

# Babuk Ransomware: The Builder

marcoramilli.com/2021/07/05/babuk-ransomware-the-builder/

View all posts by marcoramilli

July 5, 2021

```
030F4D0 FileName db 'C', ':', '\', 'U', 's', 'e', 'r', 's', '\', [REDACTED]
030F4D0 ; DATA XREF: sub_2F4650+42f0
030F4D0 ; sub_2F4650+6Efo ...
030F4D0 db [REDACTED] '\', 'D', 'e', 's', 'k', 't', 'o', 'p', '\', 't', 'e'
030F4D0 db 's', 't', '\', 'd', '_', 'w', 'i', 'n', '.', 'e', 'x', 'e', 0, 0, 0
030F4D0 db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

On April 2021, one of the most known Ransomware Gang called **Babuk**, decided to change the way they ask for ransom: no more double extortion, no more file encryption but just data exfiltration and a later announcement in case of no deal with the victim. It's a nice move forward for a Ransomware Gang that, as far as I know, followed the Maze Group double extortion paradigm since 2019. This is what reported on BleepingComputer ([HERE](#)) on April.

“Babuk changes direction, we no longer encrypt information on networks, we will get to you and take your data, we will notify you about it if you do not get in touch we make an announcement” – Babuk ransomware

*Babuk Ransomware Gang*

At the end of April, Babuk gang decided to definitely close their malicious operations making their Babuk Ransomware opensource, but so far no code was shared to the community.

Many Babuk ransomware where disclosed and analyzed during the following weeks, but when I saw this (reference follows) sample called “builder” with static signatures that reminded me the Babuk Ransomware, I decided to take a closer look checking if it definitely was the Babuk builder.

One-Time  
Monthly

**Make a one-time donation**

---

**Make a monthly donation**

---

Choose an amount

- \$1.00
- \$5.00
- \$10.00
- \$5.00
- \$15.00
- \$100.00

If you think this content is helpful, please consider to make a little donation. It would help me in building and writing additional contributions to community. By donation you will contribute to community as well. Thank you !

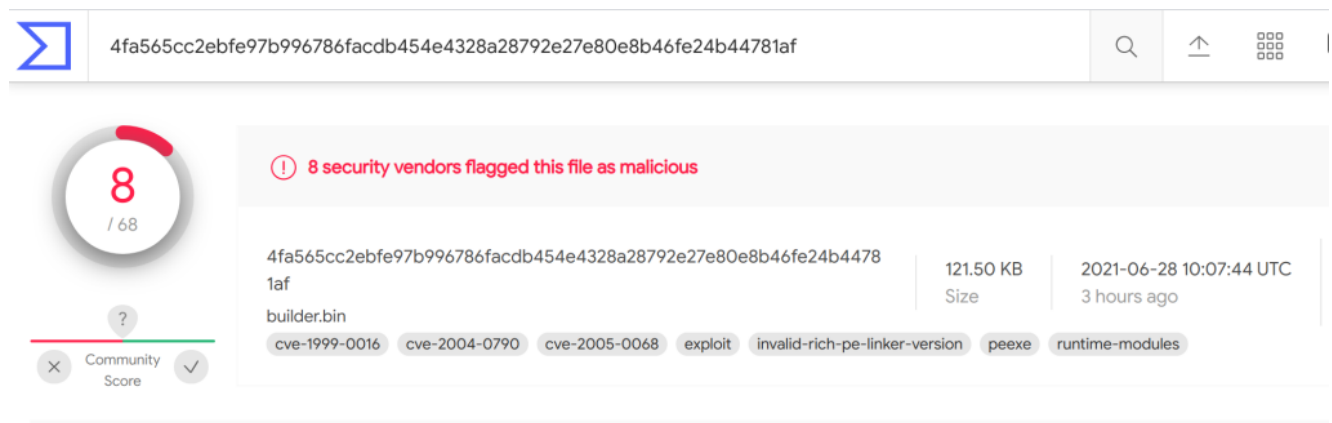
If you think this content is helpful, please consider to make a little donation. It would help me in building and writing additional contributions to community. By donation you will contribute to community as well. Thank you !

[Donate](#) [Donate monthly](#)

## Builder Analysis

So, let's take a closer look to this sample called `builder` matching signature from Babuk Ransomware. Sha256:

`4fa565cc2ebfe97b996786facdb454e4328a28792e27e80e8b46fe24b44781af` , from the time of the analysis, the sample is recognized as malicious from 8 (over 68) AV vendors. I believe it is a nice starting point to investigate our building chain ;).



The screenshot shows the VirusTotal interface for a file named `builder.bin`. The file's SHA256 hash is `4fa565cc2ebfe97b996786facdb454e4328a28792e27e80e8b46fe24b44781af`. The file size is 121.50 KB and it was uploaded on 2021-06-28 10:07:44 UTC (3 hours ago). A red warning icon indicates that 8 security vendors flagged this file as malicious. Below this, a list of detected signatures is shown: `cve-1999-0016`, `cve-2004-0790`, `cve-2005-0068`, `exploit`, `invalid-rich-pe-linker-version`, `peexe`, and `runtime-modules`. On the left, a circular progress indicator shows a score of 8 out of 68, with a 'Community Score' section below it.

## BabukRansomwareBuilder on VT

Given Name	builder.bin
SHA256	4fa565cc2ebfe97b996786facdb454e4328a28792e27e80e8b46fe24b44781af
Technology	Microsoft C/C++
Behavior	Ransomware Builder

The sample appears to be not packed. It means that on building chain the attacker need to use an external packer. It might underline a non sophisticated developer chain, in other words, the developer may have no expertise building packers or he/she decided to use external obfuscators/packers during the delivery phase of the derivated artifacts. The `builder` is a CLI utility asking for a folder. Once you give a folder name it firstly check if the folder exists, and if it doesn't, the sample creates it. It later builds up the pair keys by using elliptic curve algorithm with a randomly generated 256 key size.

