Analyzing BlackByte Ransomware's Go-Based Variants | Zscaler

By Javier Vicente Vallejo, Brett Stone-Gross

Published: 2022-05-03 · Archived: 2026-04-05 20:43:02 UTC

Key Points

- BlackByte is a full-featured ransomware family that first emerged around July 2021
- The ransomware was originally written in C# and later redeveloped in the Go programming language around September 2021
- The threat group exfiltrates data prior to deploying ransomware and leaks the stolen information if a ransom is not paid
- The group has demanded multi-million dollar ransoms from some victims
- BlackByte ransomware employs various anti-analysis techniques including a multitude of dynamic string obfuscation algorithms
- In early versions of the ransomware, file encryption utilized a hardcoded 1,024-bit RSA public key along with a 128-bit AES key that was derived from a file retrieved from a command and control server
- More recent BlackByte versions use Curve25519 Elliptic Curve Cryptography (ECC) for asymmetric encryption and ChaCha20 for symmetric file encryption

Introduction

BlackByte is a Ransomware-as-a-Service (RaaS) group that has been targeting corporations worldwide since July 2021. Previous versions of the ransomware were written in C#. More recently, the authors redeveloped the ransomware using the Go programming language. The BlackByte Go variant was used in attacks described in an FBI advisory that warned BlackByte had compromised numerous businesses, including entities in US critical infrastructure sectors. In this post, Zscaler ThreatLabz analyzes two variants of the Go-based implementation of BlackByte ransomware.

Technical Analysis

ThreatLabz has identified two variants of the Go-based variant of BlackByte. The first variant was seen in-the-wild around September 2021 and shares many similarities with the C# version including the commands executed to perform lateral propagation, privilege escalation, and file encryption algorithms. A more recent Go-based variant was introduced around February 2022. This new variant introduced many additional features and updated the file encryption algorithms. In this blog, for brevity, the Go-based BlackByte variant 1 will be referred to as BlackByte v1 and the second variant will be referred to as BlackByte v2.

Initialization

Before BlackByte performs file encryption, the ransomware first performs initialization. Most of these initialization functions are very similar or identical to the C# variant of BlackByte.

Mutex Creation

BlackByte creates a mutex using a value that is hardcoded in the malware, for example: *Global\7b55551e-a59c-4252-a34a-5c80372b3014*. If the mutex exists, BlackByte will terminate. This ensures that there is only one active instance of BlackByte running at a time.

Identify System Language

BlackByte ransomware resolves the victim's system language by comparing the language ID values with those shown in Table 1. If the system language matches any from this list, BlackByte will exit without performing file encryption.

Language ID	Language
1049	Russian
1058	Ukrainian
1059	Belarusian
1064	Tajik
1067	Armenian
1068	Azerbaijani Latin
1079	Georgian
1087	Kazakh
1090	Turkmen
1091	Uzbek Latin
2092	Azerbaijani Cyrillic
2115	Uzbek Cyrillic

Table 1. System languages avoided by BlackByte ransomware

These languages are specifically avoided by BlackByte to prevent encrypting files on systems that are located in Commonwealth of Independent States (CIS) countries. This likely indicates that the threat actors behind BlackByte are located in Eastern Europe and/or Russia. This is designed to reduce the threat that local law enforcement in those regions will pursue criminal prosecution against those responsible for BlackByte.

Enable Long Paths

The malware executes the following command to avoid issues that may occur when encrypting files with long path names:

```
C:\WINDOWS\system32\cmd.exe /c reg add HKLM\SYSTEM\CurrentControlSet\Control\FileSystem /v LongPathsEnabled /
```

Disable Controlled Folder Access

BlackByte executes the following command to disable controlled folder access:

```
Set-MpPreference -EnableControlledFolderAccess Disabled
```

The Windows controlled folder access feature is designed to protect data from malicious applications such as ransomware. When enabled, files located in the specified protected folders can not be modified by unauthorized applications.

Delete Shadow Copies

Similar to other ransomware families, BlackByte deletes shadow copies to prevent a victim from easily recovering files from backups. There are two methods that BlackByte uses to delete shadow copies. The first executes the following PowerShell command:

```
$x = [System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String('RwBlAHQALQBXAG0AaQBPAGIAagB
'AFcAaQBuADMAMgBfAFMAaABhAGQAbwB3AGMAbwBwAHkAIAB8AC'+'AARgBvAHIARQBhAGMAaAAtAE8AYgBqAGUAYwB0ACAAewAkA'+
'F8ALgBEAGUAbABlAHQAZQAoACkAOwB9AA=='));Invoke-Expression $x
```

The Base64 encoding string when decoded is the following: **Get-WmiObject Win32_Shadowcopy | ForEach-Object {$_.Delete();}**

BlackByte also executes the commands to delete shadow copies for each drive:

```
C:\WINDOWS\system32\cmd.exe /c vssadmin resize shadowstorage /for=: /on=: /maxsize=401MB
C:\WINDOWS\system32\cmd.exe /c vssadmin resize shadowstorage /for=: /on=: /maxsize=unbounded
```

Process Termination and Stop / Start Services

The following commands are executed by BlackByte to stop services that may hinder file encryption:

```
C:\WINDOWS\system32\sc.exe config SQLTELEMETRY start= disabled
C:\WINDOWS\system32\sc.exe config SQLTELEMETRY$ECWDB2 start= disabled
C:\WINDOWS\system32\sc.exe config SQLWriter start= disabled
C:\WINDOWS\system32\sc.exe config SstpSvc start= disabled
C:\WINDOWS\system32\sc.exe config MBAMService start= disabled
C:\WINDOWS\system32\sc.exe config wuauserv start= disabled
```

BlackByte will also start the following services:

```
C:\WINDOWS\system32\sc.exe config Dnscache start= auto
C:\WINDOWS\system32\sc.exe config fdPHost start= auto
C:\WINDOWS\system32\sc.exe config FDResPub start= auto
C:\WINDOWS\system32\sc.exe config SSDPSRV start= auto
C:\WINDOWS\system32\sc.exe config upnphost start= auto
C:\WINDOWS\system32\sc.exe config RemoteRegistry start= auto
```

BlackByte ransomware terminates the following processes shown in Table 2 at the beginning of the execution:

uranium	processhacker	procmon	pestudio	procmon64
x32dbg	x64dbg	cffexplorer	procexp64	procexp
pslist	tcpview	tcpvcon	dbgview	rammap
rammap64	vmmap	ollydbg	autoruns	autorunsc
regmon	idaq	idaq64	immunitydebugger	wireshark

dumpcap	hookexplorer	importrec	petools	lordpe
sysinspector	proc_analyzer	sysanalyzer	sniff_hit	windbg
joeboxcontrol	joeboxserver	joeboxserver	resourcehacker	fiddler
httpdebugger	dumpit	rammap	rammap64	vmmap
agntsvc	cntaosmgr	dbeng50	dbsnmp	encsvc
excel	firefox	firefoxconfig	infopath	isqlplussvc
mbamtray	msaccess	msftesql	mspub	mydesktopqos
mydesktopservice	mysqld	mysqld-nt	mysqld-opt	Ntrtscan
ocautoupds	ocomm	ocssd	onenote	oracle
outlook	PccNTMon	powerpnt	sqbcoreservice	sql
sqlagent	sqlbrowser	sqlservr	sqlwriter	steam
synctime	tbirdconfig	thebat	thebat64	thunderbird
tmlisten	visio	winword	wordpad	xfssvccon
zoolz	filemon	nsservice	nsctrl	

Table 2. Process names terminated by BlackByte ransomware

Many of these process names are related to business applications. BlackByte kills these processes to avoid open file handle permission issues when performing file encryption of the victim's files. In addition, the list contains a large number of malware analyst tools that can be used to reverse engineer the functionality of the ransomware.

