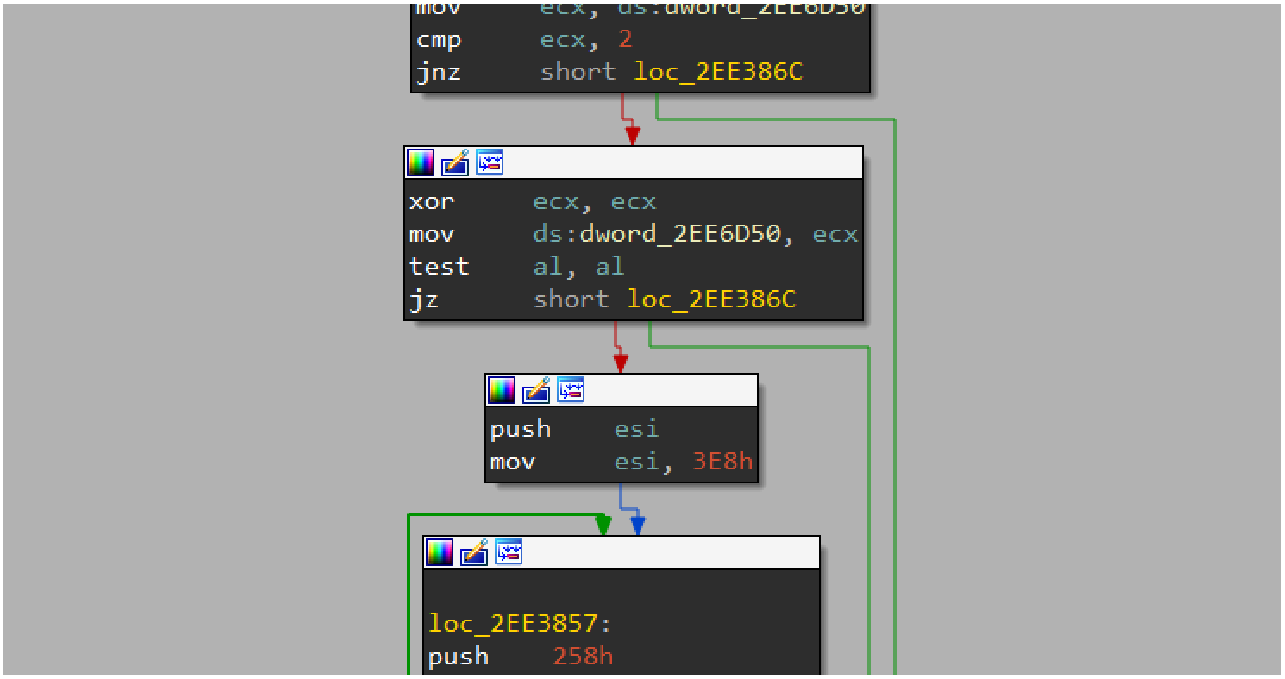
Comparative analysis between Bindiff and Diaphora - Patched Smokeloader Study Case



This article presents a comparative study case of diffing binaries using two technologies: *Bindiff* [1] and *Diaphora* [2]. We approached this topic in a Malware Analysis perspective by analyzing a (guess which malware family?) *Smokeloader* (! :D) campaign.

In August *2019*, I spotted this campaign using patched samples of *Smokeloader 2018* samples. This specific actor patched binaries to add new controllers *URL*s without needing to pay extra money (Smokeloader's seller charges extra-fee for *C2 URL* updates). This campaign was described in more detail in this previous article [3].

More details about the original samples [4][5] analyzed in this article can be found in the following tables:

Filename:	smokeloader_2018_unpatched.bin
Size:	33792 Bytes
File type:	PE32 executable (GUI) Intel 80386, for Microsoft Windows
md5:	76d9c9d7a779005f6caeaa72dbdde445
sha1:	34efc6312c7bff374563b1e429e2e29b5da119c2
sha256:	b61991e6b19229de40323d7e15e1b710a9e7f5fafe5d0ebdfc08918e373967d3

Filename:	smokeloader_2018_patched.bin
Size:	1202732 Bytes
File type:	PE32 executable (GUI) Intel 80386, for Microsoft Windows
md5:	7ba7a0d8d3e09be16291d5e7f37dcadb
sha1:	933d532332c9d3c2e41f8871768e0b1c08aaed0c
sha256:	6632e26a6970d8269a9d36594c07bc87d266d898bc7f99198ed081d9ff183b3f

The following tables hold details about the unpacked code dumped from "*explorer.exe*" used in this article [6] [7].

Filename:	explorer.exe.7e8e32c0.0x02ee0000-0x02ef3fff.dmp
Size:	81920 Bytes
File type:	data
md5:	711c02bec678b9dace09bed151d4cedd
sha1:	84d6b468fed7dd7a40a1eeba8bdc025e05538f3c
sha256:	865c18d1dd13eaa77fabf2e03610e8eb405e2baa39bf68906d856af946e5ffe1

Filename:	explorer.exe.7e8df030.0x00be0000-0x00bf3fff_patched.dmp
Size:	81920 Bytes (yes, same size)
File type:	data
md5:	d8f23c399f8de9490e808d71d00763ef
sha1:	e1daad6cb696966c5ced8b7d6a2425ff249bf227
sha256:	421482d292700639c27025db06a858aafee24d89737410571faf40d8dcb53288

Summarizing the main changes implemented by this patch are:

- wipes out the code for decrypting *C2 URL*s;
- replaces it with *NOP*s and hardcoded *C2 URL* string; and
- preserves the original size of decryption function to not disrupt offsets;

Figure 01 and 02 presents the graph of the original code. Figure 01 is the code used for indexing a table of encrypted *C2 URL*s payloads. Figure 02 lists the code used for decrypting the *C2 URL*s. This function is called in other parts of the code (not only by the function shown in Figure 01) - this is why they are not merged in one function. We labeled functions (e.g. "*__decrypt_C2_url*" and "*__decrypt_c2_algorithm*") in this assembly code to make it easier to read.

