Life on a crooked RedLine: Analyzing the infamous infostealer's backend

By Alexandre Côté Cyr

Archived: 2026-04-05 14:37:32 UTC

> UPDATE (November 12th, 2024): We clarified the information in the fourth paragraph to better reflect RedLine's functionality before versus after the takedown.

On October 28th, 2024, the Dutch National police, alongside the FBI, Eurojust, and several other law enforcement organizations, performed a takedown of the infamous RedLine Stealer malware-as-a-service (MaaS) operation, and its clone called META Stealer. This global effort, named Operation Magnus, resulted in the takedown of three servers in the Netherlands, the seizure of two domains, two people being taken into custody in Belgium, and the unsealing of charges against one of the alleged perpetrators in the United States.

Back in April 2023, ESET participated in a partial disruption operation of the RedLine malware, which consisted of the removal of several GitHub repositories used as dead-drop resolvers for the malware's control panel. Around that time, we investigated previously undocumented backend modules of this malware family in collaboration with fellow researchers at Flare. These modules don't interact directly with the malware, but rather handle authentication and provide functionality for the control panel.

Since RedLine has now been taken down, we are revealing our findings from 2023 publicly, along with some more recent discoveries that were made based on source code and samples shared with us by the Dutch National Police.

Note that most of this analysis was performed before the takedown. Additionally, there are old, cracked copies of the malware that might still work. This is why we are describing the RedLine operation as if it is an ongoing activity.

> **Key points of the blogpost:**
>
> - In 2023, ESET researchers, in collaboration with law enforcement, collected multiple modules used to run the infrastructure behind RedLine Stealer.
> - We analyzed these previously undocumented modules to provide insight into the internal workings of this malware-as-a-service empire.
> - We were able to identify over 1,000 unique IP addresses used to host RedLine control panels.
> - The 2023 versions of RedLine Stealer we investigated in detail used the Windows Communication Framework for communication between the components, while the latest version from 2024 uses a REST API.
> - Based on our analysis of the source code and backend samples, we have determined that Redline Stealer and META Stealer share the same creator.

RedLine Stealer is information stealing malware first discovered in 2020 by Proofpoint. Rather than being centrally operated, RedLine operates on a MaaS model in which anyone can buy a turnkey infostealer solution from various online forums and Telegram channels. Clients, called affiliates, can opt for a monthly subscription, or a lifetime license; in exchange for their money, they get a control panel that generates malware samples and acts as a C&C server for them. The generated samples can collect a large variety of information, including local cryptocurrency wallets; cookies, saved credentials, and saved credit card details from browsers; and saved data from Steam, Discord, Telegram, and various desktop VPN applications.

Using a ready-made solution makes it easier for the affiliates to integrate RedLine Stealer into larger campaigns. Some notable examples include posing as free downloads of ChatGPT in 2023, and masquerading as video game cheats in the first half of 2024.

> **Note on terminology used**
>
> Because of its MaaS model, any comprehensive discussion of RedLine will involve multiple different components and layers of network infrastructure. To limit any possible confusion, we will use the following terms consistently throughout the text:
>
> - **RedLine malware**: The RedLine Stealer malware or a sample thereof.
> - **RedLine panel**: GUI control panel used to manage infostealing campaigns.
> - **RedLine backend**: Collection of modules that provide authentication and functionality for the RedLine panel.
> - **RedLine**: The whole malware operation. This includes the RedLine malware, the RedLine panel, and the RedLine backend modules.
> - **Backend server**: A server on which the RedLine backend runs.
> - **Victim**: Entity targeted with the RedLine malware.
> - **Operator**: The individual or team that develops RedLine, sells licenses, and operates the licensing and associated backend infrastructure.
> - **Affiliate**: Entity that operates infostealing campaigns via an instance of the RedLine panel. They usually have a license bought from the operator, but may also use a cracked version of the panel.

