WINDOWS PACKAGE REPORT

MALWARE ANALYSIS OF JIGSAW RANSOMWARE

21PC17 JYOTHISH K S

21PC19 NANDA PRANESH S

21PC25 VARUN S

JIGSAW RANSOMWARE:

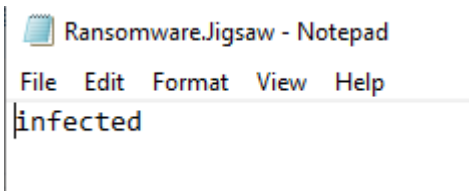
Jigsaw is a type of ransomware attack using which, an attacker tries to encrypt all the files present in the victim's machine and then demands a ransom to be paid, mostly through bitcoins for him to decrypt the files back to original. This ransomware was inspired from the movie "Jigsaw". Every ransomware works in the same model but the difference of working in Jigsaw is that, there is a countdown timer that runs in the application, giving the victim an hour time to pay the ransom or every one hour, one of the victim's files will be deleted.

TOOLS USED:

- Kali Linux
- PEid
- Process Hacker
- Regshot
- Dotpeek
- WireShark

RANSOMWARE SOURCE:

We have downloaded the malware from 'theZoo' repository in GitHub that contains a collection of all the malwares available, for budding malware analysts to understand the working of each malware and analyzing them. Every malware file is password protected, and in this case, the password was given to us in a 'PASS file'.



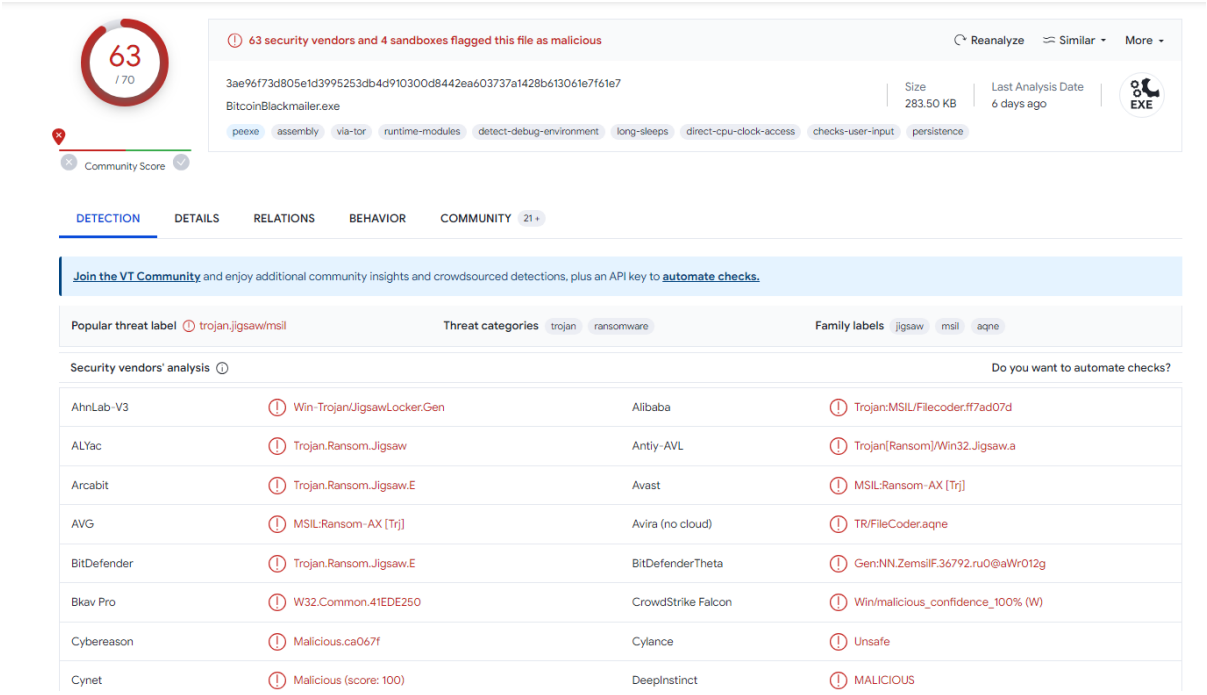
TESTING ENVIRONMENT:

We have used a Windows 10 ISO Virtual machine to analyze the malware, both static and dynamic. By using a VM to execute the malware, we safeguard our actual system files from being attacked. We also take a snapshot of the current state of the machine before executing the malware in the VM to revert back to harmless state if something happens.

STATIC ANALYSIS:

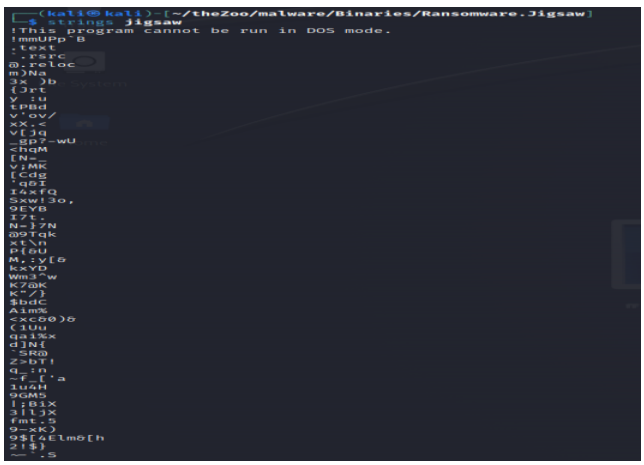
1) Scanning the malware using VirusTotal:

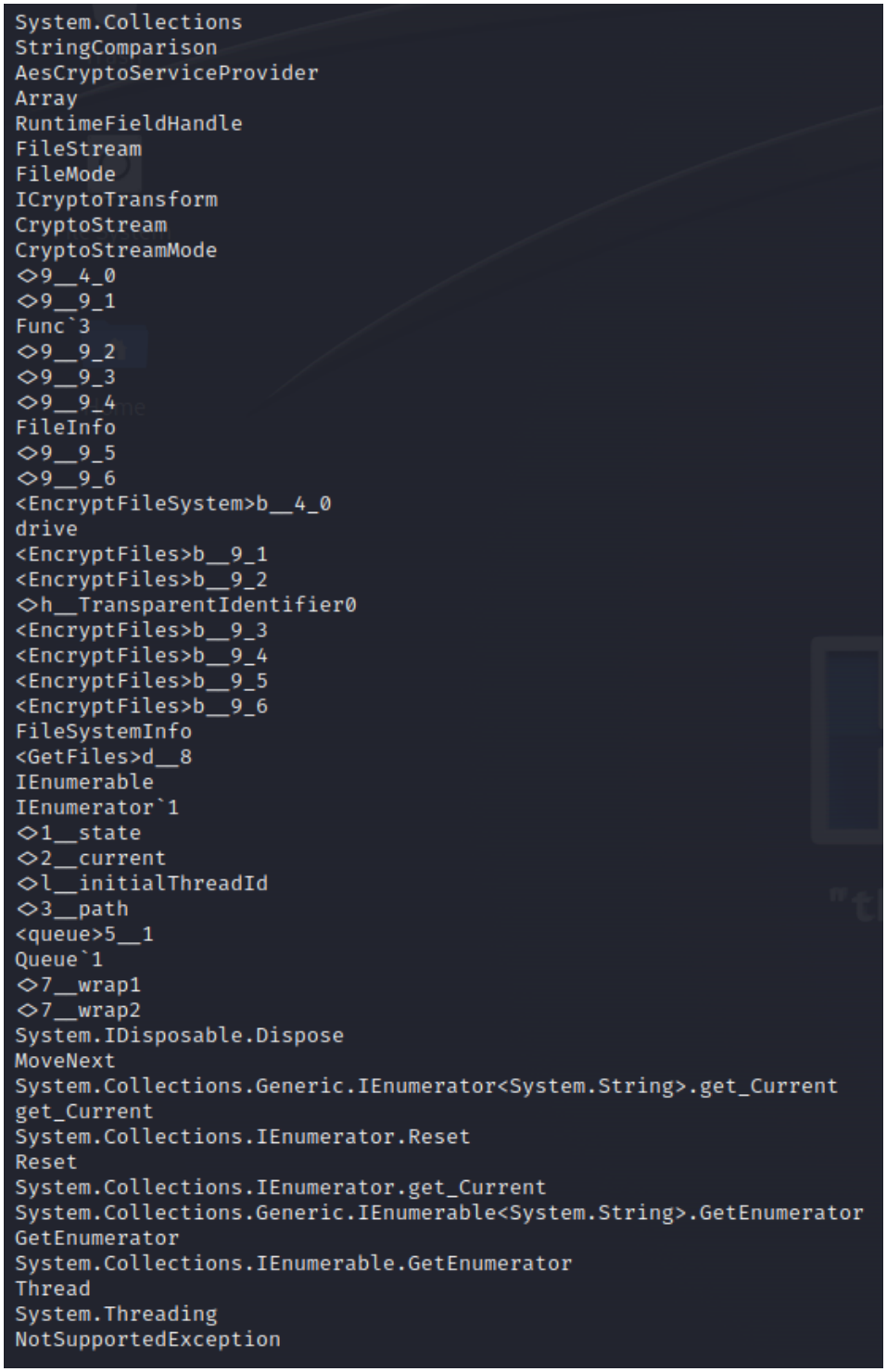
VirusTotal (www.virustotal.com) is a type of antivirus scanning website that is used to detect if the file given as input is malicious or not. In the below image, we have given the jigsaw file as input and we can observe that 63/70 vendors have marked it as malicious, i.e., about 90% of the antivirus software that has come across the file have found it to be malicious. Through this, we can infer that the jigsaw file is malicious.



2) Searching for associated strings:

For this, we are using the inbuilt strings function available in KALI LINUX to observe the different strings associated with the Jigsaw ransomware. For this, we have to clone the repository into our home directory and unzip the malware. After that, we can use the strings function to see the various strings in the malware.



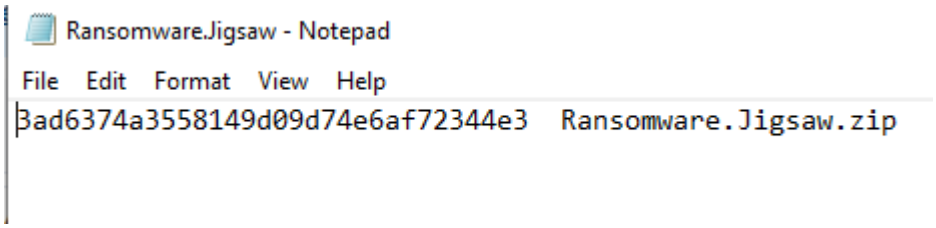


In the above screenshots, we can see the different strings associated with jigsaw file, which could be function names, URL parameters, filename, etc.,

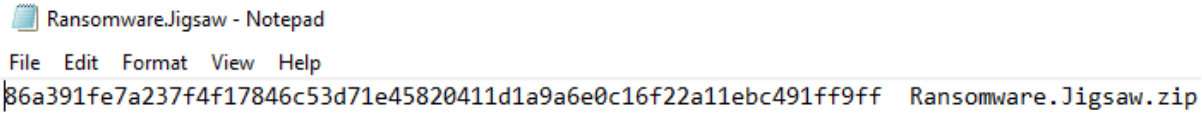
3) Identifying the hash:

Hashing in the context of static analysis is that, every malware has a unique hash associated to it and using this hash value, we can try and identify the type of malware. The MD5 and SHA256 hash values of the jigsaw ransomware we given to us.

MD5:

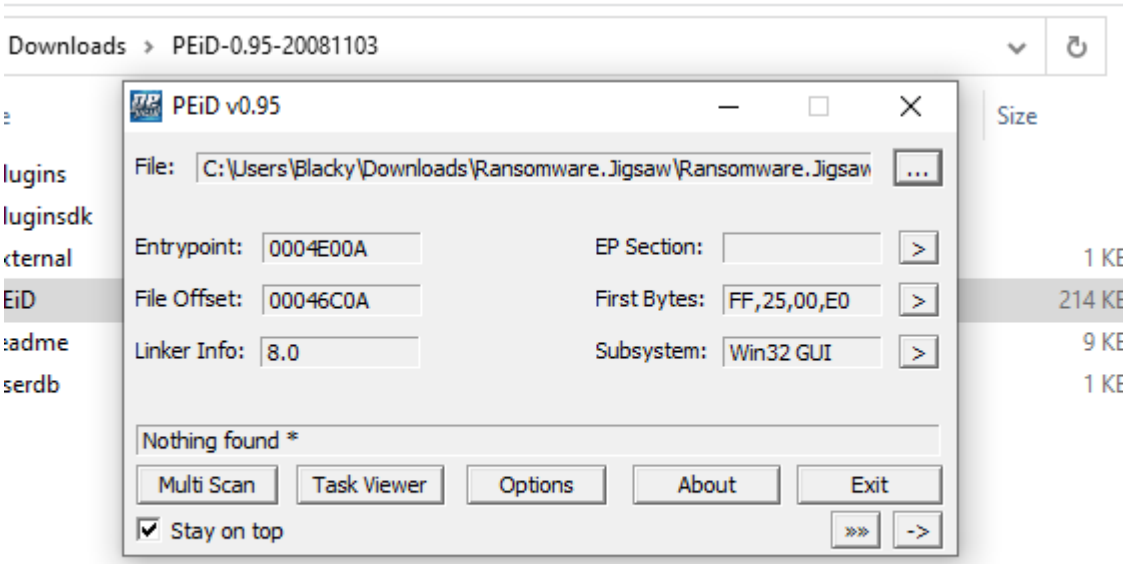


SHA256:



4) Detecting packers with PEid:

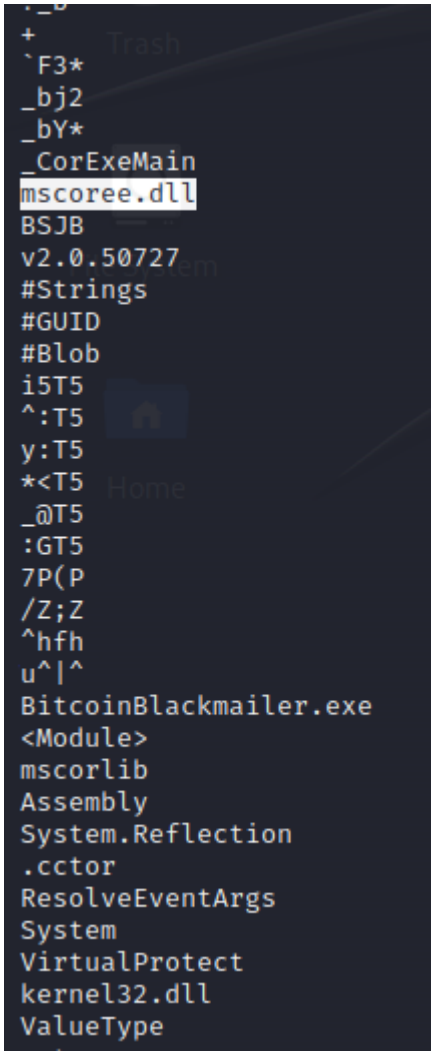
PEid is a tool that can be used to detect the type of packer or compiler used in the malware, which can be used to easily analyze the working of the malware if we could find any such packer.