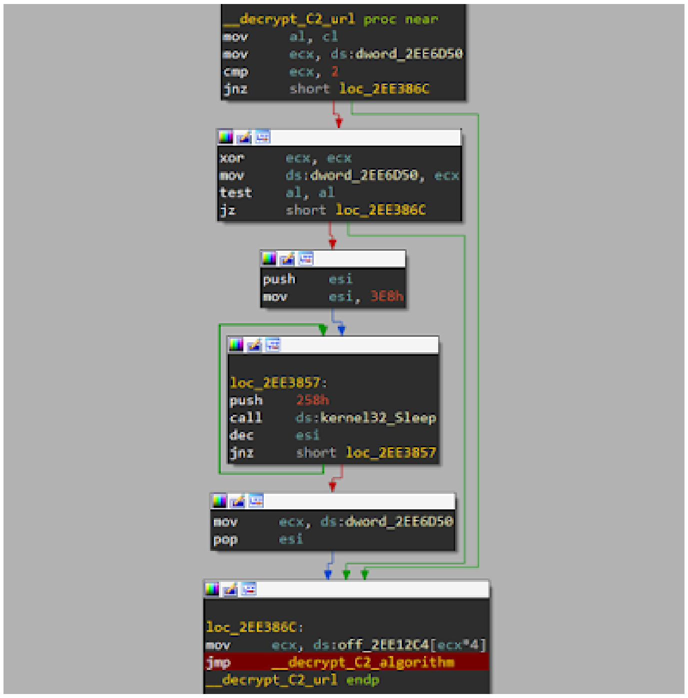


Figure 01 - Original code used for decrypting *C2 URL*s.

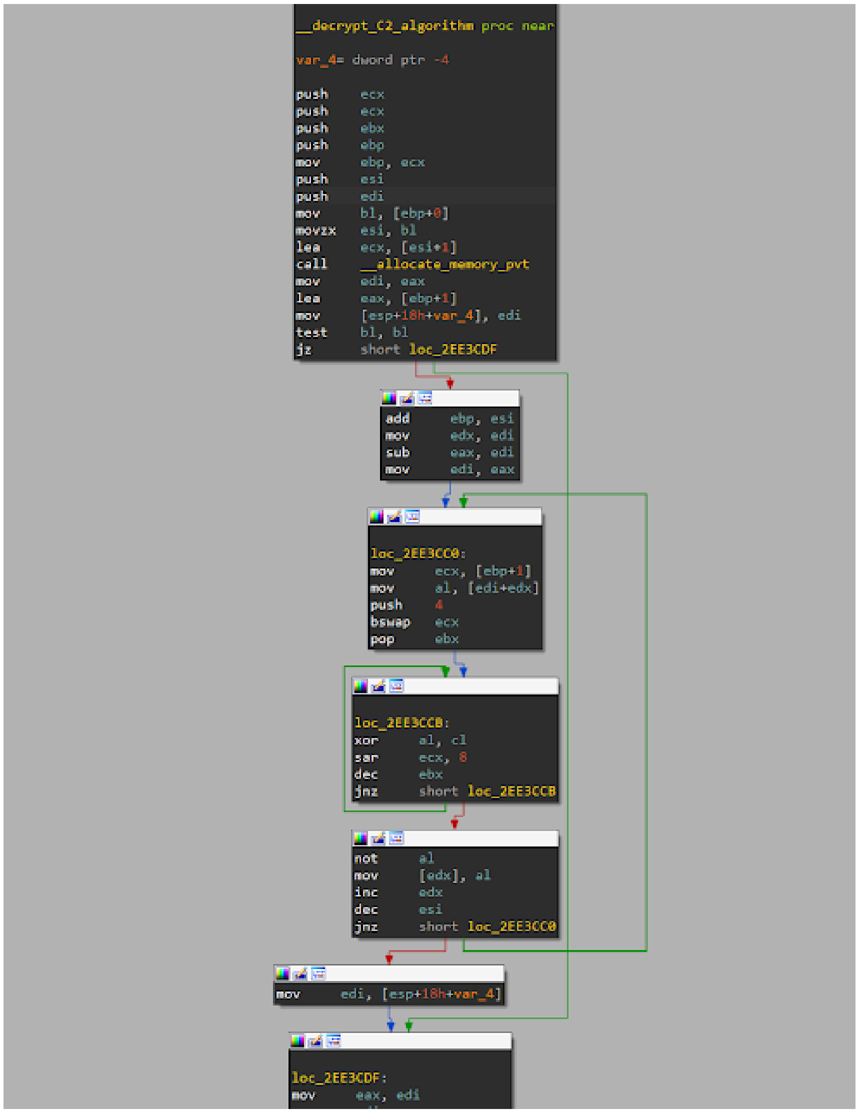


Figure 02 - Original encryption scheme used for decrypting *C2 URLs*.

Figure 03 and 04 shows the same functions on the patched version of *Smokeloader*. We can notice that the first function is the same as the unpatched version but the second function was replaced. This second function returns the address for the hardcoded *URL* in *ECX*. More details about how it works can be found in this article [3].

Figure 03 - Patched code used for decrypting *C2 URL*s.

```
seg000:00BE3BEF sub_BE3BEF      proc near               ; CODE XREF: sub_BE19EE+154↑p
seg000:00BE3BEF                                         ; sub_BE24D8+63↑p ...
seg000:00BE3BEF                 call    $+5
seg000:00BE3BF4
seg000:00BE3BF4 loc_BE3BF4:                             ; DATA XREF: sub_BE3BEF+6↓o
seg000:00BE3BF4                 pop     eax
seg000:00BE3BF5                 add     eax, (offset aHttpJnanny2PwB - offset loc_BE3BF4)
seg000:00BE3BF8                 mov     ecx, eax
seg000:00BE3BFA                 nop
seg000:00BE3BFB                 nop
seg000:00BE3BFC                 jmp     short locret_BE3C41
seg000:00BE3BFC ; ---------------------------------------------------------------------------
seg000:00BE3BFE                 align 10h
seg000:00BE3C00                 dd 0
seg000:00BE3C04                 db 2 dup(0)
seg000:00BE3C06 aHttpJnanny2PwB db 'http://jnanny2.pw/br/',0
seg000:00BE3C06                                         ; DATA XREF: sub_BE3BEF+6↑o
seg000:00BE3C1C                 dd 8 dup(0)
seg000:00BE3C3C                 db 5 dup(90h)
seg000:00BE3C41 ; ---------------------------------------------------------------------------
seg000:00BE3C41
seg000:00BE3C41 locret_BE3C41:                          ; CODE XREF: sub_BE3BEF+D↑j
seg000:00BE3C41                 retn
seg000:00BE3C41 sub_BE3BEF      endp
```

Figure 04 - Patched code returning decrypted C2 URL string.