Overview

In this blogpost we document modules running on RedLine's backend servers to provide a greater understanding of the inner workings of this MaaS empire. We also provide some information on the RedLine panel. Figure 1 contains a simplified overview of the whole RedLine operation.

Ordinarily, known samples of RedLine panel distributed to affiliates are heavily packed and virtualized after the first layer of obfuscation is applied. But as we were looking through our telemetry for activity related to RedLine Stealer and its panel, we came across an old version of the RedLine panel that was only obfuscated with .NET Reactor, making it much easier to analyze.

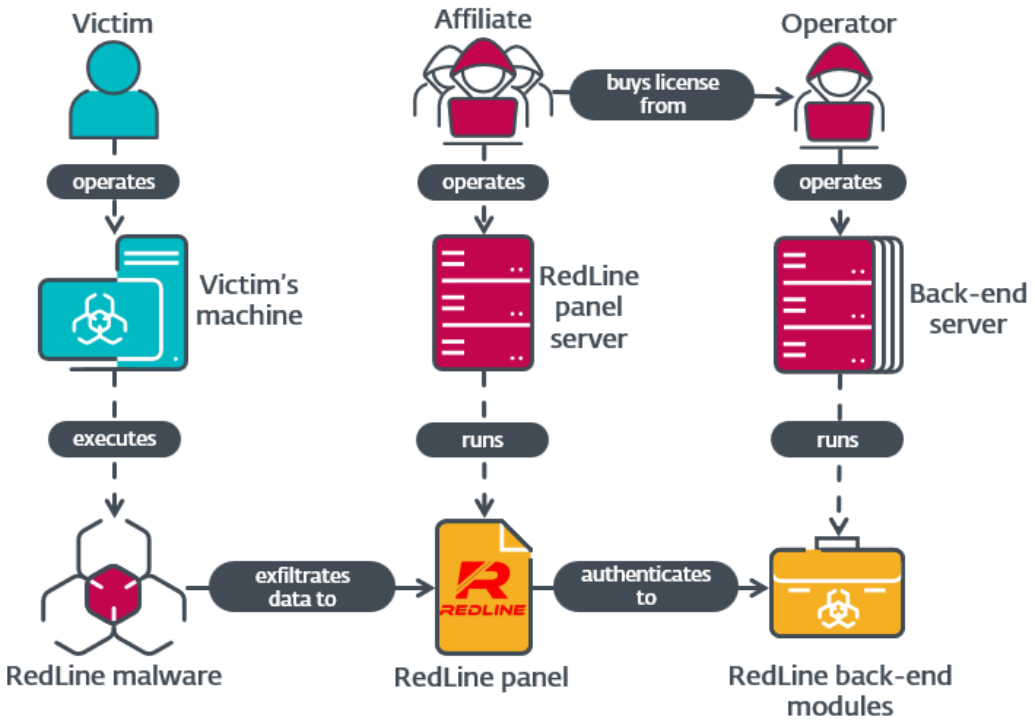


Figure 1. Overview of actors and components involved in RedLine

All the components of RedLine, from the malware itself to the backend authentication server, are written in C# with the .NET framework. The versions we analyzed used the Windows Communication Foundation (WCF) framework to communicate with each other. This framework allows one to define an API using contracts, which are statements applied to classes and interfaces to specify how objects and actions are translated between their representation in the code and the network communications. Thus, programs that interact using this framework must share definitions of these data structures or classes. These shared models proved useful in our analysis of the obfuscated components.

Note that the latest 2024 RedLine version uses a REST API instead of WCF to communicate with the backend.

RedLine panel

This control panel is what affiliates can buy on forums and Telegram channels. Licenses sell for US$150 per month or US$900 for a lifetime license. In exchange for the money, the affiliates get a GUI through which they manage their campaigns, with features to configure what information to collect, create malware samples, view and manage collected information, and integrate with a Telegram bot to sell stolen information.