Here, we can see that ‘Nothing found’ is the output and we can conclude that the tool didn’t find any packer.

5) Checking for DLL files:

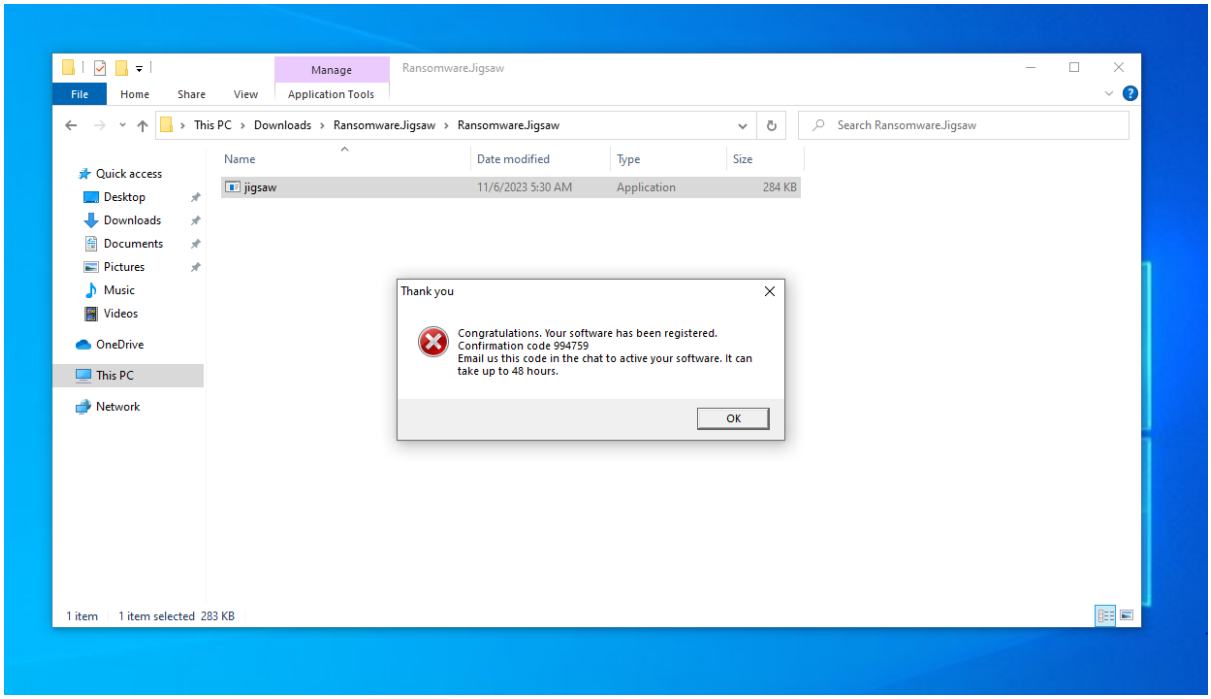
Finding about the import and export functions of any file would give us an idea about the different libraries used in the program. This can be mediated by a DLL (Dynamic Link Library) file.



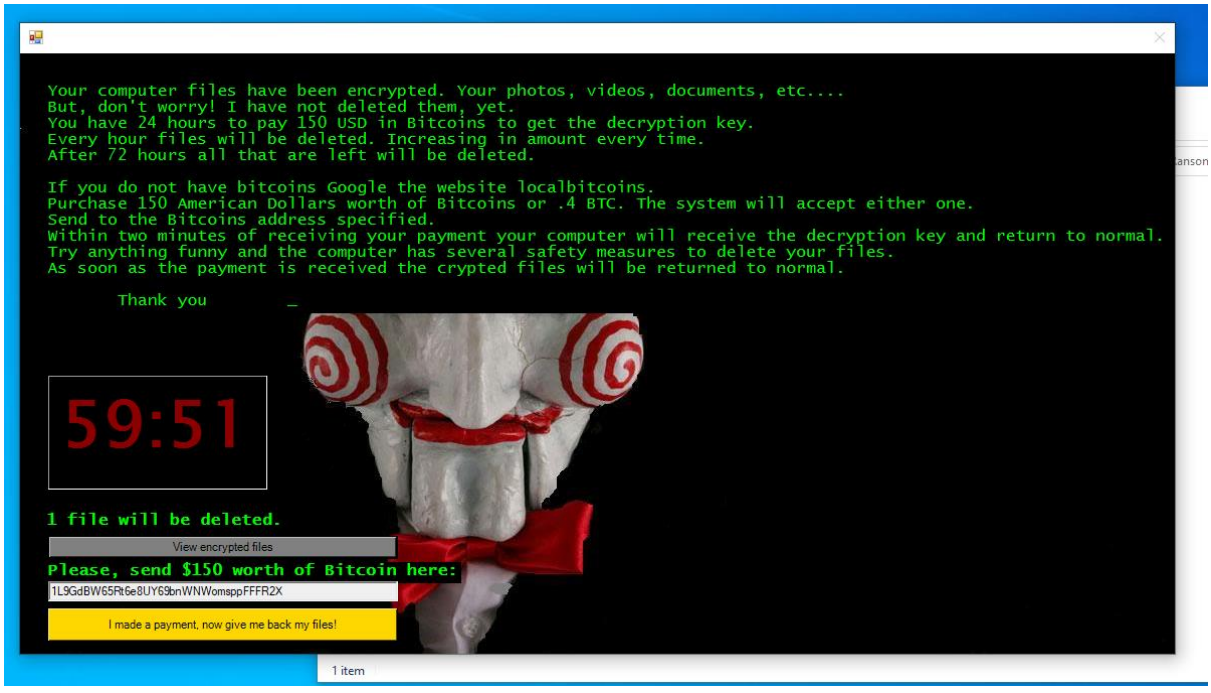
Using the strings function of KALI LINUX, we have found 2 DLL files, namely ‘mscoree.dll’ and ‘kernel32.dll’.

DYNAMIC ANALYSIS:

First, we will execute the ransomware file in our VM and then try to analyze its working. While executing the file, a popup is observed like in the below screenshot.