The next sections describe the output of *Diaphora* and *Bindiff* when diffing the samples above.

.::[Diffing using Diaphora]

Diaphora is an Open Source binary diffing tool that uses *SQLite* as an intermediate representation for storing code and characteristics of reversed binaries [8]. It implements many diffing heuristics (strategies) directly on top of this database. The main advantage of this approach is that *Diaphora* is technology agnostic - this means that it does not depend on any reversing framework such as *Ghidra*, *IDApro*, or *Binary Ninja*.

It can even compare reversing databases built up using one specific tool with projects using another tool. This characteristic facilitates collaboration among researchers. Another big advantage is that by using *SQLite* for describing its heuristics it makes the processing of adding a new heuristic as "simple" as writing a new *SQL* so more people can contribute to the growth of the project and more experimental heuristics can be quickly prototyped and verified.

In the experiment described in this section, we used *Diaphora* version *2.0.2* (released in October 2019) and *IDApro 7.5*. *Diaphora* works as an *IDApro Python* script and can be easily executed by "*File -> Script File*" or "*alt + F7*". Figure 05 shows its main interface.

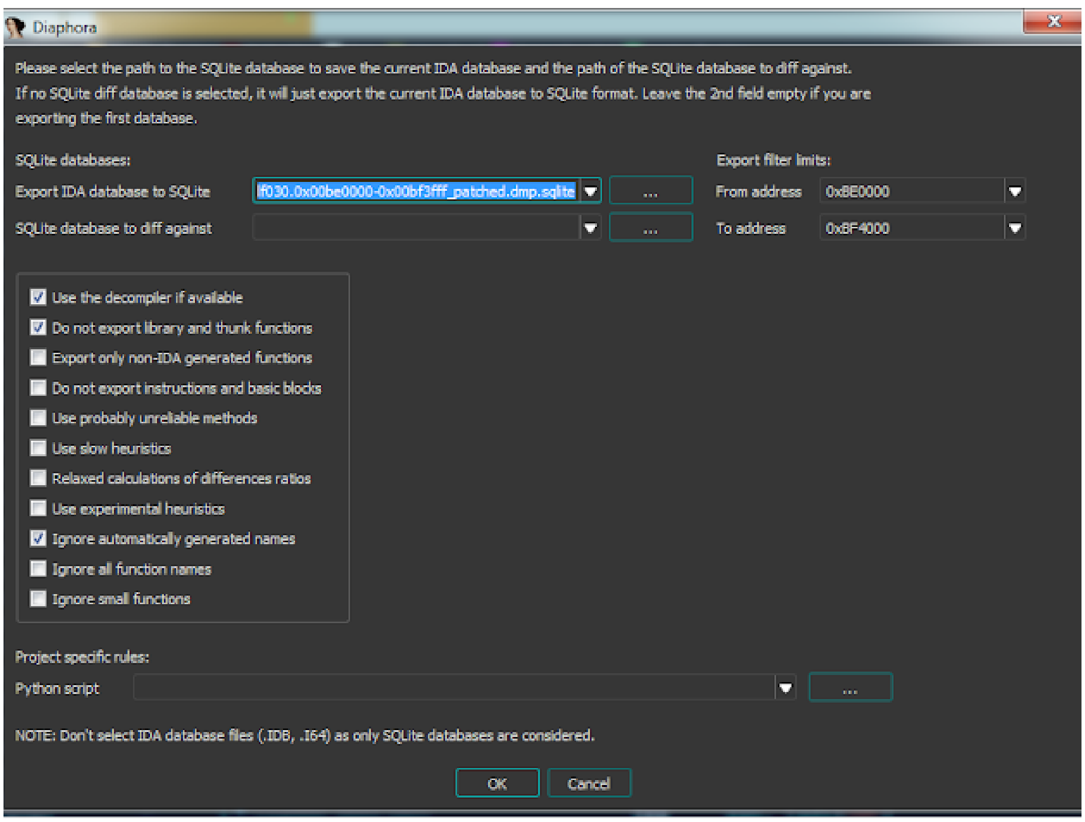


Figure 05 - Diaphora main Dialog Interface

This interface is a little bit confusing at a first sight especially for users that did not go through the documentation before trying to use it. It expects the user to input both *SQLite* databases to be compared, boundaries (for the working database), and set up some checkboxes with options.

First, we used Diaphora over the reference database (unpatched *Smokeloader*) for extracting characteristics and generating our reference *Diaphora SQLite* database. For doing this we just need to open the reference database on *IDA*, open *Diaphora*, and fill the "*Export IDA database to SQLite*" input field. *Diaphora* will export its *SQLite* database to the same base directory of the *IDA* working files.

After executing *Diaphora* on our patched *Smokeloader* using out saved labeled database of *Smokeloader 2018* as a reference in *Diaphora* we get four new tabs in *IDA*:

- *Best Matches* - common functions to both databases. The ones with *100%* match ratio;
- *Partial Matches* - all functions that are not Best matches and not unmatched;
- *Unmatched in Primary* - all functions in the first database that are not present in the second;
- *Unmatched in Secondary* - all functions in the second database that are not present in the first;

Figure 06 shows the content of the "*Best Matches*" tab in our example.