BlackByte also terminates the following services that are associated with antivirus products, backup software, and business applications including financial software, email clients, and databases as shown below in Table 3.

klvssbridge64	vapiendpoint	ShMonitor	Smcinst	SmcService
SntpService	svcGenericHost	swi_	TmCCSF	tmlisten
TrueKey	TrueKeyScheduler	TrueKeyServiceHelper	WRSVC	McTaskManag
OracleClientCache80	mfefire	wbengine	mfemms	RESvc
mfevtp	sacsvr	SAVAdminService	SAVService	SepMasterServ
PDVFSService	ESHASRV	SDRSVC	FA_Scheduler	KAVFS
KAVFSGT	kavfsslp	klnagent	macmnsvc	masvc
MBAMService	MBEndpointAgent	McShield	audioendpointbuilder	Antivirus
AVP	DCAgent	bedbg	EhttpSrv	MMS
ekrn	EPSecurityService	EPUpdateService	ntrtscan	EsgShKernel
msexchangeadtopology	AcrSch2Svc	MSOLAP$TPSAMA	Intel(R) PROSet Monitoring	msexchangeim
ARSM	unistoresvc_1af40a	ReportServer$TPS	MSOLAP$SYSTEM_BGC	W3Svc

MSExchangeSRS	ReportServer$TPSAMA	Zoolz 2 Service	MSOLAP$TPS	aphidmonitors
SstpSvc	MSExchangeMTA	ReportServer$SYSTEM_BGC	Symantec System Recovery	UI0Detect
MSExchangeSA	MSExchangeIS	ReportServer	MsDtsServer110	POP3Svc
MSExchangeMGMT	SMTPSvc	MsDtsServer	IisAdmin	MSExchangeE
EraserSvc11710	Enterprise Client Service	MsDtsServer100	NetMsmqActivator	stc_raw_agent
VSNAPVSS	PDVFSService	AcrSch2Svc	Acronis	CASAD2DWe
CAARCUpdateSvc	McAfee	avpsus	DLPAgentService	mfewc
BMR Boot Service	DefWatch	ccEvtMgr	ccSetMgr	SavRoam
RTVscan	QBFCService	QBIDPService	Intuit.QuickBooks.FCS	QBCFMonitor
YooIT	zhudongfangyu	nsService	veeam	backup
sql	memtas	vss	sophos	svc$
mepocs	wuauserv			

Table 3. Service names terminated by BlackByte ransomware

Windows Firewall

BlackByte disables the Windows firewall via the command:

```
netsh advfirewall set allprofiles state off
```

Windows Defender

The ransomware executes the following command to delete task manager, resource monitor, and stop the Windows Defender service:

cmd /c del C:\Windows\System32\Taskmgr.exe /f /q & del C:\Windows\System32\resmon.exe /f /q & powershell -command "$x = [System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String ('V'+'wBp'+'A'+'G4AR'+'AB'+'lAG'+'YAZQBuAGQA'));Stop-Service -Name $x;Set-Service -StartupType Disabled

The Base64 encoded string above decodes to *WinDefend*.

Raccine Anti-Ransomware

BlackByte terminates and uninstalls an anti-ransomware product known as Raccine. The Raccine processes that are terminated are *raccine.exe* and *raccinesettings.exe*. To uninstall Raccine, BlackByte deletes the following registry keys and values:

- **HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Raccine Tray**
- **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\Raccine**
- **HKEY_CURRENT_USER\SOFTWARE\Raccine**
- **HKEY_LOCAL_MACHINE\SOFTWARE\Raccine**

BlackByte then deletes Raccine's scheduled task via the command:

```
C:\WINDOWS\system32\schtasks.exe /DELETE /TN "\"Raccine Rules Updater\"" /F
```

Privilege Escalation

The ransomware executes the following commands to disable UAC remote restrictions:

```
C:\WINDOWS\system32\cmd.exe /c reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v Local
```

BlackByte sets the *EnableLinkedConnections* registry value to force symbolic links to be written to link logon sessions as follows:

```
C:\WINDOWS\system32\cmd.exe /c reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v Enabl
```

In BlackByte v2, an additional privilege escalation method was added that exploits the *CMSTPLUA* COM interface to bypass UAC. The *ShellExec* method of the interface *ICMLuaUtil* can be invoked with arbitrary commands with elevated privileges using the ElevationMoniker **Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}**. This allows BlackByte v2 to execute the *svchost.exe* process that it injects into with elevated privileges. This privilege escalation technique has also been utilized by other ransomware groups including REvil and LockBit.

Lateral Propagation

BlackByte ransomware performs network enumeration and can propagate across a local network. First it executes the following commands to enable network discovery and file and printer sharing:

```
C:\WINDOWS\system32\cmd.exe /c netsh advfirewall firewall set rule "group=\"Network Discovery\"" new enable=Ye
C:\WINDOWS\system32\cmd.exe /c netsh advfirewall firewall set rule "group=\"File and Printer Sharing\"" new en
```

The following commands are then executed to discover other computers and network file shares:

```
net view
arp -a
```

BlackByte loads the Active Directory module RSAT-AD-PowerShell and queries for other computers via the following commands:

```
C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe Install-WindowsFeature -Name \"RSAT-AD-PowerShell\"
```

```
powershell -command "Import-Module ActiveDirectory;Get-ADComputer -Filter * -Properties * | FT Name"
```

If the *-a* flag is passed via the command-line, BlackByte attempts to copy itself to remote computer's public folders via the administrative share *\\\c$\Users\Public*. If that attempt is unsuccessful, BlackByte will default to the path: *\\\Users\Public*. BlackByte uses the Windows task scheduler to execute the ransomware on the remote host using the following command:

```
C:\Windows\system32\schtasks.exe /Create /S  /TN  /TR "C:\Users\Public\ -s " /ru SYSTEM /sc onlogon /RL HIGHES
```

In BlackByte v2, the filename and task name are pseudorandomly generated using a function that produces eight upper and lowercase alphabetic and numeric characters (e.g., *BqgDOVYL.exe* and *KYL8EpE9*, respectively). BlackByte v1 uses a

hardcoded filename and command-line argument *complex.exe -single* and the hardcoded task name *asd*.

After scheduling the task, the remote BlackByte binary is executed using the command:

```
C:\Windows\system32\schtasks.exe /S  /Run /TN
```

After the task is executed, BlackByte deletes the remote task using the command:

```
C:\Windows\system32\schtasks.exe /Delete /S  /TN  /f
```

BlackByte then deletes the copy of itself on the remote host network share. BlackByte also attempts to access administrative shares A$ through Z$ and the folders shown in Table 4.