The 2023 versions of the panel we investigated were heavily obfuscated using DNGuard, a well-known .NET obfuscator, and BoxedApp, a commercial packer and virtualization library. Luckily for us, RedLine used the WCF framework so the panel had to share some classes and interfaces with other components. By analyzing these shared elements from the malware and backend components, which are less protected, we were able to understand a lot of the panel's functionality despite the protections.

All RedLine panels from 2023 were signed with certificates issued to AMCERT,LLC by Sectigo, which have since been revoked after we reported them. AMCERT,LLC corresponds to a company registered in Armenia that doesn't seem to have an online presence – quite unusual for a purported software developer.

Authentication

In order to use a RedLine panel, affiliates must first authenticate. Figure 2 shows the login prompt.

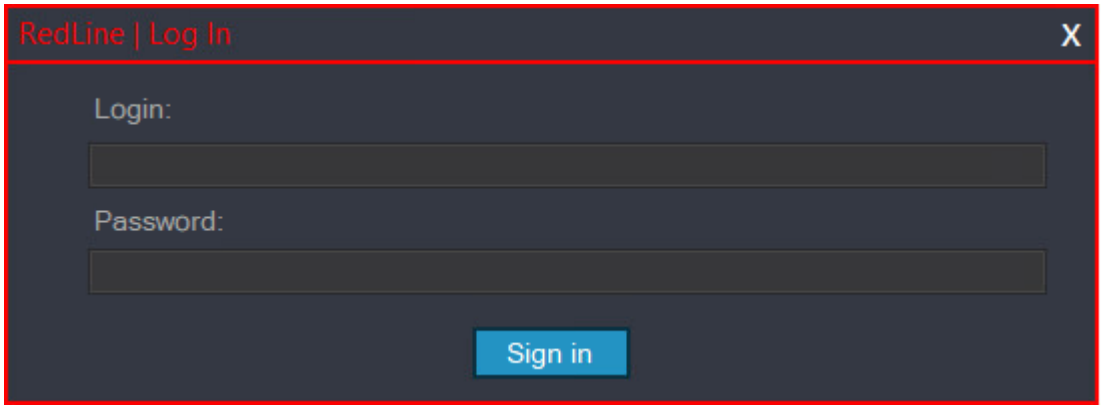


Figure 2. RedLine panel login prompt

The RedLine panels we analyzed use GitHub repositories, like the one in Figure 3, as dead-drop resolvers for their authentication servers. The address of this repository is hardcoded in the panel, but it is not the same for all versions of the panel. Finding and removing several of these repositories in cooperation with GitHub is what enabled us to temporarily disrupt RedLine operations in April 2023. While the removal did not affect the malware's backend, it forced the operators to distribute new versions of the panels. For a short while after the disruption, they moved the dead-drop resolvers to Pastebin, before choosing to use their own domains in May 2023, as described in our joint talk with Flare at Sleuthcon 2023. In the latest version of the RedLine panel, the threat actors abandoned the dead-drop resolvers completely and simply used a hardcoded URL https://fivto[.]online/secure-api/. More information on the dead-drop resolvers can be found in the *Network infrastructure* section.

By running the backend server components in a virtual network, we managed to create affiliate accounts and authenticate with our own instance without having to buy a subscription.

