REPORT

McAfee

BABUK: MOVING TO VM AND *NIX SYSTEMS BEFORE STEPPING AWAY

TABLE OF CONTENTS

INTRODUCTION

For a long time, ransomware gangs were mostly focused on Microsoft Windows operating systems. Yes, we observed the occasional dedicated Unix or Linux based ransomware, but cross-platform ransomware was not happening yet. However, cybercriminals never sleep and in recent months we noticed that several ransomware gangs were experimenting with writing their binaries in the cross-platform language Golang (Go).

Our worst fears were confirmed when Babuk announced on an underground forum that it was developing a cross-platform binary aimed at Linux/UNIX and ESXi or VMware systems. Many core backend systems in companies are running on these *nix operating systems or, in the case of virtualization, think about the ESXi hosting several servers or the virtual desktop environment.

We touched upon this briefly in our previous blog, together with the many coding mistakes the Babuk team is making.

Even though Babuk is relatively new to the scene, its affiliates have been aggressively infecting high-profile victims despite numerous problems with the binary.

In our first Babuk blog McAfee Advanced Threat Research (ATR), and our industry peers, discussed some of the Windows binary issues. It seems that Babuk has adopted live beta testing on its victims when it comes to its Golang binary and decryptor development. We have seen several victims' machines encrypted beyond repair due to either a faulty binary or a faulty decryptor.

AUTHORS

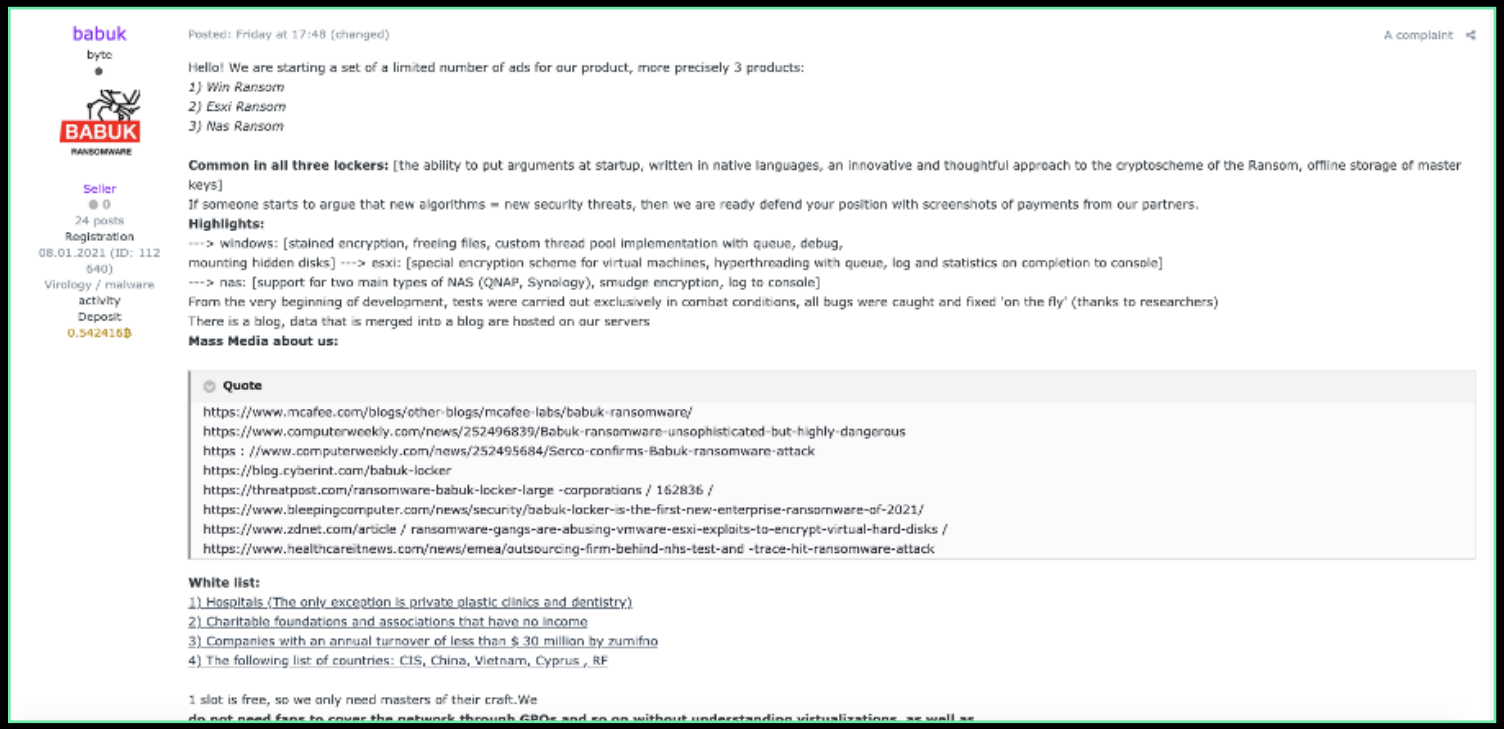
This report was researched and written by:

- Thibault Seret
- Noël Keijzer, Northwave

Subscribe to receive threat information.

CONNECT WITH US

Even if a victim gave in to the demands and was forced to pay the ransom, they still could not get their files back. We strongly hope that the bad coding also affects Babuk's relationship with its affiliates. The affiliates are the ones performing the actual compromise and are now faced with a victim that cannot get their data back even if they pay. This essentially changes the crime dynamic from extortion to destruction which, from a criminal's point of view, is much less profitable.



babuk
byte
Seller
0
24 posts
Registration
08.01.2021 (ID: 112 640)
Virology / malware activity
Deposit
0.542416฿

Posted: Friday at 17:48 (changed)
A complaint

Hello! We are starting a set of a limited number of ads for our product, more precisely 3 products:
1) Win Ransom
2) Esxi Ransom
3) Nas Ransom

