Analysis of a New HawkEye Variant

By Xiaopeng Zhang

Published: 2019-06-18 · Archived: 2026-04-05 14:44:21 UTC

Threat Analysis by FortiGuard Labs

Background

FortiGuard Labs recently captured a malware being spread by a phishing email. After a quick analysis, I discovered that it was a new variant of the HawkEye malware.

HawkEye is known as a keylogger and an application credential stealing malware. Over past few years, we have seen it spread by email, and carried in MS Word documents, Excel files, PowerPoint files, and RTF files. In this analysis, I am going to provide an overview of what this new variant can do to a victim's system.

Distribution and Download

Here is the email content, masquerading as an airline ticket confirmation, which asks the targeted victim to click on a link.

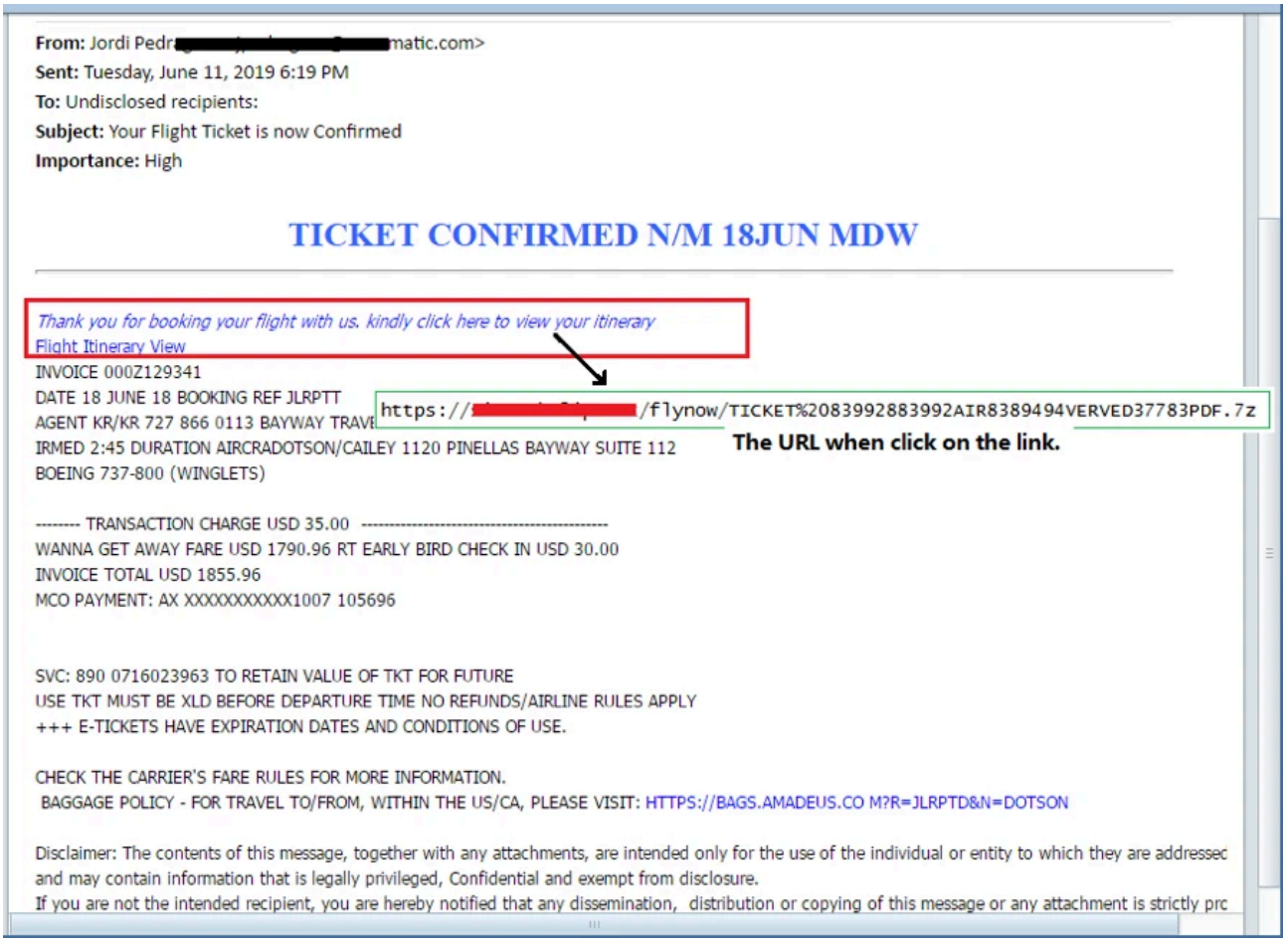

Figure 1. The email content

It was designed so that a victim downloads a 7z file from the link shown in figure 1 that contains this new variant of HawkEye and runs it on the victim's system.

Unfortunately, on initial analysis the URL was not available and I received a "404 Not Found" message in the browser.

Browsing to its main page. It turned out to be an FTP service, containing several related network folders about this campaign, with most containing the same malware sample (Figure 2).