Users	Backup	Veeam	Consejo	homes
home	media	common	Storage Server	Public
Web	Images	Downloads	BackupData	ActiveBackupForBusiness
Backups	NAS-DC	DCBACKUP	DirectorFiles	share

Table 4. Network shares targeted by BlackByte ransomware

Check for Analysis Tools

The malware checks the following DLL modules in memory shown in Table 5 and exits if they are present:

DLL Filename	**Description**
DBGHELP.DLL	Windows DbgHelp Library
SbieDll.dll	Sandboxie
SxIn.dll	Qihu 360 Total Security
Sf2.dll	Avast Antivirus
snxhk.dll	Avast Antivirus
cmdvrt32.dll	COMODO Internet Security

Table 5. DLLs Identified by BlackByte ransomware

Disable Debugging

BlackByte attempts to prevent debugging tools from monitoring and attaching to various processes by removing the following registry values under SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options:

- vssadmin.exe
- wbadmin.exe
- bcdedit.exe
- powershell.exe
- diskshadow.exe
- net.exe
- taskkill.exe
- wmic.exe
- fsutil.exe

Process Injection

BlackByte v1 injects the ransomware code in an instance of *regedit.exe*, while BlackByte v2 injects itself into an instance of *svchost.exe*. After the process is injected with the ransomware code, the file encryption is then performed in the context of the *regedit.exe* or *svchost.exe* process. BlackByte then deletes its original binary on disk by executing the command:

```
C:\Windows\system32\cmd.exe /c ping 1.1.1.1 -n 10 > Nul & Del  /F /Q
```

The ping command is used to delay the file deletion by 10 seconds. The process injection functionality may be able to bypass some security software detections.

Unmount Virtual Machine Images

In order to identify virtual machines on the victim's system, BlackByte will execute the command:

```
powershell Get-VM
```

If any virtual machine files are located, BlackByte will attempt to unmount the image by executing the following command line:

```
powershell.exe Dismount-DiskImage -ImagePath
```

Backup Volumes

The malware executes mountvol.exe to try to mount additional volumes:

```
C:\WINDOWS\system32\mountvol.exe A: \\?\Volume{[GUID]}\
C:\WINDOWS\system32\mountvol.exe B: \\?\Volume{[GUID]}\
C:\WINDOWS\system32\mountvol.exe E: \\?\Volume{[GUID]}\
C:\WINDOWS\system32\mountvol.exe F: \\?\Volume{[GUID]}\
```

This is likely an attempt to mount and encrypt backup volumes to further prevent file recovery after encryption.

File Encryption

BlackByte enumerates all physical drives and network shares skipping files that contain the following substrings in Table 6:

blackbyte	ntdetect.com	bootnxt	ntldr	recycle.bin
bootmgr	thumbs.db	ntuser.dat	bootsect.bak	autoexec.bat
iconcache.db	bootfont.bin			

Table 6. BlackByte ransomware file substring filter list

BlackByte avoids the following extensions shown in Table 7.

url	msilog	log	ldf	lock
theme	msi	sys	wpx	cpl
adv	msc	scr	key	ico
dll	hta	deskthemepack	nomedia	msu
rtp	msp	idx	ani	386

diagcfg	bin	mod	ics	com
hlp	spl	nls	cab	exe
diagpkg	icl	ocx	rom	prf
themepack	msstyles	icns	mpa	drv
cur	diagcab	cmd	shs	

Table 7. File extensions skipped by BlackByte ransomware

BlackByte will also skip files located in the following directories shown in Table 8.

bitdefender	trend micro	avast software	intel	common files
programdata	windowsapps	appdata	mozilla	application data
google	windows.old	system volume information	program files (x86)	boot
tor browser	windows	intel	perflogs	msocache

Table 8. Directories whitelisted by BlackByte ransomware

BlackByte optimizes encryption speed based on the targeted file size according to the following rules:

Filesize	**Encryption Algorithm**
Size	Encrypt the entire file
15MB >= Size > 5MB	Encrypt the first 1MB and last 1MB
150MB >= Size > 15MB	Encrypt the first 5MB and last 5MB
Size > 150MB	Encrypt the first 50MB and last 50MB

BlackByte renames encrypted files with the extension *.blackbyte*. The ransomware creates a *DefaultIcon* registry key under *HKEY_CLASSES_ROOT\.blackbyte* that points to an icon file, so that every file that is encrypted will show this icon in Windows explorer. In addition, the registry names *s1159* and *s2359* are set to *BLACKBYTE* under *HKEY_CURRENT_USER\Control Panel\International*. These registry values control the time format for AM/PM. As a result, Windows will show BLACKBYTE instead of AM/PM as shown below in Figure 2.

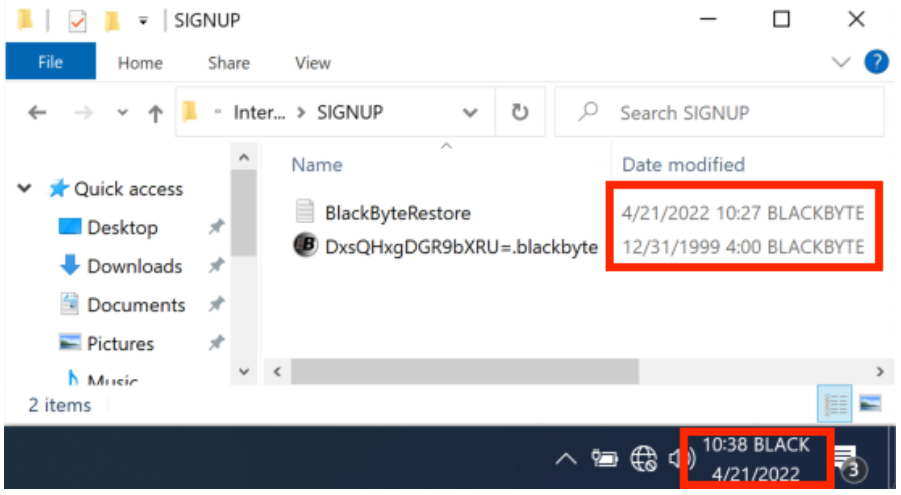


Figure 2. BlackByte AM/PM time format modification

This time format modification is performed by executing the commands:

```
reg add "HKCU\Control Panel\International" /v s1159 /t REG_SZ /d BLACKBYTE /f
reg add "HKCU\Control Panel\International" /v s2359 /t REG_SZ /d BLACKBYTE /f
```

File Encryption Algorithms (Variant 1)

BlackByte v1 must be executed with the command line argument **-single** followed by a SHA256 hash. This hash is combined with a TOR onion URL (e.g., hxxp://7oukjxwkbnwyg7cekudzp66okrchbuubde2j3h6fkpis6izywoj2eqad[.]onion/). The SHA256 hash given as an argument is concatenated to the onion URL to build the URL of the victim ransom portal that is embedded in the ransom note. This URL is substituted in the **[LINK]** field of the ransom note template.

When BlackByte v1 is executed, the malware tries to connect to a hardcoded URL that hosts a file that is involved in the construction of an AES key that is used to encrypt a victim's files. An example URL used for this purpose was hxxps://185.93.6[.]31/mountain.png. The mechanism used to build the AES key is very similar to the C# variant.