Common in all three lockers: [the ability to put arguments at startup, written in native languages, an innovative and thoughtful approach to the cryptoscheme of the Ransom, offline storage of master keys]
If someone starts to argue that new algorithms = new security threats, then we are ready defend your position with screenshots of payments from our partners.
Highlights:
---> windows: [stained encryption, freeing files, custom thread pool implementation with queue, debug, mounting hidden disks] ---> esxi: [special encryption scheme for virtual machines, hyperthreading with queue, log and statistics on completion to console]
---> nas: [support for two main types of NAS (QNAP, Synology), smudge encryption, log to console]
From the very beginning of development, tests were carried out exclusively in combat conditions, all bugs were caught and fixed 'on the fly' (thanks to researchers)
There is a blog, data that is merged into a blog are hosted on our servers
Mass Media about us:

Quote
https://www.mcafee.com/blogs/other-blogs/mcafee-labs/babuk-ransomware/
https://www.computerweekly.com/news/252496839/Babuk-ransomware-unsophisticated-but-highly-dangerous
https : //www.computerweekly.com/news/252495684/Serco-confirms-Babuk-ransomware-attack
https://blog.cyberint.com/babuk-locker
https://threatpost.com/ransomware-babuk-locker-large -corporations / 162836 /
https://www.bleepingcomputer.com/news/security/babuk-locker-is-the-first-new-enterprise-ransomware-of-2021/
https://www.zdnet.com/article / ransomware-gangs-are-abusing-vmware-esxi-exploits-to-encrypt-virtual-hard-disks /
https://www.healthcareitnews.com/news/emea/outsourcing-firm-behind-nhs-test-and -trace-hit-ransomware-attack

White list:
1) Hospitals (The only exception is private plastic clinics and dentistry)
2) Charitable foundations and associations that have no income
3) Companies with an annual turnover of less than $ 30 million by zumifno
4) The following list of countries: CIS, China, Vietnam, Cyprus , RF

1 slot is free, so we only need masters of their craft.We
do not need fans to cover the network through GPOs and so on without understanding virtualizations, as well as

FIGURE 1. POST FROM BABUK ABOUT THE LINUX VERSION OF THE RANSOMWARE

BABUK: MOVING TO VM AND *NIX SYSTEMS BEFORE STEPPING AWAY

THE BABUK THREAT ACTOR

Before giving an overview of the methodology used by the Babuk threat actor, some general background knowledge of how ransomware attacks occur and what groups are behind the attacks is necessary.

Following, we will first describe the typical phases of a ransomware attack, as well as the Ransomware-as-a-Service model that is used by Babuk ransomware. Subsequently we will show what a typical Babuk ransomware attack looks like, and what specific Threats, Tactics, and Procedures are used by the Babuk threat actor. Finally, a technical analysis of the ransomware employed by the threat actor will show that there are many flaws found within the code that result in the destruction of victim's data.

AUTHORS

This report was researched and written by:

- Thibault Seret
- Noël Keijzer, Northwave

PHASES OF A RANSOMWARE ATTACK

During a cyberattack the different steps an attacker performs can be categorised in three main categories, which we will use to describe a typical attack of a Babuk threat actor:

- Initial access (IN)
- Network propagation (THROUGH)
- Action on objectives (OUT)

IN	THROUGH	OUT
Exploiting a vulnerability from an internet-connected server. Various tools are often available for known vulnerabilities.	**Lateral movement.** Reconnaissance of the network (through vulnerable systems or insufficient access control). In this stage, the attacker hops from computer to computer.	**Encrypt files and demand ransom.** The attacker applies a cryptographic function on the files - the attacker demands the purchase of this key which is needed to revert the encryption.
Phishing email. An email with a rogue attachment or link. These emails can be tailor-made and are hard to distinguish from legitimate emails.	**Privilege escalation.** Becoming an admin through password guessing, unsafe user rights, or vulnerabilities. The attacker tries to get more rights on a computer and thus more possibilities. An attacker uses lateral movement and privilege escalation often alternately.	**Downloading sensitive data - threatening to publish.** Stealing business-sensitive data or personal data from the server. The victim must pay to avoid publication which integrates seamlessly with encryption.
Remote access software - vulnerable password. Attackers can try to guess the password if they can log in from the Internet.		**Destroying or corrupting backups.**

FIGURE 2. RANSOMWARE ATTACK PHASES

RANSOMWARE-AS-A-SERVICE SUPPLY CHAIN

Lately, there is an increasing trend observed in the cybercriminal industry called "Ransomware-as-a-Service (RaaS)." RaaS is a business model that is increasing in popularity among ransomware authors.[1] RaaS is a service, offered by ransomware developers that allows cybercriminals to rent ransomware. RaaS aims to simplify ransomware attacks for criminals that lack the technical skills to build their own ransomware in exchange for a part of the ransom acquired by the criminals. This business model allows many ransomware developers to collaborate with other seasoned cybercriminals that can distribute ransomware in large networks to which they already have access. Babuk ransomware made use of such a model before shutting down its ransomware operations at the end of April 2021.

RaaS is transforming the way a ransomware attacks works, involving several distinct actors. Generally, we can divide the supply chain for such attacks in to four stages, as shown below.