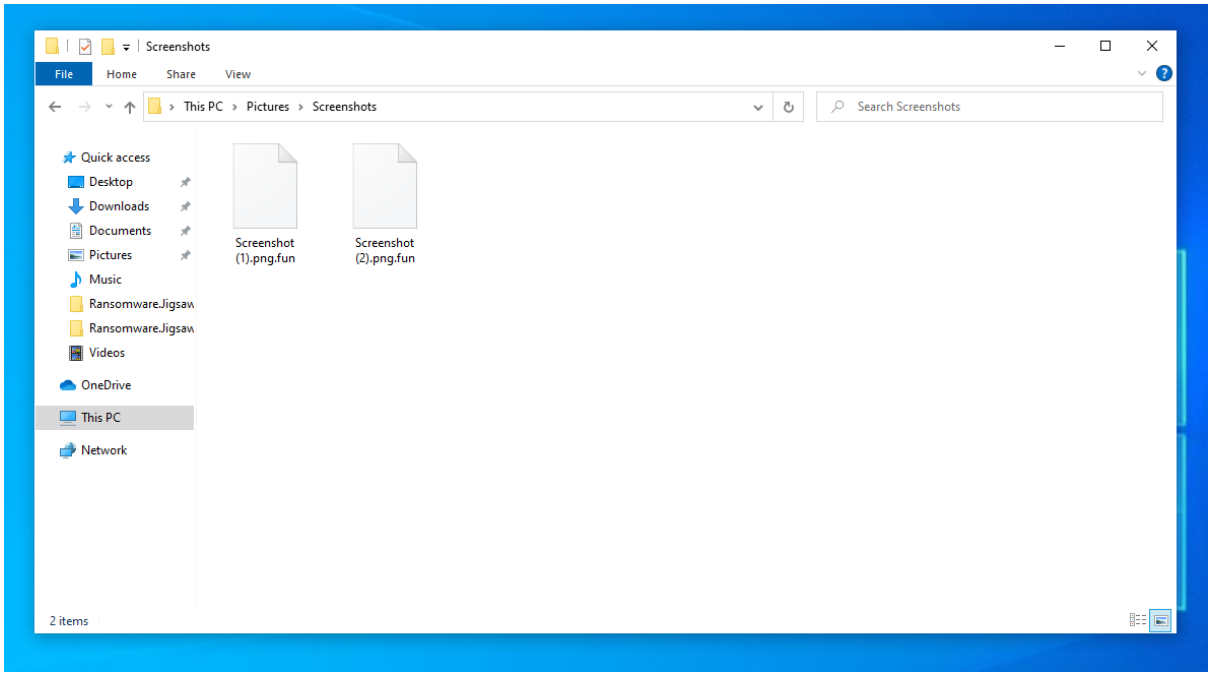


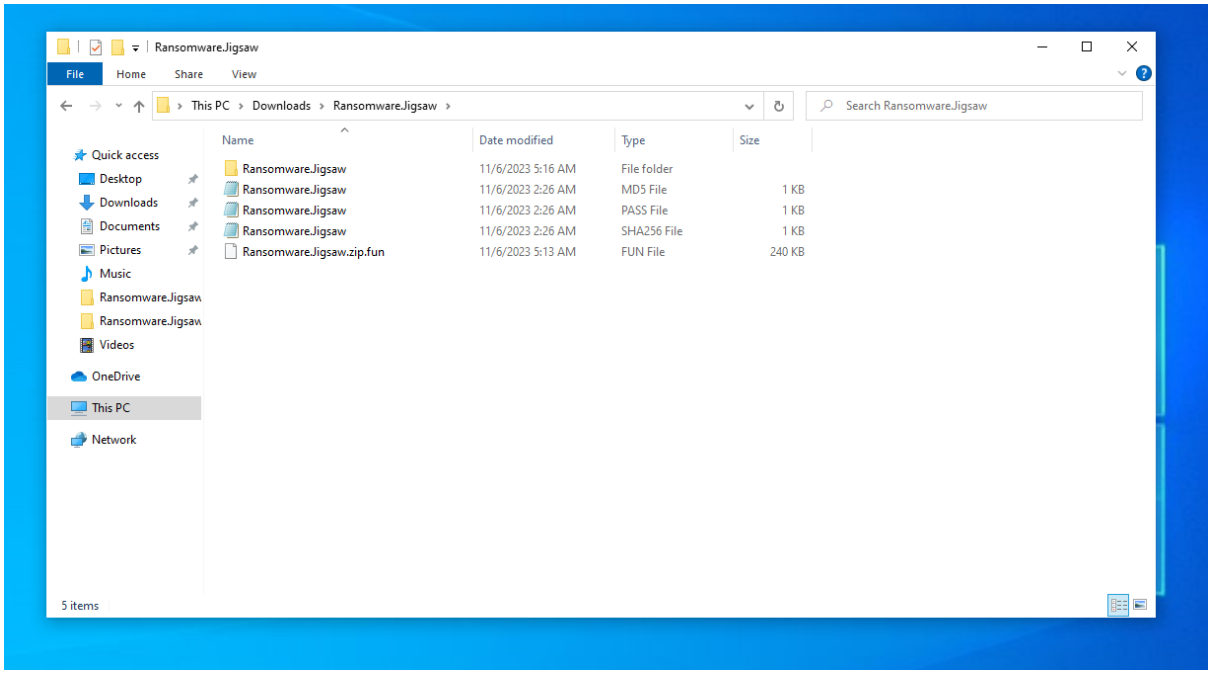
If we click ‘OK’ and wait for sometime, we can see the jigsaw program running like this.



It says that all the files have been encrypted and it asks us to pay .4 BTC for it to decrypt our files back to original and then, there is a timer running saying that, for every one hour one file will be deleted. Thus, we can see that the malware has executed successfully.

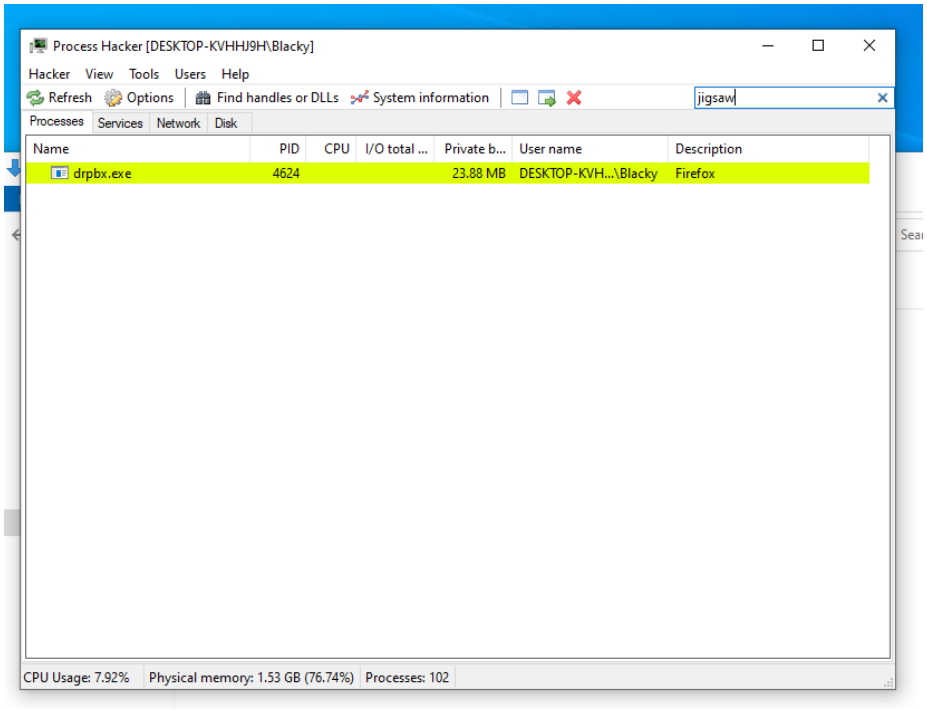
If we check the files in our system, we can see that they are all encrypted with ‘.fun’ extension.