After the content of the file mountain.png is downloaded, BlackByte reads the first 16 bytes of the file into a buffer and 24 bytes at the offset 0x410 of the file into another buffer. These 24 bytes are used as key to create and initialize a NewTripleDESCipher object from the Go Cryptographic API. This object is used to decrypt the first 16 bytes of the file mountain.png. The resulting 16-byte buffer will be used as a PBKDF2 password to derive the AES key that will be used to encrypt the victim's files. The BlackByte PBKDF2 algorithm uses SHA1 as the hashing function and 1,000 iterations to derive the AES key. The password is converted to unicode and the unicode string *BLACKBYTE_IS_COOL* is used as the salt. The following example Python code can be used to derive the AES key used for file encryption.

```python
import sys, binascii
from hashlib import pbkdf2_hmac
from Crypto.Cipher import AES
from Crypto.Cipher import DES3

def tounicode(s):
    out = bytearray()
    try: s = binascii.unhexlify(s.replace(b" ", b""))
    except: pass
    for e in s:
        if e >= 0x80:
            out += b"\xFD\xFF"
        else:
            out += (e.to_bytes(1, 'little') + b"\x00")
    return out

def decrypt_blackbyte_v1(fake_png_file, enc_data):
    seed = fake_png_file[0:16]
    tdes_iv = fake_png_file[0x410:0x418]
    tdes_key = fake_png_file[0x410:0x428]
    des3_cipher = DES3.new(tdes_key, DES3.MODE_CBC, tdes_iv)
    passwd = binascii.hexlify(des3_cipher.decrypt(seed))
    passwd = tounicode(passwd)
    salt = tounicode(b"BLACKBYTE_IS_COOL")
    hmac = pbkdf2_hmac("sha1", passwd, salt, 1000, 32)
    aes_key = hmac[0:16]
    aes = AES.new(key=aes_key, mode=AES.MODE_CBC, iv=aes_key)
    return aes.decrypt(enc_data)
```

Figure 3. Python code to decrypt BlackByte v1 files with the file (e.g., mountain.png) downloaded from the C2 server

Victim's files are encrypted with AES using CBC mode. The first 16 bytes of the PBKDF2 derived key are used as AES key, and the same 16 bytes are used as the initialization vector (IV). The same AES key is used to encrypt all the files on a victim's machine.

The PBKDF2 password is encrypted with a hardcoded 1,024-bit RSA public key and the resulting RSA-encrypted value is encoded with Base64. This Base64 encoded string is substituted in the **[KEY]** field in the ransom note template. The threat actor can decrypt the PBKDF2 password with their corresponding RSA private key, derive the AES key, and thereafter, decrypt the victim's encrypted files. The following is an example RSA public key that was hardcoded in BlackByte:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDUBwECQuQiVGorPYvHrJM11OWV
E1PS8gaBqIAfPaR1rQHUEXu3iX/da/dCtV8Z27/SIA/ZYUNhTyUsX9Snjz8zve90
QAiG1c/BS81WWRax7M7i1rESStVwOaUDAj5w6cz9GwDMGYI+wve9Qyjtw5R6hr5I
qlIEig1Wy1X27vUC2wIDAQAB
-----END PUBLIC KEY-----
```

Ransom Note and BlackByte Icon (Variant 1)

The BlackByte ransom note and an image containing an icon file are stored as Base64 encoded strings in the binary. After the encryption of the victim's files, the ransom note is written to a file named *BlackByteRestore.txt*, and the previously mentioned icon file is written to a file named *BB.ico*. An example BlackByte v1 ransom note template is shown below in Figure 4. The BlackByte logo uses the extended ASCII characters of the 8-bit code page 437 to create 3-D block letters.

All your files have been encrypted, your confidential data has been stolen, in order to decrypt files and avoid leakage, you must follow our steps.

1) Download and install TOR browser from this site: https://torproject.org/

2) Paste the URL in TOR browser and you will be redirected to our chat with all information that you need.

3) If you won't contact with us within 4 days, your access to our chat will be removed and you wont be able to restore your system.

Your URL: [LINK]

Your Key: [KEY]

Figure 4. Go-based BlackByte v1 ransom note template

File Encryption Algorithms (Variant 2)

The second variant of BlackByte ransomware does not require a network connection to start encryption. In addition, the ransomware's command-line parameters were modified. BlackByte v2 requires two command line parameters:

```
sample.exe
```

The first parameter is a flag (e.g., **-a**) that controls specific behaviors of the ransomware (e.g., to propagate across a network), while the second parameter is a passphrase (e.g., **54726956**) that is verified before file encryption commences. If BlackByte is not provided with any command-line arguments, the ransomware prints out the phrase *BlackByte ransomware, 8-th generation, the most destructive of all ransomware products, real natural disaster.* and exits.

BlackByte v2 removed the RSA and AES file encryption algorithms from the ransomware. The encryption algorithms were replaced with Curve25519 elliptic curve cryptography for asymmetric encryption and ChaCha for symmetric algorithm. The Curve25519 functions are statically compiled within BlackByte using Go library code. BlackByte generates a random 32-byte buffer per file using the Windows API function *RtlGenRandom()*. This random value is used as a file's secret key. The file's public key is calculated as follows:

file_public_key = Curve25519(file_secret_key, base_point = 0x9)

The threat actor's Curve25519 public key is hardcoded in the binary and stored as a Base64 encoded string. For the sample with the SHA256 hash ffc4d94a26ea7bcf48baffd96d33d3c3d53df1bb2c59567f6d04e02e7e2e5aaa, the hardcoded Curve25519 public key was the string:

2BSTzcpdqRW/a2DRT3TiL9lN5INRmmn1lCQWzZhkfQs=
(d81493cdca5da915bf6b60d14f74e22fd94de483519a69f5942416cd98647d0b)

The shared secret is derived as follows:

shared_secret = Curve25519(file_secret_key, blackbyte_public_key)

The shared secret is hashed with SHA256 to derive a 32-byte ChaCha encryption key. The ChaCha encryption key is then hashed again with SHA256 to derive the ChaCha nonce (using 12 bytes starting at offset 10). Once the ChaCha key parameters have been derived, they will be used to encrypt the file's content. The encrypted data is written to the file (overwriting the original content). Finally, the victim's 32-byte public key is concatenated to the encrypted content of the file. The BlackByte v2 encryption algorithm is shown below in Figure 5.

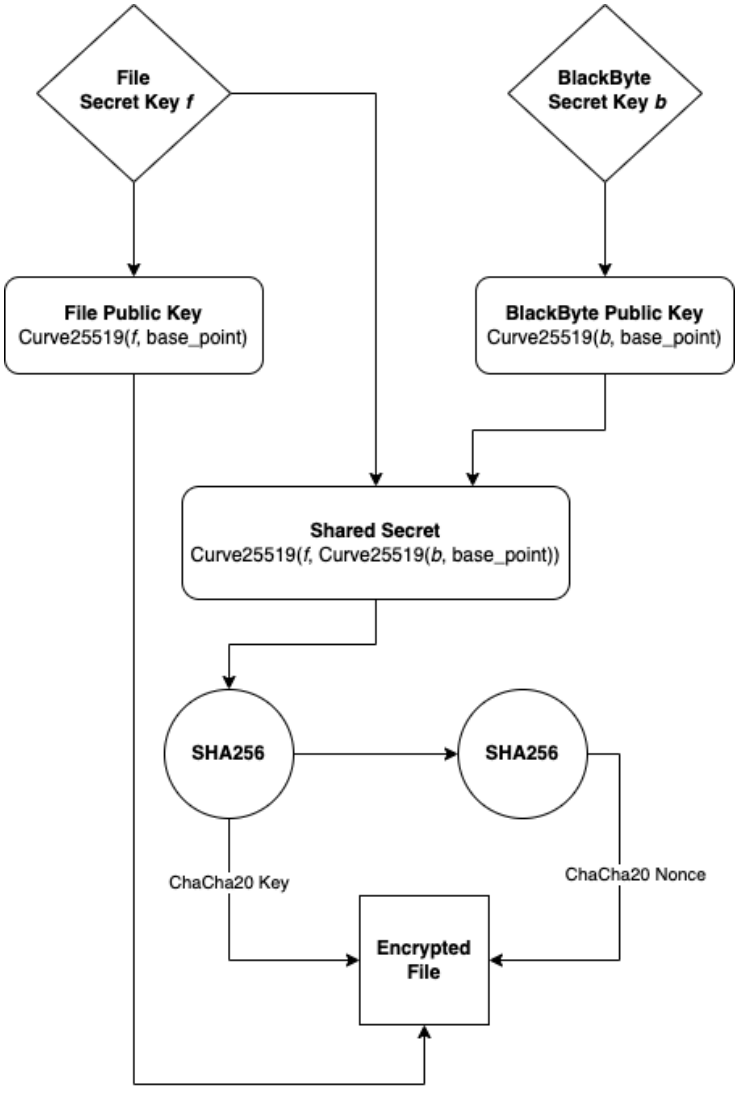


Figure 5. BlackByte v2 file encryption algorithm

The threat actor can use the file's public key together with the threat actor's secret key to recover the shared secret and use it to decrypt the encrypted data as follows:

shared_secret = Curve25519(blackbyte_secret_key, file_public_key)

The following Python code in Figure 6 can be used to decrypt BlackByte encrypted data from a file that has been encrypted if the threat actor's private key is obtained:

```python
import hashlib
from cryptography.hazmat.primitives.asymmetric.x25519 import X25519PublicKey
from cryptography.hazmat.primitives.asymmetric.x25519 import X25519PrivateKey
from cryptography.hazmat.primitives import serialization
from Crypto.Cipher import ChaCha20


def decrypt_blackbyte_v2(private_key_bytes, data):
    private_key = X25519PrivateKey.from_private_bytes(private_key_bytes)
    public_key = private_key.public_key().public_bytes(
        encoding=serialization.Encoding.Raw, format=serialization.PublicFormat.Raw
    )
    public_bytes = data[-32:]
    peer_public_key = X25519PublicKey.from_public_bytes(public_bytes)
    shared_key = private_key.exchange(peer_public_key)
    chacha_key = hashlib.sha256(shared_key).digest()
    nonce_hash = hashlib.sha256(chacha_key).digest()
    nonce = nonce_hash[10:22]
    cc = ChaCha20.new(key=chacha_key, nonce=nonce)
    return cc.decrypt(data[:-32])
```

Figure 6. Python code to decrypt BlackByte v2 files with the threat actor's private key

BlackByte v2 also encrypts the filename after encryption. The encryption is a simple XOR layer with a hardcoded key, followed by Base64 encoding as shown in Figure 7.

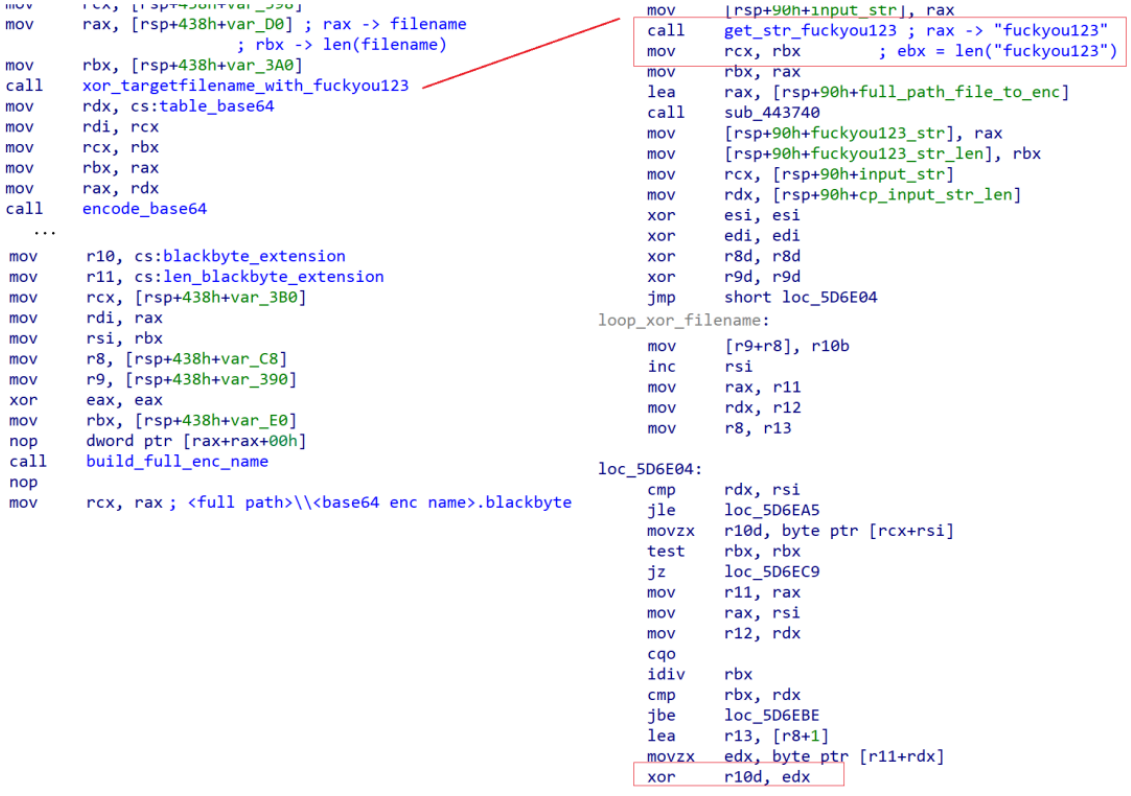


Figure 7. BlackByte v2 filename encryption

In the analyzed sample, the XOR key was *fuckyou123*. After a filename has been encrypted, the file is renamed and the .blackbyte extension is concatenated.

ThreatLabz has implemented proof-of-concept file decryption code for both BlackByte v1 and v2 here.

Ransom Note and BlackByte Icon (Variant 2)

BlackByte v2 introduced some improvements to storing the ransom note and icon file. The Base64 encoded blocks for the ransom note and icon file added an XOR-based encryption layer. The XOR key to decrypt the ransom note and icon file is embedded in the ransomware as an obfuscated string. The icon file is written to the victim's *%APPDATA%* directory using a randomly generated filename consisting of six upper and lowercase alphabetic and numeric characters (e.g., *i2uOJh.ico*).

BlackByte v2 contains a hardcoded TOR onion URL and path for the victim portal rather than relying on the command-line for the path value. BlackByte v2 also added a hardcoded password that is required to access the victim ransom portal. An example password is:

```
gkaW_#DD[Aw_JTB@luXpJBdye6eLr@{bx5pHFA)T5FpMYJC]f|@
```

The BlackByte v2 ransom note template is shown below in Figure 8. The **[LINK]** substring in the ransom note is replaced with the hardcoded BlackByte victim URL and the **[PASSW]** substring is replaced with the victim-specific password for the ransom portal.

