Unmasking AsyncRAT: Navigating the labyrinth of forks

By Nikola Knežević

Archived: 2026-04-05 19:11:29 UTC

AsyncRAT has cemented its place as a cornerstone of modern malware and as a pervasive threat that has evolved into a sprawling network of forks and variants. While its capabilities are not that impressive on their own, it is the open-source nature of AsyncRAT that has truly amplified its impact. This blogpost provides an overview and analysis of the most relevant forks of AsyncRAT, drawing connections between them and showing how they have evolved.

> **Key points of this blogpost:**
>
> - We provide unique insights into the landscape of AsyncRAT and its numerous variants in order to navigate the labyrinth of forks easily.
> - In the effort to map the vast hierarchy of AsyncRAT's forks, we uncover their unique interconnections and document how these variants can be distinguished.
> - We explore less common variants that feature unique plugins, ranging from a basic screamer plugin to a USB malware spreader.

Origins of AsyncRAT

You may have heard of AsyncRAT, short for asynchronous remote access trojan. This open-source RAT was released on GitHub in 2019 by a user going by the name of NYAN CAT. Developed in C#, it offers a wide range of typical RAT functionalities, including keylogging, screen capturing, credential theft, and more. Its simplicity and open-source nature has made it a popular choice among cybercriminals, leading to its widespread use in various cyberattacks.

But where does it come from? We believe that the groundwork for AsyncRAT was laid earlier by the Quasar RAT, which has been available on GitHub since 2015 and features a similar approach. Both are written in C#; however, their codebases differ fundamentally, suggesting that AsyncRAT was not just a mere fork of Quasar, but a complete rewrite. A fork, in this context, is a personal copy of someone else's repository that one can freely modify without affecting the original project. The main link that ties them together lies in the custom cryptography classes used to decrypt the malware configuration settings. Specifically, these are classes Aes256 and Sha256, which fall under the Client.Algorithm namespace for AsyncRAT and the Quasar.Common.Cryptography namespace for Quasar. Figure 1 shows identical code being used in both implementations of Aes256.

```
namespace Client.Algorithm
{
    public class Aes256
    {
        private const int KeyLength = 32;
        private const int AuthKeyLength = 64;
        private const int IvLength = 16;
        private const int HmacSha256Length = 32;
        private readonly byte[] _key;
        private readonly byte[] _authKey;

        private static readonly byte[] Salt =
        {
        0xBF, 0xEB, 0x1E, 0x56, 0xFB, 0xCD, 0x97, 0x3B, 0xB2, 0x19, 0x2, 0x24, 0x30, 0xA5, 0
        0x44, 0xD2, 0x1E, 0x62, 0xB9, 0xD4, 0xF1, 0x80, 0xE7, 0xE6, 0xC3, 0x39, 0x41
        };

        public Aes256(string masterKey)
        {
            if (string.IsNullOrEmpty(masterKey))
                throw new ArgumentException($"{nameof(masterKey)} can not be null or empty."

            using (Rfc2898DeriveBytes derive = new Rfc2898DeriveBytes(masterKey, Salt, 50000
            {
                _key = derive.GetBytes(KeyLength);
                _authKey = derive.GetBytes(AuthKeyLength);
            }
```

```
namespace Quasar.Common.Cryptography
{
    public class Aes256
    {
        private const int KeyLength = 32;
        private const int AuthKeyLength = 64;
        private const int IvLength = 16;
        private const int HmacSha256Length = 32;
        private readonly byte[] _key;
        private readonly byte[] _authKey;

        private static readonly byte[] Salt =
        {
            0xBF, 0xEB, 0x1E, 0x56, 0xFB, 0xCD, 0x97, 0x3B, 0xB2, 0x19, 0x2, 0x24, 0x30, 0xA5, 0x
            0x44, 0xD2, 0x1E, 0x62, 0xB9, 0xD4, 0xF1, 0x80, 0xE7, 0xE6, 0xC3, 0x39, 0x41
        };

        public Aes256(string masterKey)
        {
            if (string.IsNullOrEmpty(masterKey))
                throw new ArgumentException($"{nameof(masterKey)} can not be null or empty.");

            using (Rfc2898DeriveBytes derive = new Rfc2898DeriveBytes(masterKey, Salt, 50000))
            {
                _key = derive.GetBytes(KeyLength);
                _authKey = derive.GetBytes(AuthKeyLength);
            }
```

Figure 1. Comparison of cryptography classes between AsyncRAT (left) and Quasar (right)

The same code is mostly copied and pasted, including the same salt value and decryption settings. This class, together with Sha256, leads us to believe that AsyncRAT was to some degree influenced by the Quasar RAT.

Apart from these similarities, AsyncRAT introduced significant improvements, particularly in its modular architecture and enhanced stealth features, which make it more adaptable and harder to detect in modern threat environments. Its plugin-based architecture and ease of modification have sparked the proliferation of many forks, pushing the boundaries even further.

Fork labyrinth

Ever since it was released to the public, AsyncRAT has spawned a multitude of new forks that have built upon its foundation. Some of these new versions have expanded on the original framework, incorporating additional features and enhancements, while others are essentially the same version in different clothes.

Fork hierarchy

Figure 2 illustrates how some of the more prevalent AsyncRAT forks have evolved from one another over time.

