HawkEye Malware: Technical Analysis - ANY.RUN's Cybersecurity Blog

By Aaron Jornet Sales (RexorVc0)

Archived: 2026-04-05 13:27:07 UTC

Editor's note: The current article is authored by the threat researcher Aaron Jornet Sales, also known as RexorVc0. You can find him on X and LinkedIn.

HawkEye, also known as PredatorPain (Predator Pain), is a malware categorized as a keylogger, but over the years, it has adopted new functionalities that align it with the capabilities of other tools like stealers.

History of HawkEye

HawkEye emerged before 2010, with records of its use and sale dating back to 2008, making it quite long-lived. After several spearphishing campaigns in which this well-known malware was attached, it gained significant popularity starting in 2013.

This keylogger has been available on various dark web sites, even having dedicated websites where the tool was sold. However, this keylogger has been cracked for years and used by different actors without going through the subscription method imposed by its creators, whose price ranged between $20 and $50. This has contributed to its continued notoriety, and it has been used not only by criminal actors but also by script kiddies due to its ease of use.

Although it is not one of the most widely used malwares, it remains in active use and saw a significant resurgence during the COVID period. During this time, certain actors took advantage of the general hysteria to obtain company data through phishing campaigns.

Additionally, HawkEye has been used in conjunction with other loaders and/or malware that invoked this keylogger. Over its long trajectory, various actors and malware have been involved in attacks on companies, some of which include Galleon Gold, Mikroceen, iSPY crypter related with Gold Skyline, Remcos used on campaigns with HawkEye, Pony used on campaigns with HawkEye, etc.

The method of HawkEye's delivery has varied throughout its history, as have the types of sources behind the attacks. Nevertheless, it has been primarily involved in spearphishing campaigns, where attackers devised convincing scenarios to trick victims into downloading the malicious file, which could be a document, compressed file, or another malware acting as a loader for the keylogger.

It has also been used to target websites of portals typically accessed by companies, which were the main targets of the attacking groups. Another common method of spreading HawkEye was through "free" software, which turned out to be malware in disguise.

HawkEye's delivery methods are quite diverse compared to other malware. However, its execution and behavior have remained relatively consistent over the years. A behavior graph of what has been observed in recent months would look as follows:

HawkEye graph

During the analysis process, I typically spend weeks, even months, collecting samples to understand how they function as a whole based on the existing variants. Therefore, we may observe variations among those presented. In most executions, we encounter enormous trees of processes based on their activities.

To simplify, as you've seen in the previous graph, it's not as complex compared to other stealers or RATs. It generally consists of an executable that drops others in temporary paths, then injects code into one of them or into a .NET-related software. Later, in memory, it gathers all possible data and sends it to a C&C.

ProcDOT detonation chart

Going straight to the point, in an initial execution of one of the samples I analyzed, we see a rather extensive process—a succession of execution copies launched in temporary paths.

Process Tree execution (Image 1)

Process Tree execution (Image 2)

In this instance, they used the Roaming\Templates path, but this is highly variable depending on who created it. Generally speaking, they tend to abuse paths like AppData\Roaming and AppData\Temp, which are classic choices.

Paths commonly abused (Image 1)

Paths commonly abused (Image 2)

Paths commonly abused (Image 3)

Here's the list of paths observed for dropping files:

- C:\Users\<user>\AppData\Local\Temp\
- C:\Users\<user>\AppData\Roaming\
- C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Templates\
- C:\Users\<user>\AppData\Local\Temp\System\
- C:\Users\<user>\Music\

All of these files that are launched, and which we've observed executing in the previous step, are copies of themselves. The filenames are also highly variable, as you might expect, but they often try to have an icon that makes the victim think it's a legitimate program, or the malware description might be altered to make it seem like legitimate software.

Ultimately, after comparing the dropped files, we can see they are simple copies of the original, with the particularity that some versions launch them in hidden mode, so you can't see them unless you've enabled the "View hidden files" function in Windows.

Hidden files duplication graph

During these file droppings, we can encounter both replicas of the original file in different paths, as well as support files whose functionality is typically to establish persistence (or check if it's already done, and if not, do it) and to perform injector functions, which is a characteristic of this malware. In this case, the smaller binary is responsible for these actions.