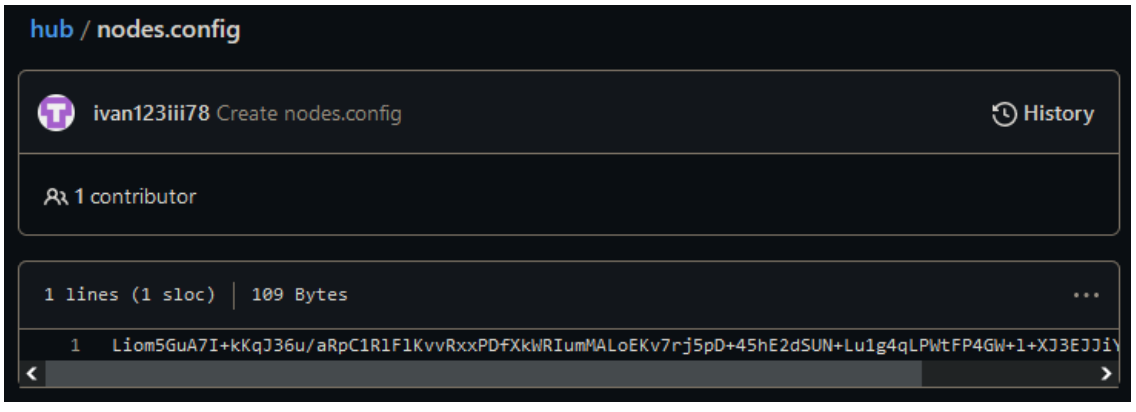


Figure 3. GitHub repository used as a dead-drop resolver

The first screen shown to affiliates upon logging in, seen in Figure 4, is filled with ads. While we do not know the exact nature of the ads served by the real authentication servers, we were able to create some in our own instance of the backend server for demonstration purposes. The backend server doesn't host the actual images present in those ads, only their URLs.

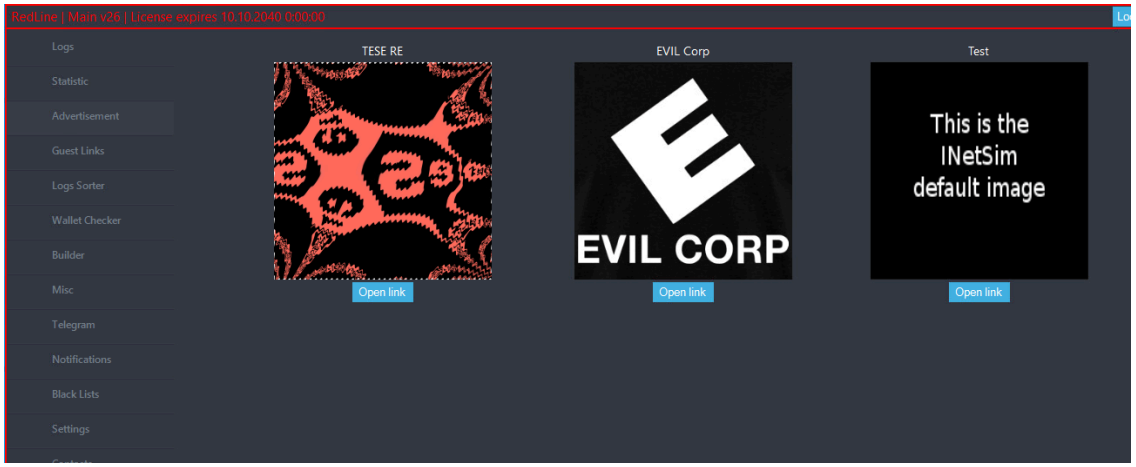


Figure 4. Advertisement *tab of the RedLine panel with fake ads displayed for demonstration purposes*

The Black Lists tab allows affiliates to ignore incoming data by country, IP address, Build ID, or HWID (a unique ID computed from a victim machine's domain name, username, and serial number). Regardless of the exclusions selected here, samples of RedLine Stealer all contain code to prevent execution if the locale is set to one of the following countries: Armenia, Azerbaijan, Belarus, Kazakhstan, Kyrgyzstan, Moldova, Tajikistan, Uzbekistan, Ukraine, and Russia.

The Telegram tab, shown in Figure 5, allows affiliates to configure a Telegram bot to post stolen data to specific chats or channels. The affiliate must provide a valid API token for the bot, and then can select which entries will be shared based on the country, Build ID, OS version, and domains found in cookies or saved logins. The bot can also be configured to share the full logs or only specific information as defined in the Message Format field, and to share statistics with the selected recipients (see Figure 6).