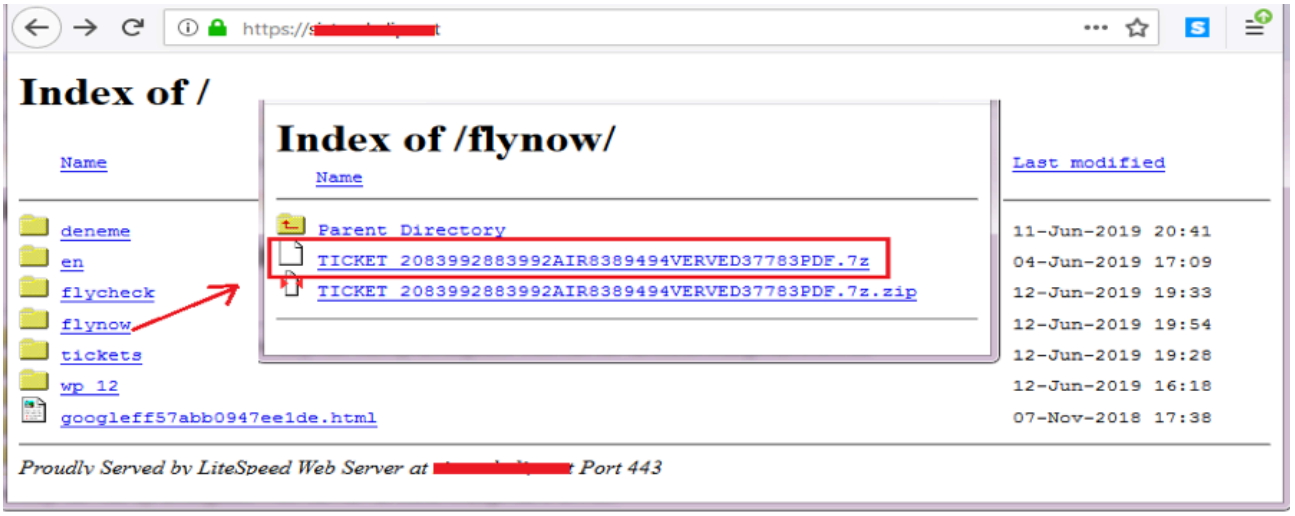

Figure 2. Screenshot of the main page

After the downloaded 7z file was decompressed, we retrieved the EXE file "TICKET%2083992883992AIR8389494VERVED37783PDF.exe", which is the new variant of HawkEye.

Start HawkEye

Once HawkEye started, it spawned a suspended child process, "RegAsm.exe", from the Microsoft .Net framework installation directory – which is a tool for Assembly Registration. Meanwhile, HawkEye extracted a PE file into its memory and then moved the PE file into "RegAsm.exe". The dynamically extracted PE file is the main program of HawkEye. It's called "HawkEye_RegAsm," to differentiate these files in the analysis. HawkEye_RegAsm began running after resuming running "RegAsm.exe" after being suspended.

HawkEye_RegAsm is a .Net written program, which is packed by ConfuserEx v1.0.0 to protect itself. This creates a big challenge for analysts to read its code and analyze it. The code was actually totally obfuscated, as shown in figure 3.

