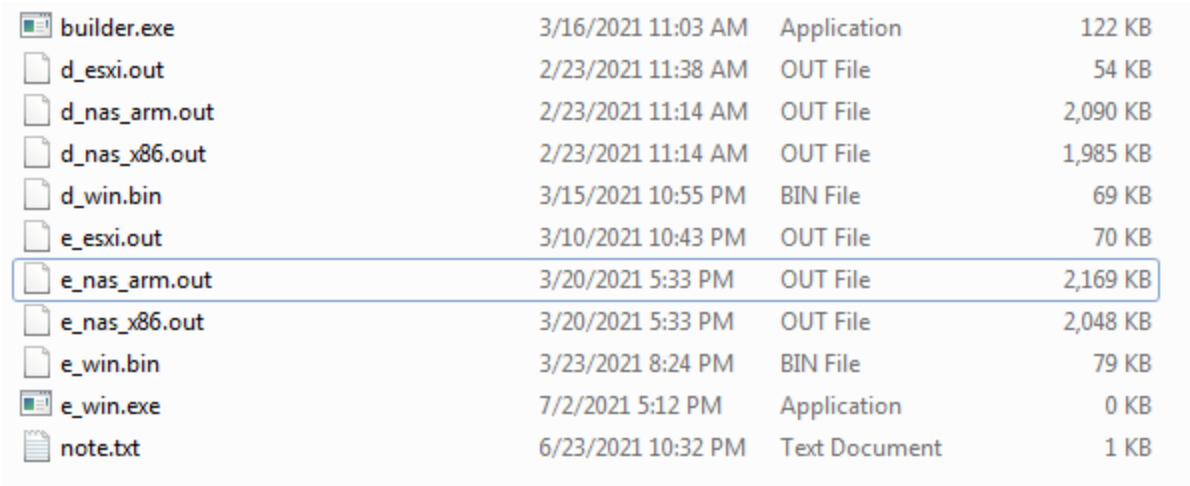
Quick review of Babuk ransomware builder

lab52.io/blog/quick-review-of-babuk-ransomware-builder/

Last week, the builder for the Babuk ransomware family was leaked online. Lab52 has obtained and analyzed this builder sample determining that it is very likely to be authentic.

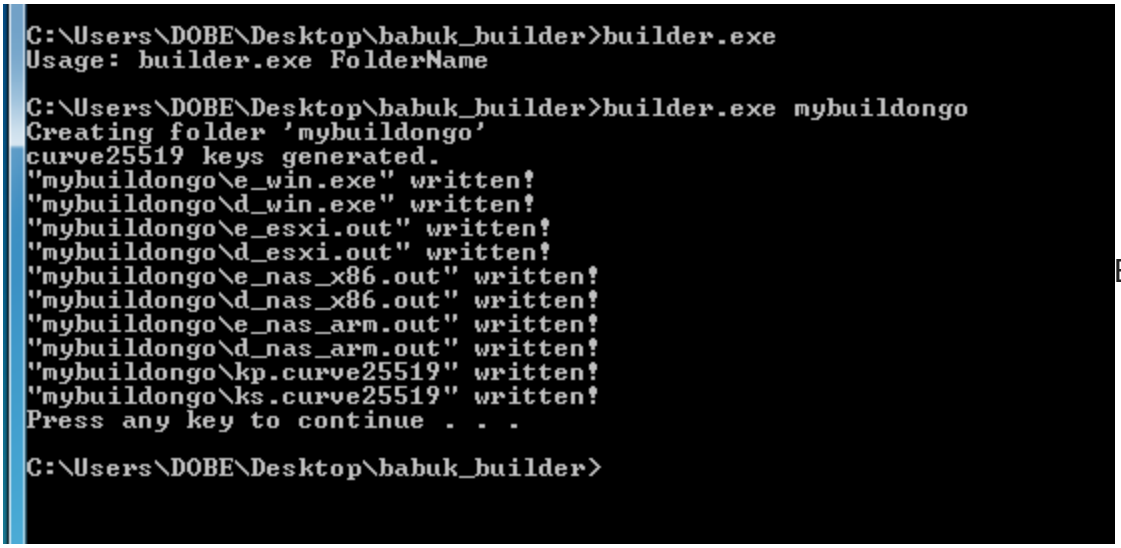
After their recent official move from Ransomware as a Service to data leaks extortions, someone uploaded to virusTotal the ransomware builder for unknown reasons, and it was soon identified as such by British researcher Kevin Beaumont.