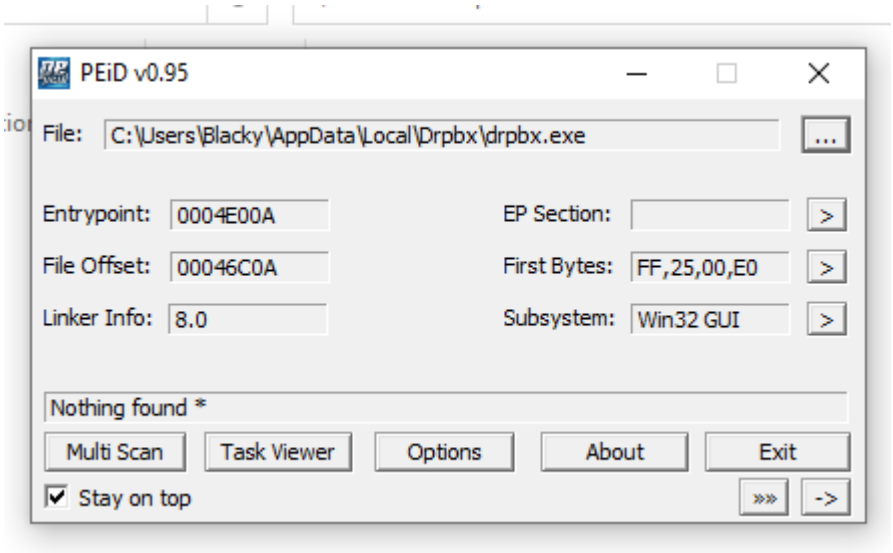


In the above screenshot, we can see that the type of a zip file is changed to FUN file as it has been encrypted.

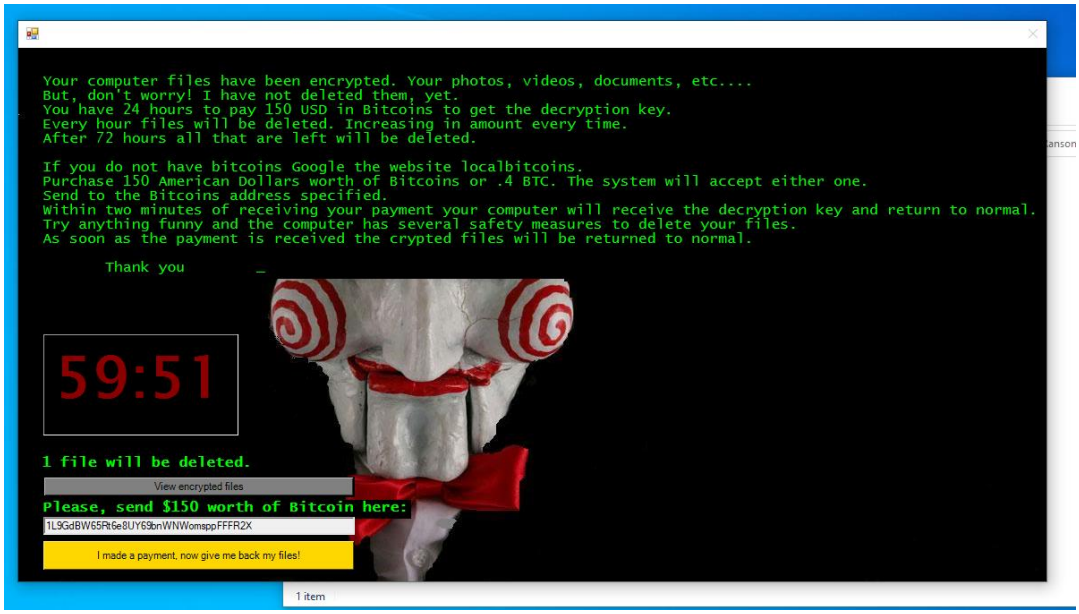
Now, we use Process Hacker tool to see what processes are running and we can notice that there is nothing under the name jigsaw. But, if we use the search option, we find that there is something called ‘drpbx.exe’ running, through which we can conclude that after execution, the file has renamed itself to ‘drpbx.exe’ .



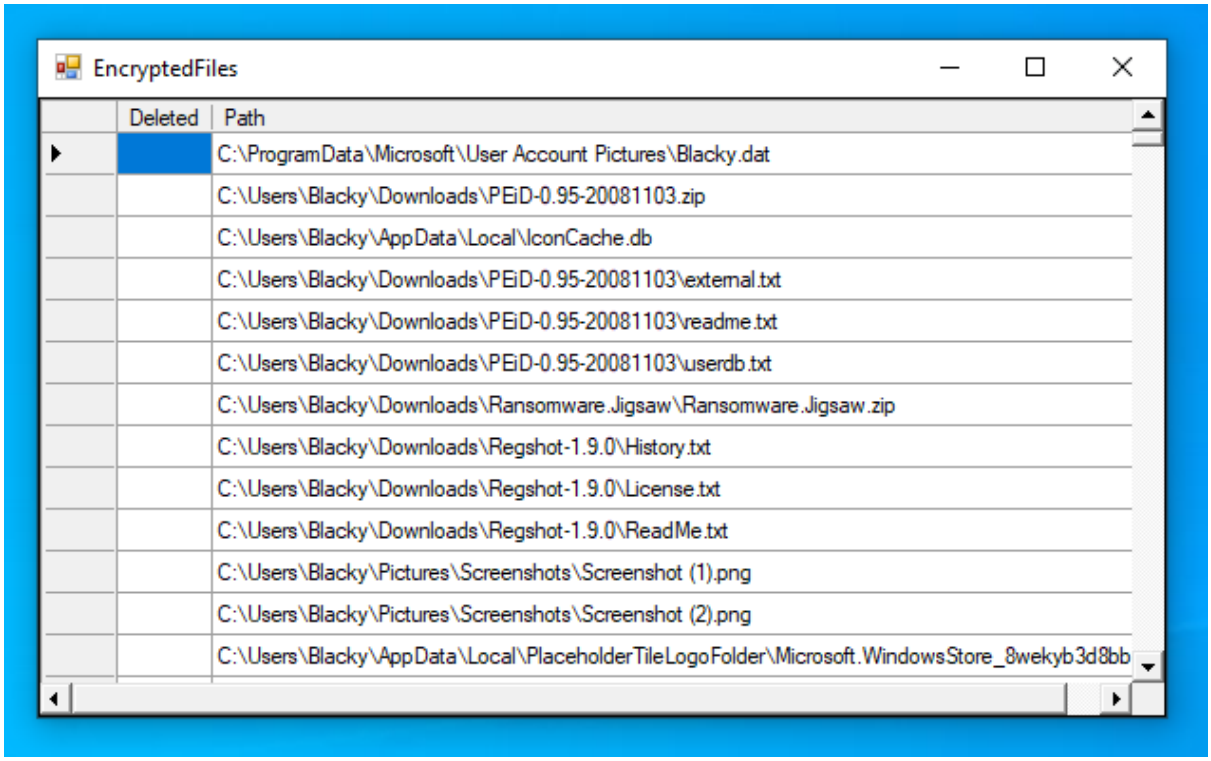
We then try to find this file's packer using PEid by giving it as input, but then there is nothing found.