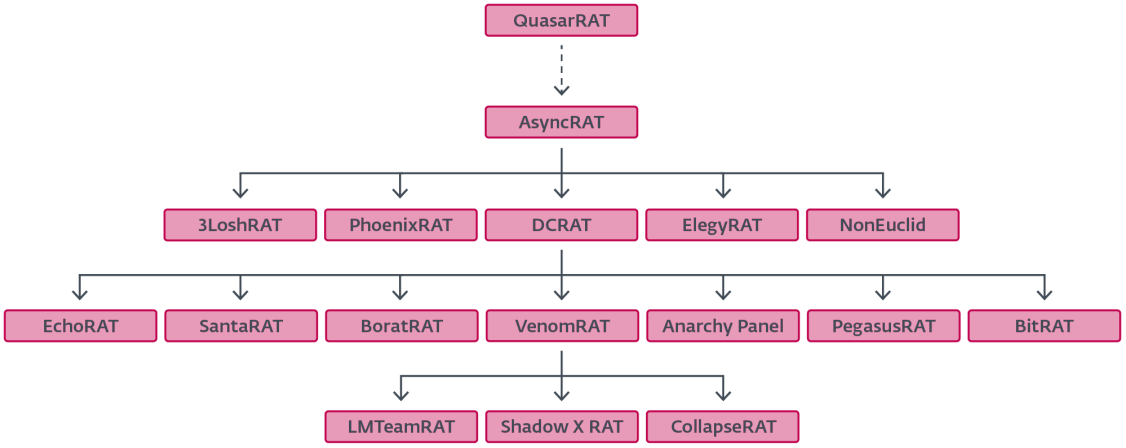

Figure 2. A small subset of forks highlighting their derivative relationships

In the middle of the tree are DcRat and VenomRAT. Our analysis has shown that they are the most widely deployed variants, together accounting for a significant number of campaigns. Other lesser-known forks occupy

smaller but nonetheless notable portions of the pie. Figure 3 depicts the distribution of the most prevalent forks according to our telemetry.

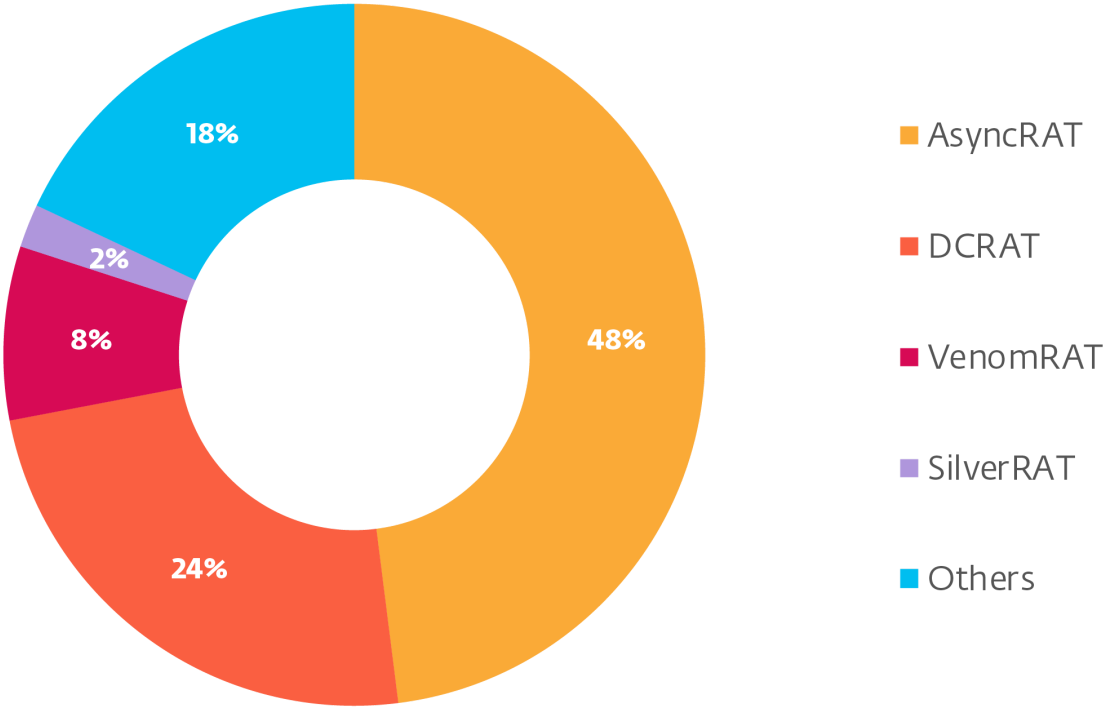

Figure 3. Q2 2024 distribution of the most common forks, as measured by the number of unique samples

DcRat offers a notable improvement over AsyncRAT in terms of features and capabilities. One of the more obvious changes is the data structure used for transferring data back and forth. It utilizes MessagePack, a well-known open-source library for more efficient binary data serialization. DcRat also implements evasion techniques like AMSI and ETW patching, which work by disabling security features that detect and log malicious behavior – AMSI patching prevents script scanning, while ETW patching blocks event tracing. Additionally, it features an antiprocess system whereby processes whose names match those in a denylist are terminated. Blocklisted programs include Taskmgr.exe, ProcessHacker.exe, MsMpEng.exe, Taskkill.exe, etc.

It's also worth noting that DcRat's plugin base builds upon AsyncRAT and further extends its functionality. Among the added plugins are capabilities such as webcam access, microphone recording, Discord token theft, and "fun stuff", a collection of plugins used for joke purposes like opening and closing the CD tray, blocking keyboard and mouse input, moving the mouse, turning off the monitor, etc. Notably, DcRat also introduces a simple ransomware plugin that uses the AES-256 cipher to encrypt files, with the decryption key distributed only once the plugin has been requested. Apart from that, there appear to be many small changes like a different choice of salt (a string instead of a binary value), deliberately changed variable names to further evade detection, dynamic API resolution, and many more.

VenomRAT, on the other hand, was likely inspired by DcRat, as evidenced in the *Identifying versions* section. The malware is packed with so many features that it could be considered a separate threat on its own. We have chosen to group it under AsyncRAT as their client parts are very similar to each other. VenomRAT's features and plugins have been documented in more detail by other vendors, so we won't dive deep into them in this blogpost.

Not all RATs are serious in nature though, and this applies equally to AsyncRAT forks. Clones like SantaRAT or BoratRAT (see Figure 4) are meant to be jokes. In the case of the former, its authors have themselves acknowledged that the project was basically "shamelessly ripped off of DcRat". Yet, despite this, we have found instances of real-world usage of them in the wild.

Figure 4. Official BoratRAT promotional logo

Identifying versions

While doing the analysis, we used various methods to identify and categorize each sample. It should be noted that the research was primarily on the client part of the malware, as this binary is what ends up on victims' machines. It contains useful information such as malware configuration and where information about the C&C can be found.