builder.exe	3/16/2021 11:03 AM	Application	122 KB
d_esxi.out	2/23/2021 11:38 AM	OUT File	54 KB
d_nas_arm.out	2/23/2021 11:14 AM	OUT File	2,090 KB
d_nas_x86.out	2/23/2021 11:14 AM	OUT File	1,985 KB
d_win.bin	3/15/2021 10:55 PM	BIN File	69 KB
e_esxi.out	3/10/2021 10:43 PM	OUT File	70 KB
e_nas_arm.out	3/20/2021 5:33 PM	OUT File	2,169 KB
e_nas_x86.out	3/20/2021 5:33 PM	OUT File	2,048 KB
e_win.bin	3/23/2021 8:24 PM	BIN File	79 KB
e_win.exe	7/2/2021 5:12 PM	Application	0 KB
note.txt	6/23/2021 10:32 PM	Text Document	1 KB

Content of Babuk builder leak

What we first find is builder.exe, along with 2 other Windows executable files with .bin extension, 4 different Unix executables, and note.txt. At a first test, we could see how we have to tell builder.exe the output folder as an argument, and we noticed that the files generated were similar to the builder folder files.

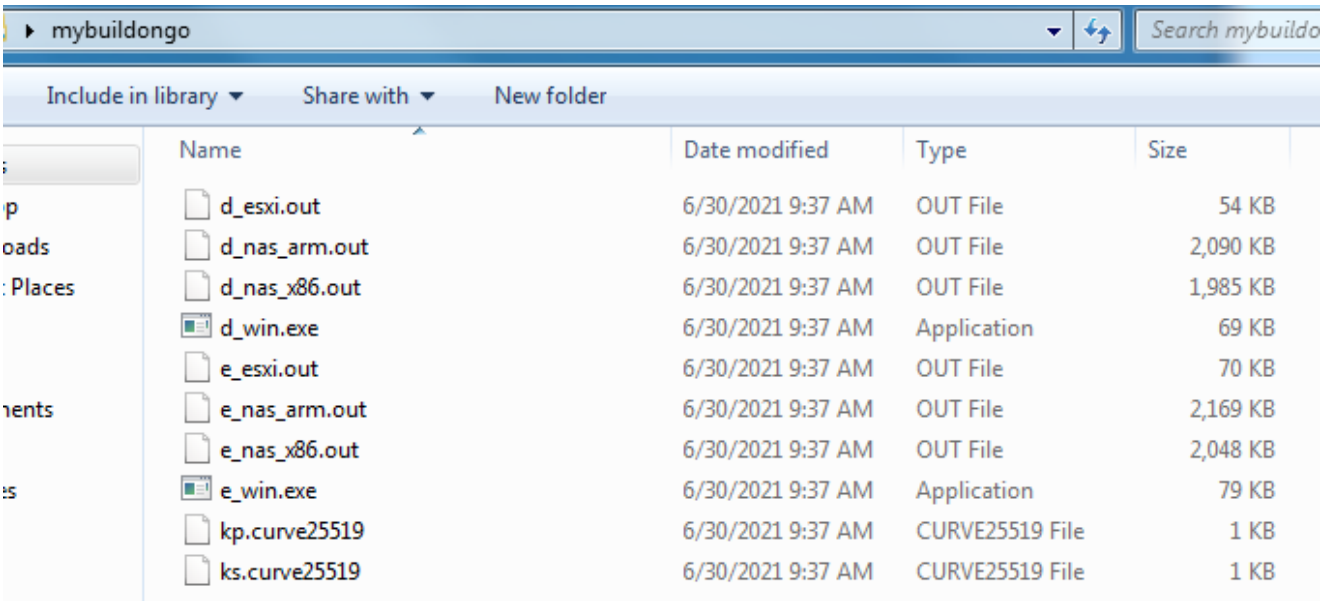


```
C:\Users\DOBE\Desktop\babuk_builder>builder.exe
Usage: builder.exe FolderName

C:\Users\DOBE\Desktop\babuk_builder>builder.exe mybuildongo
Creating folder 'mybuildongo'
curve25519 keys generated.
"mybuildongo\e_win.exe" written!
"mybuildongo\d_win.exe" written!
"mybuildongo\e_esxi.out" written!
"mybuildongo\d_esxi.out" written!
"mybuildongo\e_nas_x86.out" written!
"mybuildongo\d_nas_x86.out" written!
"mybuildongo\e_nas_arm.out" written!
"mybuildongo\d_nas_arm.out" written!
"mybuildongo\kp.curve25519" written!
"mybuildongo\ks.curve25519" written!
Press any key to continue . . .

C:\Users\DOBE\Desktop\babuk_builder>
```