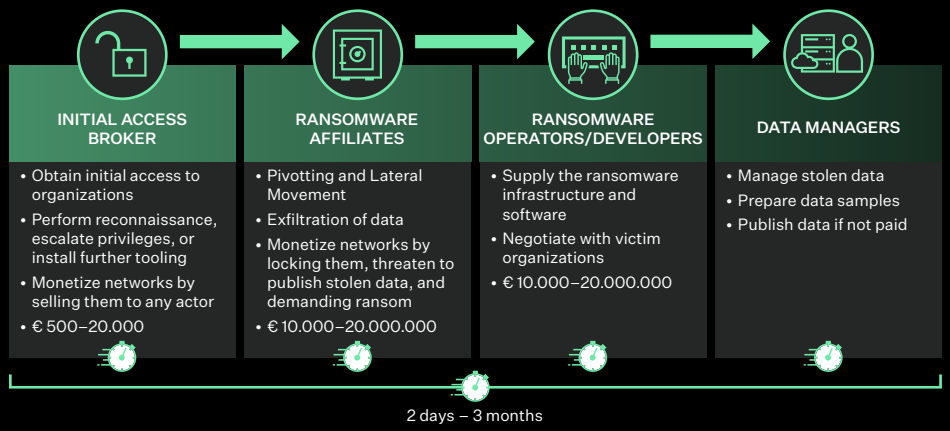


FIGURE 3. SUPPLY CHAIN SCHEMA

Technique	Tactic	Observable
Exploit public-facing application (T1190)	Initial access	Exploitation of Exchange server using CVE-2021-27065
Valid accounts (T1078)	Persistence	Group makes use of legitimate domain administrator credentials for most of their activity
Create account (T1136)	Persistence	Creates domain account named Petr
Exploitation for privilege escalation (T1068)	Privilege escalation	Zerologon exploit to obtain domain admin account, Mimikatz to obtain additional account credentials
Impair Defenses (T1562)	Defense Evasion	Usage of GMER rootkit remover to disable anti-virus solutions
Account discovery (T1087)	Discovery	Usage of ADFind to list all accounts in the domain
Remote System Discovery (T1018)	Discovery	Usage of ADFind to list all systems in the domain
Remote System Discovery (T1018)	Discovery	Usage of NetScan to identify systems on the network
File and Directory Discovery (T1083)	Discovery	Usage of LAN Search Pro to find files located on network shares
Remote Services (T1021)	Lateral Movement	Usage of RDP and SSH to move between systems
Lateral Tool Transfer (T1570)	Lateral movement	Usage of WinSCP to transfer files to linux systems
Multi-Stage Channels (T1104)	Command and Control	Usage of Cobalt strike to obtain persistence within the environment
Archive Collected Data (T1560)	Collection	Usage of WinRAR to archive data prior to exfiltration
Exfiltration Over Web Service, Transfer Data to Cloud Storage (T1567.002)	Exfiltration	Data exfiltration to MEGA using the MegaSync application as well as data exfiltration to Google Drive using Google Chrome
Data Encrypted for Impact (T1486)	Impact	Encrypt systems using ransomware

FIGURE 4. MITRE MATRIX

A TYPICAL ATTACK BY A BABUK THREAT ACTOR

IN

During a Babuk attack that Northwave investigated, the threat actor gained access to the victim's network by exploiting a vulnerability on an internet-facing server, in this case CVE-2021-27065. This vulnerability was one that was actively being exploited by the HAFNIUM threat actor before being patched by Microsoft. After a patch was issued, the vulnerability was picked up by several threat groups and Northwave has seen this vulnerability being abused by different ransomware threat actors in several investigations.

Upon gaining entry to the victim's network, the attacker remained dormant for over a week. We suspect this was because the party that gained access to the network was an initial access broker, selling access to ransomware affiliates.

THROUGH

As mentioned above, the attacker did not start reconnaissance and lateral movement on the victim's systems until a week after the initial compromise. In the paragraphs below we will describe the methodology that the threat actor used to gain complete control of the environment.

After gaining access, the threat actor placed a Cobalt Strike backdoor on the system. Cobalt Strike is frequently employed by attackers to obtain persistence on their victim's networks. Northwave found that the threat actor placed Cobalt Strike backdoors on several key systems within the network. Furthermore, Northwave found that the attacker made use of GMER, which is a rootkit removal tool. This tool was likely used to remove or disable anti-virus solutions on the victim's system. The threat actor was also found to have downloaded Metasploit, though they did not actually use it during the attack on this victim.

The threat actor used a custom version of zer0dump[2] to obtain domain administrator credentials. This tool uses the Zerologon[3] exploit to elevate privileges by compromising the domain controller. The threat actor did not create a new domain account, neither did they change the credentials of any existing accounts. Instead, the threat actor opted to use already existing domain admin accounts, with their original credentials. The threat actor used Mimikatz to obtain access to credentials of other domains present on the victim's network. During later stages of the attack, the threat actor opted to create a new local administrator account on some of the systems as a means of additional persistence.

Lateral movement between Windows systems was achieved using RDP. For connections to Linux systems, the attacker made use of SSH (using Putty). Moving files to Linux systems was done using WinSCP from Windows systems, while tools used on Windows systems were downloaded from the internet. The threat actor made use of the "temp.sh" and "wdfiles.ru" file hosting websites to host most of his tools. Other tools were downloaded directly from GitHub or the websites of their respective developers.
The attacker downloaded a tool to scan for systems vulnerable to the EternalBlue exploit, however, the attacker did not seem to use it during the attack.

Reconnaissance of the environment was done using ADFind, NetScan, and LAN Search Pro. ADFind is a tool we frequently see in investigations that enables the threat actor to dump a list of all systems and users in a domain. NetScan is an administrative tool that can perform a scan to map out a network, including logged-on users, installed software, and various other information about remote machines. LAN Search Pro is a utility that allows a user to search for files on network shares of the local network.

OUT

Before starting to roll out ransomware on the victim's network, the threat actor exfiltrated data. The threat actor used WinRAR to compress data and staged the exfiltrated data on the fileserver that the data originated from. The threat actor then exfiltrated this data to both Mega and Google Drive. Data exfiltration to Mega was done using the MegaSync application whilst exfiltration to Google Drive was done through the Chrome browser.

After obtaining full control of the environment and exfiltrating data from the victim, the threat actor proceeded to destroy the victim's backups by first deleting any backup related files and then deploying ransomware on their backup systems.

Finally, after the threat actor had ensured that there was no way for the victim to recover from the attack using backups, they moved to the victim's ESXi hosts and deployed a precompiled ransomware binary. The ransomware binary would proceed to encrypt all the victim's virtual machines. Unfortunately, this ransomware binary turned out to be very poorly implemented and contained several different design flaws that resulted in the irreversible corruption of data. This binary is analysed in the sections below.

RECENT DEVELOPMENTS SURROUNDING THE BABUK THREAT ACTOR

At the end of April, Babuk announced that it would cease operations and switch to a different business model. The group would no longer encrypt systems but would instead focus exclusively on data exfiltration.[4,5,6] Furthermore, the group stated that it would publish the code to its ransomware as an open-source project. The threat actor indicated that it would focus on publishing data from victims that were unresponsive to its ransom demands. Furthermore, the threat actor indicated that it would host and publish data for other groups. As such, the Babuk threat actor seems to be moving towards a data management position.