The quickest and most straightforward way to identify a fork is to peek directly into the malware's configuration, which can usually be found in the InitializeSettings function. The configuration values are encrypted with AES-256 and stored as base64 strings in the Settings class. In most cases, the correct fork name is readily available and conveniently labeled as Version. In about 90% of our analyzed samples, the Version field contains some meaningful description of either the fork's name or the malware author's pseudonym. The remaining samples had this field intentionally left blank. Figure 5 illustrates the typical configuration initialization procedure found in DcRat and its derivatives (VenomRAT in this case).

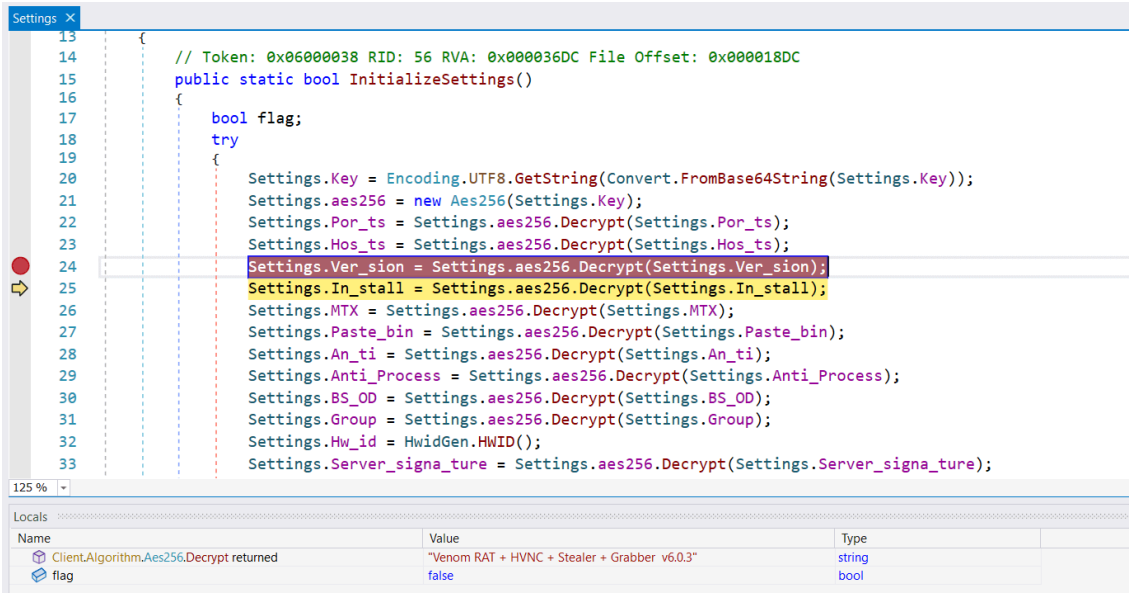

Figure 5. Initialization of VenomRAT configuration values

If the Version field is empty, sometimes it's possible to get another clue by looking at the Salt value used for encrypting the configuration. Attackers often neglect this parameter when copy-pasting their own fork. The Salt value can be found in the Client.Algorithm.Aes256 class, as seen in Figure 6.

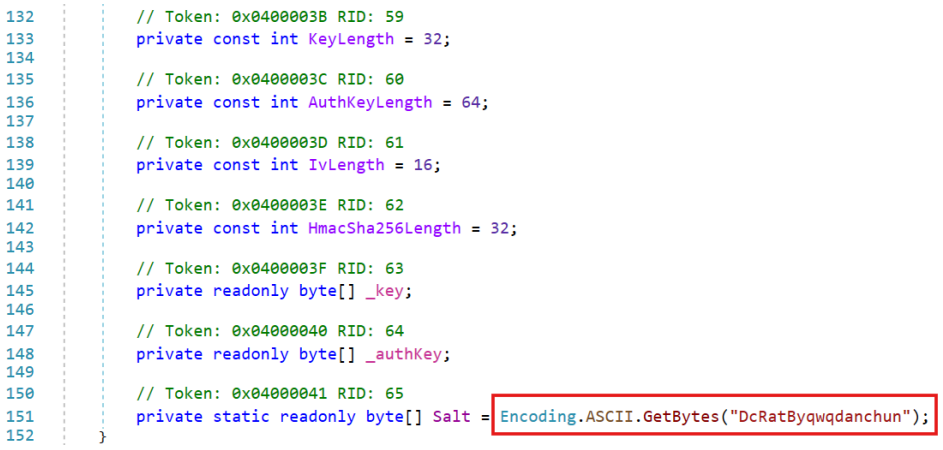

Figure 6. Extraction of the Salt *value in the constructor of VenomRAT's cryptography class*

Yet another way to get more insight is to look for the embedded certificate used to authenticate the C&C server. It's also located in the configuration as a base64-encoded value. Unpacking this value often reveals further information about the server, such as common name, organization, and organizational unit. If a particular fork has its own name in the Version field, it is often possible to trace back the previous fork upon which it was likely based by looking at the CN field. Figure 7 shows a DER-encoded certificate that reveals the BoratRAT fork, after extraction and decoding.

```
Version:          3 (0x02)
Serial number:    122387568940541051439889742627299520269178564770 7 (0x00d6607be5831138a000ff519932eafe33ece5665b)
Algorithm ID:     SHA512withRSA
Validity
  Not Before:     23/09/2023 17:38:21 (dd-mm-yyyy hh:mm:ss) (230923173821Z)
  Not After:      02/07/2034 17:38:21 (dd-mm-yyyy hh:mm:ss) (340702173821Z)
Issuer
  CN = BoratRat Server
  OU = Sa8XOfH1BudX
  O  = BoratRat By Sa8XOfH1BudX
  L  = SH
  C  = CN
Subject
  CN = BoratRat
```

Figure 7. Client certificate after extraction

The techniques mentioned above primarily apply to trivial cases where malware authors either did not bother to remove traces or used a default certificate. A more sophisticated method for identifying AsyncRAT servers exists, which involves sending a specially crafted packet to the C&C server. This approach is explained in detail in this Axel Mahr blogpost.