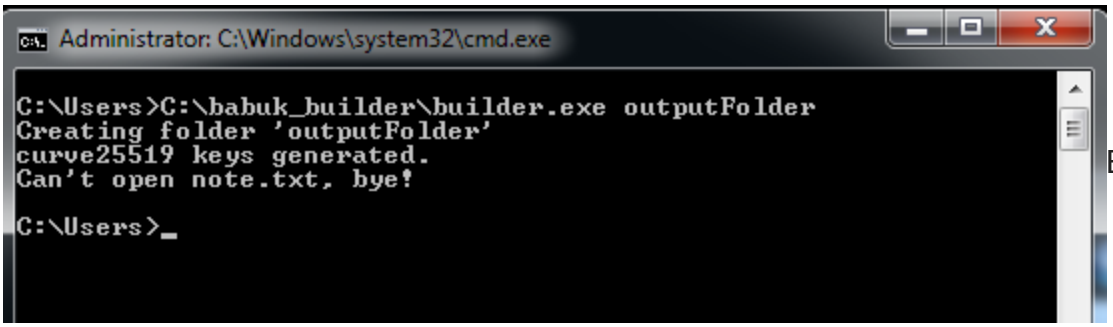
Babuk builder usage



Babuk builder output

After the successful execution, we get the two eliptic curve keys generated for encryption, 3 encryption executables for Windows, ARM-based NAS decives, and VMWare ESXi servers respectively, together with its corresponding decryption executables.

One interesting thing that we found after these firsts test was that builder.exe would look for its files in the folder from where it is called, causing an error in case we want to execute builder.exe with an absolute path from a different location, which could be considered a bug or, at least, a not so much elegant implementation.



Babuk builder path bug

We decided to compare the Windows crypter executable with real samples uploaded to public sandboxes, and we could first see useful information that was already suggesting that the builder could indeed be authentic.

property	value	REAL SAMPLE
md5	F0D4C7D334633A72A3C7BD722E12C378	
sha1	5240F71F60C473B5F9BA100D2CE1D6EFFDBC08C1	
sha256	1F2EDDA243404918B78AA6123AA1FC5B18DD9506E4042C7A1547B565334527E1	
md5-without-overlay	n/a	
sha1-without-overlay	n/a	
sha256-without-overlay	n/a	
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 0	
first-bytes-text	M Z ..... @	
file-size	80896 (bytes)	
size-without-overlay	n/a	
entropy	6.144	
imphash	202FA14F574C71C2F95878E40A79322D	
signature	n/a	
entry-point	55 8B EC 81 EC 90 00 00 00 E8 32 52 00 00 E8 CD 81 00 00 E8 A8 9E FF FF A3 60 42	
file-version	n/a	
description	n/a	
file-type	executable	
cpu	32-bit	
subsystem	GUI	
compiler-stamp	0x604FD6A7 (Mon Mar 15 22:50:31 2021 - UTC)	
debugger-stamp	0x604FD6A7 (Mon Mar 15 22:50:31 2021)	
resources-stamp	n/a	
exports-stamp	n/a	

