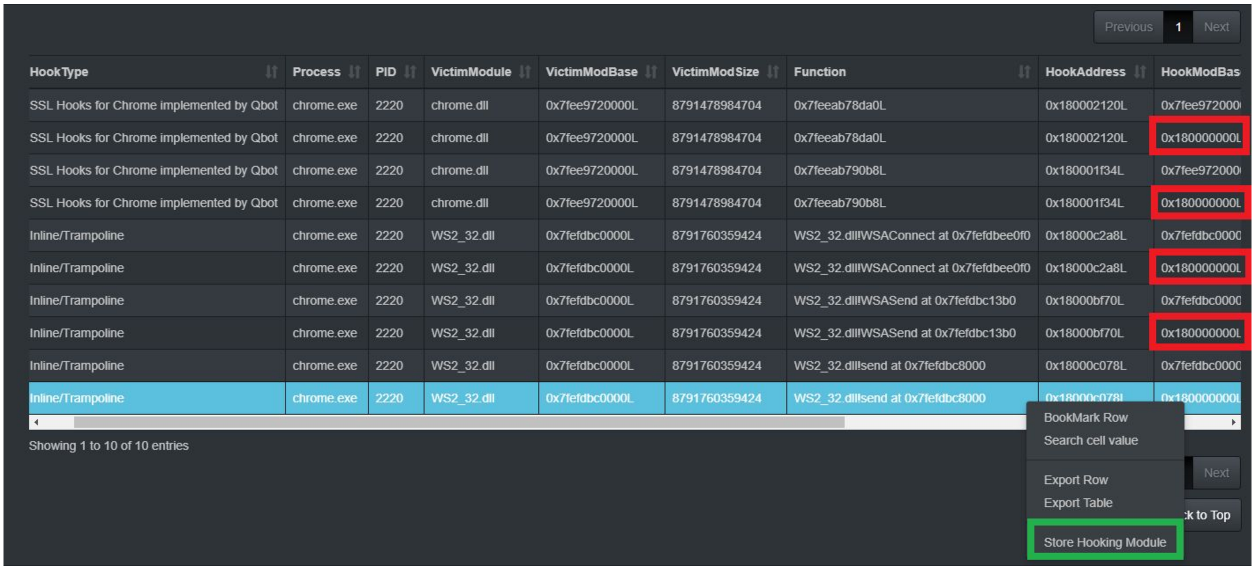
The original apihooks plugin (that browserhooks is based on) does not support wow64 modules and therefore does not detect hooks in WoW64 processes. We were able to overcome this obstacle and implemented wow64 module support - feature that might be very useful even in original apihooks plugin. While Google is now offering 64-bit Chrome for download to a visitor with 64-bit machine by default. It is still very common for 32 bit browsers (Firefox is 32-bit on all systems by default, older installation of Chrome will stay 32 bit unless manually reinstalled, ...) to run on 64 bit machines and therefore wow64 is necessary. Internet Explorer, Mozilla Firefox and Microsoft Edge works in all cases due to functions being exported. Regarding Chrome, we tested the SSL VMT lookup.

Integration with VolUtility

We prefer this GUI for VF, because it can be easily customized for our purposes.