Injector written in temporary folder

I check to see if there is any shared information between the two binaries and notice that certain parts of the code match the original. This will become relevant later, as right now we're seeing them separately, but everything will make sense afterward.

Comparison of the injector and the Hawkeye bin

After this step, we can see how persistence is established. PredatorPain isn't just a malware that establishes persistence once—it's been observed to check and establish persistence up to three different times, depending on the phases (Loader > Injector > Payload).

This makes it clear that the malware is determined to persist on the system, one way or another. At this stage, to avoid revealing persistence mechanisms through strings, it obfuscates a string and then decodes it to introduce, in this case, one of the binaries launched earlier. This practice isn't as common and adds a level of sophistication not found in other samples.

Hawkeye persistence in registers

Not only does it create persistence in the registry, but we also find samples that establish persistence in tasks using commands like the following:

```
schtasks.exe /Create /TN "<Path>\<TaskName>" /XML "<File>"
```

After observing its behavior in the early stages, we delve deeper into the entire execution thread throughout the analysis phase with debugging. I've followed several samples, and they're mostly similar—samples in .NET, sometimes obfuscated with tools like Confuser, Eaz, Reactor, or similar, which are relatively easy to deobfuscate.

Hawkeye code obfuscated

In most samples, I noticed heavy interaction with resources, which will become crucial shortly since I observed a significant amount of data in these resources across most of the samples I found.

Resources data content (Image 1)

Resources data content (Image 2)

In the malware's initial phases, it looks for the running process (which will be the previously prepared copy), where it will check the PID to access the resources. Within these resources, we see two distinct types of code: the initial part, which acts as a key, and the data chunk, which is what will be deobfuscated. To achieve this, it uses XOR + Poly, and at the end of the process, it extracts a Portable Executable.

Graph of binary load from resources

It can do this in various ways depending on the sample, but we see the same extraction of a binary from a resource as we do from obfuscated code in memory, like the example shown below.

Graph of PE extraction from memory

The result of this phase is two extracted files—one will be the injector, and the other will be the Keylogger.

Extracted Injector

Extracted Keylogger

I compared both files, and they're entirely different, in size, in structure—the only common factor is that both are .NET binaries.

Binary comparison

To highlight the difference between the injector dropped on disk (Right) and the one extracted from memory (Left), we can compare the extended content. We can observe how the memory-extracted injector includes imports

related to injection that the disk version doesn’t (such as ZwUnmapViewOfSection, VirtualAllocEx, WriteProcessMemory, etc.).

Extracted and dropped injector comparison

Extracted and dropped injector comparison

Here we can observe various functionalities while extracting the binaries, such as self-deletion. This is done to maintain evasion and avoid revealing its location, as it drops replicas of the original binary in various locations, as we saw earlier.

Self-deletion and self-copy of the original binary (Image 1)

Self-deletion and self-copy of the original binary (Image 2)

Self-deletion and self-copy of the original binary (Image 3)

Self-deletion and self-copy of the original binary (Image 4)

One of the dropped files, the smaller one, acts as the injector. When extracted from memory, it has more functionalities than the one seen on disk. This is because the injection tasks are carried out during runtime, but the written file is actually a portion of this, triggering the main binary located in the temporary path.

It checks persistence and restarts the entire process, including injection. Therefore, it’s a part of the file without revealing all of its functionalities. I’ll show you how it performs injection using Process Hollowing.

Graph of the process injection

In essence, the injector doesn’t have much more functionality. It includes a phase where it checks running processes, which is an interesting technique to detect analysis tools or to determine if the process is already running. If not, it launches the process, adds it to the registry (as seen earlier), and restarts the execution.

Process collection routine (Image 1)

Process collection routine (Image 2)

Process collection routine (Image 3)

Lastly, we only have the second extraction left to observe, which is HawkEye itself. I’ve encountered many versions of it, as the modules included will vary significantly based on what the creator configures in the builder of the Keylogger itself.

Learn to analyze cyber threats

See a detailed guide to using ANY.RUN's Interactive Sandbox for malware and phishing analysis

Read full guide

We'll talk more about this later, but you can see all the functionalities that can be added during its creation, which will impact the modules incorporated into it.