property	value	BUILT SAMPLE
md5	E722820869D7F3A67E353FFDA66766B1	
sha1	EC128361C408090D9558BF35C00C88E2E25E59FC	
sha256	03DBBA2D38695589059DE72E4BCD4CA3421117A78E3C8F179F062D8DA1B44E4A	
md5-without-overlay	n/a	
sha1-without-overlay	n/a	
sha256-without-overlay	n/a	
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 0	
first-bytes-text	M Z ..... @	
file-size	80896 (bytes)	
size-without-overlay	n/a	
entropy	5.673	
imphash	202FA14F574C71C2F95878E40A79322D	
signature	n/a	
entry-point	55 8B EC 81 EC 88 00 00 00 E8 12 52 00 00 E8 AD 81 00 00 E8 A8 9E FF FF A3 60 42	
file-version	n/a	
description	n/a	
file-type	executable	
cpu	32-bit	
subsystem	GUI	
compiler-stamp	0x605A4000 (Tue Mar 23 20:22:40 2021 - UTC)	
debugger-stamp	0x605A4000 (Tue Mar 23 20:22:40 2021)	
resources-stamp	n/a	
exports-stamp	n/a	

Comparison *between real Babuk sample and built sample*

We also compared the encryption timing between two samples, getting similar times, which would be a reinforcement about its authenticity since Babuk is Top 3 fastest ransomware encryption speed since they updated their efficiency “flaws” identified by Chuong Dong during his great analysis of the three versions of Babuk. We were also able to also identify that this was a builder for their last version.

As the final comparison to ensure the authenticity of the sample, we compared the assembly code of both files using a plugin for IDA pro named Diaphora, and resulting to be almost identical.

Partial matches | Unmatched in primary

Line	Address	Name
00000	00404b20	sub_404B20

Not matching functions between real Babuk sample and built sample

Partial matches | Unmatched in primary | Best matches

Line	Address	Name	Address 2	Name 2	Ratio
00001	0040aab0	sub_40AAB0	0040aad0	sub_40AAD0	0.990
00000	0040aba0	start	0040abc0	start	0.800

Partial matching functions real Babuk sample vs. built sample

Partial matches | Unmatched in primary | Best matches

Line	Address	Name	Address 2	Name 2	Ratio
00000	0040a7c0	StartAddress	0040a7e0	StartAddress	1.000
00001	0040b150	sub_40B150	0040b150	sub_40B150	1.000
00002	0040b190	sub_40B190	0040b190	sub_40B190	1.000
00003	0040d930	sub_40D930	0040d930	sub_40D930	1.000
00004	0040fde0	sub_40FDE0	0040fde0	sub_40FDE0	1.000
00005	0040fdf0	sub_40FDF0	0040fdf0	sub_40FDF0	1.000
00006	0040fe80	sub_40FE80	0040fe80	sub_40FE80	1.000
00007	004101e0	sub_4101E0	004101e0	sub_4101E0	1.000
00008	00410550	sub_410550	00410550	sub_410550	1.000
00009	004105a0	sub_4105A0	004105a0	sub_4105A0	1.000
00010	004107e0	sub_4107E0	004107e0	sub_4107E0	1.000
00011	00410a50	sub_410A50	00410a50	sub_410A50	1.000
00012	00412d80	sub_412D80	00412d80	sub_412D80	1.000
00013	00412d90	sub_412D90	00412d90	sub_412D90	1.000
00014	00412dc0	sub_412DC0	00412dc0	sub_412DC0	1.000

Perfect matching function from real Babuk sample vs. built sample

As it could be expected, the builder would take the content of note.txt and use it as the ransomware note that it would be dropped in the infected machines. Since Babuk decided not to use any packing mechanism, we could also spot in clear text the ransom note and the rest of the space reserved for the ransomware note inside the built binaries.