Scenario 1

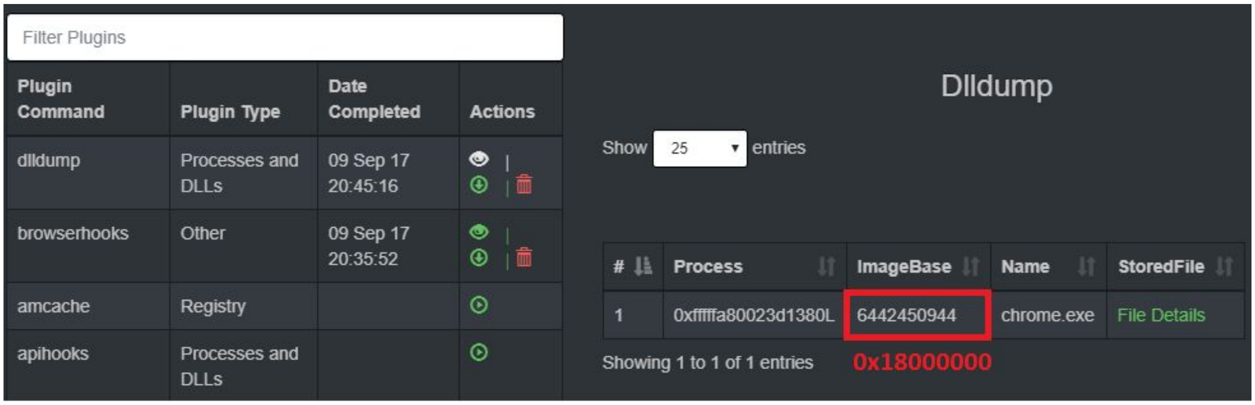
We had a Win7SP1x64 system compromised with a banking Trojan. After running browserhooks we can identify crucial hooks (custom by Win{32;64}/Qbot and some standard inline ones):



HookType	Process	PID	VictimModule	VictimModBase	VictimModSize	Function	HookAddress	HookModBas
SSL Hooks for Chrome implemented by Qbot	chrome.exe	2220	chrome.dll	0x7fee9720000L	8791478984704	0x7feeab78da0L	0x180002120L	0x7fee972000
SSL Hooks for Chrome implemented by Qbot	chrome.exe	2220	chrome.dll	0x7fee9720000L	8791478984704	0x7feeab78da0L	0x180002120L	0x180000000L
SSL Hooks for Chrome implemented by Qbot	chrome.exe	2220	chrome.dll	0x7fee9720000L	8791478984704	0x7feeab790b8L	0x180001f34L	0x7fee972000
SSL Hooks for Chrome implemented by Qbot	chrome.exe	2220	chrome.dll	0x7fee9720000L	8791478984704	0x7feeab790b8L	0x180001f34L	0x180000000L
Inline/Trampoline	chrome.exe	2220	WS2_32.dll	0x7fefdbc0000L	8791760359424	WS2_32.dll!WSAConnect at 0x7fefdbee0f0	0x18000c2a8L	0x7fefdbc0000
Inline/Trampoline	chrome.exe	2220	WS2_32.dll	0x7fefdbc0000L	8791760359424	WS2_32.dll!WSAConnect at 0x7fefdbee0f0	0x18000c2a8L	0x180000000L
Inline/Trampoline	chrome.exe	2220	WS2_32.dll	0x7fefdbc0000L	8791760359424	WS2_32.dll!WSASend at 0x7fefdbc13b0	0x18000bf70L	0x7fefdbc0000
Inline/Trampoline	chrome.exe	2220	WS2_32.dll	0x7fefdbc0000L	8791760359424	WS2_32.dll!WSASend at 0x7fefdbc13b0	0x18000bf70L	0x180000000L
Inline/Trampoline	chrome.exe	2220	WS2_32.dll	0x7fefdbc0000L	8791760359424	WS2_32.dll!send at 0x7fefdbc8000	0x18000c078L	0x7fefdbc0000
Inline/Trampoline	chrome.exe	2220	WS2_32.dll	0x7fefdbc0000L	8791760359424	WS2_32.dll!send at 0x7fefdbc8000	0x18000c078L	0x180000000L

Showing 1 to 10 of 10 entries

Some lines of the output represents the intermediate, trampoline, step. However, the lines with the base of the hooking module equal 0x1800000000 look suspicious. Note the green option in the right click menu: "Store Hooking Module". This would run ""dlldump" with the corresponding PID and `HookModBase` parameters.