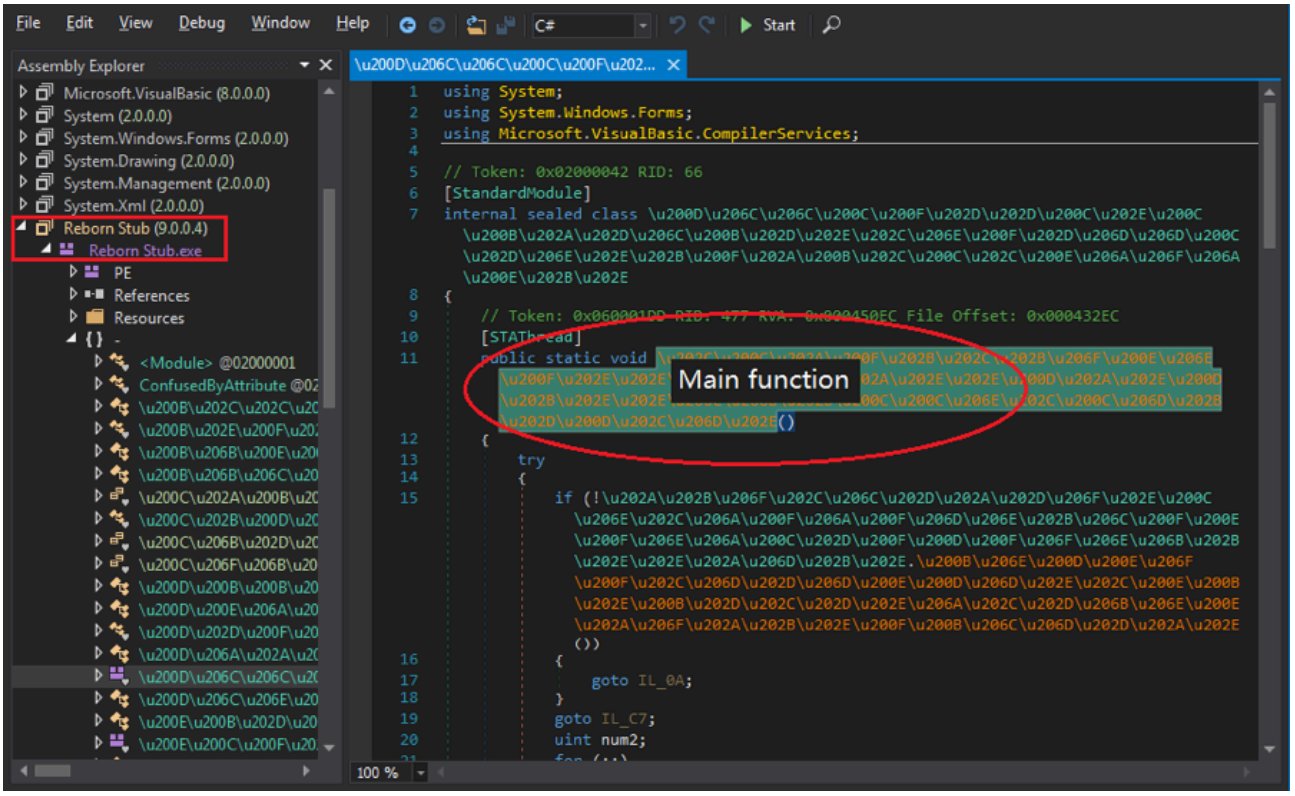

Figure 3. Obfuscated Entry function of HawkEye_RegAsm

After sleeping 10 seconds, HawkEye_RegAsm starts its work on the victim's system. Through analysis so far, it appears to mainly perform the following functions:

1> Set up clipboard logger
2> Set up keyboard logger
3> Spawn another two child processes "vbc.exe", both from the .Net framework directory as well.
4> Send collected data to an email address using SMTP from time to time (every 10 minues).

HawkEye_RegAsm starts a thread to perform the above tasks, and then every 10 minutes it sends its collected information to its Yandex email address.

HawkEye_RegAsm sets up a clipboard and keyboard logger using Windows-native APIs (such as SetWindowsHookEx, SetClipboardViewer, etc.) Its local functions can record victim's behaviors when the victim types on the keyboard as well as when copying data into the system clipboard.

Figure 4 shows an example of the information that HawkEye_RegAsm collected from its keyboard and clipboard logger, as well as the software title from when the event occurred.

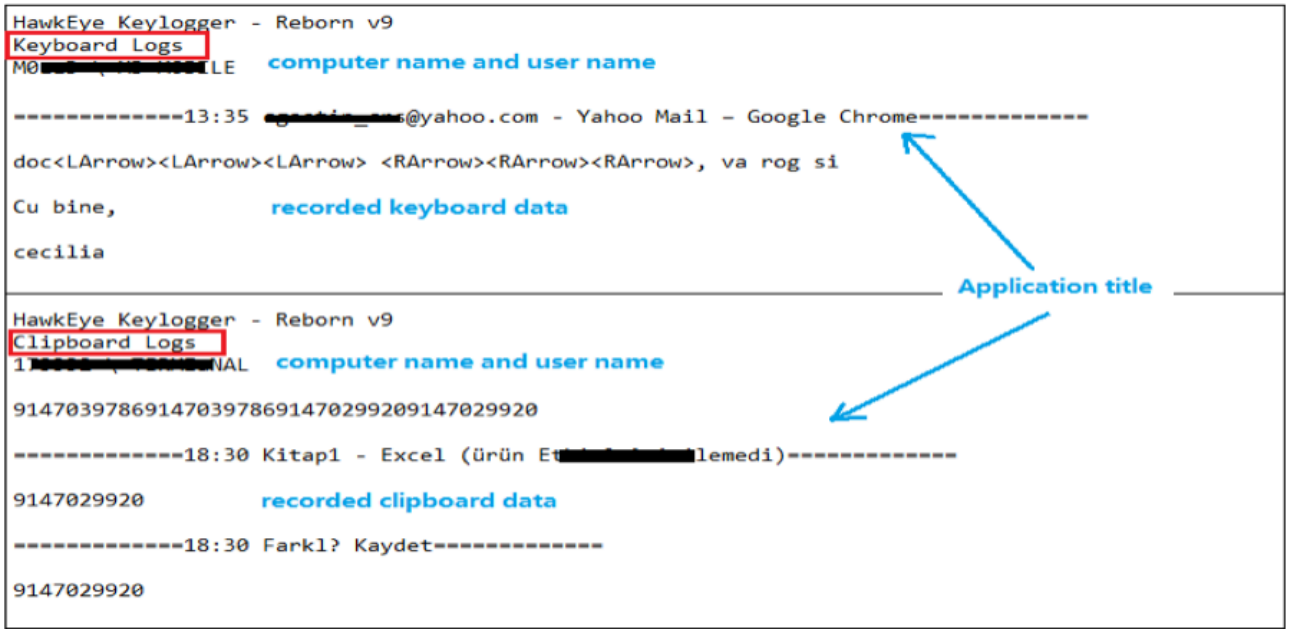

Figure 4. Example of collected Clipboard and Keyboard data

Collecting Credentials from Saved Credential Storage

HawkEye_RegAsm performs a similar task as to the RegAsm.exe. It spawns two suspended child processes, “vbc.exe”, which are from the same directory as RegAsm.exe. HawkEye dynamically extracts two PE files into its memory, which are then copied into the two newly created child processes of “vbc.exe”. It also modifies its ThreadContext data (It calls the API, SetThreadContext) and makes its entry point to the transfered PE file. When “vbc.exe” resumes running it can be executed. It’s a trick that malware often performs to camouflage itself behind of a normal process.