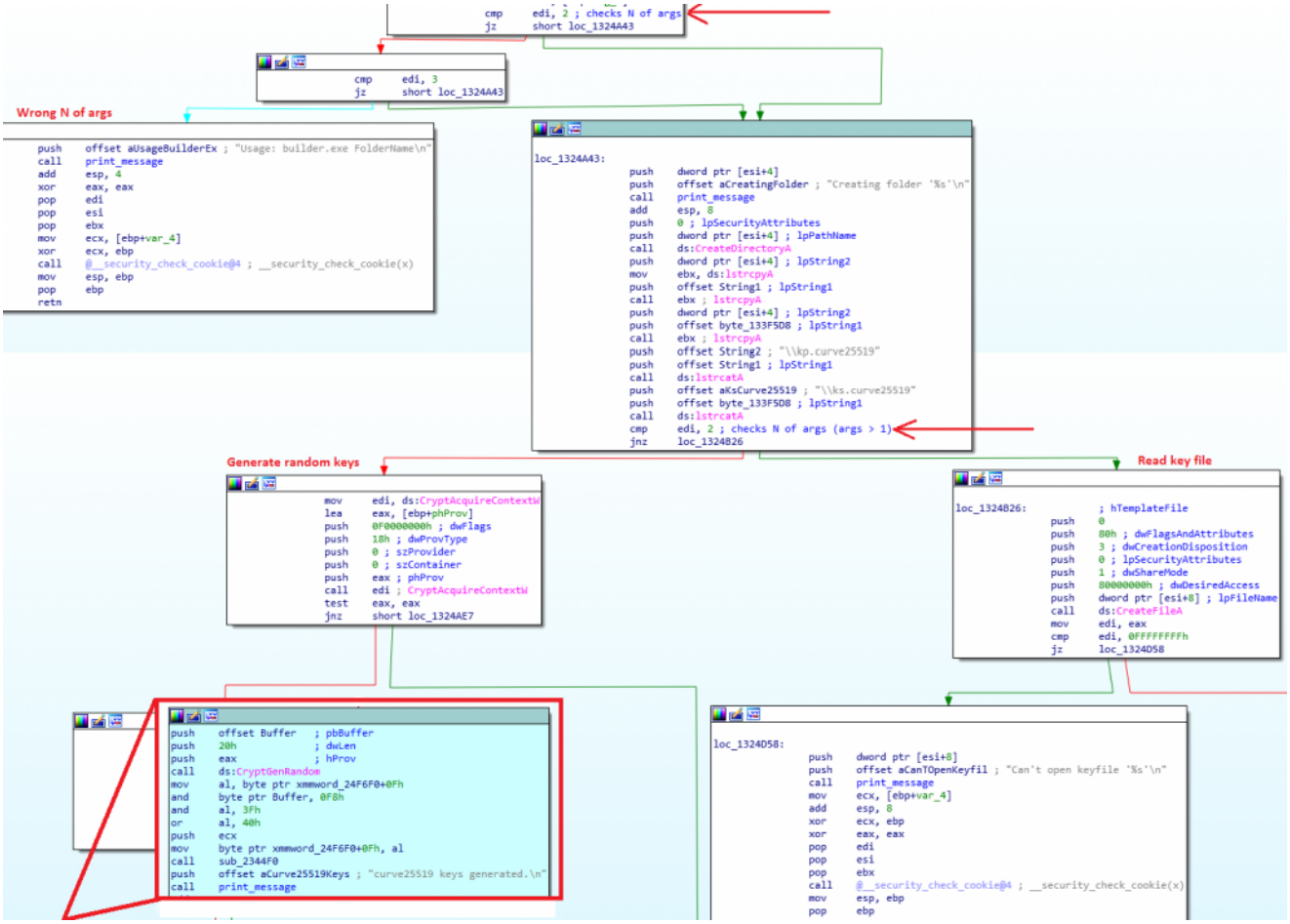
ascii	14	0x00000A60	-	-	IsWow64Process
ascii	30	0x00000A90	x	-	Wow64DisableWow64FsRedirection
ascii	29	0x00000B34	-	-	Wow64RevertWow64FsRedirection
ascii	14	0x00000B58	-	-	, Error Code:
ascii	4	0x00000B68	-	-	->
ascii	17	0x00000E5C	-	-	Can't OpenProcess
ascii	215	0x00000E70	-	-	LAB52 rules. Please, folks, do not do this at home. Be nice people and don't be evil, a thieft ...
ascii	15	0x00002E84	-	-	Can't RmGetList
ascii	25	0x00002E94	-	-	Can't RmRegisterResources
ascii	20	0x00002EB0	-	-	Can't RmStartSession

