Babuk Ransomware

chuongdong.com/reverse engineering/2021/01/03/BabukRansomware/

Chuong Dong

January 3, 2021

Reverse Engineering · 03 Jan 2021

Overview

This is my report for the new Babuk Ransomware that recently appears at the beginning of 2021.

Since this is the first detection of this malware in the wild, it's not surprising that Babuk is not obsfuscated at all. Overall, it's a pretty standard ransomware that utilizes some of the new techniques we see such as multi-threading encryption as well as abusing the Windows Restart Manager similar to Conti and REvil.

For encrypting scheme, Babuk uses its own implementation of SHA256 hashing, ChaCha8 encryption, and Elliptic-curve Diffie–Hellman (ECDH) key generation and exchange algorithm to protect its keys and encrypt files. Like many ransomware that came before, it also has the ability to spread its encryption through enumerating the available network resources.

alt text

Figure 1: RaidForums Babuk leak

IOCS

Babuk Ransomware comes in the form of a 32-bit *.exe* file.

MD5: e10713a4a5f635767dcd54d609bed977

SHA256: 8203c2f00ecd3ae960cb3247a7d7bfb35e55c38939607c85dbdb5c92f0495fa9

Sample:
https://bazaar.abuse.ch/sample/8203c2f00ecd3ae960cb3247a7d7bfb35e55c38939607c85dbdb5c92f0495fa9/

alt text

Figure 2: VirusTotal result

Ransom Note

alt text

Figure 3: Babuk's ransom note

alt text

Figure 4: Babuk's Website

(Pretty unprofessional from the Babuk team since they did not remove the chat log between them and a victim)

Code Analysis

Command-line Arguments

Babuk can work with or without command line paramters. If no parameter is given, it’s restricted to only encrypting the local machines.

alt text

Figure 5: Argument parsing

If a parameter is given, it will process these arguments upon execution and behave accordingly.

CMD Args	Functionality
-lanfirst	Same as no parameter given, encrypting locally
-lansecond	Encrypting network shares after encrypting locally
-nolan	Same as no parameter given, encrypting locally

Terminating Services

Babuk’s authors hard-coded a list of services to be closed before encryption.

Before terminating a service, Babuk will calls **EnumDependentServicesA** to retrieve the name and status of each service that depends on that specified service.

It will then call **ControlService** with the control code *SERVICE_CONTROL_STOP* to stop them before terminating the main service the same way.

alt text

Figure 6: Terminating serivces

Here is the list of services to be closed.

```
vss, sql, svc$, memtas, mepocs, sophos, veeam, backup, GxVss, GxBlr, GxFWD, GxCVD, GxCIMgr,
DefWatch, ccEvtMgr,
ccSetMgr, SavRoam, RTVscan, QBFCService, QBIDPService, Intuit.QuickBooks.FCS, QBCFMonitorService,
YooBackup,
YooIT, zhudongfangyu, sophos, stc_raw_agent, VSNAPVSS, VeeamTransportSvc, VeeamDeploymentService,
VeeamNFSSvc,
veeam, PDVFSService, BackupExecVSSProvider, BackupExecAgentAccelerator, BackupExecAgentBrowser,
BackupExecDiveciMediaService, BackupExecJobEngine, BackupExecManagementService,
BackupExecRPCService,
AcrSch2Svc, AcronisAgent, CASAD2DWebSvc, CAARCUpdateSvc,
```

Terminating Running Processes

The author also hard-coded a list of processes to be closed.

Using calls to **CreateToolhelp32Snapshot**, **Process32FirstW**, and **Process32NextW** to examine all of the processes running on the system, Babuk can loop through and look for processes needed to be closed. Upon finding any, it will call **TerminateProcess** to terminate it.

alt text

Figure 7: Terminating processes

Here is the list of processes to be closed.

```
sql.exe, oracle.exe, ocssd.exe, dbsnmp.exe, synctime.exe, agntsvc.exe, isqlplussvc.exe,
xfssvccon.exe, mydesktopservice.exe, ocautoupds.exe, encsvc.exe, firefox.exe, tbirdconfig.exe,
mydesktopqos.exe, ocomm.exe, dbeng50.exe, sqbcoreservice.exe, excel.exe, infopath.exe,
msaccess.exe,
mspub.exe, onenote.exe, outlook.exe, powerpnt.exe, steam.exe, thebat.exe, thunderbird.exe,
visio.exe, winword.exe, wordpad.exe, notepad.exe
```

Deleting Shadow Copies

Babuk attempts to delete shadow copies before and after encryption.

First, it calls **Wow64DisableWow64FsRedirection** to disables file system redirection before calling **ShellExecuteW** to execute this command

```
cmd.exe /c vssadmin.exe delete shadows /all /quiet
```

After deleting the shadow copies, Babuk checks if the system is running under an 64-bit processor. If it is, then **Wow64RevertWow64FsRedirection** is called to enable file system redirection again.

alt text

Figure 8: Deleting Shadow Copies

Encryption

Key Generation

First, Babuk uses **RtlGenRandom** to generate 4 random buffers. Two of which are used as ChaCha8 keys, and the other two are used as ChaCha8 nonces.

alt text

Figure 9: Randomly generating ChaCha8 keys and nonce

Next, it will encrypt the second ChaCha8 key using the first key and nonce. After that, the first key is then encrypted using the encrypted second key and nonce.

This encrypted first key is treated as the Elliptic-curve Diffie–Hellman (ECDH) private key for the local machine.

alt text

Figure 10: Randomly generating ECDH private key

From here, Babuk generate a local ECDH public key from the private key using the code from this ECDH library.

Then, it generates a shared secret using the local private key and the author’s hard-coded public key.

This shared secret goes thorugh a SHA256 hashing algorithm to generate 2 ChaCha8 keys, which are used to encrypt files later.