Should everything else fail, determining the sample origin can ultimately be done the old-fashioned way, by manually inspecting the code. This involves a detailed analysis of the code's structure, syntax, and functionality, comparing them against the patterns of previously categorized samples.

Extensive fork list

We have highlighted here some of the more prominent AsyncRAT forks. Due to the sheer number of available forks, it is not feasible to cover every single one. For completeness, Figure 8 provides an extended list of AsyncRAT forks known to be used for malicious purposes, as seen in ESET telemetry to date.

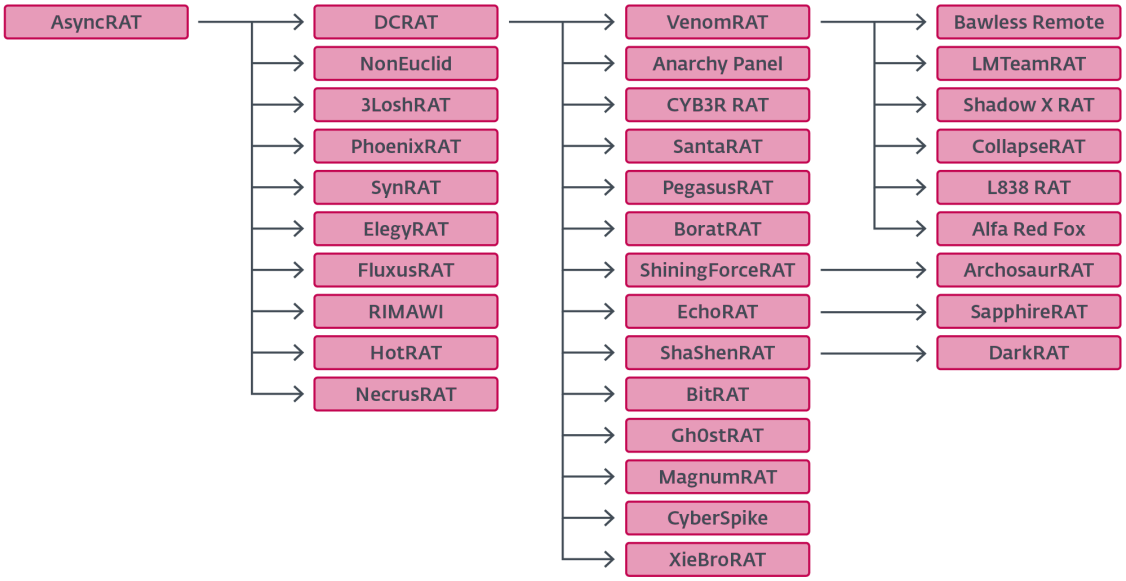

Figure 8. Extended fork hierarchy list

Exploring lesser-known variants

So far, we've mentioned some of the major forks that dominate the landscape. In this section, we have cherry-picked some lesser-known forks that enhance AsyncRAT's functionality beyond the features included in the

default versions. These exotic forks are often the work of one person or group and they make up less than 1% of the volume of AsyncRAT samples.

NonEuclid RAT

This fork stands out primarily for its inclusion of new plugins, on top of the default ones. While some plugins might seem trivial or geared towards “fun stuff”, others, like WormUsb.dll, have distinctly malicious purposes. Table 1 lists a selection of NonEuclid RAT plugins that deviate from the standard plugin base seen in regular forks.

Table 1. Selection of NonEuclid RAT plugins we deemed interesting

Plugin name	Description
Screamer.dll	Jump scare plugin.
Piano.dll	Generic audio player.
Service.dll	Windows services management.
Maps.dll	Collects geolocation info from the user.
WormUsb.dll	Malware spreader plugin.
Brute.dll	SSH and FTP brute forcer.
Signature Antivirus.dll	Simple signature-based file matcher.
cliper.dll	Replaces clipboard data with attacker’s cryptocurrency wallet addresses.

Screamer.dll

There are five jump scare images built into the plugin. An attacker sends a command that indicates which image they want to use, along with the WAV file to be played, and the delay after which the jump scare is triggered. Figure 9 shows the first three prebundled images an attacker may choose from.

Piano.dll

This plugin plays arbitrary WAV files. All sound files are stored in %appdata%\Piano. piano.dll supports three commands:

- SetSound – adds a new sound file to %appdata%\Piano,
- PlayMisc – plays a requested sound file from %appdata%\Piano, and
- ClientAdd – retrieves multiple sound files from the C&C server.

Service.dll

This plugin facilitates managing Windows services, such as starting, stopping, and pausing services.

Maps.dll

This is a simple plugin to collect geolocation information from the victim. It uses the .NET GeoCoordinateWatcher class to register a callback function to collect data each time the location is available. Among the collected information are latitude, longitude, username, and computer name.

WormUsb.dll

This plugin compromises PE files with an arbitrary payload specified by the attacker.

Despite the term Usb in WormUsb.dll, this plugin targets several locations based on the command provided:

- InfectExe – compromises an individual PE file,
- InfectExeInWindows – targets PE files in personal folders (Desktop, Documents, Downloads, My Music), and
- InfectUsbExe – targets PE files in all drives excluding the C drive.