```

loc_404A43:
push    dword ptr [esi+4]
push    offset aCreatingFolder ; "Creating folder '%s'\n"
call    sub_404620             ; Call Procedure
add     esp, 8                 ; Add
push    0                     ; lpSecurityAttributes
push    dword ptr [esi+4] ; lpPathName
call    ds:CreateDirectoryA   ; Indirect Call Near Procedure
push    dword ptr [esi+4] ; lpString2
mov     ebx, ds:lstrcpyA
push    offset String1 ; lpString1
call    ebx ; lstrcpyA ; Indirect Call Near Procedure
push    dword ptr [esi+4] ; lpString2
push    offset byte_41F5D8 ; lpString1
call    ebx ; lstrcpyA ; Indirect Call Near Procedure
push    offset String2 ; "\\kp.curve25519"
push    offset String1 ; lpString1
call    ds:lstrcatA ; Indirect Call Near Procedure
push    offset aKsCurve25519 ; "\\ks.curve25519"
push    offset byte_41F5D8 ; lpString1
call    ds:lstrcatA ; Indirect Call Near Procedure
cmp     edi, 2 ; Compare Two Operands
jnz    loc_404B26 ; Jump if Not Zero (ZF=0)

```

#### Private and Public Elliptic Curve Keys

```

push    offset Buffer ; pBuffer
push    32 ; dwLen
push    eax ; hProv
call    ds:CryptGenRandom ; Indirect Call Near Procedure

```

#### Random Key Generation For Elliptic Curve 32×8 (256) Key

After the key generation phase the builder saves such a keys inside the given folder in two separated files : `kp.curve25519` and `ks.curve25519` which are public and secret parameters for the Montgomery curve. The builder then checks for components in the current folder in order to build the output samples. The needed components are:

- `note.txt` : a simple text file wrapping ransom note.
- `e_esxi.out` , `e_nas_x86.out` , `e_nas_arm.out` , `e_win.bin` : specific encryption payloads for different targets (ESXI, NAS and Windows Machines)

- `d_esxi.out` , `d_nas_x86.out` , `d_nas_arm.out` , `d_win.bin` : specific decryption payloads for different targets (ESXI, NAS and Windows Machines)

The following image shows the main builder function in where it looks for external files (representing payloads) and saves them on local hard drive ready to be implemented into the victims system.