```
BLACKBYTE

+-------------------------------------------------------------------------+
| All your files have been encrypted, your confidential data has been stolen, |
| in order to decrypt files and avoid leakage, you must follow our steps.     |
+-------------------------------------------------------------------------+

+-------------------------------------------------------------------------------------------------------------------------+
| 1) Download and install TOR Browser from this site: https://torproject.org/
|
| 2) Paste the URL in TOR Browser and you will be redirected to our chat with all information that you need.
|
| 3) If you do not contact us within 3 days, your chat access key won't be valid.
|    Also, your company will be posted on our blog, darknet and hacker forums,
|    which will attract unnecessary attention from journalists and not only them.
|    You are given 3 days to think over the situation, and take reasonable actions on your part.
+-------------------------------------------------------------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------+
| Warning! Communication with us occurs only through this link, or through our mail on our blog. |
| We also strongly DO NOT recommend using third-party tools to decrypt files,                    |
| as this will simply kill them completely without the possibility of recovery.                  |
| I repeat, in this case, no one can help you!                                                   |
+-----------------------------------------------------------------------------------------+


Your URL: [LINK]

Your Key to access the chat: [PASSW]

Find our blog here (TOR Browser): http://dlyo7r3n4qy5fzv4645nddjwarj7wjdd6wzckomcyc7akskkxp4glcad.onion/
```

Figure 8. BlackByte v2 ransom note template

An example ransom note when populated after file encryption has been performed for BlackByte v2 is shown in Figure 9.

Figure 9. BlackByte v2 ransom note

A copy of the BlackByte v2 ransom note can be found in the ThreatLabz GitHub repository. After BlackByte encrypts files, the ransom note is written to each directory, the encrypted files are renamed, and their icons are replaced by the BlackByte icon.

Ransom Portal and Leak Site

When a victim accesses the link in the ransom portal, they are instructed to enter the access key from the ransom note as shown in Figure 10.

Figure 10. BlackByte victim ransom portal

After a victim authenticates, they are provided the ransom demand and instructions how to purchase Bitcoin. There is also a live chat feature as shown in Figure 11.

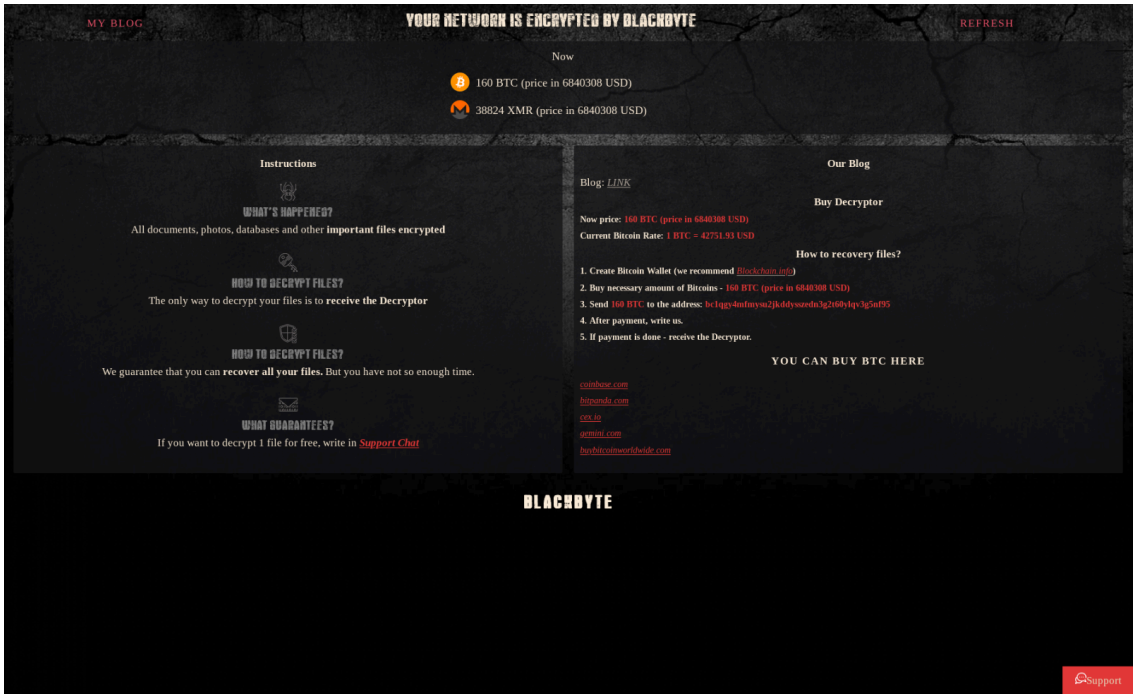


Figure 11. BlackByte ransom negotiation portal

Victims are further pressured to pay the ransom, or risk having their data publicly leaked on their TOR hidden service as shown in Figure 12.

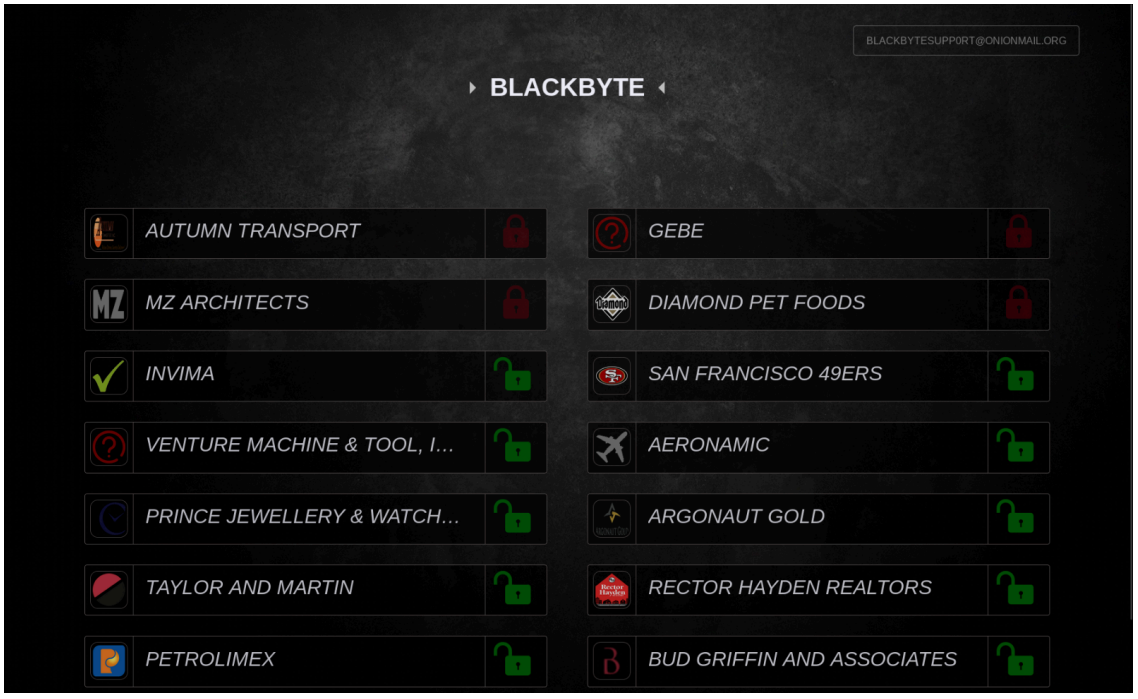


Figure 12. BlackByte victim leak site

Print Bombing

In addition to dropping a ransom note on the victim's machine, the ransomware sends a message to be printed by any connected printers. The printed ransom message is an RTF file with the content shown below:

```
{\rtf1\ansi\ansicpg1251\deff0\nouicompat\deflang1049{\fonttbl{\f0\fnil\fcharset0 Calibri;}}
{\*\generator Riched20 10.0.19041}\viewkind4\uc1
```

```
\pard\sa200\sl276\slmult1\qc\f0\fs56\lang9 Your HACKED by BlackByte team.\par
Connect us to restore your system.\fs22\par
\fs56 Your HACKED by BlackByte team.\par
Connect us to restore your system.\fs22\par
\fs56 Your HACKED by BlackByte team.\par
Connect us to restore your system.\fs22\par
\fs56 Your HACKED by BlackByte team.\par
Connect us to restore your system.\fs22\par
\fs56 Your HACKED by BlackByte team.\par
Connect us to restore your system.\fs22\par
\fs56 Your HACKED by BlackByte team.\par
Connect us to restore your system.\fs22\par