Under the hood, it works by moving the original file to a temporary location. Then it drops a small stub in place of the original file. This stub's resource section is then populated to contain both the original file and the specified payload, both of which are compressed and encrypted with a per-file key, generated at the time of construction. Following this, the malware then obfuscates the stub by introducing proxy methods, adding custom control flow obfuscation, and variable renaming. As a final touch, it embeds the original icon and metadata in the modified stub. Figure 10 shows the function, with the original method names, responsible for compromising a single file.

```
public static void Infect(string FileInfect)
{
    try
    {
        ModuleDefMD moduleDefMD2;
        ModuleDefMD moduleDefMD = (moduleDefMD2 = ModuleDefMD.Load(Resource1.Stub, null));
        try
        {
            string text = Path.GetTempFileName() + ".exe";
            File.WriteAllBytes(text, File.ReadAllBytes(FileInfect));
            File.Delete(FileInfect);
            string icon = InfectingFile.GetIcon(text);
            try
            {
                InfectingFile.WriteSettings(moduleDefMD, FileInfect, text);
                ControlFlowObfuscation.Execute(moduleDefMD);
                RenamerPhase.Execute(moduleDefMD);
                ProxyString.Execute(moduleDefMD);
                moduleDefMD.Write(FileInfect);
                moduleDefMD.Dispose();
                InfectingFile.WriteAssembly(FileInfect, text);
                IconInjector.InjectIcon(FileInfect, icon);
            }
            catch
            {
                InfectingFile.WriteAssembly(FileInfect, text);
                IconInjector.InjectIcon(FileInfect, icon);
            }
        }
```

Figure 10. Compromise function of a WormUsb.dll *plugin*

When such a compromised file is executed, it first decrypts, unpacks and runs the payload program, then proceeds to do the same with the original program.

Brute.dll

This plugin supports brute forcing of both SSH and FTP protocols from the client side. The attacker feeds it three parameters: host, login, and password, and the plugin will try to connect using those credentials. If the connection succeeds, the credentials are sent back to the attacker with a flag indicating success. It's not difficult to imagine a scenario whereby an attacker might use this sort of attack to distribute brute forcing across a large pool of compromised machines, thus circumventing restrictions based solely on the IP address.

Signature Antivirus.dll

The name of the plugin implies it might have something to do with antivirus functionality. While this is technically true, it is also a case of the most primitive, manual antivirus solution ever created. The plugin receives a list of MD5 hashes from the attacker and compares them to the hashes of all EXE files it finds on every disk. If a matching file is found, it triggers the oddly named DetectVirus function, which merely deletes the file without any further analysis. This makes the name of the plugin very dubious at best. In the hands of the malware author, it may have been used to delete competitor malware, or really just any arbitrary file.

cliper.dll

This is a standalone clipper that continuously monitors the victim's clipboard, and if a cryptocurrency wallet address is detected, it is replaced with one provided by the attacker. Attacker-provided wallets are only sent when the plugin is first requested; they are not hardcoded in the plugin. Additionally, in Figure 11, we can also see some credit card entries. This plugin contains an extensive list of regexes that can detect both cryptocurrency wallets and credit cards, and in the case of the latter they just get sent back to the attacker.

```
RegexPatterns.MyWallet["Amex Card"] = msgPack.ForcePathObject("Amex Card").GetAsString();
RegexPatterns.MyWallet["BCGlobal"] = msgPack.ForcePathObject("BCGlobal").GetAsString();
RegexPatterns.MyWallet["bch"] = msgPack.ForcePathObject("bch").GetAsString();
RegexPatterns.MyWallet["btc"] = msgPack.ForcePathObject("btc").GetAsString();
RegexPatterns.MyWallet["Carte Blanche Card"] = msgPack.ForcePathObject("Carte Blanche Card").GetAsString();
RegexPatterns.MyWallet["Diners Club Card"] = msgPack.ForcePathObject("Diners Club Card").GetAsString();
RegexPatterns.MyWallet["eth"] = msgPack.ForcePathObject("eth").GetAsString();
RegexPatterns.MyWallet["Express Card"] = msgPack.ForcePathObject("Express Card").GetAsString();
RegexPatterns.MyWallet["Insta Payment Card"] = msgPack.ForcePathObject("Insta Payment Card").GetAsString();
RegexPatterns.MyWallet["JCB Card"] = msgPack.ForcePathObject("JCB Card").GetAsString();
RegexPatterns.MyWallet["KoreanLocalCard"] = msgPack.ForcePathObject("KoreanLocalCard").GetAsString();
RegexPatterns.MyWallet["Laser Card"] = msgPack.ForcePathObject("Laser Card").GetAsString();
RegexPatterns.MyWallet["ltc"] = msgPack.ForcePathObject("ltc").GetAsString();
RegexPatterns.MyWallet["Maestro Card"] = msgPack.ForcePathObject("Maestro Card").GetAsString();
RegexPatterns.MyWallet["Mastercard"] = msgPack.ForcePathObject("Mastercard").GetAsString();
RegexPatterns.MyWallet["Solo Card"] = msgPack.ForcePathObject("Solo Card").GetAsString();
RegexPatterns.MyWallet["Switch Card"] = msgPack.ForcePathObject("Switch Card").GetAsString();
RegexPatterns.MyWallet["Union Pay Card"] = msgPack.ForcePathObject("Union Pay Card").GetAsString();
RegexPatterns.MyWallet["Visa Card"] = msgPack.ForcePathObject("Visa Card").GetAsString();
RegexPatterns.MyWallet["Visa Master Card"] = msgPack.ForcePathObject("Visa Master Card").GetAsString();
RegexPatterns.MyWallet["xlm"] = msgPack.ForcePathObject("xlm").GetAsString();
```