The two “vbc.exe” processes collect credentials from the victim’s system. One is used to collect the credentials of browsers. The other one focuses on email clients and IM clients to steal credentials and profiles. Both PE files injected into “vbc.exe” have the same code framework. They first call a function to collect credentials and save them in memory, and second, it reads the collected data, formats it, and saves it to a tmp file from its command line parameter.

Figure 5 shows HawkEye calling the CreateProcess API to start one of the two “vbc.exe” processes, with the parameter shown below in the “Locals” sub-tab. You can see the full path of “vbc.exe”. “/stext ""C:\Users*********\AppData\Local\Temp\tmpBE3D.tmp""" is the parameter passed to it. The tmp file name is random and different from the two “vbc.exe” processes, which temporarily saves collected credentials.

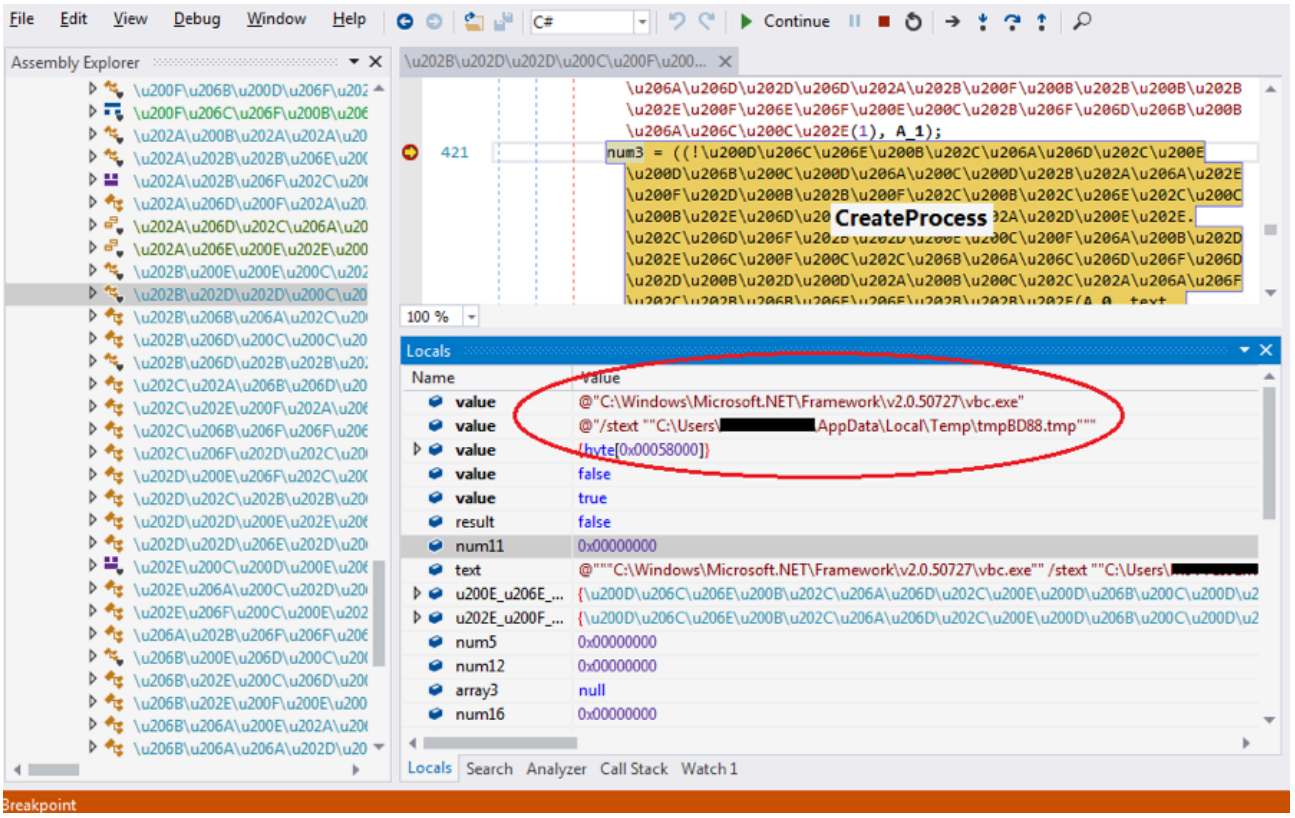

Figure 5. Break on when calling CreateProcess to start a “vbc.exe”

The two PE files are not packer protected and not .Net written program.