IDA View-A | Unmatched in secondary | Unmatched in primary | Partial matches | Best matches

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00000	00be3e3a	sub_BE3E3A	02ee3ee0	__hashing	1.000	4	4	Function hash
00001	00be39a1	sub_BE39A1	02ee3a47	__create_7char_alpha	1.000	5	5	Function hash
00002	00be3636	sub_BE3636	02ee36dc	__check_if_bit_domain	1.000	6	6	Function hash
00003	00be3d76	sub_BE3D76	02ee3e1c	__crypt	1.000	8	8	Function hash
00004	00be3a4a	sub_BE3A4A	02ee3af0	__get_proc_address	1.000	9	9	Function hash
00005	00be1e35	sub_BE1E35	02ee1e45	__talk_to_c2	1.000	43	43	Same rare KOKA hash

Figure 06 - Diaphora Best Matches tab

Each row shows function labels in primary and second databases, matching ratio (which goes from *0* to *1*), amount of *basic blocks* in each database and information about which heuristic was used to compare both functions. *Diaphora* provides features to importing features (such as commends and function labels) from the reference database to the target database. Usually, you will want to copy all annotations from one database to another in this "*Best Matches*" tab. By checking this tab we can see that few core functions are kept intact in the patched version and we are facing two very similar applications.

Figure 07 presents the "*Partial Matches*" functions.

IDA View-A | Unmatched in secondary | Unmatched in primary | Partial matches | Best matches

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks	Description
00002	00be2bfa	sub_BE2BFA	02ee2c4c	???__executing_loaded_code	0.995	7	7	Same rare KOKA hash
00004	00be2c76	sub_BE2C76	02ee2cc8	__wrapper_to_CoInitializeSecurity	0.995	17	17	Same KOKA hash and constants
00017	00be3842	sub_BE3842	02ee38e8	__creates_mapped_section	0.995	14	14	Same rare KOKA hash
00019	00be2f6e	sub_BE2F6E	02ee2fc0	__loads_content_into_mapped_memory	0.992	25	25	Same rare KOKA hash
00007	00be38d9	sub_BE38D9	02ee397f	__checks_process_paging	0.990	7	7	Same rare KOKA hash
00011	00be3e58	sub_BE3E58	02ee3efe	__find_substring	0.990	10	10	Same rare KOKA hash
00018	00be3ae2	sub_BE3AE2	02ee3b88	__load_library_and_get_proc_addr	0.988	16	16	Same rare KOKA hash
00012	00be3673	sub_BE3673	02ee3719	__resolve_domain	0.985	10	10	Same rare KOKA hash
00009	00be3702	sub_BE3702	02ee37a8	__get_proxy_configuration	0.983	9	9	Same rare KOKA hash
00024	00be1818	sub_BE1818	02ee1828	__load_procs_from_module	0.983	5	5	Mnemonics small-primes-product
00008	00be3d16	sub_BE3D16	02ee3dbc	__fetch_data	0.980	8	8	Same rare KOKA hash
00025	00be2f05	sub_BE2F05	02ee2f57	???__something_inside_handle_response	0.980	5	5	Mnemonics small-primes-product
00005	00be3792	sub_BE3792	02ee3838	__decrypt_C2_url	0.978	7	7	Same rare KOKA hash
00000	00be3ea5	sub_BE3EA5	02ee3f4b	__calc_MD5_hash	0.973	5	5	Same constants
00014	00be29d9	sub_BE29D9	02ee2a2b	__write_or_read_file	0.973	11	11	Same rare KOKA hash
00026	00be3173	sub_BE3173	02ee31c5	__connect_to_c2	0.973	5	5	Mnemonics small-primes-product
00006	00be2806	sub_BE2806	02ee2858	__inject_shellcode_into_process	0.970	7	7	Same rare KOKA hash
00003	00be310b	sub_BE310B	02ee315d	__enum_windows_handler	0.968	7	7	Same rare KOKA hash
00027	00be381a	sub_BE381A	02ee38c0	__terminate_process	0.965	5	5	Mnemonics small-primes-product
00010	00be3c42	sub_BE3C42	02ee3ce8	__fetch_and_load_user_agent	0.963	10	10	Same rare KOKA hash
00023	00be39c9	sub_BE39C9	02ee3a6f	__get_current_process_privileges	0.963	3	3	Mnemonics small-primes-product
00013	00be1c98	sub_BE1C98	02ee1ca8	__build_profile	0.960	11	11	Same rare KOKA hash
00015	00be3074	sub_BE3074	02ee30c6	__anti_debug_thread_001	0.960	12	12	Same rare KOKA hash
00016	00be166f	sub_BE166F	02ee167f	__load_libraries_anti_debug	0.943	13	13	Same rare KOKA hash
00021	00be2b02	sub_BE2B02	02ee2b54	__execute_cmd_and_schedule_exec_using_autoupdate	0.932	3	3	Mnemonics small-primes-product
00020	00be2abe	sub_BE2ABE	02ee2b10	__schedule_exec_using_autoupdate	0.930	2	2	Mnemonics small-primes-product
00022	00be37d2	sub_BE37D2	02ee3878	__terminate_processes	0.920	3	3	Mnemonics small-primes-product
00001	00be390f	sub_BE390F	02ee39b5	__set_file_attributes	0.820	1	1	Same constants

Figure 07 - Partial Matches functions and our "*__decrypt_C2_url*" function right there with *0.978* matching ratio.

Diaphora provides very fine level diffing and most of these functions are basically unmatching constants. These matches are marked in yellow in the graph diff view. Figure 08 and 09 shows the "*__set_file_attributes"* function and its match in the primary database. We can clearly see that they actually are the same function.