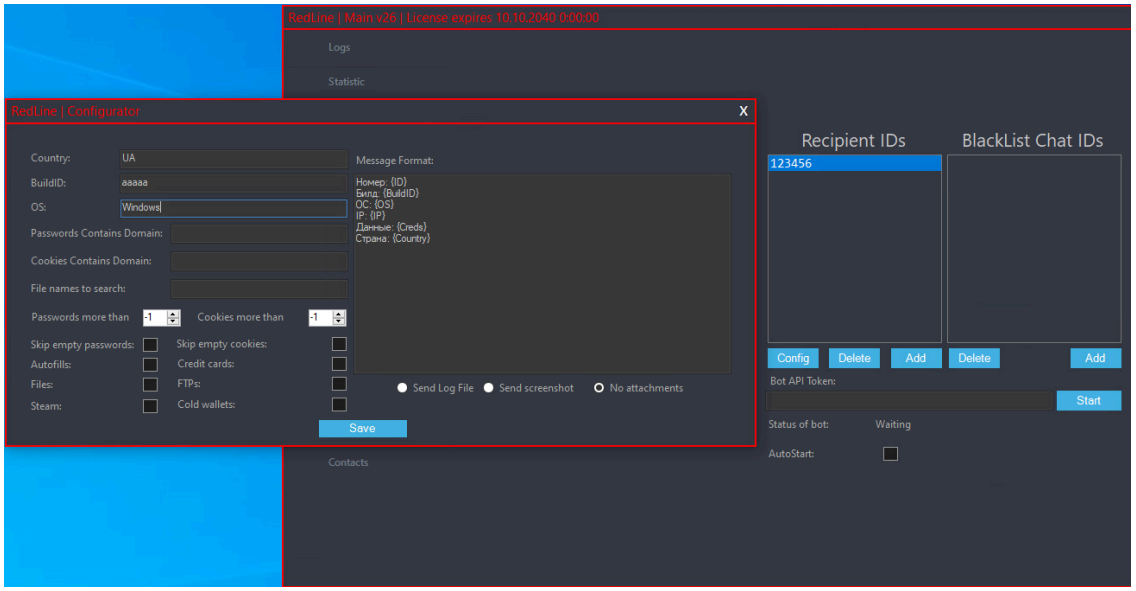


Figure 5. Telegram bot configuration in the RedLine panel

```
if (recipients.Contains(argument.Message.Chat.Id.ToString())
    &&
!blackRecipients.Contains(argument.Message.Chat.Id.ToString()))
{
    StringBuilder stringBuilder = new StringBuilder();
    lock (counterLock)
    {
        stringBuilder.AppendLine($" Total logs: {LogsDataController.Instance.Count}");
        stringBuilder.AppendLine($" Passwords: {Counters.Passwords}");
        stringBuilder.AppendLine($" Cookies: {Counters.Cookies}");
        stringBuilder.AppendLine($" Cold Wallets: {Counters.ColdWallets}");
        stringBuilder.AppendLine($" AutoFills: {Counters.AutoFills}");
        stringBuilder.AppendLine($" FTPs: {Counters.FTPs}");
        stringBuilder.AppendLine($" Files: {Counters.Files}");
        stringBuilder.AppendLine($" Credit cards: {Counters.CreditCards}");
    }
    await client.PreSafeSendTextMessage(argument.Message.Chat.Id, stringBuilder.ToString(), ParseMode.Default);
}
```

Figure 6. Telegram bot code for sharing statistics

Creating malware samples in the front end

The Builder tab, shown in Figure 7, allows affiliates to create new RedLine Stealer samples by providing a RedLine panel server address, a Build ID, an error message to display, and an image to be used as the icon for the created sample. These last two are optional, with the image serving as the icon of the software RedLine is impersonating, while the error message can be used to mislead the victim as to why the expected application wasn't started.

The Build ID is used as a campaign identifier and is sent by the samples along with stolen information. While some previously leaked versions of the RedLine panel included an executable to create builds locally, in newer versions this is performed by the backend server. This change makes any leaked or cracked RedLine panel much less useful since affiliates won't have the ability to create samples to use with it. We describe the build process in the *RedLine.Nodes.LoadBalancer* section.