Figure 11. Wallets and cards monitored by cliper.dll

JasonRAT

Identified in 2024, this variant shows continued signs of activity. It is interesting in that it employs obscure variable-naming conventions reminiscent of "satanic" terms from what the malware author refers to as the Book of Jason. In Figure 12, you can see typical AsyncRAT configuration values (in base64), but with renamed variables, while Figure 13 shows the logic of the main entry point of the malware. Besides the usual configuration values, this variant further extends the client by introducing country targeting.

```
static JasonIntructions()
{
    JasonIntructions.GateOfDarkness = "Jwjx54u8jdgTjMYEoSTCrrJJ/2nRFFTTVBbpU46GbyTvM3zBcr6TenlXQLb82WDYBDTw8lv7Ik2EF3mkiIoOVA==";
    JasonIntructions.DarkHost = "cxdnBEG7Or9CIbZvkSBtt7Om4Wb2ZctjxNrh77akKjGv9VH39q7ALJaTOFWerTDNpqV7XR0Dyc5/Iq0Whg+V9w==";
    JasonIntructions.JasonAge = "UhNGwVBc3eUvpeP8m9ZmilxX7Dmql3PfNzElAWXF/pRT2jGiDjduN3RewMUtta1l+8twZ6vxlhZVeqRjxL7Bgg==";
    JasonIntructions.TakeOverHisBody = "zrkKEs4hC6A6TcrvSnTNXAAgsZcoa5VmBDxH1Ci2ZXiahdjBYS6/7HEMUMZi3d62AREW6lkxaQX7dFmSg0SU8A==";
    JasonIntructions.HoleOfDarkness = "AppData";
    JasonIntructions.ElementOfPath = "Knmshnjsloj";
    JasonIntructions.KeysOfTheUnderworld = "SmFzb27imKBMZXRUaGVyZUJlQ2FybmFnZQ==";
    JasonIntructions.ElementOfRecognition = "pKHgi+WOXQ/Mzq96k9eNzDgo3c9xSU1nRgUCHM1M8w6r3bMLkBSRtGXKj4q2lkq6kzp0smSIhFa0eUCHL9KaQNb2VV2HQ2XBT4uCeylLWK8=";
    JasonIntructions.BookOfJason = "HuLFsd5rh4QxQrpE0glTphVpISzgBSo5LFgHiyKh8auNDDV/xlHZvO2yRFr7P1Pyfzp1jG+BkLxwi8OZZBdq/f2+0HrABSxlysmPadjJcToCD/
      Rw6xthYZOT40N212PbPT8GGqinZ1Bc9oW7CWptrICaMbPP
      +VFf1tsdI4TG2rJmLStnO6DZMDLhr1ygS8oV88UayO04iHSywVTkrIOzlsQAq1wCxx8hPBniWKdVNohSpPrUVHl4Dye0bgY26MJ1hibIRGkGvG0xELe
      +Juv2a5J6U2lKhbw4rGKry9GlaQ4DvIn3Y43bwUYOg7g4AdJiWU+0MgbIf9jiLVIejQxoC2XUDGLzbL+PtnFxL5ewvesBUl7ZMfz43bOCIOKtyadkW
      +xI12iaGaSrwUfJjJgsJqCYzXvUaMrIZxtNSSVybk0HLAGzDHH1ZBPiBacvsYEjEbvq/GwOfH8T4ehtPJyGwQtGkZHe3KWVrin5dtJjMbuI1jJmAoipsLS
      +krmMemM9mHaB3rWryuuJVMcmGdZd2cbHenBURa2vyO7ruhSTX9yCSuUdXOFu1GzbvklD8Ldv9m42JSNSId8tZf2CTaHCjjqH6yEPPaiLo4/B/f1iVl/CGKEVjM/6pQY8f6cQ3ke2DVQ7nNhy3Ty/fJZQ
      +HJ8Q/FV/JuqnCTcM3ktgbet+Fp5Egqdn1lwvb7x/5zsHDVS/uOsxQifYpCjkAfbB7Nig9+qKUW7vuF0jn6y3n9RPiQn4WIdGfi6m0TO9hElOYYjmhhDh+QAfFINyg3Ue3djTaoEhCOIGkT5pPC3KqP/
      FlkpVDD+U4Ge8cFzFepBi8/xXXIqVzwY+JeW7J3LHYxWuackUGskL47GbYNFMhEUWHOBUB2GXS8KKp7EcTt5zflKAz6uOyTQT25xpgL3xuiqSa6exAQ2XpMNIPP+ulVNwxrjgbc2rSSzoIk1jmHVZ/
      bHQLNh1PNkHrOsjhL0qQfTqWaqOG+piRWrJnVRnrJBXxpcsWUMWhEzr0Eitu41cbfPh/hwpDeEgjwBwT33rDU/wOOEsW86xk7H6Cp5f/
      n93YUnPXt578uSdzd2s/3FmD7GjZCRHmsymxgTTdO0deIcVG6p1Tqau2hsJmytjwMRVZ8=";
    JasonIntructions.TortureGoodSouls = "I4i9tyh6ZOSUmBPSfuh3qImMKbop49xTsDVFjPHaeazufBMzWQinVezU9m4bNf501oz/n7SkCOxi2ezQmLEbi+5xjdkl9wxYqIDnPuIc3Leb4txxoS
      +T3yDWGZxFezjeP3ek1e24WOHW+qS5iKvcCu7YBYRS21icEa+R+c8b5nIXVc5aJwEK4sili/EfvfrbohZNHZCsJvRN5zEknHjXwgsasLQD8jv4DAx7Fd8ZbhqVSei23B++HlRVUw1G6YJu8AkUuNu+
      +QicMXZqbXXhmKVyuBnXWAcUNB0fJq8DQlo=";
    JasonIntructions.IfGateNotExistCreateIt = "YDu53xcQC+UGjuFkoA3nKOEHFOUw4ykc4r3qiBuFb7HN7VXsgmpvD22Ch+LURGxNNXFcyoUxJL17vqJEwiFieQ==";
    JasonIntructions.NoUnderstanding = "uk7SNkhouHuu9ZNYzHJxSyjgye/9LsDsqNCI596cG/BAy3Np9BTBSwMY/n5P3aJTBs7/yTJ3fk3L5B1GCpruZQ==";
    JasonIntructions.HeardOfSin = null;
    JasonIntructions.KarpTorture = "1";
    JasonIntructions.PosessionLevel = "yJKLs6bHwpeQBqtKk55jiLNmUdgSaFzVSqdLtzqxY3ybNn6bjW7qzeLf/RdfQ5uP/m4NDuNsmf97vDTFBJ+gbA==";
    JasonIntructions.AwakeTheQuardian = "PlrXNzMwUMy01Dp2t7HUL2mB5b6nxF8P1SL5FUr4DHuml X9VFq2WWKE6jQCBV58/HeFNpBXencbgdO9sT0KHFQ==";
    JasonIntructions.NoEscape = "Y5pw7FoHIcpbYBfvl1ggDk2U/mrilalBe3aXL3Nic0H9FRziUwVJW+fWJYggTkW7Frh0EayKdr71mHatf9b8QQ==";
    JasonIntructions.Exclude = "false";
    JasonIntructions.OffLog = "true";
    JasonIntructions.Corona = "false";
    JasonIntructions.TargetstringValue = "false";
    JasonIntructions.boolValue7 = bool.Parse(JasonIntructions.TargetstringValue);
    JasonIntructions.CountryLock = JasonIntructions.boolValue7;
    JasonIntructions.TargetCountry = "%TARGET%";
    JasonIntructions.Amitgail = "false";
}
```