Plugin Command	Plugin Type	Date Completed	Actions
dlldump	Processes and DLLs	09 Sep 17 20:45:16	
browserhooks	Other	09 Sep 17 20:35:52	
amcache	Registry		
apihooks	Processes and DLLs		

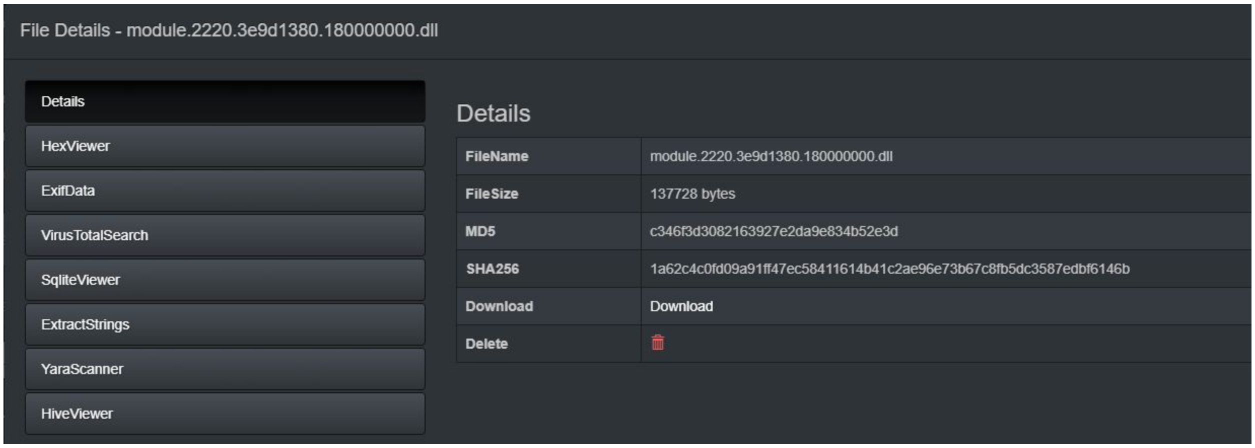
Dlldump

Show 25 entries

#	Process	ImageBase	Name	StoredFile
1	0xfffffa80023d1380L	6442450944	chrome.exe	File Details

Showing 1 to 1 of 1 entries

Now we can discovered more about the library by clicking "File Details":

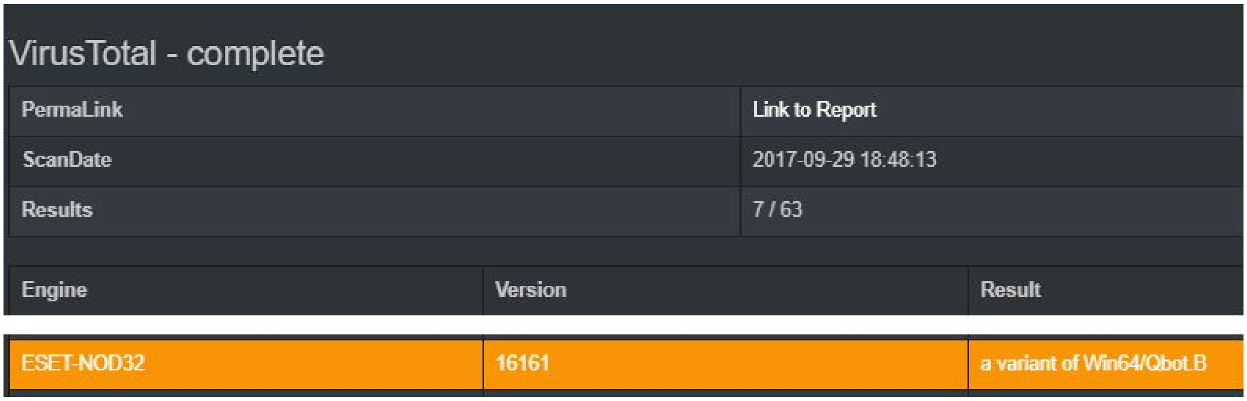


File Details - module.2220.3e9d1380.180000000.dll

Details

FileName	module.2220.3e9d1380.180000000.dll
FileSize	137728 bytes
MD5	c346f3d3082163927e2da9e834b52e3d
SHA256	1a62c4c0fd09a91ff47ec58411614b41c2ae96e73b67c8fb5dc3587edbf6146b
Download	Download
Delete	

We see the relatively small file size and the option to download the dump. But let us first peek into the Virustotal results:



VirusTotal - complete

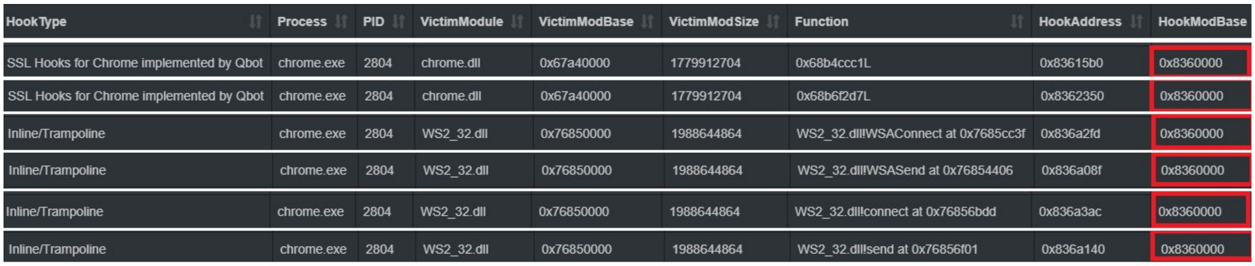
PermaLink	Link to Report
ScanDate	2017-09-29 18:48:13
Results	7 / 63

Engine	Version	Result
ESET-NOD32	16161	a variant of Win64/Qbot.B

We can see that the file is highly suspicious now (e.g. ESET detects it as a variant of Win64/Qbot.B, which might lead to more detailed description "http://www.virusradar.com/en/Win64_Qbot.B/description" if available. In a similar fashion for AV engines and their virus description websites). The case could be closed very easily as a machine infected with a recent banking Trojan.

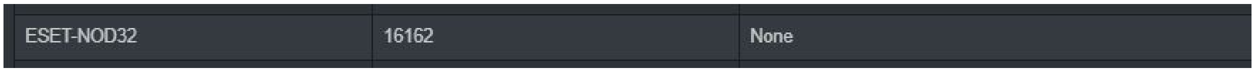
Scenario 2

The same infection by Qbot like the previous one, but now running as a Wow64 process and still detected:



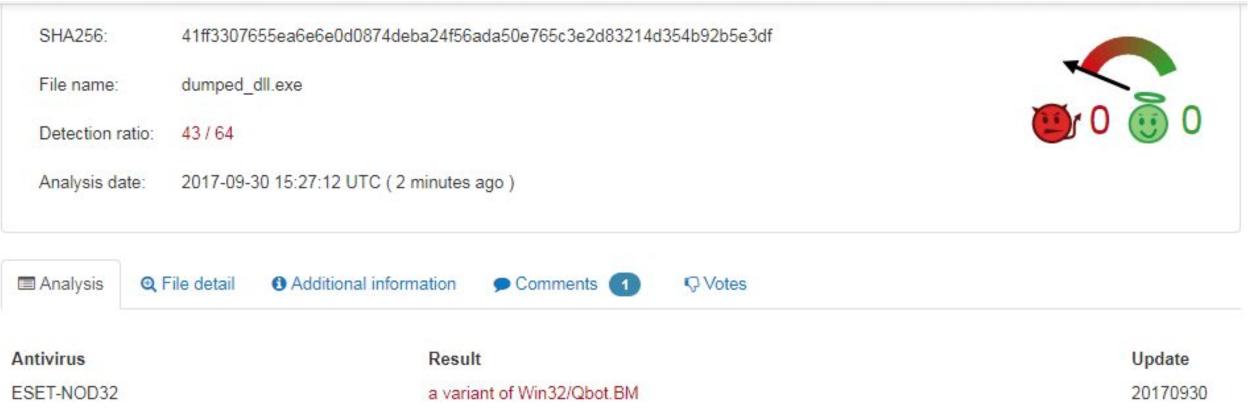
HookType	Process	PID	VictimModule	VictimModBase	VictimModSize	Function	HookAddress	HookModBase
SSL Hooks for Chrome implemented by Qbot	chrome.exe	2804	chrome.dll	0x67a40000	1779912704	0x68b4ccc1L	0x83615b0	0x8360000
SSL Hooks for Chrome implemented by Qbot	chrome.exe	2804	chrome.dll	0x67a40000	1779912704	0x68b6f2d7L	0x8362350	0x8360000
Inline/Trampoline	chrome.exe	2804	WS2_32.dll	0x76850000	1988644864	WS2_32.dll!WSAConnect at 0x7685cc3f	0x836a2fd	0x8360000
Inline/Trampoline	chrome.exe	2804	WS2_32.dll	0x76850000	1988644864	WS2_32.dll!WSASend at 0x76854406	0x836a08f	0x8360000
Inline/Trampoline	chrome.exe	2804	WS2_32.dll	0x76850000	1988644864	WS2_32.dll!connect at 0x76856bdd	0x836a3ac	0x8360000
Inline/Trampoline	chrome.exe	2804	WS2_32.dll	0x76850000	1988644864	WS2_32.dll!send at 0x76856f01	0x836a140	0x8360000