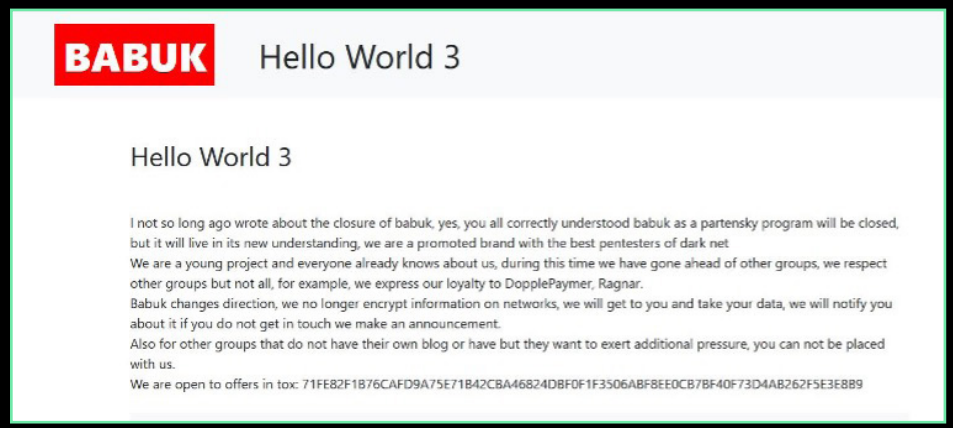


FIGURE 5. POST ON BABUK'S SITE

Given the poor design of its ransomware, a fair number of victims should be saved from completely losing their data when being attacked by Babuk. As mentioned in the previous sections, Northwave has seen threat actors slowly move from a scheme extorting victims by encrypting their data towards a double-extortion scheme where the threat actors both encrypt the victim's data and exfiltrate it as well. It is interesting to see threat actors now moving towards a scheme where their sole source of pressure to extort victims is the exfiltration of sensitive data.

FROM RANSOMWARE DEVELOPERS TO DATA LEAK MANAGERS?

As mentioned previously on its website, the Babuk team has moved from the ransomware environment to becoming a data leak discloser:

FIGURE 6. BABUK'S NEW WEBSITE -> PAYLOAD.BIN

Firstly, it released the source code for highly anticipated game Cyberpunk 2077 at the end of May 2021. The team behind the game, CD ProjektRed, is a polish video game developer, publisher, and distributer. The leak contains the source code for Cyberpunk 2077 on various video game platforms including PS5, PC, etc:

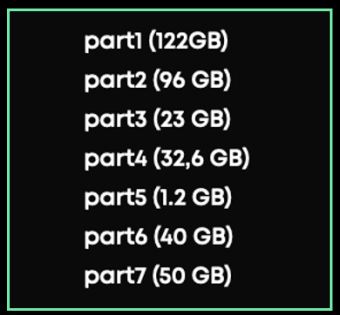


FIGURE 7. ALL LEAKED CYBERPUNK 2077 SOURCE CODE

Since this first leak, we have not seen any new activity from the actor.

TECHNICAL ANALYSIS

The malware is written in the open-source programming language Golang, most likely because it allows developers to have a single codebase to be compiled into all major operating systems. This means that, thanks to static linking, code written in Golang on a Linux system can run on a Windows or Mac system. This presents a large advantage to ransomware gangs looking to encrypt a whole infrastructure comprised of different systems architecture.

Babuk sample:

Filename	Babuk_nas.bin
File Type	ELF 32-bit LSB executable
File Size	2MB
SHA256	e43462670c467d4899ce61e2d202d93049499b10fcaaf05d98d53616567a4276
Sections	23

FIGURE 8. BABUK SAMPLE SUMMARY

As we know, for Windows, Babuk replaced Chacha encryption with the HC-128 algorithm in mid-January, but for the Linux version it still uses Chacha and Curve25519 algorithms:

```
Package main: F:/AAA_PROJ/BABUK_LOCK_curve25519/nas/enc
File: main.go
        encrypt_file Lines: 717 to 805 (88)
        main Lines: 805 to 825 (20)
        mainfunc1 Lines: 825 to 827 (2)
Package golang.org/x/crypto/curve25519: C:/Users/Krunker/go/src/golang.org/x/crypto/curve25519
File: curve25519.go
        init0 Lines: 45 to 45 (0)
File: curve25519_generic.go
        feZero Lines: 18 to 29 (11)
        feAdd Lines: 29 to 35 (6)
        feSub Lines: 35 to 41 (6)
        feCopy Lines: 41 to 50 (9)
        feCSwap Lines: 50 to 73 (23)
        feFromBytes Lines: 73 to 153 (80)
        feToBytes Lines: 153 to 269 (116)
        feMul Lines: 269 to 507 (238)
        feSquare Lines: 507 to 660 (153)
        feMul121666 Lines: 660 to 718 (58)
        feInvert Lines: 718 to 779 (61)
        scalarMultGeneric Lines: 779 to 795 (16)
Package golang.org/x/crypto/chacha20: C:/Users/Krunker/go/src/golang.org/x/crypto/chacha20
File: chacha_generic.go
        newUnauthenticatedCipher Lines: 80 to 128 (48)
        quarterRound Lines: 128 to 184 (56)
        (*Cipher)XORKeyStream Lines: 184 to 256 (72)
        (*Cipher)xorKeyStreamBlocksGeneric Lines: 256 to 352 (96)
        hChaCha20 Lines: 352 to 388 (36)
```