Figure 12. Partially obfuscated JasonRAT configuration values

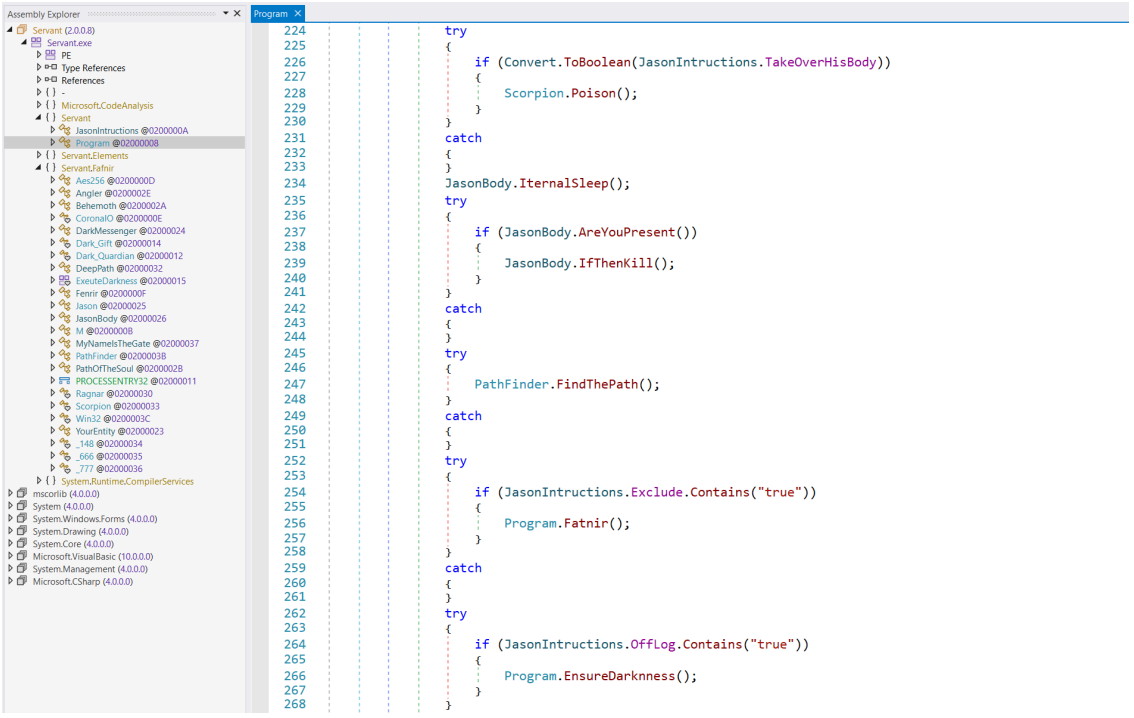

Figure 13. Main JasonRAT entry point showing renamed function names

Another strange feature is the choice of string obfuscation. A subset of the strings employs an extra layer of obfuscation by utilizing an extended variant of Morse code. Both uppercase and lowercase letters are included, as well as some special characters. Figure 14 shows the encoded registry key string using an extended mapping.

```
if (JasonIntructions.Amitgail.Contains("true"))
{
    string location = Assembly.GetExecutingAssembly().Location;
    if (!new WindowsPrincipal(WindowsIdentity.GetCurrent()).IsInRole(WindowsBuiltInRole.Administrator))
    {
        Jason.IronKnife();
    }
    else
    {
        Jason.OpenRegistryKey(M.D("-.-. *^** *^ *** *** * *** *_+^^# *_+^^# ^^ *** @~_* *** * ^ ^ ** ^* ^^* *** *_+^^# *_+^^# ***
          **** * *^** *^** *_+^^# *_+^^# ^^^ *^^* * ^* *_+^^# *_+^^# ^*^* ^^^ ^^ ^^ *^ ^* ^**"), true).SetValue("", "",
          RegistryValueKind.String);
    }
    try
    {
        Program.XorAE(M.D("-.-. *^** *^ *** *** * *** *_+^^# *_+^^# ^^ *** @~_* *** * ^ ^ ** ^* ^^* *** *_+^^# *_+^^# *** **** * *^**
          *^** *_+^^# *_+^^# ^^^ *^^* * ^* *_+^^# *_+^^# ^*^* ^^^ ^^ ^^ *^ ^* ^**")).DeleteSubKeyTree(location);
    }
    catch
    {
    }
}
```

Figure 14. Extended Morse code used as string obfuscation in JasonRat

XieBroRAT

This is a RAT with Chinese localization. It introduces a new plugin, BrowserGhost.dll, which is a browser-credential stealer. Another plugin, Abstain.dll, provides interaction with Cobalt Strike servers by making a reverse connection.

To increase the coverage, the malware provides the delivery chain in several different languages. The standard .NET client binary can be wrapped and distributed via shellcode, VBS, or JavaScript.

Finally, the author further extended the malware by borrowing heavily from open-source projects, integrating tools like mimikatz, SharpWifiGrabber, SharpUnhooker, etc.

Conclusion

AsyncRAT's rise and its subsequent forks highlight the inherent risks of open-source malware frameworks. Our analysis revealed a diverse and evolving ecosystem of derivatives, ranging from persistent threats like DcRat and VenomRAT to lesser-known novelty forks like JasonRAT and BoratRAT, which seem to serve more as curiosities than credible threats. All of these forks not only extend AsyncRAT's technical capabilities but also demonstrate how quickly and creatively threat actors can adapt and repurpose open-source code.