However, this time ESET does not flag the threat, because the dumped file was not reconstructed well. This holds in general and it depends on how strict AV engines are in checking the integrity of the executable. Adding yara rules for chosen banking bot families may fix this.



ESET-NOD32	16162	None

To conclude this case, one can extract strings from the module, choose those looking interesting and try an internet search. Picking "\AppData\LocalLow\", "\\.\pipe\%ssp", "data_inject", "data_after", "data_before" would lead to an online analysis of a similar file and the following VT report: https://www.reverse.it/sample/41ff3307655ea6e6e0d0874deba24f56ada50e765c3e2d83214d354b92b5e3df



SHA256: 41ff3307655ea6e6e0d0874deba24f56ada50e765c3e2d83214d354b92b5e3df

File name: dumped_dll.exe

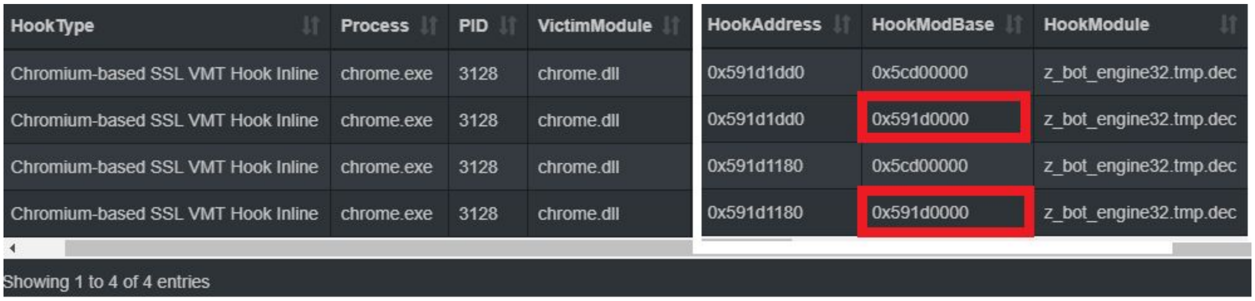
Detection ratio: 43 / 64

Analysis date: 2017-09-30 15:27:12 UTC (2 minutes ago)

Antivirus	Result	Update
ESET-NOD32	a variant of Win32/Qbot.BM	20170930

Scenario 3

In this case we discover SSL VMT inline hooks (Chrome version 55.0.2883.87):

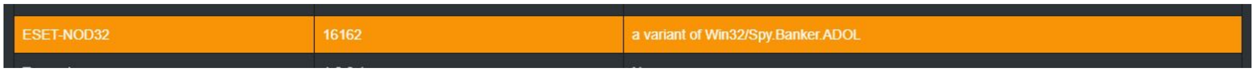


HookType	Process	PID	VictimModule	HookAddress	HookModBase	HookModule
Chromium-based SSL VMT Hook Inline	chrome.exe	3128	chrome.dll	0x591d1dd0	0x5cd00000	z_bot_engine32.tmp.dec
Chromium-based SSL VMT Hook Inline	chrome.exe	3128	chrome.dll	0x591d1dd0	0x591d0000	z_bot_engine32.tmp.dec
Chromium-based SSL VMT Hook Inline	chrome.exe	3128	chrome.dll	0x591d1180	0x5cd00000	z_bot_engine32.tmp.dec
Chromium-based SSL VMT Hook Inline	chrome.exe	3128	chrome.dll	0x591d1180	0x591d0000	z_bot_engine32.tmp.dec

Showing 1 to 4 of 4 entries

The name of the hooking module is usually unknown because the attackers load their executables almost always customly. Here, we loaded it into Chrome by calling the LoadLibrary WINAPI for the *z_bot_engine32.tmp.dec*, therefore the name is visible.