Strings of built sample

```
00000E50  2E 00 2E 00 00 00 00 00 2E 00 00 00 43 61 6E 27  ............Can'
00000E60  74 20 4F 70 65 6E 50 72 6F 63 65 73 73 00 00 00  t.OpenProcess...
00000E70  4C 41 42 35 32 20 72 75 6C 65 73 2E 20 50 6C 65  LAB52.rules..Ple
00000E80  61 73 65 2C 20 66 6F 6C 6B 73 2C 20 64 6F 20 6E  ase,.folks,.do.n
00000E90  6F 74 20 64 6F 20 74 68 69 73 20 61 74 20 68 6F  ot.do.this.at.ho
00000EA0  6D 65 2E 20 42 65 20 6E 69 63 65 20 70 65 6F 70  me..Be.nice.peop
00000EB0  6C 65 20 61 6E 64 20 64 6F 6E 27 74 20 62 65 20  le.and.don't.be.
00000EC0  65 76 69 6C 2C 20 61 20 74 68 69 65 66 74 20 61  evil,.a.thieft.a
00000ED0  6E 64 20 61 6C 6C 20 74 68 6F 73 65 20 74 68 69  nd.all.those.thi
00000EE0  6E 67 73 20 74 68 61 74 20 72 61 6E 73 6F 6D 77  ngs.that.ransomw
00000EF0  61 72 65 20 6F 70 65 72 61 74 6F 72 73 20 61 72  are.operators.ar
00000F00  65 2E 20 53 74 61 79 20 73 61 66 65 20 61 6E 64  e..Stay.safe.and
00000F10  20 74 61 6B 65 20 74 68 65 20 76 61 63 63 69 6E  .take.the.vaccin
00000F20  65 2E 20 41 6C 73 6F 2C 20 70 75 74 20 6F 6E 69  e..Also,.put.oni
00000F30  6F 6E 20 69 6E 20 73 70 61 6E 69 73 68 20 6F 6D  on.in.spanish.om
00000F40  65 6C 65 74 74 65 2E 00 00 00 00 00 00 00 00 00  elette.........
00000F50  00 00 00 00 00 00 00 00 08 78 78 78 78 78 78 78  .........xxxxxxx
00000F60  78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78  xxxxxxxxxxxxxxxx
00000F70  78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78  xxxxxxxxxxxxxxxx
00000F80  78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78  xxxxxxxxxxxxxxxx
00000F90  78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78  xxxxxxxxxxxxxxxx
00000FA0  78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78  xxxxxxxxxxxxxxxx
00000FB0  78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78  xxxxxxxxxxxxxxxx
00000FC0  78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78  xxxxxxxxxxxxxxxx
```