FIGURE 9. MAIN GO FILES USED BY BABUK

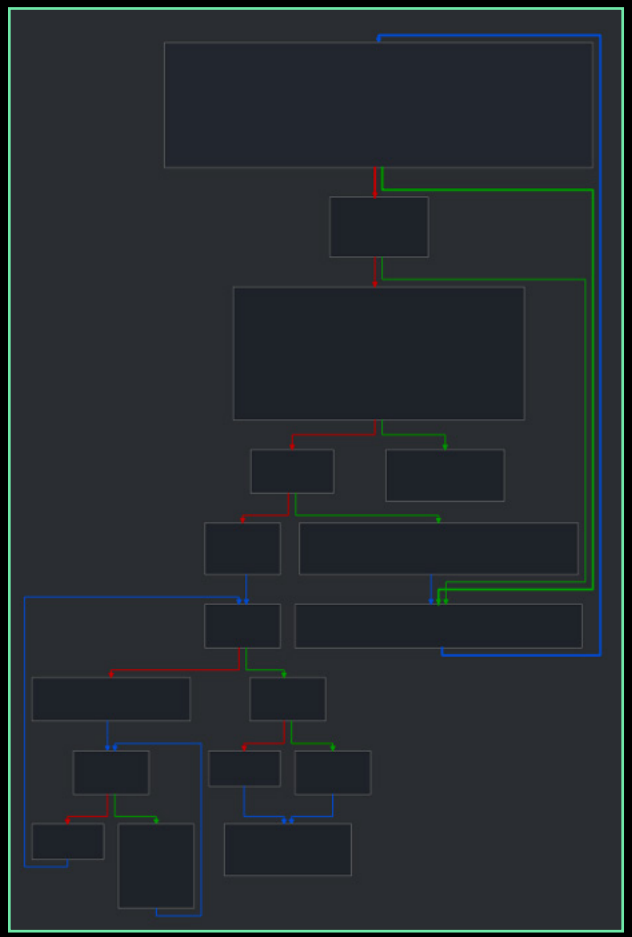
Before starting the encryption process, the sample will check if the processor is allowing MMX instructions because Go requires MMX support for it to compile properly:

```
;-- str.This_program_can_only_be_run_on_processors_with_MMX_support.:
;-- bad_proc_msg:
0x0812ce60          .string "This program can only be run on processors with MMX support.\n" ; len=62
0x0812ce9e      add byte [eax], al
```

FIGURE 10. BABUK CHECKS FOR MMX INSTRUCTIONS SUPPORT

MMX instructions allow a single instruction to execute against multiple items of data concurrently, providing the program can be expressed in that form.

It also continues to map the CPU by looking for virtual processors by using “getproccount” and “sched_getaffinity” system calls, to avoid multiple calls and touching the filesystem:



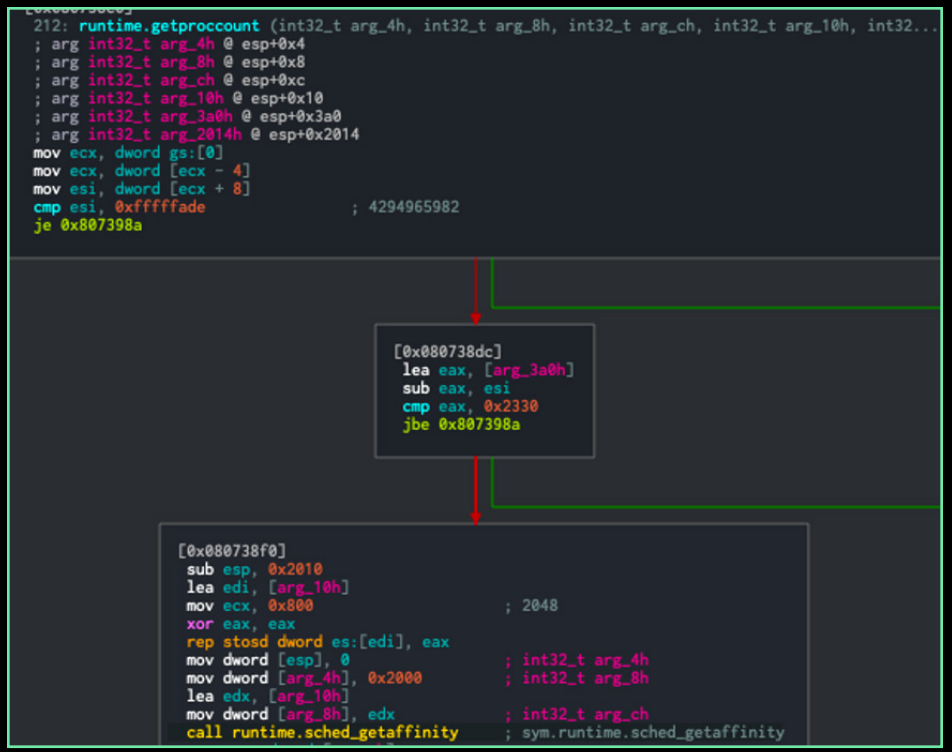


FIGURE 11. CPU CHECKING FLOW AND MAPPING

After the processor recognition, the sample will set the environment to run correctly (set GCC (GNU Compiler Collection), increase the default stack size of goroutine, implement synchronization algorithms with atomics, etc).

Babuk manipulates the buffers a lot from the victim computer, notably with the garbage collector (GC) process to free memory while other goroutines modify it. For example, if GC is freeing memory, goroutines report all their memory writes to it. The goal is to prevent concurrent memory modifications being missed by the current freeing phase. To do it, Golang uses a “write barrier” which performs the write and informs GC.

In this sample, before writing to memory, the code checks some variables:

- If the “write barrier” is activated and calls “runtime.gcWriteBarrier”:

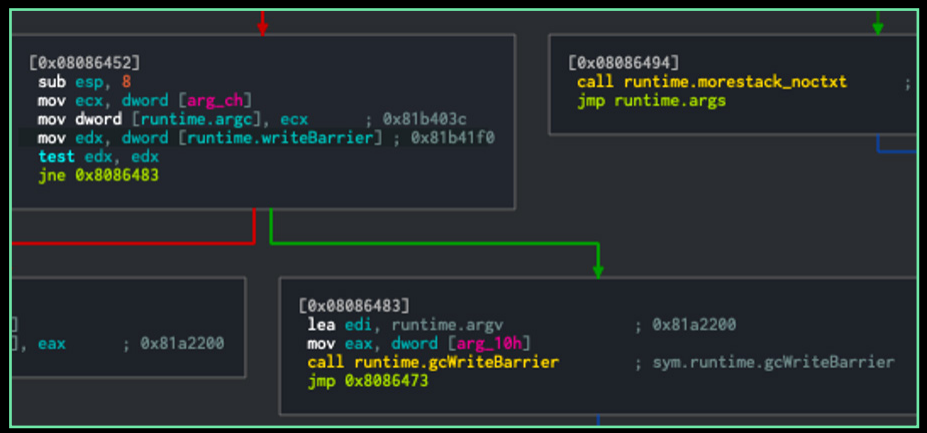


FIGURE 12. WRITE BARRIER CHECKS

- If that pointer writes, follow the CGO (a tool used by Go) rules, used to import a pseudo-package “C.” The Go code can then refer to types such as C.size_t, variables such as C.stdout, etc. If the import of “C” is immediately preceded by a comment, that comment (called the preamble), is used as a header when compiling the C parts of the package:

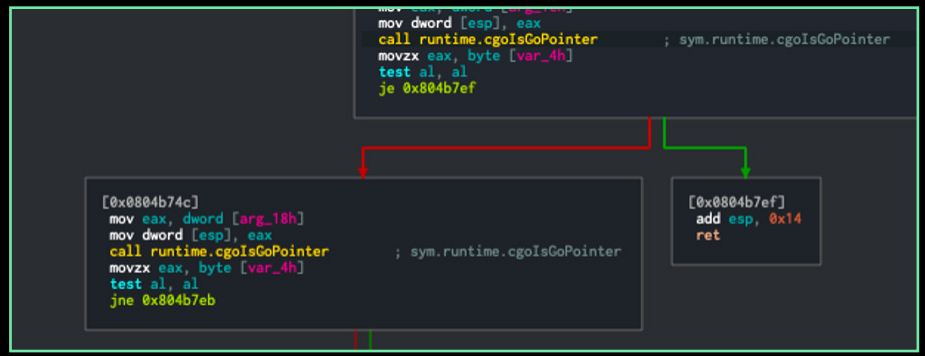


FIGURE 13. PSEUDO C PACKAGE IMPORT

All the checks for CGO are made using atomic.

The sample implements a “write barrier” slow path. The write barrier has what we call a fast path that enqueues to a per-P write barrier buffer, which is written in assembly and does not clobber any general-purpose registers, so it does not have the usual overheads of a Go call. The slow path is used when the buffer fills up; the write barrier invokes the slow path “wbBufFlush” to flush the buffer to the GC work queues: this path spills all registers and disallows any GC safe points that could observe the stack frame.

One point of note is that the samples check for the HugePages size, a mechanism that enables memory pages that are greater than their default size, to optimize operations. To do it, it checks the path “/sys/kernel/mm/transparent_hugepage/hpage_pmd_size” by using the variable “sysTHPSizePath”, declared as:

```
var sysTHPSizePath =[]byte("/sys/kernel/mm/transparent _ hugepage/
hpage _ pmd _ size\x00")
```

MEMORY ALLOC:

Next, the sample starts to initiate the memory allocation process by using several Golang functions. First, it uses “runtime.mallocinit” and checks the physical page size by using “physPageSize” a couple of times for mapping and unmapping operations:

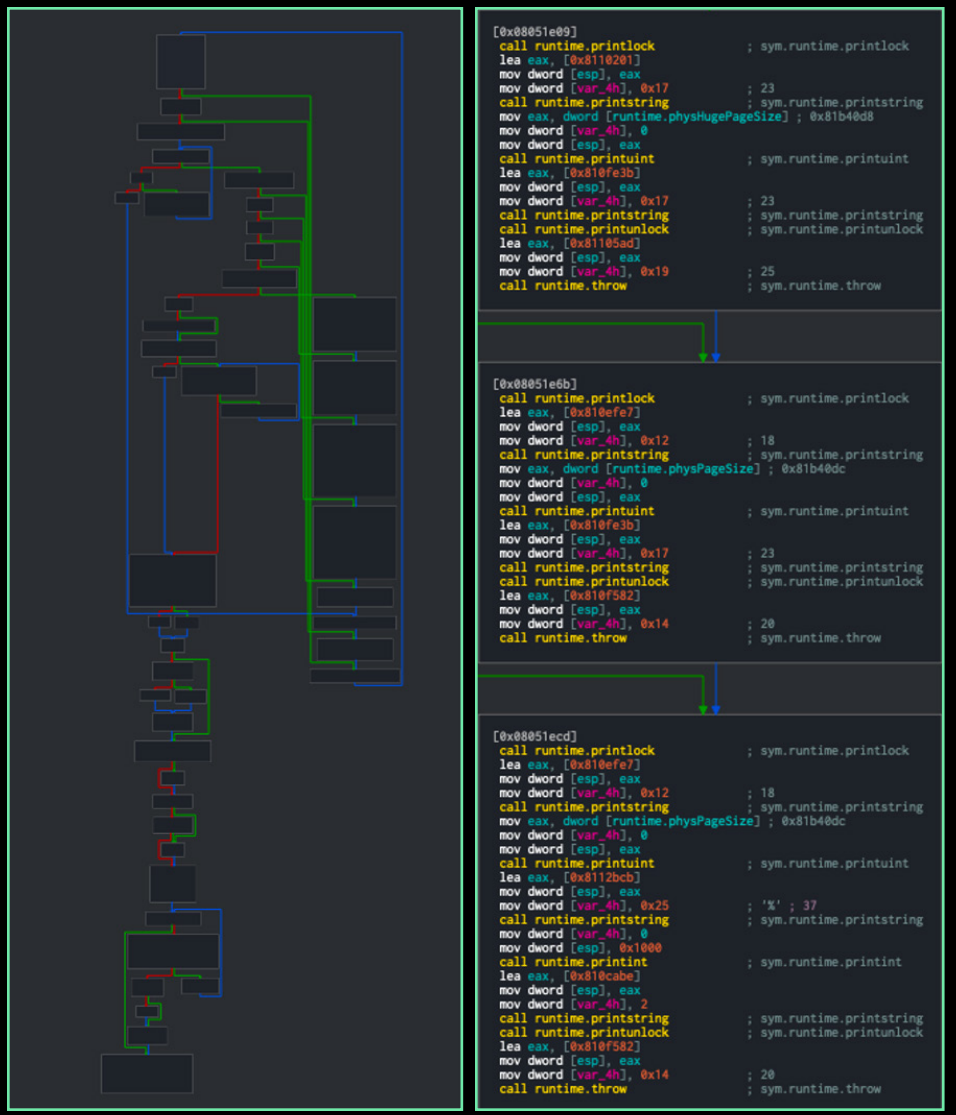


FIGURE 14. MEMORY ALLOCATION + “PHYSPAGESIZE” CHECK

Here is a code example:

```
// Check physPageSize.

        if physPageSize == 0 {

                // The OS init code failed to fetch the physical page
size.

                throw("failed to get system page size")

        }

        if physPageSize > maxPhysPageSize {

                print("system page size (", physPageSize, ") is larger
than maximum page size (", maxPhysPageSize, ")\n")

                throw("bad system page size")

        }

        if physPageSize < minPhysPageSize {

                print("system page size (", physPageSize, ") is small-
er than minimum page size (", minPhysPageSize, ")\n")

                throw("bad system page size")

        }

        if physPageSize&(physPageSize-1) != 0 {

                print("system page size (", physPageSize, ") must be a
power of 2\n")

                throw("bad system page size")

        }

        if physHugePageSize&(physHugePageSize-1) != 0 {

                print("system huge page size (", physHugePageSize, ")
must be a power of 2\n")

                throw("bad system huge page size")
```

Then "mallocinit" is used to reserve virtual memory for future allocations and initializes the "mheap" global variable used as central storage for all memory-related objects.

As expected, the heap is used to allocate memory by initializing an allocator and calls the "fixAlloc_Alloc" function each time the sample wants to allocate new mcache, mspan, etc. It allocates memory but, instead of allocating the actual size of the structure (f.size bytes), it sets "_FixAllocChunk" bytes. The rest of the available space is stored in the allocator.

Finally, the cache is initialized as:

```
_g_ := getg()

_g_.m.mcache = allocmcache()
```

The "allocmcache" function calls "fixAlloc_Alloc" to initialize a new mcache struct. The mcache field is initialized only for those threads that are currently executed and it is relocated to another thread whenever a process switch occurs.

A lot of settings are made by the sample to prepare the system before encryption; we are not going deeper on this here as it is not the most interesting part for this sample.

ENCRYPTION

Directories and files are listed by using the package “filepath.” Strangely, the sample reads only the first 250 bytes of each file using the “ReadAtLeast” function, which is unusual and not documented by the Babuk team.

Initially, “io.ReadAtLeast()” will read as many bytes as byteSlice can hold. Here is an example:

```
byteSlice := make([]byte, 512)

    minBytes := 8

    numBytesRead, err := io.ReadAtLeast(file, byteSlice, minBytes)

    if err != nil {

        log.Fatal(err)
```

So, theoretically, an implementation issue is present in this sample.

Then, Babuk instantiates Curve25519 for the key generation and exchange algorithm to protect the key and encrypt files.

```
lea eax, [var_84h]
mov dword [esp], eax
lea ecx, [var_a4h]
mov dword [var_4h], ecx
lea edx, golang.org/x/crypto/curve25519.basePoint ; 0x818e0e0
mov dword [var_8h], edx
call golang.org/x/crypto/curve25519.scalarMultGeneric ; sym.golang.org_x_crypto_curve25519.scalar...
nop
nop
lea eax, [var_64h]
mov dword [esp], eax
lea ecx, [var_a4h]
mov dword [var_4h], ecx
lea ecx, main.m_publ              ; 0x818e100
mov dword [var_8h], ecx
call golang.org/x/crypto/curve25519.scalarMultGeneric ; sym.golang.org_x_crypto_curve25519.scalar...
mov dword [esp], 0
mov eax, dword [arg_200h]
mov dword [var_4h], eax
mov ecx, dword [arg_204h]
mov dword [var_8h], ecx
lea edx, [0x810cea1]
mov dword [var_ch], edx
mov dword [var_10h], 6
call runtime.concatstring2        ; sym.runtime.concatstring2
mov eax, dword [var_14h]
mov ecx, dword [var_18h]
mov edx, dword [arg_200h]
mov dword [esp], edx
mov ebx, dword [arg_204h]
mov dword [var_4h], ebx
mov dword [var_8h], eax
mov dword [var_ch], ecx
call os.rename                    ; sym.os.rename
```

FIGURE 15. CURVE25519 INSTANTIATED

Then it uses the Chacha algorithm for the encryption part, by using the keys generated from the Curve25519 algorithm and SHA256 hash:

```
[0x080eb684]
 mov eax, dword [ebx + 0x20]
 mov dword [esp], ebp
 call eax
 mov eax, dword [var_8h]
 mov dword [var_58h], eax
 mov ecx, dword [var_4h]
 mov dword [var_30h], ecx
 mov edx, dword [var_8h]
 mov dword [var_34h], edx
 lea ebx, main.1                ; 0x81b4110
 mov dword [esp], ebx
 call sync.(*Mutex).Lock        ; sym.sync.__Mutex_.Lock
 lea eax, [var_64h]
 mov dword [esp], eax
 mov dword [var_4h], 0x20       ; 32
 mov dword [var_8h], 0x20       ; 32
 call crypto/sha256.Sum256      ; sym.crypto_sha256.Sum256
 lea edi, [var_e4h]
 lea esi, [var_ch]
 call fcn.080a22f0
 lea eax, [var_e4h]
 mov dword [esp], eax
 mov dword [var_4h], 0x20       ; 32
 mov dword [var_8h], 0x20       ; 32
 call crypto/sha256.Sum256      ; sym.crypto_sha256.Sum256
```