The first “vbs.exe” collects credentials from victim’s browsers and the system credential manager for IE.

In my analysis, this variant of HawkEye focuses on the following browsers:

Microsoft Internet Explorer, Google Chrome, Apple Safari, Opera, Mozilla Sunbird, Mozilla Firefox, Mozilla Portable Thunderbird, Mozilla SeaMonkey, YandexBrowser, Vivaldi browser, and more.

Figure 6 shows some strings defined in the ASM code of the browsers that the HawkEye malware wants to collect credentials from.

The collected credentials are then saved into the tmp file from its command line parameter. HawkEye_RegAsm keeps checking this tmp file, and once the credentials are collected, it is done. HawkEye_RegAsm then reads the entire data of this tmp file into its memory and the deletes it immediately.

```
0044B89A                align 10h
0044B8A0 ; wchar_t aYandexYandexbr
0044B8A0 aYandexYandexbr:                        ; DATA XREF: sub_411E4C+430↑o
0044B8A0                                         ; sub_411E4C+44F↑o
0044B8A0                unicode 0, <Yandex\YandexBrowser\User Data\Default\Login Data>
0044B904                align 8
0044B908 ; wchar_t aVivaldiUserDat
0044B908 aVivaldiUserDat:                        ; DATA XREF: sub_411E4C+4EC↑o
0044B908                                         ; sub_411E4C+50B↑o
0044B908                unicode 0, <Vivaldi\User Data\Default\Login Data>,0
0044B952                align 4
0044B954 ; wchar_t aGoogleChromeUs
0044B954 aGoogleChromeUs:                        ; DATA XREF: sub_411E4C+616↑o
0044B954                                         ; sub_411E4C+632↑o
0044B954                unicode 0, <Google\Chrome\User Data>,0
0044B984 ; wchar_t aGoogleChromeSx
0044B984 aGoogleChromeSx:                        ; DATA XREF: sub_411E4C+66A↑o
0044B984                                         ; sub_411E4C+686↑o
0044B984                unicode 0, <Google\Chrome SxS\User Data>,0
0044B9BC ; wchar_t aChromiumUserDa
0044B9BC aChromiumUserDa:                        ; DATA XREF: sub_411E4C+6BE↑o
0044B9BC                                         ; sub_411E4C+6DA↑o
0044B9BC                unicode 0, <Chromium\User Data>,0
0044B9E2                align 4
0044B9E4 ; wchar_t aOperaOperaWand
0044B9E4 aOperaOperaWand:                        ; DATA XREF: sub_411E4C+823↑o
0044B9E4                                         ; sub_411E4C+83F↑o
0044B9E4                unicode 0, <Opera\Opera\wand.dat>,0
0044BA0E                align 10h
0044BA10 ; wchar_t aOperaOpera7Pro
0044BA10 aOperaOpera7Pro:                        ; DATA XREF: sub_411E4C+88D↑o
0044BA10                                         ; sub_411E4C+8A9↑o
0044BA10                unicode 0, <Opera\Opera7\profile\wand.dat>,0
0044BA4C ; wchar_t aOpera
0044BA4C aOpera:                                 ; DATA XREF: sub_411E4C+92E↑o
0044BA4C                                         ; sub_411E4C+94A↑o
0044BA4C                unicode 0, <Opera>,0
0044BA58 ; wchar_t aOperaSoftwareO
0044BA58 aOperaSoftwareO:                        ; DATA XREF: sub_411E4C+A56↑o
0044BA58                                         ; sub_411E4C+A72↑o
0044BA58                unicode 0, <Opera Software\Opera Stable\Login Data>,0
0044BAA6                align 4
```

Figure 6. Browsers' information defined in the first PE file

The second PE file in "vbc.exe"collects profile and credential information of the email and IM software client installed on a victim's machine.

The clients it targets are:

Qualcomm Eudora, Mozilla Thunderbird, MS Office Outlook, IncrediMail, Groupmail, MSNMessenger, Yahoo!Pager/Yahoo!Messenger and Windows Mail.

Below is an example list that HawkEye stole from the Chrome browser on my test machine. As you can see, it includes login URL, Browser name, User name, Password, Created time, and the full path of the file where the collected information came from.

```
================================================
URL             : https://www.amazon.com/ap/signin
Web Browser     : Chrome
User Name       : [redacted]@gmail.com
Password        : [redacted]1
Password Strength : Strong
User Name Field   : email
Password Field    : password
Created Time      : 11/8/2018 5:15:46 PM
Modified Time     :
Filename    : C:\Users\*******\AppData\Local\Google\Chrome\User Data\Default\Login Data
================================================
```