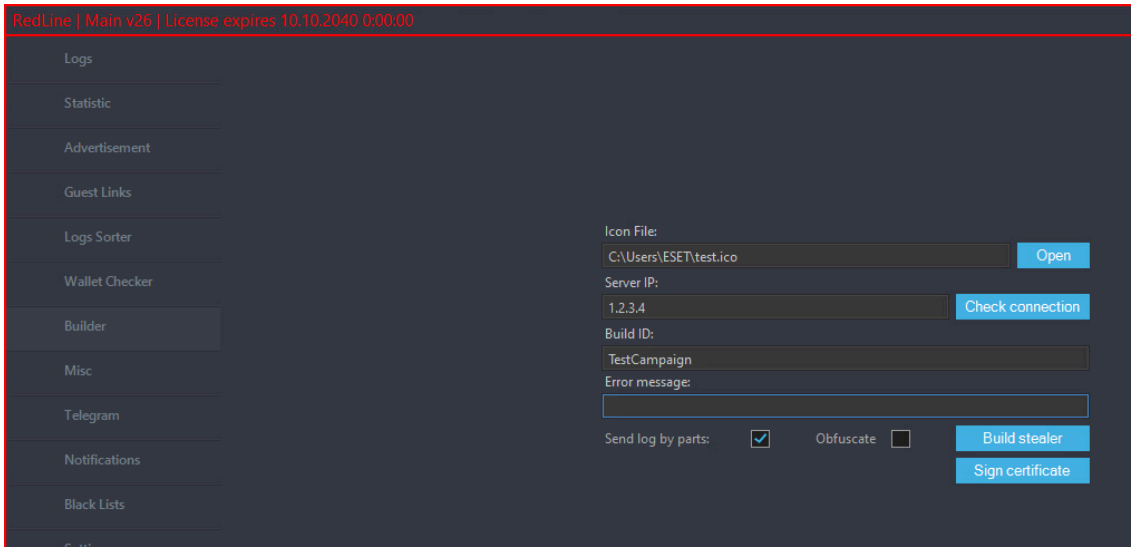


Figure 7. Builder *tab of the RedLine panel*

RedLine backend

The RedLine backend we analyzed in 2023 consists of two modules. The first one, named RedLine.Nodes.DbController, manages affiliate and advertisement data. Despite its name, this module doesn't use a traditional database, but rather stores records as Protobuf-encoded objects in individual files, with a specific subdirectory for each type of data. The second module, named RedLine.Nodes.LoadBalancer, functions as the server and provides most of the functionality used by the RedLine panel.

We also found a module called RedLine.MainServer, which is probably the ancestor of RedLine.Nodes.DbController and RedLine.Nodes.LoadBalancer. In later versions, it was split off as a separate module that handles user and advertisement data.

In the 2024 version of the backend, DbController and LoadBalancer have been replaced by a single module named Nodes.Api.

RedLine.Nodes.DbController

In DbController, affiliate data is represented by a class named ClientData, detailed in Table 1.

Table 1. Description of ClientData *properties*

Attribute	**Description**
ID	Unique numeric ID.
Login	Username.
Password	Password.
RegistrationDate	Timestamp of the affiliate account's creation.
Activated	Whether the affiliate has bought a license.
LastActive	Timestamp of the affiliate's last activity.
LastIP	IP from which the affiliate's last activity occurred. Used together with LastActive to prevent authentication if the affiliate's account was active from another IP address in the last five minutes. We believe this is to prevent account sharing.
Premium	*Unused in the modules we analyzed.*
ActivateUntil	Expiration timestamp of the affiliate's license.
BuildKey	Unique string used to identify samples created by the affiliate.

This is mostly straightforward but presents a few interesting quirks, mainly the way passwords and lifetime licenses are handled.