Checking the VT results show:

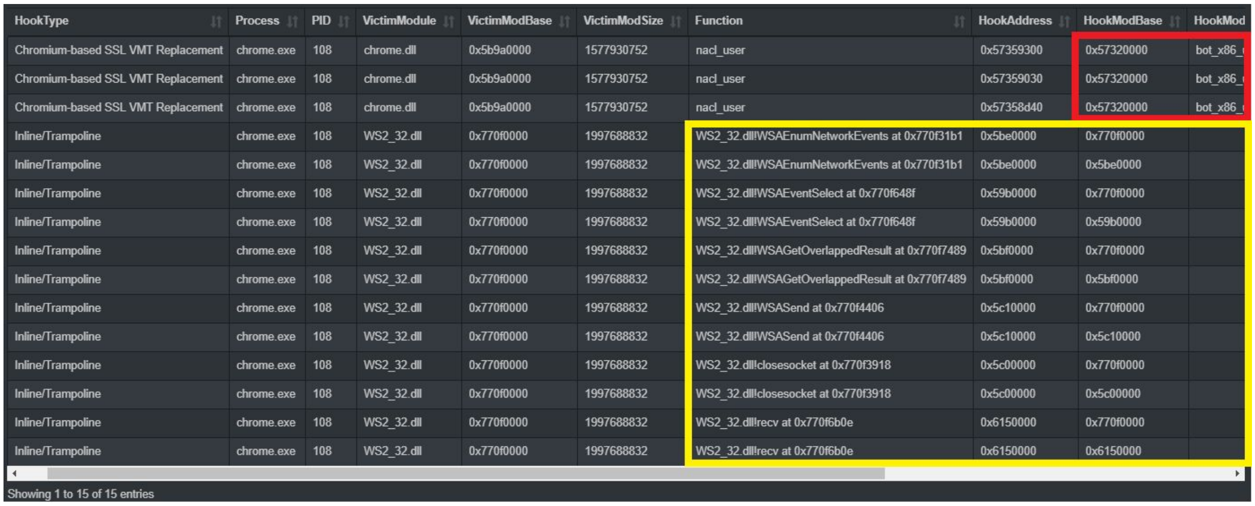


ESET-NOD32	16162	a variant of Win32/Spy.Banker.ADOL

ESET detects the dump as a variant of Win32/Spy.Banker.ADOL (which was a predecessor of Win32/Tinukebot coming from the same code base).