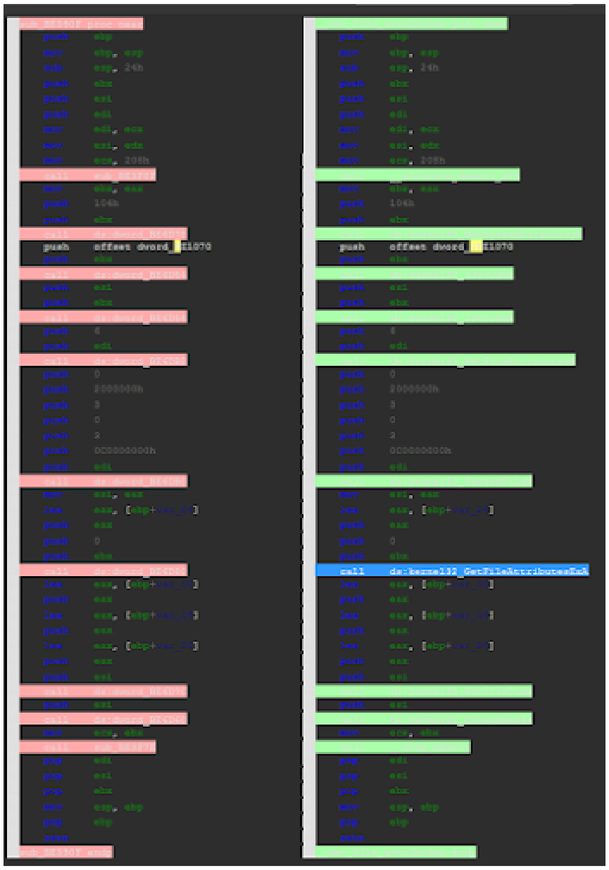


Figure 08 - Diffing "*__set_file_attributes*" function assembly view.

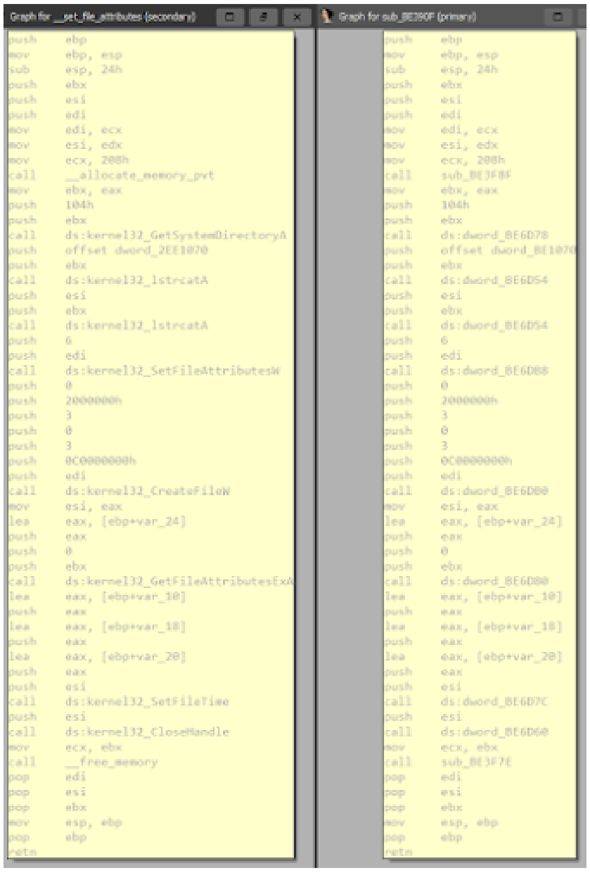


Figure 09 - Diffing "__set_file_attributes" function using graph view.

Figure 10 shows the relevant patch we are interested in the "*__decrypt_C2_url*" function. As we can see both functions are basically identical until the *jump* instruction. The function jumps to what I labeled as "*__decrypt_C2_algorithm*" and is used as a function in other places around the code.

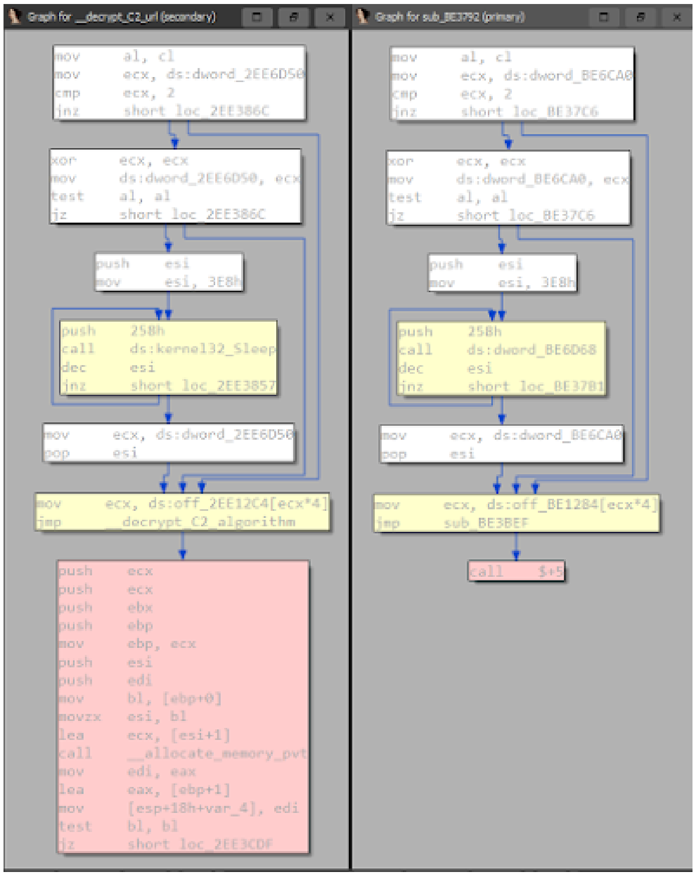
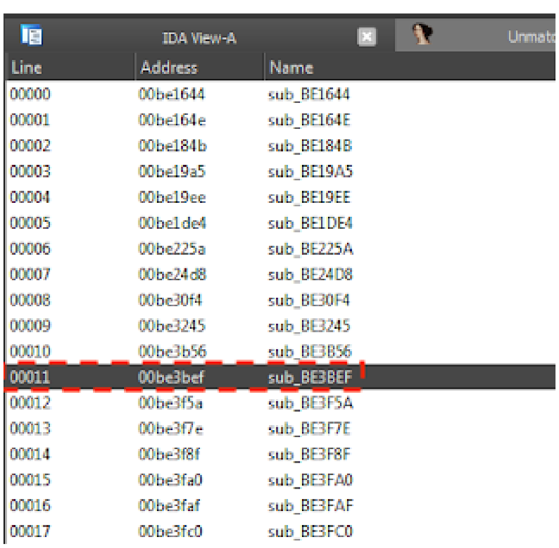