The widespread availability of such frameworks significantly lowers the barrier to entry for aspiring cybercriminals, enabling even novices to deploy sophisticated malware with minimal effort. This democratization of malware development – especially considering the rising popularity of LLMs and potential to misuse their capabilities – further accelerates the creation and customization of malicious tools, contributing to a rapidly expanding and increasingly complex threat landscape.

In light of these trends, it is reasonable to anticipate that future forks may incorporate more advanced obfuscation, modularity, and evasion capabilities. This potential evolution underscores the importance of proactive detection strategies and deeper behavioral analysis to effectively address emerging threats.

> *For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.*

> *ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.*

IoCs

A comprehensive list of indicators of compromise (IoCs) can be found in our GitHub repository.

Files

SHA-1	Filename	Detection	Description
F8E31B338123E38757F8 B7099797119A038A3538	Screamer.dll	MSIL/AsyncRAT.C	NonEuclid jump scare plugin.
98223D2F8DF2F9E832AE 081CD6E072A440C9A3CD	Piano.dll	MSIL/AsyncRAT.C	NonEuclid audio player plugin.
CDEC9A1C73E3E21B1D70 DDAA6BF139D8D2A197A5	Maps.dll	MSIL/AsyncRAT.C	NonEuclid geolocation plugin.
932C49EEE087D432D0DA 10CC0640B11FD2C91203	Service.dll	MSIL/AsyncRAT.C	NonEuclid Windows service management plugin.
2FA98D088486BAC57FF6 0E072E28FEE5830E7B28	WormUsb.dll	MSIL/AsyncRAT.C	NonEuclid malware spreader plugin.
62C9FEFA84067F695032 A6939F07C3799AAD80A3	Brute.dll	MSIL/AsyncRAT.C	NonEuclid SSH and FTP brute forcer plugin.
FAD946F7ACF017F0C50C 81BF379AABA3528AFBB3	Signature Antivirus.dll	MSIL/AsyncRAT.C	NonEuclid signature-based file matcher plugin.
51B8A5818B7031EDB59A 2B2ECF160A78505880BA	cliper.dll	MSIL/AsyncRAT.C	NonEuclid clipboard hijacker plugin.
4FB0CAAD6E345947EE2D 30E795B711F91C6A4819	Stub.exe	MSIL/AsyncRAT.A	AsyncRAT client.
FD9CF01CEA7DE8631C34 B988A7AAD55587A162FA	Stub.exe	MSIL/AsyncRAT.A	3LoshRAT client.
B8AB93E958E0DE4BE276 6B2537832EDB37030429	Client.exe	MSIL/AsyncRAT.A	DcRat client.

SHA-1	Filename	Detection	Description
68B58483D0E4E7CC2478 D6B4FC00064ADE3D7DB3	Microsoft_Edge _Driver.exe	MSIL/AsyncRAT.A	VenomRAT client.
4F69E0CE283D273B724C E107DF89F11C556A7A4E	Client.exe	MSIL/AsyncRAT.C	BoratRAT client.
E4F87568473536E35006 D1BD4D4C26A8809F3F91	Client.exe	MSIL/AsyncRAT.A	Anarchy Panel client.
D10B8197732437E9BF84 0FEA46A30EFF62892A4E	Client.exe	MSIL/AsyncRAT.A	CollapseRAT client.
0DC28EA51F0D96E0D1BC 78DF829C81A84332C5F1	dwm.exe	MSIL/AsyncRAT.A	Shadow X RAT client.
E5B511E7550CBADE74E7 5EADE8F413A89D963FE5	ClientAny.exe	MSIL/AsyncRAT.A	LMTeamRAT client.
3124F58428184FDF75E2 1B1E5A58CADF9DD2BA03	Stub.exe	MSIL/AsyncRAT.A	PhoenixRAT client.
8402AA507CF5B1BBFAB5 3E3BF7A7D4500796A978	Client.exe	MSIL/AsyncRAT.A	EchoRAT client.
AB2C6F9695346FAA9495 B4AB837085C1524FFDDF	Client.exe	MSIL/AsyncRAT.A	XieBroRAT client.
3E6CD9D07B8ECE706697 F332AC9F32DE5ECAF086	tempClient.exe	MSIL/AsyncRAT.C	NonEuclid RAT client.
FF4592A8BCB58F5CF6BD 70B882E886EC6906EECD	Servant.exe	MSIL/AsyncRAT.A	JasonRAT client.

MITRE ATT&CK techniques

This table was built using version 17 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Defense Evasion	T1562.001	Impair Defenses: Disable or Modify Tools	DcRat terminates security tools such as Taskmgr.exe and MsMpEng.exe.
	T1562.004	Impair Defenses: Disable or Modify System Firewall	DcRat leverages AMSI and ETW bypass techniques to evade detection.

Tactic	ID	Name	Description
	T1027.013	Obfuscated Files or Information: Encrypted/Encoded File	JasonRAT employs modified Morse code and obscure variable names to hinder analysis.
Credential Access	T1539	Steal Web Session Cookie	DcRat leverages a plugin to steal Discord tokens from compromised machines.
	T1555.003	Credentials from Password Stores: Credentials from Web Browsers	XieBroRAT uses a plugin to collect browser credentials.
	T1110.003	Brute Force: Password Spraying	NonEuclid uses a plugin to brute force SSH and FTP credentials.
Discovery	T1614.001	System Location Discovery: System Language Discovery	NonEuclid uses a plugin that collects geolocation data from compromised systems.
Collection	T1123	Audio Capture	DcRat has a microphone plugin that enables audio capture from the victim's device.
	T1125	Video Capture	DcRat includes a webcam plugin that allows remote access to the victim's camera.
	T1115	Clipboard Data	NonEuclid uses a plugin that monitors the clipboard to intercept and replace cryptocurrency wallet addresses.
Impact	T1486	Data Encrypted for Impact	DcRat features a ransomware plugin capable of encrypting files on the victim's system.