Looking at the code used to authenticate affiliates (shown in Figure 8) makes it obvious that passwords are stored in cleartext. The submitted password is compared directly to the stored one without any hashing function being involved.

```
string text = (string.IsNullOrWhiteSpace(client.Login) ? "NULL" : client.Login);
string text2 = (string.IsNullOrWhiteSpace(client.Password) ? "NULL" : client.Password);
string text3 = (string.IsNullOrWhiteSpace(currentIp) ? "NULL" : currentIp);
bool flag = false;
try
{
    if (text == "NULL" || text2 == "NULL")
    {
        return flag;
    }
    object obj = UsersDataController.DataBaseLock;
    lock (obj)
    {
        IEnumerable<ClientData> dbInstance = UsersDataController.DbInstance;
        Func<ClientData, bool> func;
        if ((func = <>9__1) == null)
        {
            func = (<>9__1 = (ClientData x) => x.Login == client.Login && x.Password == client.Password && x.Activated && x.ActivateUntil >= DateTime.Now);
        }
        client = dbInstance.FirstOrDefault(func);
```

Figure 8. Excerpt of the function to handle affiliate logins in DbController

Figure 9 shows that a lifetime license, which affiliates can buy for US$900, is represented by setting the expiration date of a given license to any date after 2025. This is alternatively referred to as a Pro License elsewhere in the code.

```
DateTime result = this.OnCheckExpire(client.Login).Result;
Console.WriteLine(string.Format("Trying to sign file {0}, expires at {1}", client.Login, result));
if (result.Year > 2025)
{
    this.AddNotify("Signed file by " + client.Login + "|" + client.LastIP);
    return Signer.SignFile(exe);
}
return Encoding.UTF8.GetBytes("Message: Only for lifetime license!");
```

Figure 9. Code from the LoadBalancer *module used to check whether an affiliate has a lifetime license*

RedLine.Nodes.LoadBalancer

As seen in Figure 10, the LoadBalancer module listens on port 8778. This screenshot also shows the handlers defined in the MainPanelService WCF contract. The handlers that are related to affiliate and advertisement data – namely OnSignInInternal, OnConnect, OnCheckExpire, and OnGetPartners – delegate requests to the corresponding handler of DbController. The OnCheckConnect handler is used by the Redline panel to check whether its C&C server is reachable from the exterior. The rest of the handlers all relate to the creation of malware samples.

```
public WcfServiceHost(IMainServer mainServer)
{
    MainPanelService.OnSignInInternal = (OnSignInInternalHandler)Delegate.Combine(MainPanelService.OnSignInInternal, new OnSignInInternalHandler
      (this.OnSignInInternal));
    MainPanelService.OnCheckConnect = (OnCheckHandler)Delegate.Combine(MainPanelService.OnCheckConnect, new OnCheckHandler(this.OnCheckConnect));
    MainPanelService.OnConnect = (OnConnectHandler)Delegate.Combine(MainPanelService.OnConnect, new OnConnectHandler(this.OnConnect));
    MainPanelService.OnExpireCheck = (OnCheckExpireHandler)Delegate.Combine(MainPanelService.OnExpireCheck, new OnCheckExpireHandler(this.OnCheckExpire));
    MainPanelService.OnRequestPartners = (OnRequestAdHandler)Delegate.Combine(MainPanelService.OnRequestPartners, new OnRequestAdHandler(this.OnGetPartners));
    MainPanelService.OnCreateClipper = (OnClipperHandler)Delegate.Combine(MainPanelService.OnCreateClipper, new OnClipperHandler(this.OnCreateClipper));
    MainPanelService.OnCreateLastBuild = (OnCreateLastBuildHandler)Delegate.Combine(MainPanelService.OnCreateLastBuild, new OnCreateLastBuildHandler
      (this.OnCreateLastBuild));
    MainPanelService.OnSignFile = (OnSignFileHandler)Delegate.Combine(MainPanelService.OnSignFile, new OnSignFileHandler(this.OnSingFile));
    Uri uri = new Uri(string.Format("net.tcp://{0}:{1}", "0.0.0.0", "8778"));
```

Figure 10. Initialization code for the MainPanelService *WCF service and its handlers*