Figure 10 - "*__decrypt_C2_url*" graph diff pointed in the "*Partial Matches*" tab

What disappointed me a little bit was that *Diaphora* did not include the rest of the code of "*__decrypt_C2_algorithm*" in this function. This move bumped up the matching ratio and this was misleading when prioritizing what to manually analyze. The "*__decrypt_C2_algorithm*" function shows up in the "*Unmatched in Secondary*" tab. This is a good thing as functions in this tab should be the priority in this kind of analysis. In our example, we got *18* functions (out of 52) to analyze marked in the "*Unmatched in Secondary*" tab. Figure 11 shows this tab and Figure 12 shows the graph view of this function.



Line	Address	Name
00000	00be1644	sub_BE1644
00001	00be164e	sub_BE164E
00002	00be184b	sub_BE184B
00003	00be19a5	sub_BE19A5
00004	00be19ee	sub_BE19EE
00005	00be1de4	sub_BE1DE4
00006	00be225a	sub_BE225A
00007	00be24d8	sub_BE24D8
00008	00be30f4	sub_BE30F4
00009	00be3245	sub_BE3245
00010	00be3b56	sub_BE3B56
00011	00be3bef	sub_BE3BEF
00012	00be3f5a	sub_BE3F5A
00013	00be3f7e	sub_BE3F7E
00014	00be3f8f	sub_BE3F8F
00015	00be3fa0	sub_BE3FA0
00016	00be3faf	sub_BE3FAF
00017	00be3fc0	sub_BE3FC0

Figure 11 - "*Unmatched in Secondary*" tab and patched "*__decrypted_C2_algorithm*" function

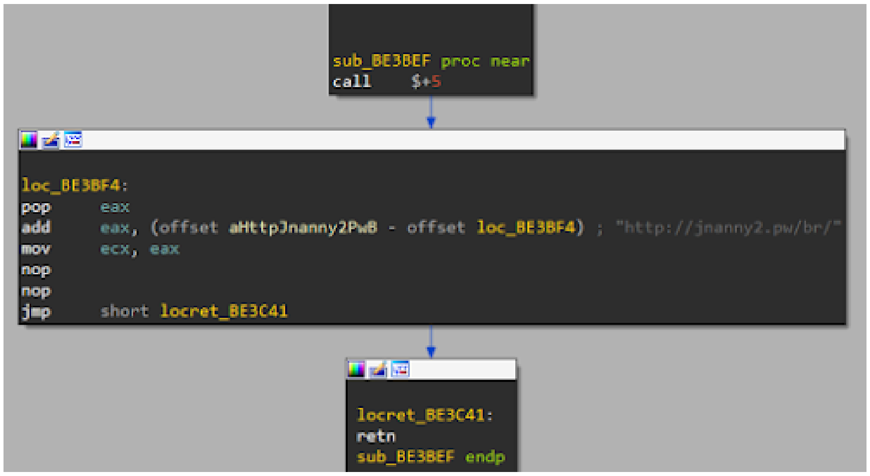


Figure 12 - Patched version of "*__decrypt_C2_algorithm*" function

The nicest thing about using *Diaphora* is that all this analysis could be easily shared by sharing compressed *SQLite* databases around. So that is it for the *Diaphora* side.

.:: [Diffing using BinDiff]

BinDiff is an executable-comparison tool created by *Zynamics* [9] (called *SABRE* in earlier days) in *2007*. Zynamics was acquired by Google in *2011* and BinDiff became freeware in *2016* [10][11]. *BinDiff* is a plugin of *IDAPro* and works directly over *IDBs* (*IDA pro Database*). Because of this design, *BinDiff* requires users to have an *IDA* license (#notcool). *BinDiff 6.0* supports also *Ghidra* using this extra plugin called *BinExport* [12] which implements something similar to *Diaphora*'s design.

OALabs released a very didactic video tutorial on *BinDiff* [13]. This video also teaches how to install BinDiff. We used *BinDiff 6.0* and *IDAPro 7.5 i*n this experiment. BinDiff adds a new option to the "*File*" menu in *IDA* and to compare two executable is as easy as opening a new *IDB* on top of the current one. Figure 13 shows the *BinDiff* option in *IDA*.

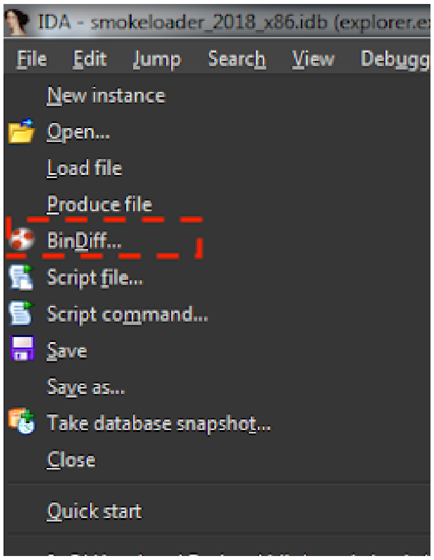
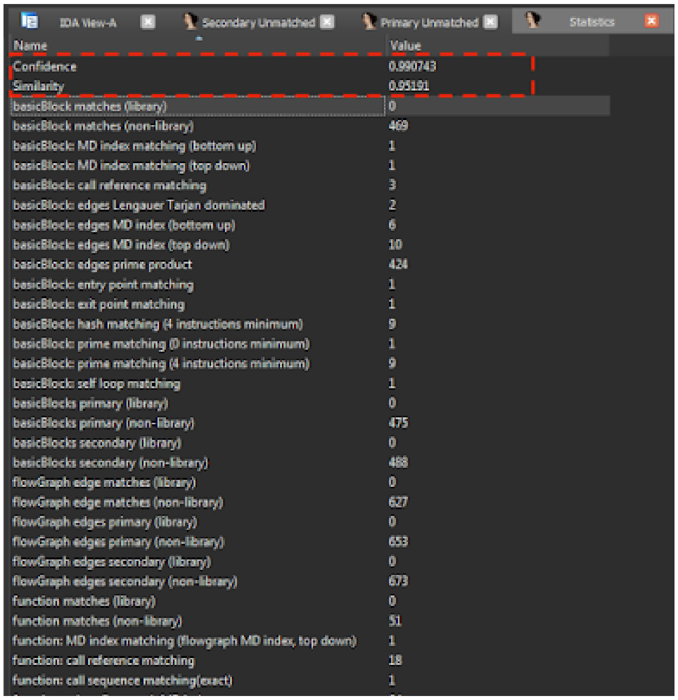