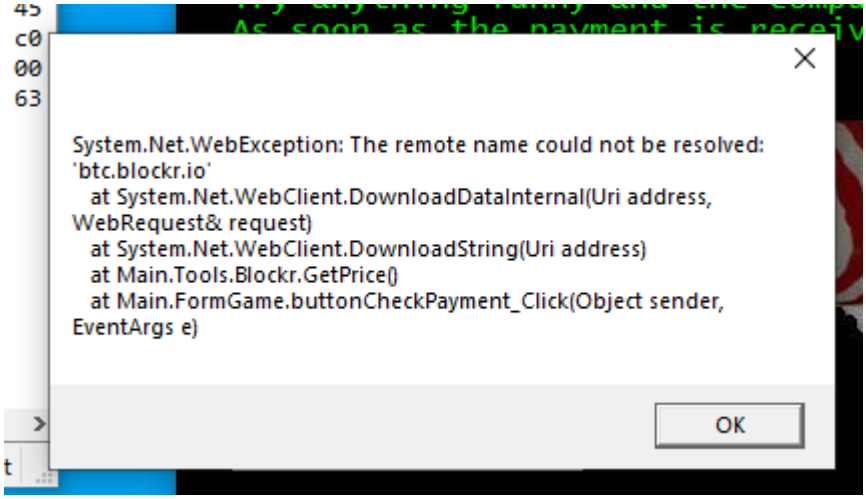
In the executed malware, we can find 2 buttons, one saying "View all encrypted files" and the other saying "I made my payment, now give me back my files!".



If we click on “View encrypted files”, it provides us with a list of system files that are encrypted.



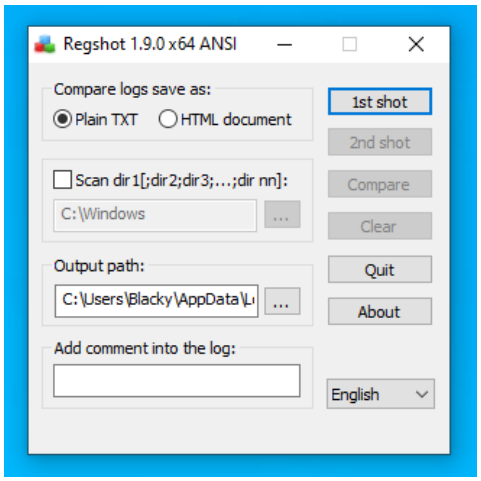
We then use WireShark to analyze the traffic of the network, then we click the other button “I made my payment, now give me back my files!” and check the packets being sent. This is the prompt we get when we try to click the button.



Analyzing the traffic, we can see that a DNS packet is being sent to “btc.blockr.io”. This basically checks with the host if the payment is done or not.

```
    [Coloring Rule String: udp]
▾ Ethernet II, Src: PcsCompu_ba:19:f7 (08:00:27:ba:19:f7), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
  › Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
  › Source: PcsCompu_ba:19:f7 (08:00:27:ba:19:f7)
    Type: IPv4 (0x0800)
› Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.29.1
› User Datagram Protocol, Src Port: 57303, Dst Port: 53
▾ Domain Name System (query)
    Transaction ID: 0xac22
  › Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ btc.blockr.io: type A, class IN
        Name: btc.blockr.io
        [Name Length: 13]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
    [Response In: 2]
```

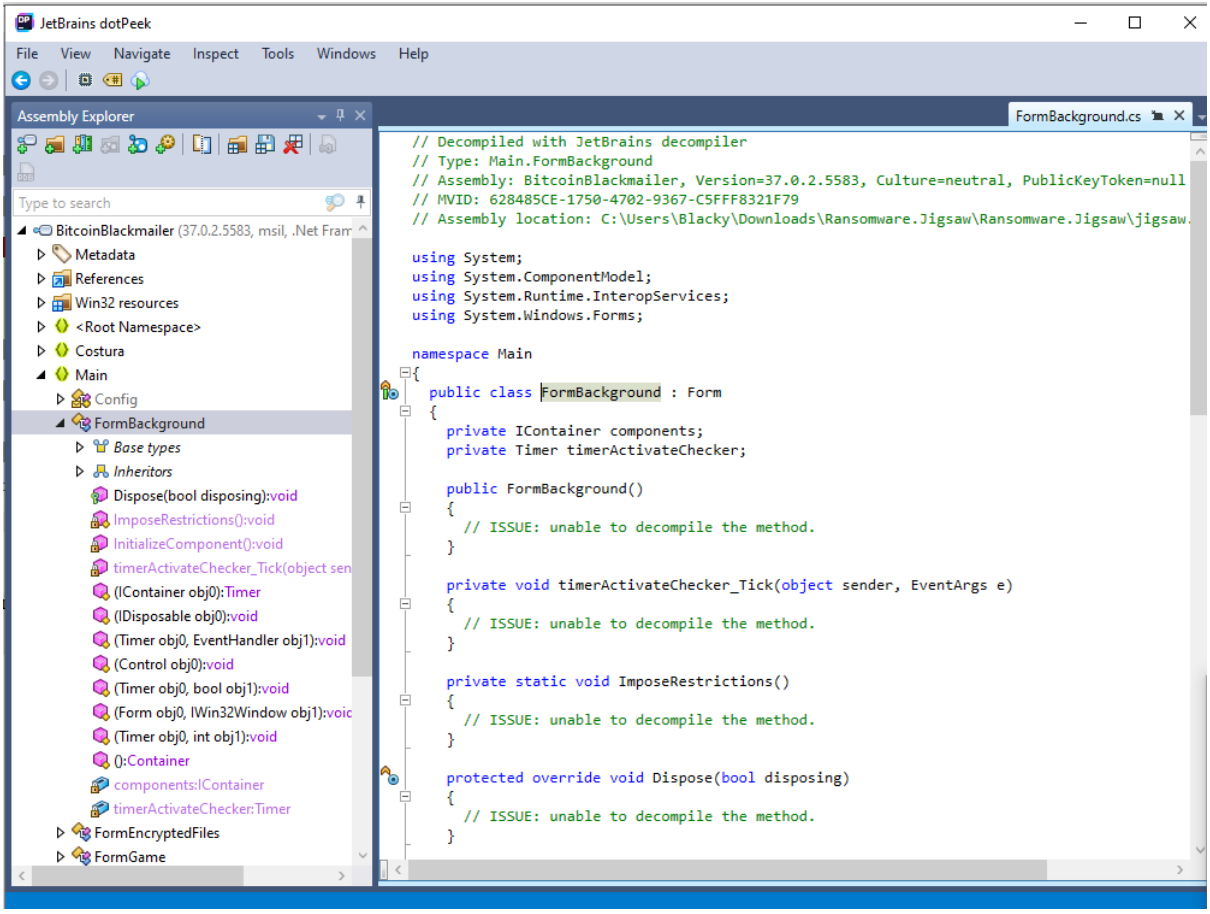
Then, we use the RegShot tool to analyze the change in register values.



After clicking first shot, then second shot and then comparing both values, we can observe that there has been changes in 4 register values.