\pard\sa200\sl276\slmult1\par
}
```

In BlackByte v1, the message is written to the file *C:\Users\tree.dll* and the following command is executed to print it:

```
C:\\Windows\\System32\\cmd.exe /c for /l %x in (1,1 ,75) do start wordpad.exe /p C:\\Users\\tree.dll
```

In addition, a task named *Task* is created to print the message every hour:

```
C:\WINDOWS\system32\schtasks.exe /create /np /sc HOURLY /tn Task /tr "C:\Windows\System32\cmd.exe
/c for /l %x in (1,1,75) do start wordpad.exe /p C:\Users\tree.dll" /st 07:00
```

In BlackByte v2, the text of the message is written to a file with a random name consisting of six upper and lowercase alphabetic and numeric characters. The task name is also created randomly consisting of eight upper and lowercase alphabetic and numeric characters. An example task command to print the ransom message is shown below:

```
C:\WINDOWS\system32\schtasks.exe /create /np /sc HOURLY /tn 4y77VPNo /tr "C:\Windows\System32\cmd.exe
/c for /l %x in (1,1,75) do start %SystemDrive%\Program Files\Windows NT\Accessories\WordPad.exe /p
C:\Users\1HoWkK.dll" /st 07:00
```

Anti-Analysis / Anti-Forensics Techniques

String Obfuscation

Both Go-based BlackByte variants encrypt most strings using a tool similar to AdvObfuscator. Each string is decrypted using a unique algorithm with polymorphic code that implements different operations *xor*, *addition*, *subtraction*, etc. In the examples below, the encrypted strings in Figure 13 are built and decrypted from arguments on the stack.

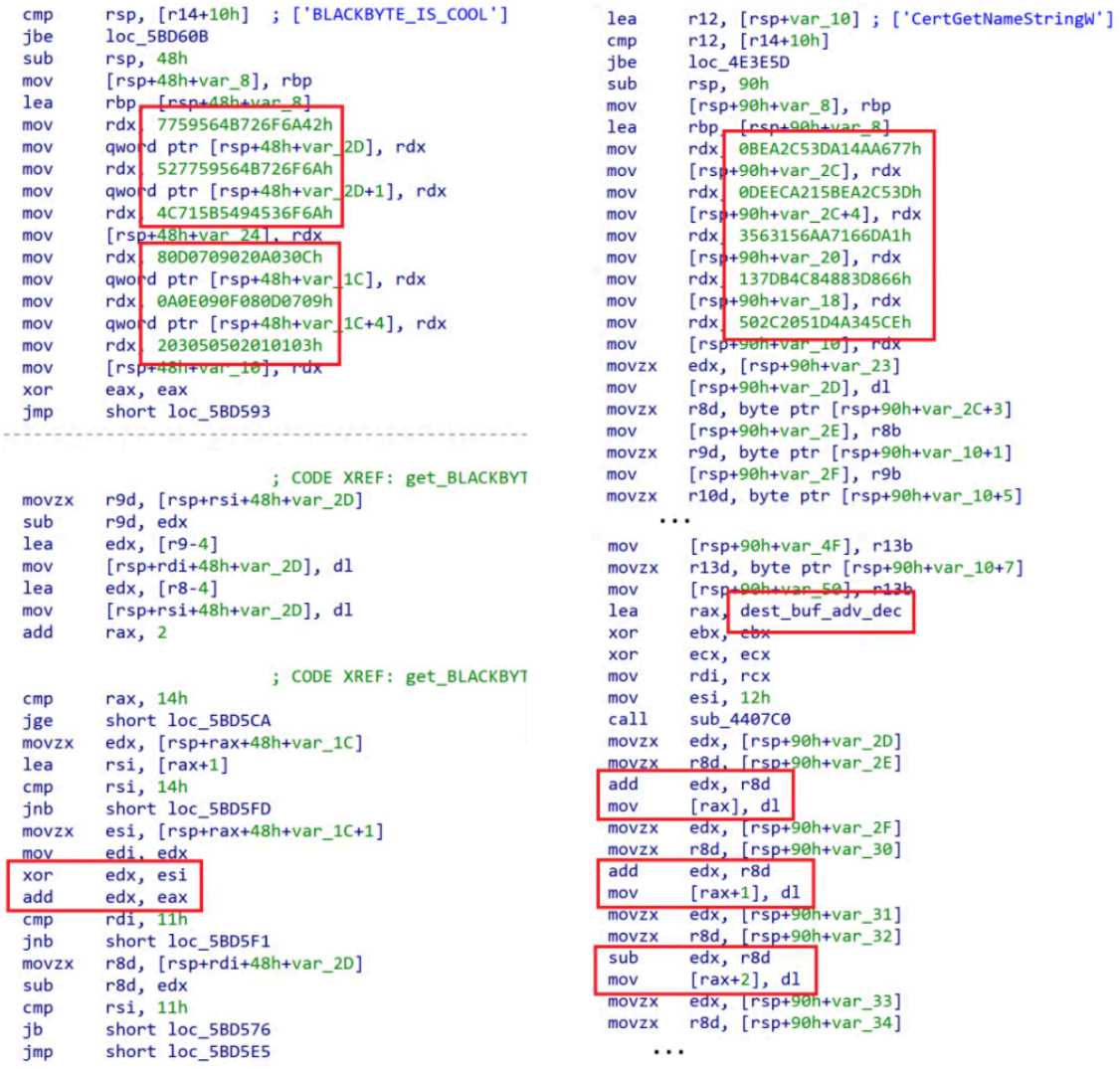


Figure 13. BlackByte string obfuscation examples

Modified UPX Packer

In addition to string obfuscation, BlackByte samples are typically packed with UPX. In BlackByte v1, all of the samples observed by ThreatLabz were packed with the standard UPX packer and could be unpacked via the command-line parameter *-d*. The early samples of BlackByte v2 were also packed with the standard UPX packer. However, the most recent BlackByte samples (since March 2022) are packed with a modified version of UPX. The names of the sections have been renamed from *UPX0* and *UPX1* to *BB0* and *BB1*, respectively. Figure 14 shows an example BlackByte v2 sample with the modified UPX headers.

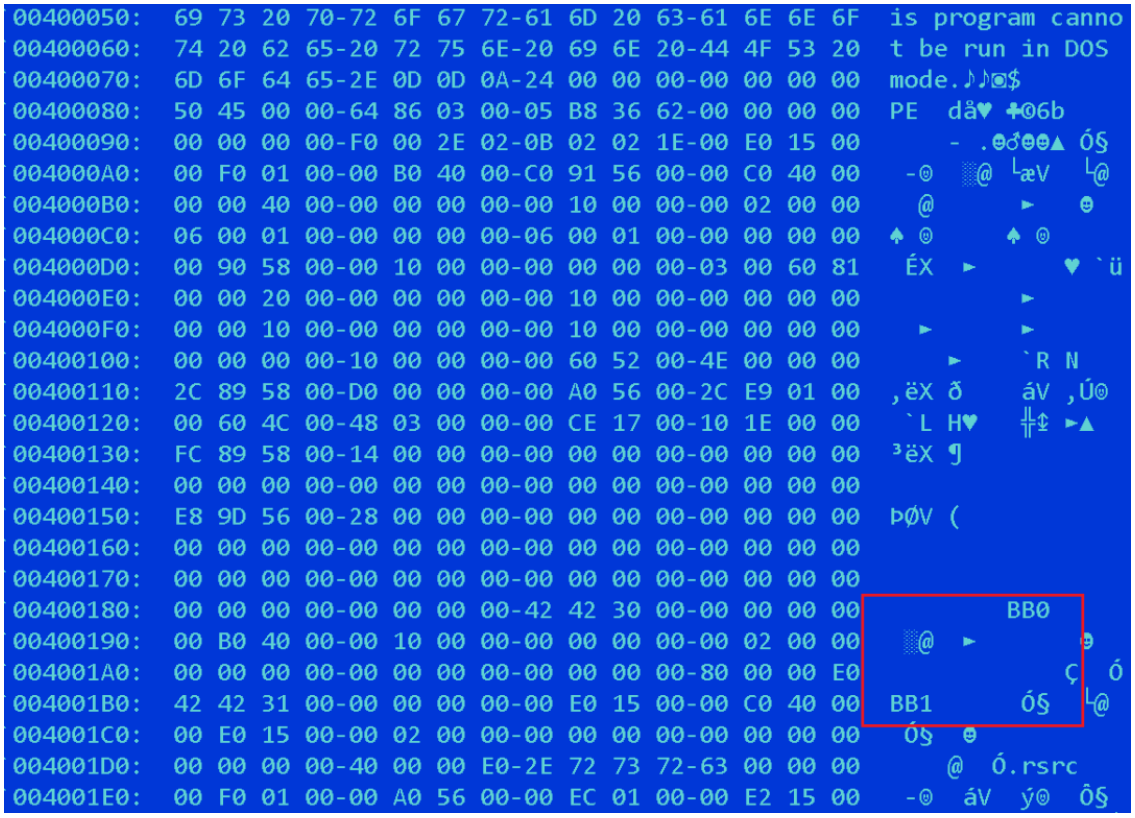


Figure 14. BlackByte v2 altered UPX header

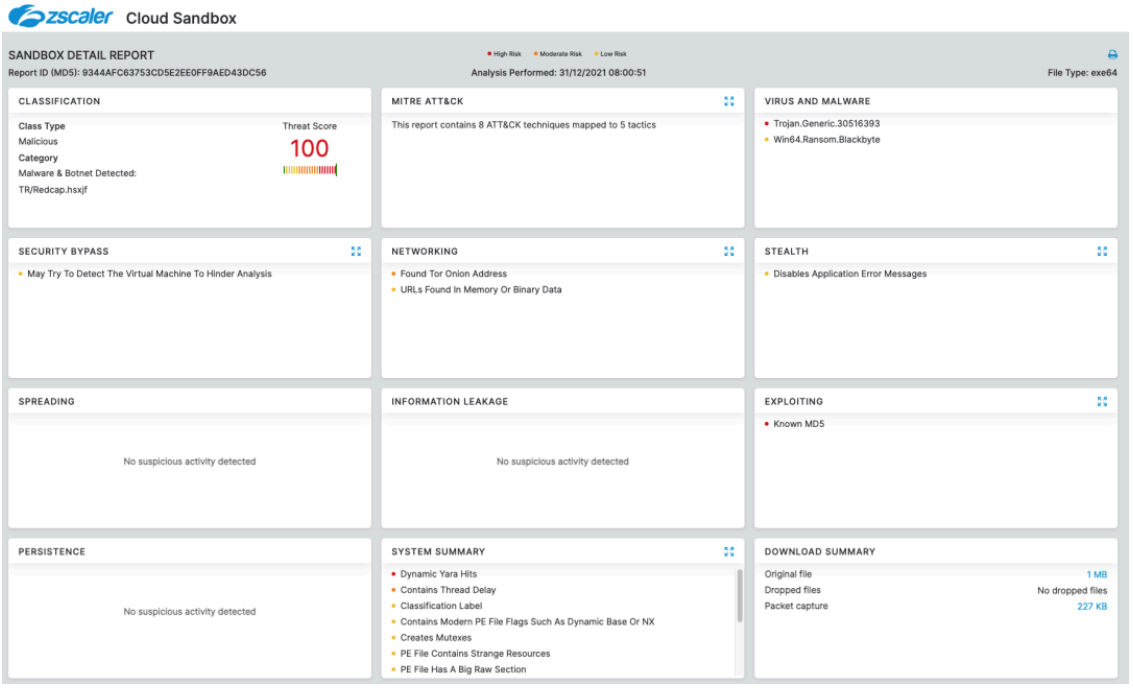
Antivirus Detection