Scenario 4

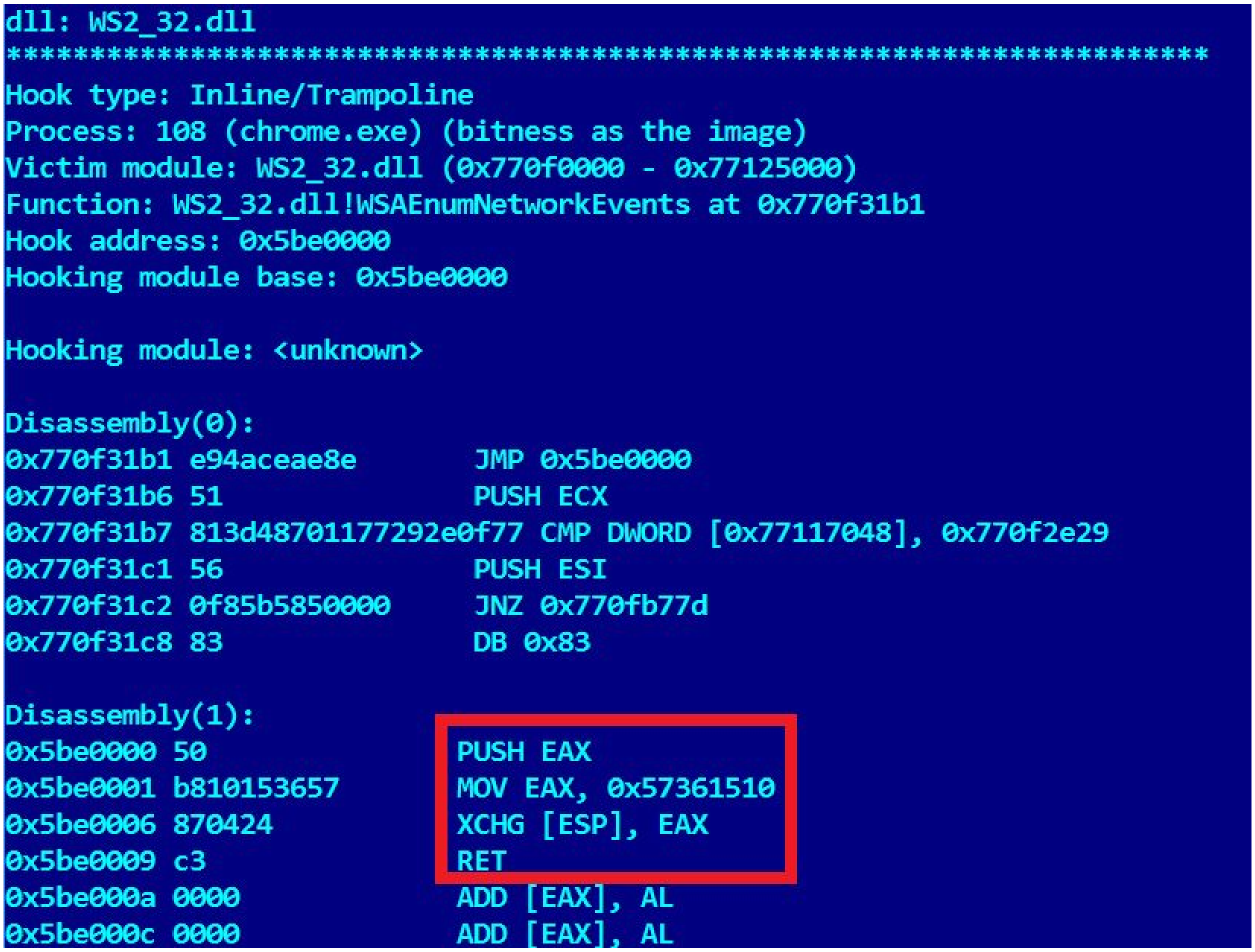
This example demonstrates SSL VMT replacements together with a more stealthy hooks of exported WINAPI attack points. After running browserhooks we get:



HookType	Process	PID	VictimModule	VictimModBase	VictimModSize	Function	HookAddress	HookModBase	HookMod
Chromium-based SSL VMT Replacement	chrome.exe	108	chrome.dll	0x5b9a0000	1577930752	nacl_user	0x57359300	0x57320000	bot_x86_
Chromium-based SSL VMT Replacement	chrome.exe	108	chrome.dll	0x5b9a0000	1577930752	nacl_user	0x57359030	0x57320000	bot_x86_
Chromium-based SSL VMT Replacement	chrome.exe	108	chrome.dll	0x5b9a0000	1577930752	nacl_user	0x57358d40	0x57320000	bot_x86_
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!WSAEnumNetworkEvents at 0x770f31b1	0x5be0000	0x770f0000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!WSAEnumNetworkEvents at 0x770f31b1	0x5be0000	0x5be0000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!WSAEventSelect at 0x770f648f	0x59b0000	0x770f0000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!WSAEventSelect at 0x770f648f	0x59b0000	0x59b0000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!WSAGetOverlappedResult at 0x770f7489	0x5bf0000	0x770f0000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!WSAGetOverlappedResult at 0x770f7489	0x5bf0000	0x5bf0000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!WSASend at 0x770f4406	0x5c10000	0x770f0000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!WSASend at 0x770f4406	0x5c10000	0x5c10000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!closesocket at 0x770f3918	0x5c00000	0x770f0000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!closesocket at 0x770f3918	0x5c00000	0x5c00000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!recv at 0x770f6b0e	0x6150000	0x770f0000	
Inline/Trampoline	chrome.exe	108	WS2_32.dll	0x770f0000	1997688832	WS2_32.dll!recv at 0x770f6b0e	0x6150000	0x6150000	