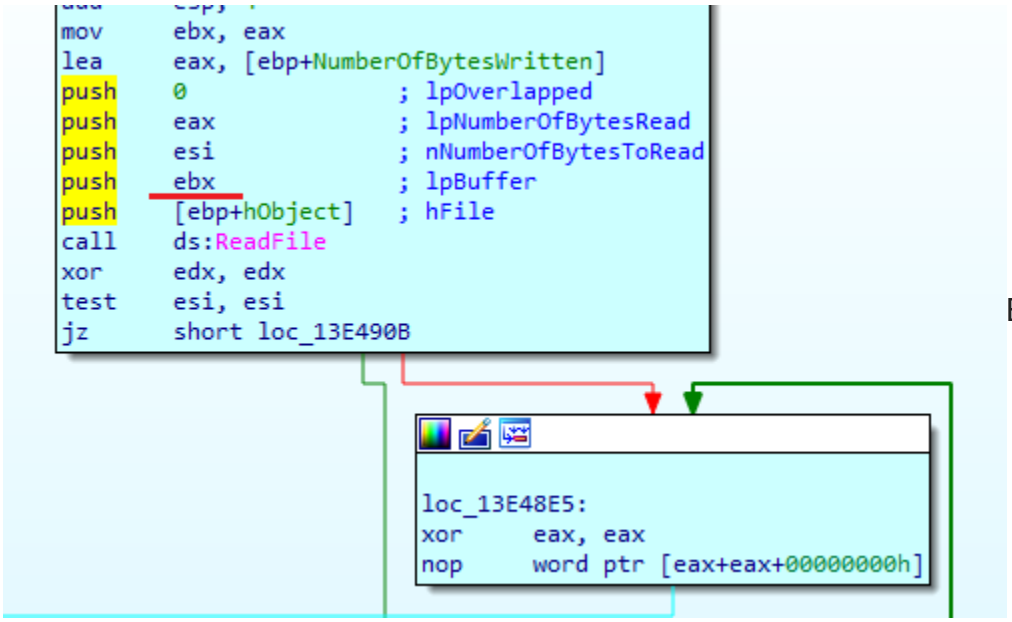
Hex content of built sample

After this, we decided to take a deeper look into the actual builder executable, and we found out that we could pass as a second argument an actual eliptic curve encyption key, instead of letting the builder generate it for us, allowing the ransomware operator to use the same decryption executable for different builds. Furthermore, it has been observed that, if no encyption key is specified as an argument, the key would be generated randomly.

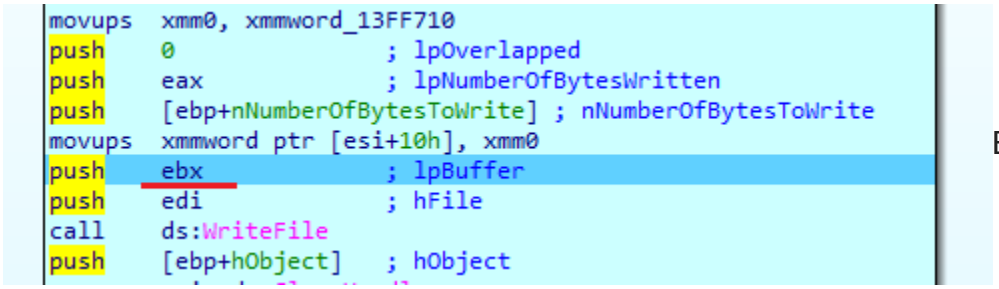


Argument parsing of Babuk builder

It could also be predictable that the builder would use the binary files as templates, and we could identify this operations within the assembly code, where it would first read the “template” file, modify it, and finally write the modification as a new file in the specified folder.



Babuk builder read of binary files as Templates



Babuk builder write of output binary files

Since Babuk binaries did not use any packer, anyone having these files and a deep knowledge about them, could have written this builder. However, according to the compilation dates which seem legit, we do not think this is the actual scenario.

About the decrypter, we have not analyzed its code, but during the tests we realized that it does not contain the eliptic curve keys hardcoded, therefore it needs to be run from a command prompt located in the same folder than these generated keys. We could also identify that it works, but it takes a ridiculous amount of time to decrypt go through the whole disk and decrypt all the files.

This could be considered important since new ransomware gangs could try to take advantage of this leak for their own Raas “startup”. However, it is also valuable for researchers since it will allow us to generate better detection rules, or even track new unofficial variations of the ransomware family.