Due to BlackByte's anti-analysis features, polymorphic code, and heavy obfuscation many antivirus products have very low detection rates. For example, the BlackByte sample with the SHA256 *534f5fbb7669803812781e43c30083e9197d03f97f0d860ae7d9a59c0484ace4* has an antivirus detection rate of 4/61 at the time of publication.

Conclusion

BlackByte is a full-featured ransomware family operated by a threat group that continues to breach organizations and demand large ransom amounts. The threat group also performs double extortion attacks by stealing an organization's files and leaking them online if the ransom is not paid. The ransomware code itself is regularly updated to fix bugs, bypass security software, and hinder malware analysis. The encryption algorithms have also been improved to be more secure and prevent file recovery. This demonstrates that the threat group will likely continue to improve the ransomware and remain a significant threat to organizations.

Cloud Sandbox Detection



Zscaler's multilayered cloud security platform detects indicators at various levels, as shown below:

- Win64.Ransom.Blackbyte

Indicators of Compromise

IoC Type	Value
BlackByte v1 Packed Sample	1df11bc19aa52b623bdf15380e3fded56d8eb6fb7b53a2240779864b1a6474ad
BlackByte v1 Packed Sample	388163c9ec1458c779849db891e17efb16a941ca598c4c3ac3a50a77086beb69
BlackByte v1 Unpacked Sample	44a5e78fce5455579123af23665262b10165ac710a9f7538b764af76d7771550
BlackByte v1 Unpacked Sample	6f36a4a1364cfb063a0463d9e1287248700ccf1e0d8e280e034b02cf3db3c442
BlackByte v2 Packed Sample	ffc4d94a26ea7bcf48baffd96d33d3c3d53df1bb2c59567f6d04e02e7e2e5aaa
BlackByte v2 Packed Sample	9103194d32a15ea9e8ede1c81960a5ba5d21213de55df52a6dac409f2e58bcfe
BlackByte v2 Packed Sample	e434ec347a8ea1f0712561bccf0153468a943e16d2cd792fbc72720bd0a8002e
BlackByte v1 Onion URL	hxxp://7oukjxwkbnwyg7cekudzp66okrchbuubde2j3h6fkpis6izywoj2eqad.]onion
BlackByte v2 Onion URL	hxxp://fyk4jl7jk6viteakzzrxntgzecnz4v6wxaefmbmtmcnscsl3tnwix6yd.]onion
BlackByte v2 Onion URL	hxxp://p5quu5ujzzswxv4nxyuhgg3fjj2vy2a3zmtcowalkip2temdfadanlyd.]onion
BlackByte v1 AES Key Seed URL	hxxps://185.93.6[.]31/mountain.png

References

- https://redcanary.com/blog/blackbyte-ransomware/
- https://www.ic3.gov/Media/News/2022/220211.pdf
- https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/blackbyte-ransomware-pt-1-in-depth-analysis/
- https://www.bleepingcomputer.com/forums/t/755181/blackbyte-ransomware-blackbyte-support-topic/

Explore more Zscaler blogs