FIGURE 16. SHA256 USED WITH THE KEY GENERATED

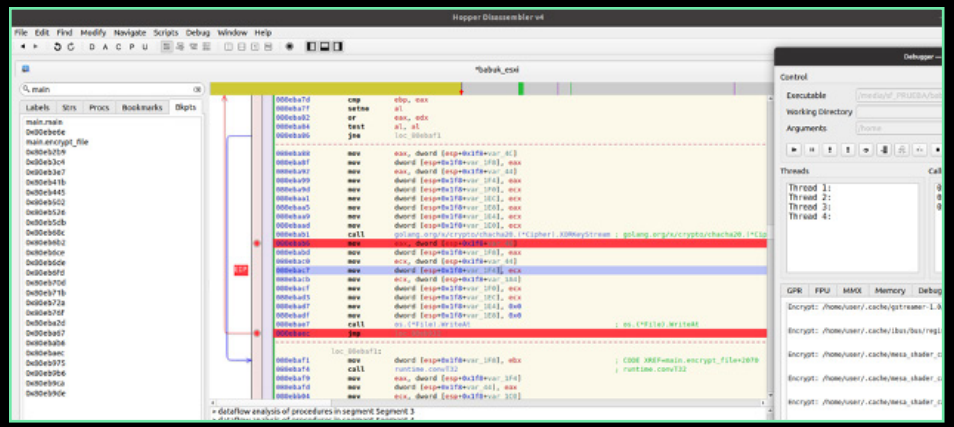


FIGURE 17. BABUK ENCRYPTION EXAMPLE

KEY FINDINGS

The encrypted files that we received do not belong to this sample as this one encrypts more than 512 bytes (0x200 in hex; other versions seem to encrypt only 0x250 bytes, and this sample does not add the text “chu...” to the end of the files, we received).

The decryptor seems to belong to the same sample but, as we saw earlier, we have a limit in the maximum number of bytes that will decrypt, which is strange.

Overall, the decryptor is poor as it only checks for the extension “.babyk” which will miss any files the victim may have renamed in an attempt to recover them. Also, the decryptor checks if the file is more than 32 bytes in length, as the last 32 bytes are combined later with other hardcoded values to get the final key. This is bad design as those 32 bytes could be trash, instead of the key, as the customer could make things, etc. It does not operate efficiently by checking the paths that are checked in the malware, instead it analyzes everything. Another error we noticed was that the decryptor tries to remove a ransom note name that IS NOT the same that the malware creates in each folder. This does not make any sense unless, perhaps, the Babuk developers/operators are delivering a decryptor that works for a different version and/or sample.

Another important thing is this sample is designed to be launched manually or with some script with an argument as the path to encrypt; with this path the malware calls to the OS function of “/path/filepath.Walk” that needs one argument that is a callback function executed per file/folder found as a new g thread (a Golang mechanism to speed the process). In the case that there is no argument, the malware finishes reporting the usage in the terminal. This callback function checks that the file/folder does not exist in some paths of the operating system, the name of the ransom note, creates the ransom note if it is needed, and launches a new g thread to encrypt the file. This procedure of using g threads makes the ransomware very quick to encrypt. They control the synchronization with a mutex mechanism (lock and unlock) and additionally, for some critical parts, they use the Go library “wait” command to control all g threads. Each encrypted file will display information in the terminal.

CONCLUSION

The Babuk threat actor, albeit having been active for only a short time, caused a lot of damage by operating with faulty ransomware. This blog has shown the modus operandi of the threat actor and analyzed the ransomware used by it. Several flaws were identified that showed how the decryption process fails in certain instances, causing irrecoverable damage. We suspect that this poor design of the ransomware was the reason that the threat actor decided to move towards a data management position.

YARA RULE

```
rule RANSOM _ BabukLocker _ NAS _ Apr2021 {

meta:

description = “Rule to detect BabuLocker Linux edition”

author = “TS @ McAfee ATR”

date = “04-27-2021”

hash = “a564da1ed886756e375de5f56241699e”

malware _ type = “Ransom”


strings:

$s1 = “BABUK _ LOCK _ curve25519” wide ascii

$s2 = “crypto/chacha20” wide ascii

$s3 = “filepath.Walk” wide ascii

$s4 = “/sys/kernel/mm/transparent _ hugepage/hpage _ pmd _ size” wide
ascii


condition:

filesize >= 1MB and filesize <= 3MB and

4 of ($s*)

}
```

IOCS: BABUK NAS LOCKER

```
8c6f768c90103857c59f031fb043d314db6a7371908a1f45bc2e86cf2ad68268

8daf429bb21285cfcf24dcc4797501ee3d4daf73364055ee5b002492ad55a3e1

e505b24de50b14aed35cf40725dc0185cab06fed90269d445ec7a4b36de124b6

e8cee8eab4020e1aadd4631ed626ab54d8733f8b14d683ca943cd4e124eeef55
```

Endnotes

1 http://essay.utwente.nl/81595/1/Keijzer_MA_EEMCS.pdf
2 https://github.com/bb00/zer0dump
3 https://www.secura.com/blog/zero-logon
4 https://www.databreaches.net/babuk-closes-one-shop-switches-to-raas/
5 https://heimdalsecurity.com/blog/babuk-ransomware-leaks-personal-data-of-metropolitan-police-officers/
6 https://www.bleepingcomputer.com/news/security/babuk-ransomware-readies-shut-down-post-plans-to-open-source-malware/

ABOUT MCAFEE

McAfee is the device-to-cloud cybersecurity company. Inspired by the power of working together, McAfee creates business and consumer solutions that make our world a safer place. By building solutions that work with other companies' products, McAfee helps businesses orchestrate cyber environments that are truly integrated, where protection, detection, and correction of threats happen simultaneously and collaboratively. By protecting consumers across all their devices, McAfee secures their digital lifestyle at home and away. By working with other security players, McAfee is leading the effort to unite against cybercriminals for the benefit of all.

www.mcafee.com

MCAFEE ATR

The McAfee® Advanced Threat Research Operational Intelligence team operates globally around the clock, keeping watch of the latest cyber campaigns and actively tracking the most impactful cyber threats. Several McAfee products and reports, such as MVISION Insights and APG ATLAS, are fueled with the team's intelligence work. In addition to providing the latest Threat Intelligence to our customers, the team also performs unique quality checks and enriches the incoming data from all of McAfee's sensors in a way that allows customers to hit the ground running and focus on the threats that matter.

Subscribe to receive our Threat Information.

6220 America Center Drive
San Jose, CA 95002
888.847.8766
www.mcafee.com