```

loc_404790:                                     ; lpString1
push      dword ptr [eax+4]
push      offset byte_41F3C8 ; lpString1
call     ebx ; lpString1 ; Indirect Call Near Procedure
mov      edi, #1 ; lpString1
push      offset a51a5b5a ; "via_win.exe"
push      offset byte_41F3C8 ; lpString1
call     edi ; lpString1 ; Indirect Call Near Procedure
mov      ecx, offset a51a5b5a ; "e_win_bin"
call     sub_404790 ; Call Procedure
push      dword ptr [eax+4] ; lpString2
push      offset fileName ; lpString1
call     ebx ; lpString1 ; Indirect Call Near Procedure
push      offset a51a5b5a ; "via_win.exe"
push      offset fileName ; lpString1
call     edi ; lpString1 ; Indirect Call Near Procedure
mov      ecx, offset a51a5b5a ; "d_win_bin"
call     sub_404550 ; Call Procedure
push      dword ptr [eax+4] ; lpString2
push      offset byte_41F3C8 ; lpString1
call     ebx ; lpString1 ; Indirect Call Near Procedure
push      offset a51a5b5a ; "via_exe1.exe"
push      offset byte_41F3C8 ; lpString1
call     edi ; lpString1 ; Indirect Call Near Procedure
mov      ecx, offset a51a5b5a ; "e_exe1.out"
call     sub_404790 ; Call Procedure
push      dword ptr [eax+4] ; lpString2
push      offset fileName ; lpString1
call     ebx ; lpString1 ; Indirect Call Near Procedure
push      offset a51a5b5a ; "via_exe1.exe"
push      offset fileName ; lpString1
call     edi ; lpString1 ; Indirect Call Near Procedure
mov      ecx, offset a51a5b5a ; "d_exe1.out"
call     sub_404550 ; Call Procedure
push      dword ptr [eax+4] ; lpString2
push      offset byte_41F3C8 ; lpString1
call     ebx ; lpString1 ; Indirect Call Near Procedure
push      offset a51a5b5a ; "via_exe_x86.out"
push      offset byte_41F3C8 ; lpString1
call     edi ; lpString1 ; Indirect Call Near Procedure
mov      ecx, offset a51a5b5a ; "e_exe_x86.out"
call     sub_404790 ; Call Procedure
push      dword ptr [eax+4] ; lpString2
push      offset fileName ; lpString1
call     ebx ; lpString1 ; Indirect Call Near Procedure
push      offset a51a5b5a ; "via_exe_x86.exe"
push      offset fileName ; lpString1
call     edi ; lpString1 ; Indirect Call Near Procedure
mov      ecx, offset a51a5b5a ; "d_exe_x86.out"
call     sub_404550 ; Call Procedure
push      dword ptr [eax+4] ; lpString2
push      offset byte_41F3C8 ; lpString1
call     ebx ; lpString1 ; Indirect Call Near Procedure
push      offset a51a5b5a ; "via_exe_arm.out"
push      offset byte_41F3C8 ; lpString1
call     edi ; lpString1 ; Indirect Call Near Procedure
mov      ecx, offset a51a5b5a ; "e_exe_arm.out"
call     sub_404790 ; Call Procedure
push      dword ptr [eax+4] ; lpString2
push      offset fileName ; lpString1
call     ebx ; lpString1 ; Indirect Call Near Procedure
push      offset a51a5b5a ; "via_exe_arm.exe"
push      offset fileName ; lpString1
call     edi ; lpString1 ; Indirect Call Near Procedure
mov      ecx, offset a51a5b5a ; "d_exe_arm.out"
call     sub_404550 ; Call Procedure
mov      edi, #1 ; CreateFile
push      0 ; dwFlagsAndAttributes
push      0 ; dwCreationDisposition
push      0 ; dwDesiredAccess
push      1 ; dwShareMode
push      0 ; dwSecurityAttributes
push      offset str1 ; lpFileName
mov      [ebp+0F70v], 0
call     edi ; CreateFile ; Indirect Call Near Procedure
mov      ebx, eax
cmp      ebx, #FFFFFFFF ; Compare Two Operands
jr      loc_404791 ; Jump If Zero (JZ)

```

Babuk main building function

The ransomware generation is a simple process. It firstly takes the external payload and it later starts a `lstrcp` (which is dynamically loaded) to copy the external payload to files, implementing the final ransomware. The following image shows the main saving function.