Showing 1 to 15 of 15 entries

We observe three hook functions (in red) with the hooking module name resolved (because we injected it like in the previous case via LoadLibrary). However there are more hooked functions (in yellow) without such module name. After checking the hooking data:



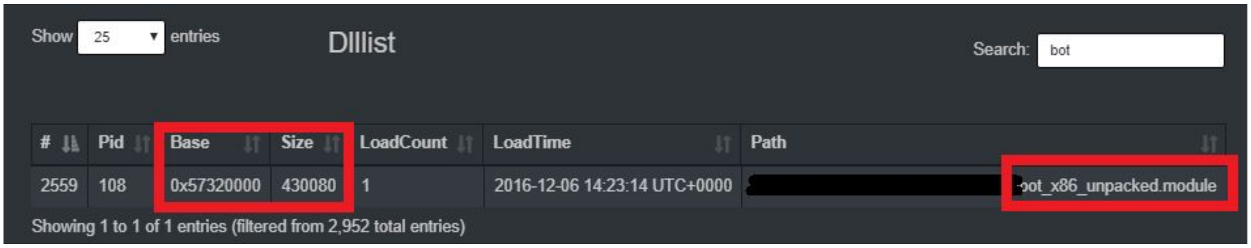
```
dll: WS2_32.dll
************************************************************************
Hook type: Inline/Trampoline
Process: 108 (chrome.exe) (bitness as the image)
Victim module: WS2_32.dll (0x770f0000 - 0x77125000)
Function: WS2_32.dll!WSAEnumNetworkEvents at 0x770f31b1
Hook address: 0x5be0000
Hooking module base: 0x5be0000

Hooking module: <unknown>

Disassembly(0):
0x770f31b1 e94aceae8e       JMP 0x5be0000
0x770f31b6 51               PUSH ECX
0x770f31b7 813d48701177292e0f77 CMP DWORD [0x77117048], 0x770f2e29
0x770f31c1 56               PUSH ESI
0x770f31c2 0f85b5850000     JNZ 0x770fb77d
0x770f31c8 83               DB 0x83

Disassembly(1):
0x5be0000 50               PUSH EAX
0x5be0001 b810153657       MOV EAX, 0x57361510
0x5be0006 870424           XCHG [ESP], EAX
0x5be0009 c3               RET
0x5be000a 0000             ADD [EAX], AL
0x5be000c 0000             ADD [EAX], AL
```

we can see that code flow redirects to the same module that was identified thanks to the SSL VMT replacements:



Show 25 entries

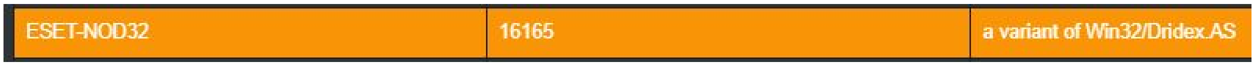
Dlllist

Search: bot

#	Pid	Base	Size	LoadCount	LoadTime	Path
2559	108	0x57320000	430080	1	2016-12-06 14:23:14 UTC+0000	[redacted] ot_x86_unpacked.module

Showing 1 to 1 of 1 entries (filtered from 2,952 total entries)

Dumping the hooking module leads to the complete discovery of the threat (even more AV engines confirm this):



ESET-NOD32	16165	a variant of Win32/Dridex.AS

Future work

Some improvements may be the following:

- Cleaning up of the code.
- Updating the SSL VMT lookup.
- Discovering additional new hooking methods by banking Trojans.

References

[1] P. Kálnai, M. Poslušný, “Browser attack points still abused by banking trojans,” In Proceedings of the 27th Virus Bulletin International Conference, Madrid, October 2017. (*accepted*)