Figure 13 - BinDiff option in *IDAPro*

After loading an annotated *IDA* database using *BinDiff*, it will add *4* new tabs:

- *Primary Unmatched* - functions in the primary database that did not match any function in the secondary database;
- *Secondary Unmatched* - functions in the secondary database that did not match any function in the primary database;
- *Statistics* - general similarity information about both executables;
- *Matched Functions* - all functions with matches and their respective similarity index.

The two first tabs hold the same information as their correlated tabs in *Diaphora*. Statistics tab provides high-level information about the matching process. Information in this tab can be used for quick knowing if the binary is a variation of the reference database. Figure 13 shows the data presented in the Statistics tab after loading our reference *Smokeloader* database against the patched one using *BinDiff*.

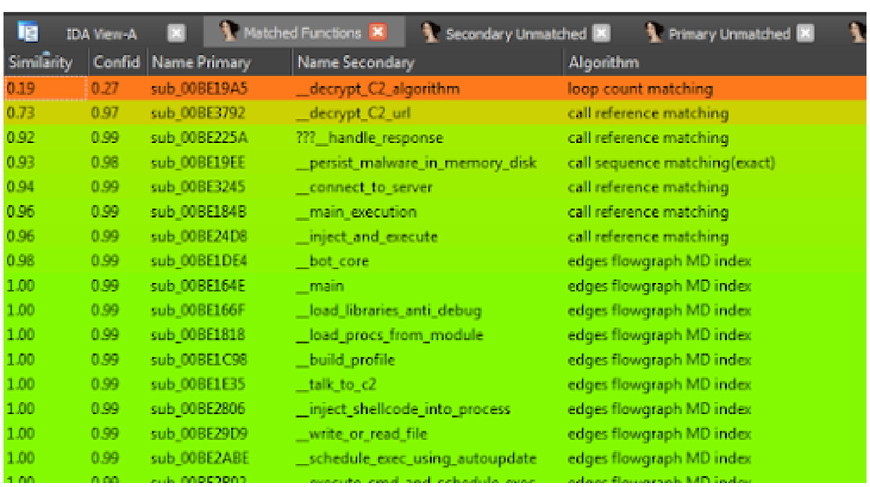


Name	Value
Confidence	0.990743
Similarity	0.95191
basicBlock matches (library)	0
basicBlock matches (non-library)	469
basicBlock: MD index matching (bottom up)	1
basicBlock: MD index matching (top down)	1
basicBlock: call reference matching	3
basicBlock: edges Lengauer Tarjan dominated	2
basicBlock: edges MD index (bottom up)	6
basicBlock: edges MD index (top down)	10
basicBlock: edges prime product	424
basicBlock: entry point matching	1
basicBlock: exit point matching	1
basicBlock: hash matching (4 instructions minimum)	9
basicBlock: prime matching (0 instructions minimum)	1
basicBlock: prime matching (4 instructions minimum)	9
basicBlock: self loop matching	1
basicBlocks primary (library)	0
basicBlocks primary (non-library)	475
basicBlocks secondary (library)	0
basicBlocks secondary (non-library)	488
flowGraph edge matches (library)	0
flowGraph edge matches (non-library)	627
flowGraph edges primary (library)	0
flowGraph edges primary (non-library)	653
flowGraph edges secondary (library)	0
flowGraph edges secondary (non-library)	673
function matches (library)	0
function matches (non-library)	51
function: MD index matching (flowgraph MD index, top down)	1
function: call reference matching	18
function: call sequence matching(exact)	1

Figure 14 - *Statistics* tab and confidence and similarity indexes

BinDiff calculated a similarity index of *95%* (with *99%* confidence). This means that we are likely dealing with two versions of the same software. The other metrics are more about counters and general information about both databases.

Figure 14 shows the *Matched Functions* tab and the similarity index for each match. It is also possible to see the heuristic used for each match.



Similarity	Confid	Name Primary	Name Secondary	Algorithm
0.19	0.27	sub_00BE19A5	_decrypt_C2_algorithm	loop count matching
0.73	0.97	sub_00BE3792	_decrypt_C2_url	call reference matching
0.92	0.99	sub_00BE225A	???_handle_response	call reference matching
0.93	0.98	sub_00BE19EE	_persist_malware_in_memory_disk	call sequence matching(exact)
0.94	0.99	sub_00BE3245	_connect_to_server	call reference matching
0.96	0.99	sub_00BE184B	_main_execution	call reference matching
0.96	0.99	sub_00BE24D8	_inject_and_execute	call reference matching
0.98	0.99	sub_00BE1DE4	_bot_core	edges flowgraph MD index
1.00	0.99	sub_00BE164E	_main	edges flowgraph MD index
1.00	0.99	sub_00BE166F	_load_libraries_anti_debug	edges flowgraph MD index
1.00	0.99	sub_00BE1818	_load_procs_from_module	edges flowgraph MD index
1.00	0.99	sub_00BE1C98	_build_profile	edges flowgraph MD index
1.00	0.99	sub_00BE1E35	_talk_to_c2	edges flowgraph MD index
1.00	0.99	sub_00BE2806	_inject_shellcode_into_process	edges flowgraph MD index
1.00	0.99	sub_00BE29D9	_write_or_read_file	edges flowgraph MD index
1.00	0.99	sub_00BE2ABE	_schedule_exec_using_autoupdate	edges flowgraph MD index

Figure 15 - *Matched Functions* tab

BinDiff managed to match *51* out of *52* functions. This is a very good result. Besides that *Bindiff* managed to find out the two most affected functions by the patch: (i) "*__decrypt_C2_algorithm*" and (*ii*) "*__decrypt_C2_url*". It is possible to zoom in and visualize changes in graphs of the function matched.