In order to be able to decrypt files, Babuk stores the local public key in the file **ecdh_pub_k.bin** in the **APPDATA** folder.

Because of ECDH’s mechanism, the ransomware author can generate the shared secret using his own private key and the victim’s public key to decrypt files. This makes it impossible for the victim to decrypt on their own unless they can capture the randomly-generated private key in the malware before it finishes encryting.

alt text

Figure 11: Generating ChaCha8 keys from ECDH shared secret

Multithreading

From a programmer’s point of view, Babuk’s approach to multithreading is pretty mediocre.

First, it determines the number of threads to spawn by doubling the number of cores on the victim’s machine and allocates an array to store all of the thread handles.

alt text

Figure 12: Thread initialization

The first problem with this approach has to do with thread’s concurrency in an OS. A huge amount of threads can potentially be created for each process. However, in an ideal situation, it’s better to have one thread running per processor to avoid having threads competing with each other for the processor’s time and resource during encryption.

However, that, by itself, is not that big of a problem if the author implemented a queue-like structure to process encrypting requests to utilize 100% of the victim processing power. Unfortunately, they decided to only spawn one encrypting thread per existing drive.

alt text

Figure 13: Launching encrypting threads

In the case where the number of drives is less than the number of processors (which is highly likely), Babuk won’t spawn as many threads as possible to encrypt.

Since each thread is responsible for an entire drive, this forces it to use the traditional recursive approach to traverse through its own folders, which results in a longer encryption time due to the huge workload.

The workload for each thread varies based on the size of the drive it’s encrypting, so the average encrypting time will just be approximately near the time it takes for one thread to encrypt the largest drive. This is inefficient and really defeats the purpose of using multithreading to encrypt drives.

Folder Traversing

As discussed above, Babuk uses a recursion method to traverse and encrypt files. Using **FindFirstFileW** and **FindNextFileW** calls, it goes through each directory to look for files and sub-directories.

When encountering a directory, it recursively calls the **main_encrypt** function again. However, Babuk only goes down 16 directory layers deep, so it potentially does not encrypt every single folders in the drive to save time.

When encountering a file, it will check if the file name is **How To Restore Your Files.txt** or if the file extension is **.__NIST_K571__** to avoid encrypting the ransom note or encrypted files.

alt text

Figure 14: Traversing through folders

Kill File Owner

Similar to Conti or REvil ransomware, Babuk utilizes the Windows Restart Manager to terminate any process that is using files. This ensures that nothing prevents it from opening and encrypting the files.

This is accomplished through the calls **RmStartSession**, **RmRegisterResources**, and **RmGetList** to get a list of processes that are using the a specified file. If the process is not **explorer.exe** or a critical process, then Babuk will call **TerminateProcess** to kill it.

alt text

Figure 15: Killing processes that are using files

File Encryption

Babuk's file encryption is divided into 2 different types - small file encryption and large file encryption.

For small files that are les than 41943040 bytes or roughly 41 MB in size, the file is mapped entirely and encrypted with ChaCha8 two times.

alt text

Figure 16: Small file encryption

With large files, encryption is a bit different. To save time, the entire file is divided into three equally-large regions.

For each of these regions, only the first 10485760 bytes or 10 MB will be encrypted.

alt text

Figure 17: Large file encryption

For encryption, Babuk uses the two ChaCha8 keys generated from the ECDH shared secret's SHA256 hash as the encrypting keys and the first 12 bytes of the shared secret as nonce.

Remote File Encryption

To encrypt the remote drives from the victim machine, Babuk calls **WNetGetConnectionW** to retrieves the name of the network resources associated with those drives and pass them to the encrypting thread.

alt text

Figure 18: Encrypting remote drives

It also encrypts network shares on the machine’s LAN given the correct parameter.

Babuk calls **WNetOpenEnumW** and **WNetOpenEnumW** to traverse through remote folders on the network and encrypts file using the similar recursive method mentioned above.

alt text

Figure 19: LAN Encryption

Key Findings

Babuk is a new ransomware that started at the beginning of this year. Despite the amateur coding practices used, its strong encryption scheme that utilizes Elliptic-curve Diffie–Hellman algorithm has proven effective in attacking a lot of companies so far.

Because the malware authors are using one private key for each Babuk sample, it’s clear that their main target is large corporations instead of normal computer users. So far, according to the website embedded in the ransom note as well as the leaks on **Raidforums**, they have sucessfully compromised 5 different companies in the world.

Message to newer victims

I recently notice I’m getting a lot more traffic from Europe on this page, which I’m assuming newer victims are viewing this to better their understanding of the ransomware.

This blog post is really out of date because Babuk has evolved a lot, and the malware is drastically different from what I talk about here.

If recent Babuk victims are interested in getting more information about the newer version of this ransomware or require any assistance with analyzing any sample, feel free to reach out to me through my email **cdong49@gatech** or Twitter!

YARA Rule

```
rule BabukRansomware {
        meta:
                description = "YARA rule for Babuk Ransomware"
                reference =
"http://chuongdong.com/reverse%20engineering/2021/01/03/BabukRansomware/"
                author = "@cPeterr"
                date = "2021-01-03"
                rule_version = "v1"
                malware_type = "ransomware"
                tlp = "white"
        strings:
                $lanstr1 = "-lanfirst"
                $lanstr2 = "-lansecond"
                $lanstr3 = "-nolan"
                $str1 = "BABUK LOCKER"
                $str2 = ".__NIST_K571__" wide
                $str3 = "How To Restore Your Files.txt" wide
                $str4 = "ecdh_pub_k.bin" wide
        condition:
                all of ($str*) and all of ($lanstr*)
}
```

References

https://twitter.com/Arkbird_SOLG/status/1345569395725242373