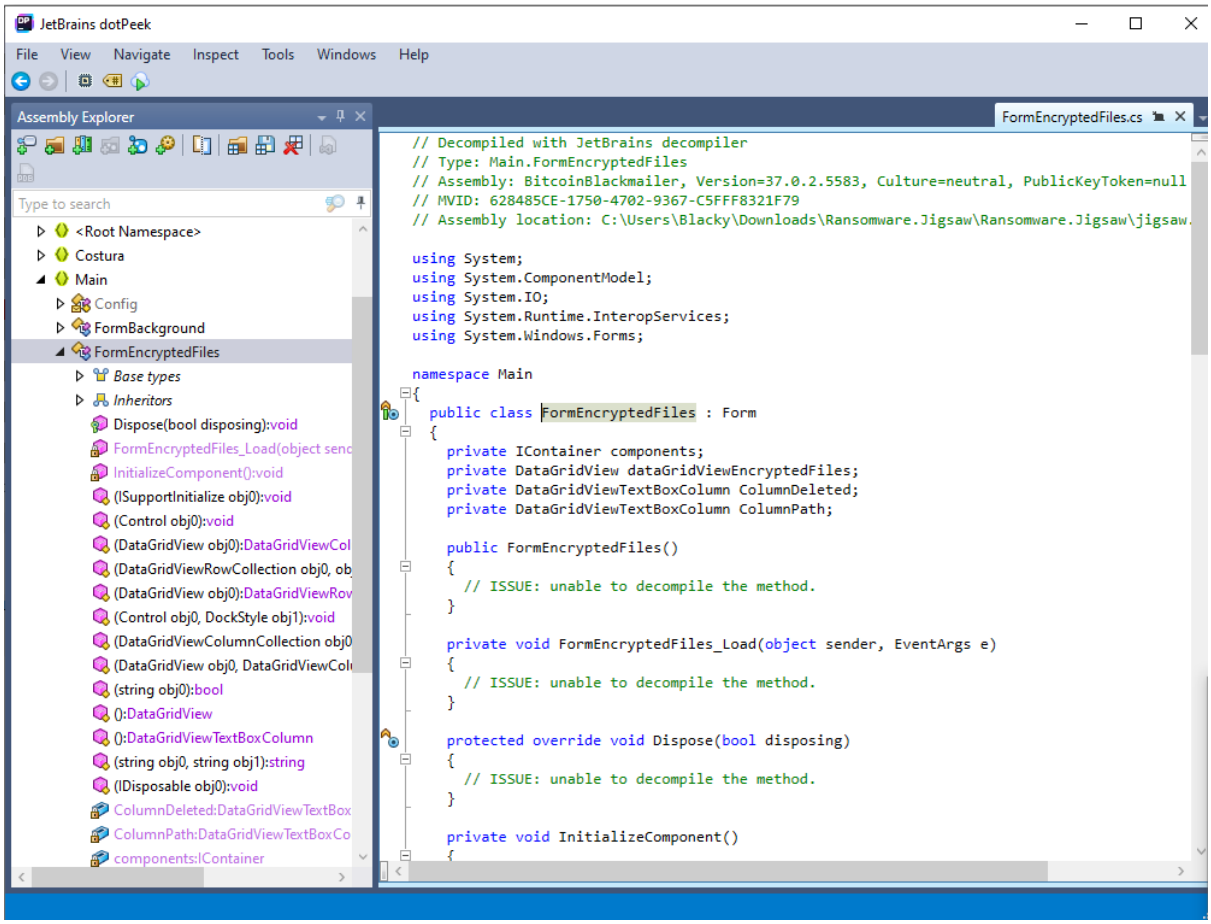
```
~res-x64 - Notepad
File  Edit  Format  View  Help
Regshot 1.9.0 x64 ANSI
Comments:
Datetime: 2023/11/6 14:38:06  ,  2023/11/6 14:38:53
Computer: DESKTOP-KVHHJ9H , DESKTOP-KVHHJ9H
Username: Blacky , Blacky

----------------------------------
Values modified: 4
----------------------------------
HKLM\SOFTWARE\Microsoft\Multimedia\Audio\Journal\Render:  53 00 57 00 44 00 5C 00 4D 00 4D 00 44 00 45 00 56 00 41 00 50 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
HKLM\SOFTWARE\Microsoft\Multimedia\Audio\Journal\Render:  53 00 57 00 44 00 5C 00 4D 00 4D 00 44 00 45 00 56 00 41 00 50 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\VFUProvider\StartTime:  4C 29 28 D7 BE 10 DA 01
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\VFUProvider\StartTime:  0D C0 FD FA BE 10 DA 01
HKU\S-1-5-20\SOFTWARE\Microsoft\Windows\CurrentVersion\DeliveryOptimization\Usage\CPUpct: "0.000000"
HKU\S-1-5-20\SOFTWARE\Microsoft\Windows\CurrentVersion\DeliveryOptimization\Usage\CPUpct: "0.006499"
HKU\S-1-5-20\SOFTWARE\Microsoft\Windows\CurrentVersion\DeliveryOptimization\Usage\MemoryUsageKB:  B0 12 00 00 00 00 00 00
HKU\S-1-5-20\SOFTWARE\Microsoft\Windows\CurrentVersion\DeliveryOptimization\Usage\MemoryUsageKB:  E4 12 00 00 00 00 00 00

----------------------------------
Total changes: 4
----------------------------------
```

Since the ransomware is designed using .net framework, we use Dotpeek tool, which is a .net decompiler. This tool is widely used because it is more user-friendly. Opening the file in the decompiler, we can observe the following under the main.

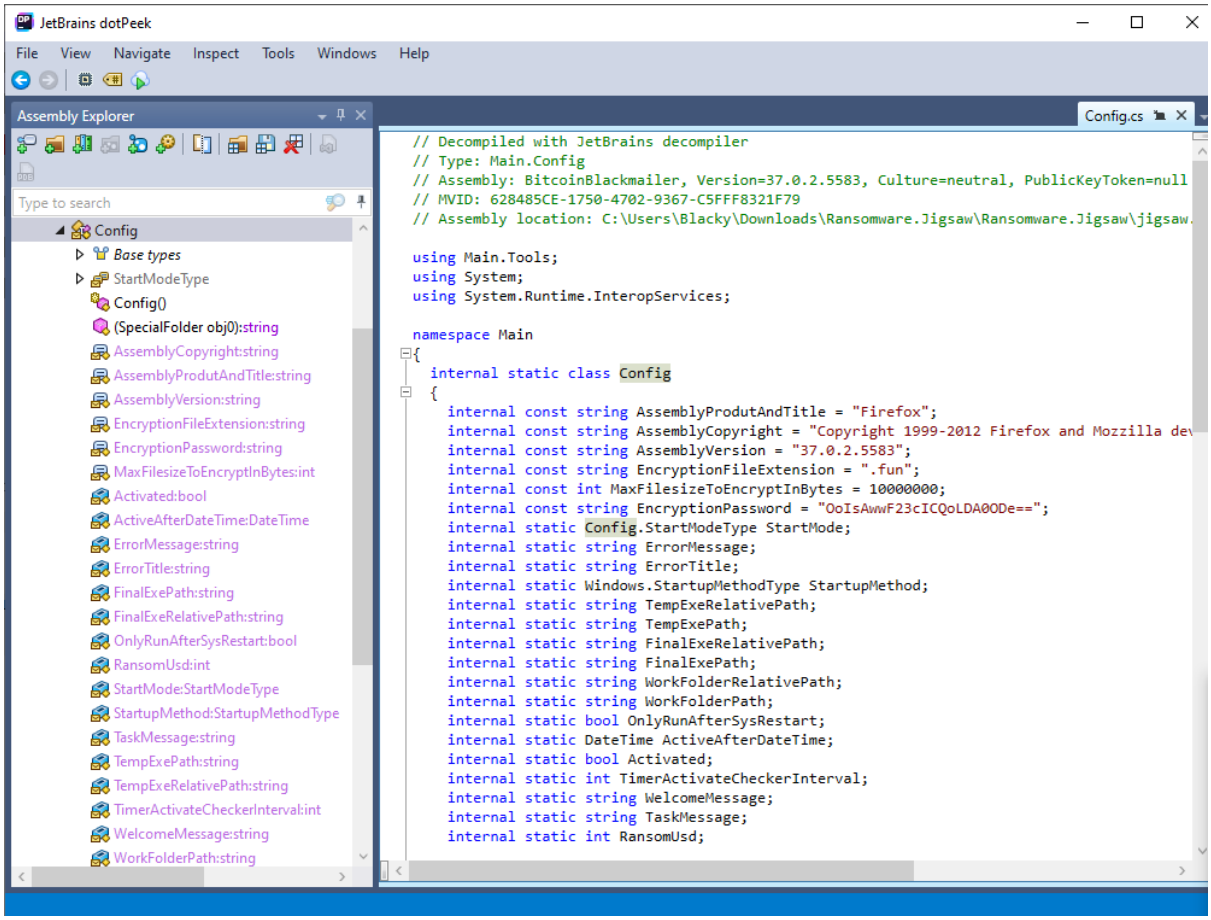


From the above screenshot, we can observe the name of the program as ‘BitcoinBlackmailer’, which we already found using the strings function of KALI LINUX. Also, we found the “FormBackground” class which we can observe to be responsible for the UI interface and the timer of the ransomware program.