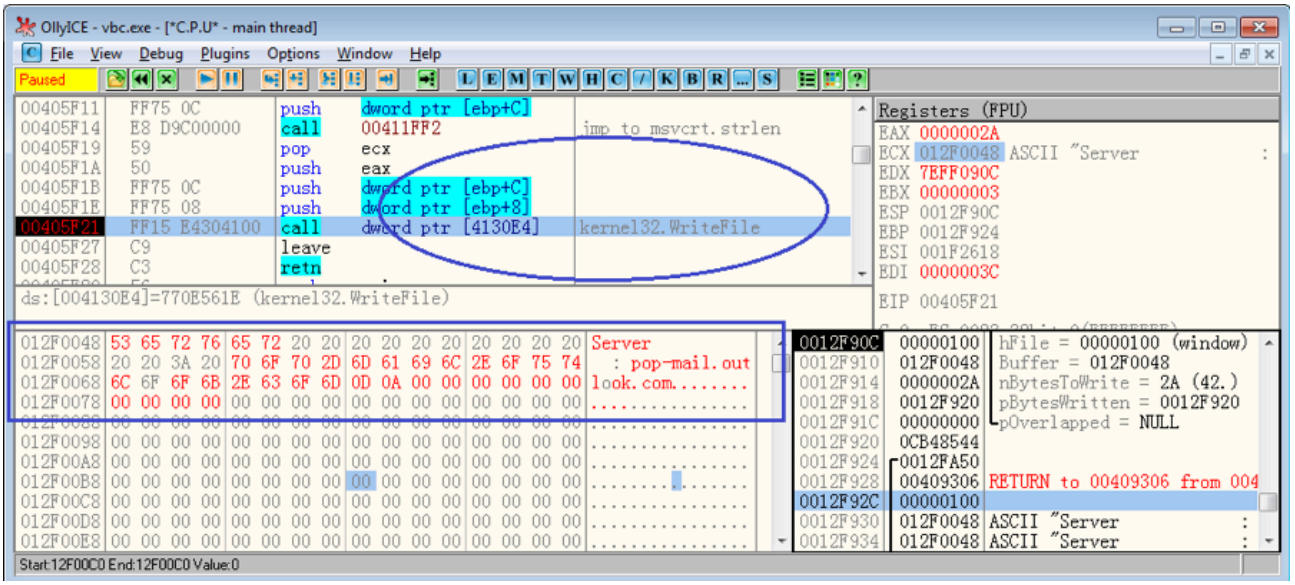

Figure 7. “vbc.exe” saves collected server address information to tmp file

The second PE file in “vbc.exe” not only collects the client’s login username and password, but also profile information, such as the recipent Server address, recipient Server Port, protocol Type (POP3), SMTP Server, SMTP Port, etc. Figure 7 shows a screenshot of Ollydbg when “vbc.exe” was about to write the collected recipient Server addresses into its tmp file. It writes one line once. The same tmp file is finally read by HawkEye_RegAsm and then deleted.

On my test machine, I only installed MS Outlook with one account. My test account and server profile were collected and put in the structure shown below, which would normally be sent to the attacker’s email box.

```
==============================================
Name          : Myemail
Application     : MS Outlook 2002/2003/2007/2010
Email         : test_my21@outlook.com
Server        : pop-mail.outlook.com
Server Port     : 995
Secured       : No
Type          : POP3
User          : test_my21@outlook.com
Password        : Test,_1234
Profile       : Outlook
Password Strength : Strong
SMTP Server     : smtp-mail.outlook.com
SMTP Server Port  : 587
==============================================
```

Sending Collected Data to the Attacker via SMTP

Ok. Now let's go back to the main process of HawkEye_RegAsm, which controls all tasks of HawkEye and sends the victim's credentials. In its main program, it calls Thread.Sleep(600000), and pauses while collecting credentials every 10 minutes. That is, it reports the collected data to attacker the once every 10 minutes.

It first sends an HTTP request, *http://bot.whatismyipaddress.com*, to ask for my machine's public IP. This is a way to ensure that the victim's machine is able to access the internet. If it did not reply with a public IP, it stops sending collected data to the email box. In addition, the IP appears in the email subject so it can identify victims.

The attacker's email is in Yandex.mail, whose email account and password are used when sending collected data through the Yandex SMTP server. That's why I was able to get the attacker's email credentials while tracking the main program. You can see the screenshot in figure 8 when I was debugging it.