Comparison between crack and extracted keylogger features (Image 1)

Comparison between crack and extracted keylogger features (Image 2)

At this point, I conducted tests with several builders to verify this theory, as I had extracted multiple samples to the final phase, and almost none of them resembled each other too much. I tested by removing or adding options, and even with the same sample, there were significant differences, so you can imagine how different it can be if it’s not exactly the same version of the keylogger and different elements were selected during its creation.

Comparison between crack and extracted keylogger

At this stage, we just need to examine the payload's functionalities. Upon first glance, we can see strings that reveal its nature—this sample didn't expect anyone to reach this point, as it has three well-defined phases that conceal its tracks, but here we can see many indicators of what it is.

Overview of the extracted HawkEye (Image 1)

Overview of the extracted HawkEye (Image 2)

During the execution of this specific module, we can observe it invoking vbc.exe as it injects the payload into this process, using the same techniques we've previously seen.

Execution of HawkEye's final stage (Image 1)

Execution of HawkEye's final stage (Image 2)

Execution of HawkEye's final stage (Image 3)

Regarding the modules it brings, I compared three different samples, and they are quite similar in terms of what they can do. The general functionalities that typically match include:

- Keylogging (Monitoring and stealing keyboard and clipboard data)
- System information gathering (OS, HW, Network)
- Credential theft (Mail, FTP, browsers, video games, etc.)
- Wallet theft
- Screenshot capture
- Security software detection
- Analysis tools detection (Dbg, traffic, etc.)
- Persistence (usually via registry keys or Tasks)
- Information exfiltration through various methods (FTP, HTTP, SMTP, etc.)

Graph of payload module diffing

Calling HawkEye a keylogger is really an oversimplification, as it performs more functions than many stealers I've seen. Once injected into vbc.exe or other processes, it carries out various actions mentioned above.

Graph of HawkEye functionality

Outro

As we discussed earlier, different groups have used this keylogger, as well as independent criminals or even script kiddies. In my research, I found different places where this keylogger was sold—there were up to 4-5 different sites, as it changed developers and domains over time, which is quite common.

HawkEye webpage

It has also been distributed through cracks, where it was sold or offered on forums to members, avoiding the usual membership fees or markets, offering it for very low payments compared to the standard price, which as we mentioned earlier, ranged from $20 to $50.

HawkEye product sales

It's always important with these kinds of tools to locate the original software in different versions to understand how it works from both the victim's and the attacker's perspectives, so we can get a complete view of the malware

Here, we can see that the builder provides a multitude of configuration options, allowing us to choose where to send the stolen information (email, FTP, etc.), what we want to collect (browser info, FTP credentials, mail, etc.), whether to check for certain tools, establish persistence, delete data, download from a domain (this could function as a downloader for other malware), change the payload data to make it appear like legitimate software (e.g., changing the icon, description, etc.). As you can see, it's incredibly comprehensive. After compiling, we'll have our complete Keylogger, Stealer, or Downloader (call it what you will, as it does everything) ready to use.

Graph of HawkEye builder

I don’t want to repeat myself too much, but when comparing the versions we’ve seen and extracted with the ones we created ourselves, they function exactly the same—same injections, persistence, data theft (or whatever was chosen in the builder). Therefore, in telemetry, we won’t find any surprises, as you can see below.

Graph of HawkEye builded execution

After analyzing all of this, I hope you are as impressed as I am by the sheer versatility and longevity HawkEye has displayed over the decades. It's truly a tremendously powerful and easy-to-use tool that, unfortunately, we will continue to see in security incidents from actors of all types.

Finally, I would like to thank you for reading this analysis and for supporting me.

About ANY.RUN

ANY.RUN helps more than 500,000 cybersecurity professionals worldwide. Our interactive sandbox simplifies malware analysis of threats that target both Windows and Linux systems. Our threat intelligence products, TI Lookup, YARA Search and Feeds, help you find IOCs or files to learn more about the threats and respond to incidents faster.