Figure 16 shows changes in function "*__decrypt_C2_url*" (*73%* of similarity and *97%* of confidence).

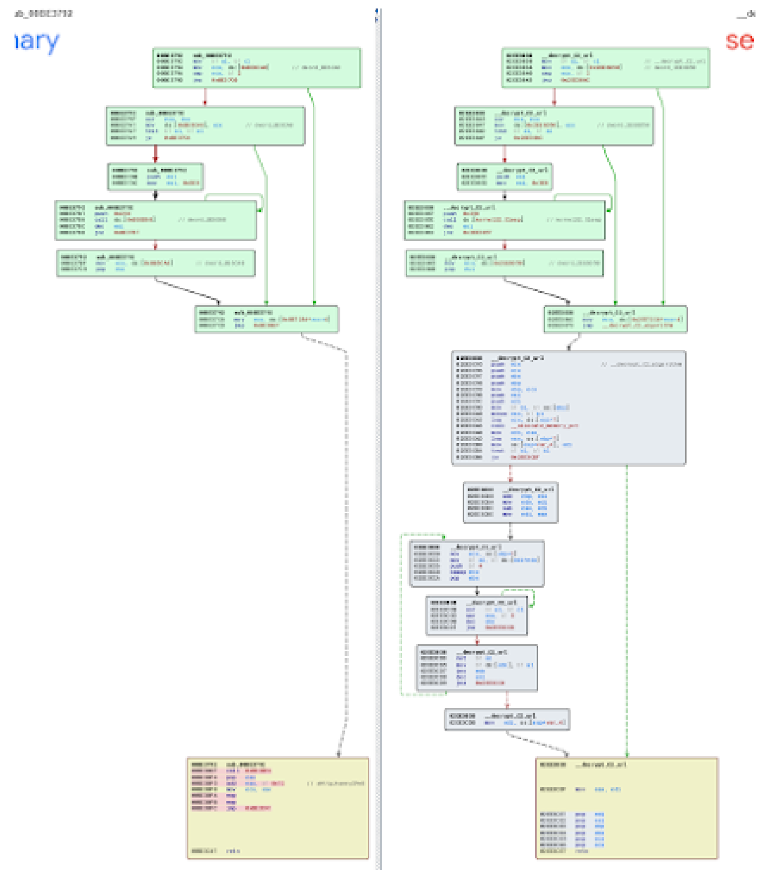


Figure 16 - Changes in function "*__decrypt_C2_url*"

As we can see, *BinDiff* hits the bullseye and detects exactly the changes without splitting the function into two parts. The way *BinDiff* organizes its graphs makes changes really easy to visualize.

Bindiff also matches the "*__decrypt_C2_algorithm*" function with some other function located at "*0x00BE19A5*" using the "*loop count*" heuristic but the similarity index is only *19%* and confidence is *27%.* This means that Bindiff matched the "*__decrypted_C2_algorithm"* function, which was wiped out with the patch, with some other random function. I don't even consider this a false-positive as results clearly state that this match has low confidence. It is useful to get a spotlight pointed to this function - this for sure is a good place to start an analysis in this specific scenario.

.:: [Conclusions]

For the specific malware analysis problem discussed in this article, we definitely got better results using *BinDiff* than *Diaphora*. I also feel that all fine program analysis used in heuristics applied by *BinDiff* makes a big difference in the final result.

In terms of general design (not taking in consideration heuristics), I think *Diaphora* has more advantages than *Bindiff*, because of its intermediate representation using an open specification format as *SQLite* and

SQL for modeling heuristics. *Diaphora* is also *Open Source* so there is a task force sustained by a community in order to improve heuristics and this is the way to go *IMHO*.

Diaphora also takes boundaries as parameters and this can be very useful in case of analyzing big databases especially when analyzing patches for vulnerability development purposes. In line with that, this article focuses on malware analysis but these same technologies could also be used for analyzing vulnerabilities and its patches - ammunition to another blog post.

.:: [References]

[1] https://www.zynamics.com/bindiff.html
[2] http://diaphora.re/
[3] http://security.neurolabs.club/2019/08/smokeloaders-hardcoded-domains-sneaky.html
[4] https://www.virustotal.com/gui/file/b61991e6b19229de40323d7e15e1b710a9e7f5fafe5d0ebdfc08918e373967d3
[5] https://www.virustotal.com/gui/file/6632e26a6970d8269a9d36594c07bc87d266d898bc7f99198ed081d9ff183b3f
[6] https://www.virustotal.com/gui/file/865c18d1dd13eaa77fabf2e03610e8eb405e2baa39bf68906d856af946e5ffe1
[7] https://www.virustotal.com/gui/file/421482d292700639c27025db06a858aafee24d89737410571faf40d8dcb53288
[8] https://www.youtube.com/watch?v=eAVfRxp99DM (BSides Joxean)
[9] https://www.zynamics.com/company.html
[10] https://security.googleblog.com/2016/03/bindiff-now-available-for-free.html
[11] https://www.zynamics.com/bindiff/manual/#N20140
[12] https://github.com/google/binexport/tree/v11/java/BinExport
[13] https://www.youtube.com/watch?v=BLBjcZe-C3I (BinDiff OALabs)