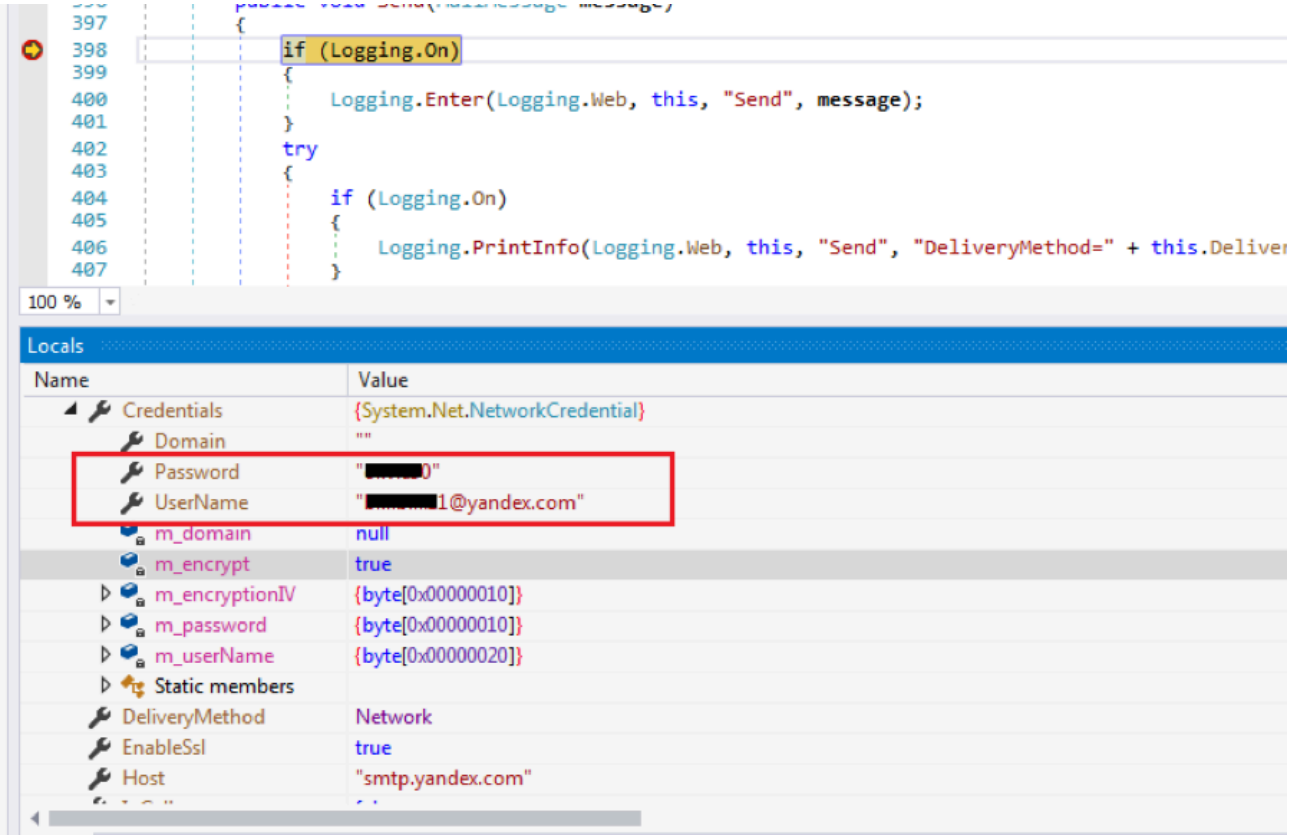

Figure 8. Sending collected data to the attacker's email box

Every ten minutes it sends packets such as that shown in Figure 9 to tell the attacker about what it has collected from the victim's machine using the keylogger, clipboard, browser credentials, and IM and email client credentials and profiles.