With ANY.RUN you can:

- Detect malware in seconds
- Interact with samples in real time
- Save time and money on sandbox setup and maintenance
- Record and study all aspects of malware behavior
- Collaborate with your team
- Scale as you need

Request free trial of ANY.RUN's products →

Detection Opportunities

[TA0005][T1036] Duplication of original files in temporary paths

- (WriteFile) C:\Users\<user>\AppData\Local\Temp*.exe
- (WriteFile) C:\Users\<user>\AppData\Roaming*.exe
- (WriteFile) C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Templates*.exe
- (WriteFile) C:\Users\<user>\AppData\Local\Temp\System*.exe
- (WriteFile) C:\Users\<user>\Music*.exe

[TA0003][T1053] Scheduled Task persistence

- schtasks.exe /Create /TN "<Path>\<TaskName>" /XML "<TempPath>\<File>"

[TA0003][T1547.001] Registry Run Keys persistence

- (Registry) HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- (ValueData) <Path Used on [TA0005][T1036] Duplication of original files in temporary paths>

[TA0005][T1055.012] Process injection on vbc or itself

- From file in temporary folder > injection > vbc.exe
- From file in temporary folder > injection > Other unidentified file in same temporary path

[TA0009][T1074.001] Save stolen info on txt files

- \vbc.exe /stext "*\AppData\Local\Temp\holdermail.txt"

[TA0009][T1113] Saving screenshots of the victim's screen

- (WriteFile / Regex NameFile) screenshot\d{1}.jpeg
-

[TA0006][T1555] Queries to browser paths or third-party software to obtain user account information

- (Registry/Path query) Web Data | login data | Accounts | Profiles | \Cookies\index.dat | profiles.ini | *.oeaccount

TTPs

[TA0001][T1566.001] SpearPhishing

[TA0002][T1204] User Execution

[TA0003][T1053] Scheduled Task/Job

[TA0003][T1547.001] Registry Run Keys / Startup Folder

[TA0005][T1112] Modify Registry

[TA0005][T1564.001] Hidden Files and Directories

[TA0005][T1055] Process Injection

[TA0005][T1562] Impair Defenses

[TA0005][T1027] Obfuscated Files or Information

[TA0005][T1140] Deobfuscate/Decode Files or Information

[TA0005][T1036] Masquerading

[TA0005][T1497] Virtualization/Sandbox Evasion

[TA0006][T1552] Unsecured Credentials

[TA0006][T1555] Credentials from Password Stores

[TA0007][T1087] Account Discovery

[TA0007][T1518.001] Security Software Discovery

[TA0007][T1033] System Owner/User Discovery

[TA0007][T1012] Query Registry

[TA0007][T1016] System Network Configuration Discovery

[TA0007][T1518] Software Discovery

[TA0007][T1082] System Information Discovery

[TA0009][T1074.001] Local Data Staging

[TA0009][T1005] Data from Local System

[TA0009][T1560] Archive Collected Data

[TA0009][T1114] Email Collection

[TA0009][T1115] Clipboard Data

[TA0009][T1113] Screen Capture

[TA0011][T1105] Ingress Tool Transfer

[TA0011][T1071] Application Layer Protocol

[TA0011][T1571] Non-Standard Port

[TA0042][T1583.008] Malvertising

IOCs

60fabd1a2509b59831876d5e2aa71a6b

defc51f31f6c4fa89cc6a39a62d8a08f

dea59d578e0e64728780fb67dde7d96d

040058f70ffdee6398f7b64ae1ea46d3

e651dca5c850451cdba7f25cbb4134e7

de823ba5d67de8682e6d7b8b472dbbcb

25a2d98dfcf6a12ea6459882c56aa2e0

179b219afa2ac15b14affd399273148b

38a3cb547a0a19a61534792f572f08b0

addcd85e0126e63e46da09eb8ea97120

0a2f6501a36c1b13532139e3c1843109

addcd85e0126e63e46da09eb8ea97120

06916c9505da82f63a73768c6f336192

ab264deb2563dc4df8b281b18e0861ba

66[.]147[.]236[.]46

204[.]141[.]42[.]56

129[.]204[.]194[.]84

Aaron Jornet Sales (RexorVc0)

I am a threat researcher who spends his working time analysing TTPs and malwares of criminal groups and APTs and in my spare time, I usually focus on the same kind of stuff.

Aaron's website

I am a threat researcher who spends his working time analysing TTPs and malwares of criminal groups and APTs and in my spare time, I usually focus on the same kind of stuff.

Aaron's website

Source: https://any.run/cybersecurity-blog/hawkeye-malware-technical-analysis/