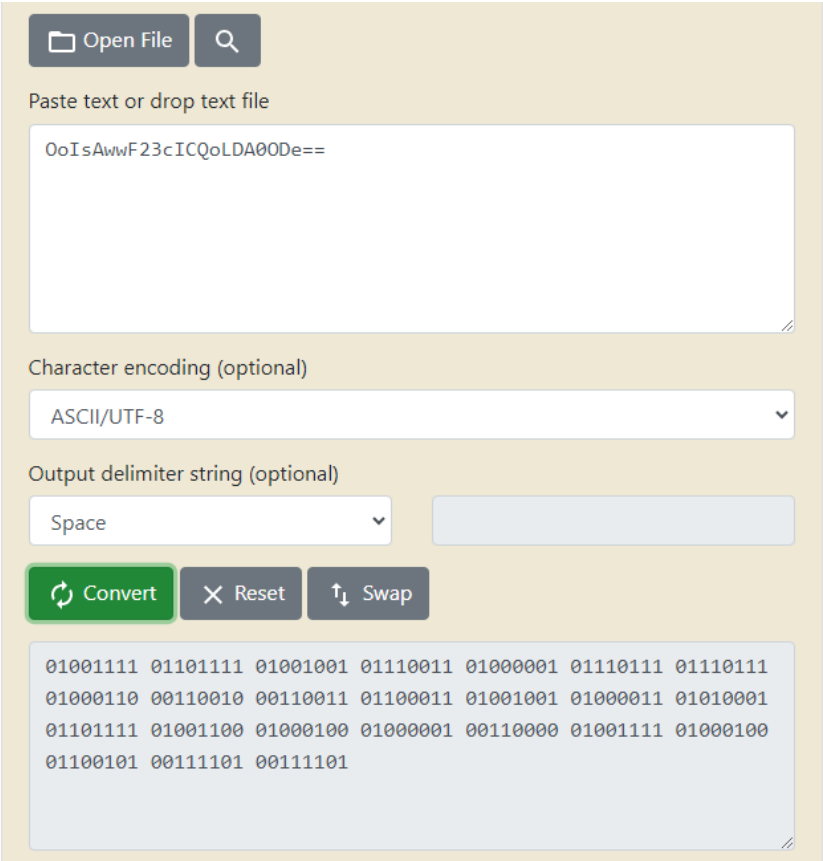
We can also see a class called “FormEncryptedFiles” which upon observing, we can conclude that it uses the same key for both encryption and decryption. Thus, this is a symmetric cipher.



We then observe the config class and we can find the extension of the encrypted files, i.e., “.fun” and also the password used for encryption. It is given as a constant string and we can infer that the value does not change at any point of time. So from our above observation of the program using symmetric cipher, we might have found our key.



Converting the password string to binary, we get the following.



From the above screenshot, we can find that the binary is of the length 192-bits and thus we can infer that the cipher uses a 192-bit key for encryption and decryption.

Upon searching in the program for the different strings we found using KALI LINUX, we found the functions “AesCryptoServiceProvider” and “CreateDecryptor” which we can observe to be responsible for encryption and decryption of files. Thus, we can say that the malicious program uses AES block cipher to encrypt the files.

```
[SecurityCritical]
public AesCryptoServiceProvider()
{
  string providerName = "Microsoft Enhanced RSA and AES Cryptographic Provider";
  if (Environment.OSVersion.Version.Major == 5 && Environment.OSVersion.Version.Minor == 1)
    providerName = "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)";
  this.m_cspHandle = CapiNative.AcquireCsp((string) null, providerName, CapiNative.ProviderType.RsaAes, CapiNative.CryptAcquireContextFlags.VerifyContext, true);
  this.FeedbackSizeValue = 8;
  int defaultKeySize = 0;
  if (AesCryptoServiceProvider.FindSupportedKeySizes(this.m_cspHandle, out defaultKeySize).Length == 0)
    throw new PlatformNotSupportedException(SR.GetString("Cryptography_PlatformNotSupported"));
  this.KeySizeValue = defaultKeySize;
}

public override byte[] Key
{
  [SecuritySafeCritical] get
  {
    if (this.m_key == null || this.m_key.IsInvalid || this.m_key.IsClosed)
      this.GenerateKey();
    return CapiNative.ExportSymmetricKey(this.m_key);
  }
  [SecuritySafeCritical] set
  {
    byte[] key = value != null ? (byte[]) value.Clone() : throw new ArgumentNullException(nameof (value));
    if (!this.ValidKeySize(key.Length * 8))
      throw new CryptographicException(SR.GetString("Cryptography_InvalidKeySize"));
    SafeCapiKeyHandle safeCapiKeyHandle = CapiNative.ImportSymmetricKey(this.m_cspHandle, AesCryptoServiceProvider.GetAlgorithmId(key.Length * 8), key);
    if (this.m_key != null)
      this.m_key.Dispose();
    this.m_key = safeCapiKeyHandle;
    this.KeySizeValue = key.Length * 8;
  }
}

public override int KeySize
{
  get => base.KeySize;
  [SecuritySafeCritical] set
  {
    base.KeySize = value;
    if (this.m_key == null)
      return;
    this.m_key.Dispose();
  }
}
```

```
[SecuritySafeCritical]
public override ICryptoTransform CreateDecryptor()
{
  if (this.m_key == null || this.m_key.IsInvalid || this.m_key.IsClosed)
    throw new CryptographicException(SR.GetString("Cryptography_DecryptWithNoKey"));
  return this.CreateDecryptor(this.m_key, this.IVValue);
}

[SecuritySafeCritical]
public override ICryptoTransform CreateDecryptor(byte[] key, byte[] iv)
{
  if (key == null)
    throw new ArgumentNullException(nameof (key));
  if (!this.ValidKeySize(key.Length * 8))
    throw new ArgumentException(SR.GetString("Cryptography_InvalidKeySize"), nameof (key));
  if (iv != null && iv.Length * 8 != this.BlockSizeValue)
    throw new ArgumentException(SR.GetString("Cryptography_InvalidIVSize"), nameof (iv));
  byte[] key1 = (byte[]) key.Clone();
  byte[] iv1 = (byte[]) null;
  if (iv != null)
    iv1 = (byte[]) iv.Clone();
  using (SafeCapiKeyHandle key2 = CapiNative.ImportSymmetricKey(this.m_cspHandle, AesCryptoServiceProvider.GetAlgorithmId(key1.Length * 8), key1))
    return this.CreateDecryptor(key2, iv1);
}
```

CONCLUSION:

Using different tools, we have performed static and dynamic analysis of the Jigsaw ransomware and have observed and found many things such as cipher used, key, etc., which can be crucial for a reverse engineer to decode the malicious program and break the ransomware.