```

EIP> 1473E movups xmm0, xmmword_F2F6F0 ; Move Unaligned Four Packed Single-FP
14745 push edi ; lpBuffer
14746 mov edi, [ebp+hObject]
14749 push edi ; hFile
1474A movups xmmword ptr [esi+10h], xmm0 ; Move Unaligned Four Packed Single-FP
1474E call ds:WriteFile ; Indirect Call Near Procedure
14754 push [ebp+hFile] ; hObject
14757 mov esi, ds:CloseHandle
1475D call esi ; CloseHandle ; Indirect Call Near Procedure
1475F push edi ; hObject
14760 call esi ; CloseHandle ; Indirect Call Near Procedure
14762 push offset FileName

```

### Main Saving Function

The following images show the variables wrapping out the file name (on the left) and the result of the “stringCopy” function (on the right) before saving them to the hard drive.

```

030F4D0 FileName db 'C', ':', '\', 'U', 's', 'e', 'r', 's', '\', ██████████
030F4D0 ; DATA XREF: sub_2F4650+42fo
030F4D0 ; sub_2F4650+6Efo ...
030F4D0 db ██████████ '\', 'D', 'e', 's', 'k', 't', 'o', 'p', '\', 't', 'e'
030F4D0 db 's', 't', '\', 'd', '-', 'w', 'i', 'n', '.', 'e', 'x', 'e', 0, 0, 0
030F4D0 db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

```

---

```

00560840 D5 6A D6 54 5E 1E 00 18 4D 5A 90 00 03 00 00 00 0jÖT^...MZ .....
00560850 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 ....ÿÿ.....
00560860 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 @.....
00560870 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00560880 00 00 00 00 D8 00 00 00 0E 1F BA 0E 00 B4 09 CD ....0.....º..î
00560890 21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72 !.LÍ!This·progr
005608A0 61 6D 20 63 61 6E 6E 6F 74 20 62 65 20 72 75 6E am·cannot·be·run
005608B0 20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A ·in·DOS·mode...
005608C0 24 00 00 00 00 00 00 00 EC 62 1D AF A8 03 73 FC $......ìb.~.sü

```

Once the building function terminates its run the user (actually the attacker) finds the given folder full of ransomware ready to be deployed to victims. The following image shows the built ransomware which happens to be ready to be spread on `Windows` system, `Linux` systems on `ESX` and specific `NAS` within ARM core.

Name	Date modified	Type
d_esxi.out	6/29/2021 8:47 AM	OUT File
d_nas_arm.out	6/29/2021 8:47 AM	OUT File
d_nas_x86.out	6/29/2021 8:47 AM	OUT File
d_win.exe	6/29/2021 8:47 AM	Application
e_esxi.out	6/29/2021 8:47 AM	OUT File
e_nas_arm.out	6/29/2021 8:47 AM	OUT File
e_nas_x86.out	6/29/2021 8:47 AM	OUT File
e_win.exe	6/29/2021 8:09 AM	Application
kp.curve25519	6/29/2021 8:47 AM	CURVE25519 File
ks.curve25519	6/29/2021 8:47 AM	CURVE25519 File

Final Built Folder

## Conclusion

Babuk Builder looks like to be a quite simple piece of code. But even if it's a simple code it holds some interesting characteristics which could be helpful to compare to future builders. For example the loading sections and the way the builder uses the `lstrcp` function to copy the loaded payloads. The main building loop and the used algorithm to generate key-pairs.

Name	Sha256
d_esxi.out	a7a832dd999f4d147087231731ac040be03a26859cfc03f948b092b5a8c259d6
d_nas_arm.out	fa2b76dde88f2306b280586b5cf40671f4f08b83e9e095f7d52608e6ed1dd7bf
d_nas_x86.out	45b26897e7d81f2e1905cbb3d227a94bc7991f14a4a24f4aa4752083602be41e
d_win.exe	0221b06e7aa462206039db6366bee9b31838d736dff9145ee54811e2abee7128
e_esxi.out	1afd6bbf62fa0906da0fc4ebd55bf7339aeb3d8beb539df9be4d016efabf3a12
e_nas_arm.out	1a3b213f0303ff5f676df39217abae197f8af689de4c884cd0acc96aedb1a328
e_nas_x86.out	af2727be8ff8eb40b4e6eb0ba3c3d0594e4e902e698875a0ef4e3a4ef06f2a86
e_win.exe	3d554fe3ed824df5bb625bcff4ddde834866164088358ddabc4e5aec8a6562b0

Babuk Builder IoC