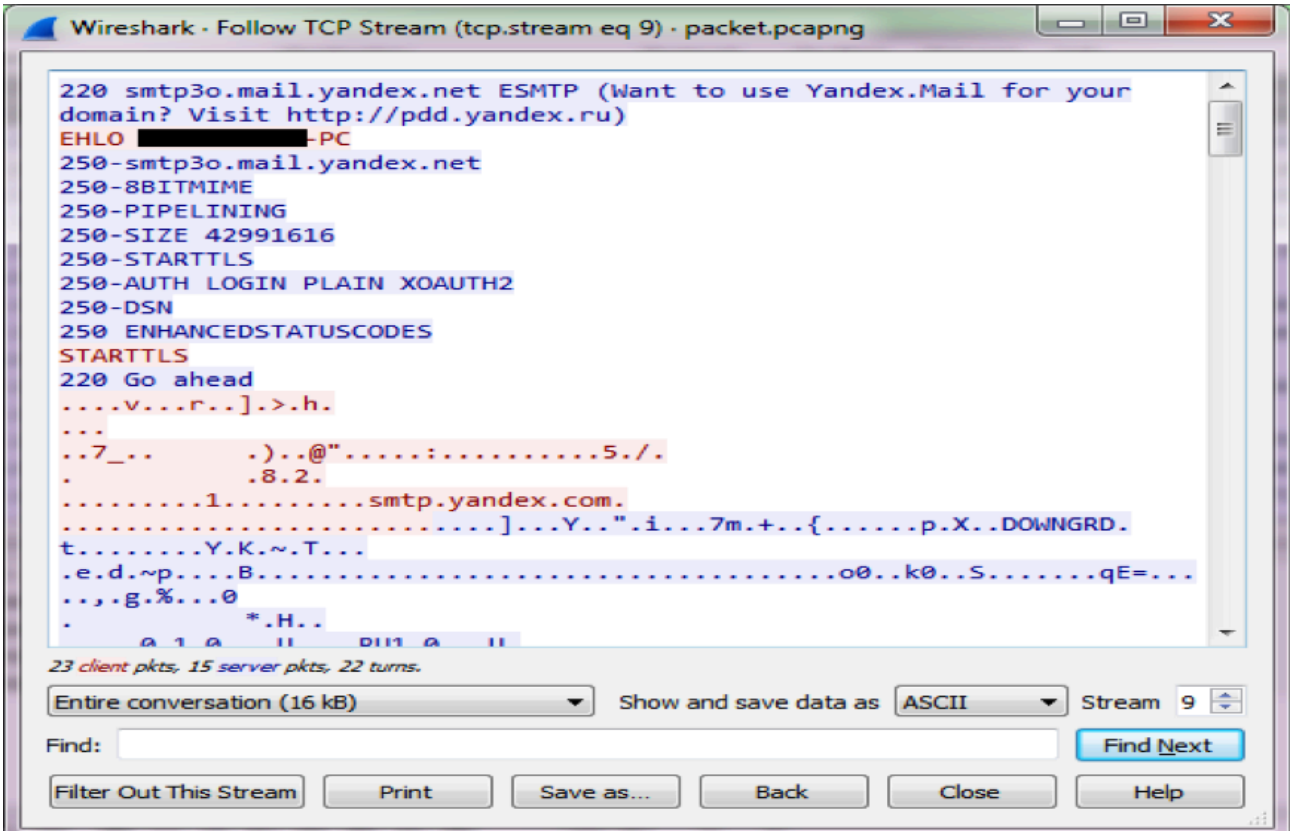

Figure 9. Sending collected data to the attacker's Yandex email address over SMTP

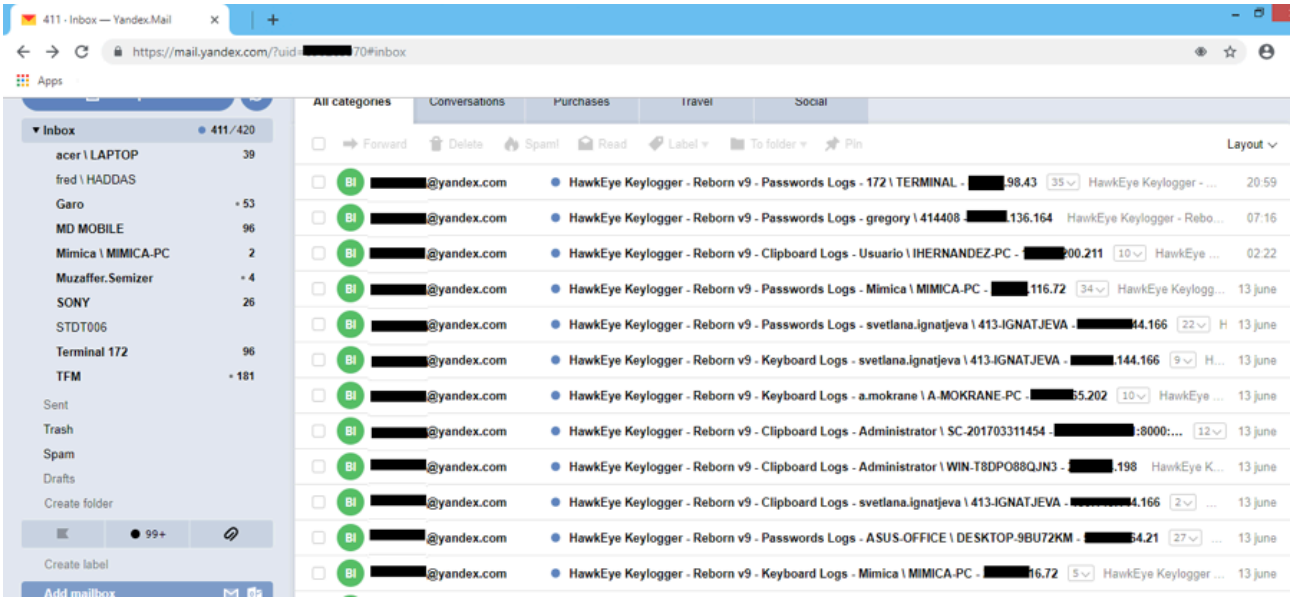

Figure 10. Glancing at an attacker's harvest

Visiting the attacker's account, we can see what the attacker has harvested, as shown in figure 10.

Solutions

The original URL in the email has been rated as "**Malicious Websites**" by the FortiGuard Web Filtering service. The decompressed exe file is detected as "**AutoIt/Injector.EAH!tr**" by the FortiGuard Antivirus service.

Sample SHA256

[TICKET%2083992883992AIR8389494VERVED37783PDF.exe]
3E7AD2A554F89B2A5E52E5C4843111342182DA4409A038CF800570B65A13F875

[Ticketmasterconfirmation3883948383948394.7z]
BBB46F812126FAEB543B02D143EF450887A043185AF98210D8F827924B31CF7A

[TKT8839483993993fligh booking ticket confirmationupdate.7z]
F2B921726D728037F9BA0C63FB6C31F77983C3A6E3938B46C411E80C218A2E84

Learn more about FortiGuard Labs and the FortiGuard Security Services portfolio. Sign up for our weekly FortiGuard Threat Brief.

Read about the FortiGuard Security Rating Service, which provides security audits and best practices.