Sample creation in Redline backend

Since it was first documented in 2020, RedLine Stealer has been rewritten to use the WCF framework, and later a REST API, for network communication. As seen in Figure 11, the internal name for these rewritten versions appears to be RedLine.Reburn.

```
if (ProVersion)
{
    solution_path = Path.Combine(Directory.GetCurrentDirectory(), "RedLine.Reburn" + version);
}
else
{
    solution_path = Path.Combine(Directory.GetCurrentDirectory(), "RedLine.Reburn" + version);
}
string project_path = string.Empty;
Path.Combine(solution_path, "RedLine.Reburn.sln");
```

Figure 11. Code to obtain the path to the RedLine.Reburn *solution*

To create samples of the RedLine Stealer malware, the OnCreateLastBuild handler shown in Figure 10 uses the CreateBuild method from the custom VSBuilder class. When a CreateBuild request is received, the cleartext password for the affiliate's account is logged to the console. This shows a casual disregard for well-known security practices.

If in the request no port is specified for the C&C server, the builder defaults to port 6677. When building a sample, the code in Figure 12 is used to derive a token from the campaign's Build ID and the affiliate account's Build Key. We believe this corresponds to the value of the ns1.Authorization header used in communication between the samples and panel. This header seems to be used by the panel to filter out connections from samples not created by the current account.

```
Md5Helper.GetMd5Hash(Md5Helper.GetMd5Hash(client.BuildKey).ToLower() + buildId + "-
a=sj9odflaISDH0s@^&sas").ToLower()
```

Figure 12. Code used to derive token values for RedLine Stealer samples

The VSBuilder.CreateBuild method uses a local C# solution to create samples. Some of the source files are read into memory to replace specific values before being written back to disk. While we were unable to collect the project's files, the decompiler output for this method along with that of a RedLine Stealer sample provide enough context to understand that the modified values are:

- C&C address (encrypted),
- Build ID (encrypted),
- fake error message (encrypted),
- decryption key,
- flag to indicate whether to send stolen information in parts or all at once,
- authentication token, and
- assembly name.

Once these values are replaced, Visual Studio is invoked to build the project.

If the affiliate has selected the corresponding option, and has a lifetime license, the compiled executable is obfuscated with the commercially available Babel Obfuscator. In the 2024 version, obfuscation is implemented for all users and is done via .NET Reactor instead.

Finally, a self-signed certificate is generated with the code from Figure 13, and used to sign the executable. The fields of its Distinguished Name (DN) are filled with random dictionary words. The certificates generated this way use the hardcoded password 123321 and have a validity period extending from seven days before their generation to 10 years after.

```
public static X509Certificate2 GenerateCertificate()
{
    X509Certificate2 x509Certificate;
    using (CryptContext cryptContext = new CryptContext())
    {
        cryptContext.Open();
        x509Certificate = cryptContext.CreateSelfSignedCertificate(new SelfSignedCertProperties
        {
            IsPrivateKeyExportable = true,
            KeyBitLength = 4096,
            Name = new X500DistinguishedName(string.Concat(new string[]
            {
                "CN=",
                new WordNames().Get(1),
                ";E=",
                new WordNames().Get(1).ToLower(),
                new WordNames().Get(1).ToLower(),
                "@gmail.com;C=",
                StringHelper.GetRandom(2).ToUpper(),
                ";O=",
                new WordNames().Get(1),
                " ",
                new WordNames().Get(1),
                " Inc."
            })),
            ValidFrom = DateTime.Today.AddDays(-7.0),
            ValidTo = DateTime.Today.AddYears(10)
        });
    }
    return x509Certificate;
}
```

Figure 13. Function that generates self-signed certificates with random words in the DN

The LoadBalancer module also provided an OnCreateClipper handler. As the name suggests, this was used to generate clipboard hijacking malware; in this case, it was used to hijack cryptocurrency transactions by replacing wallet addresses in the clipboard with those of attacker-controlled wallets. This functionality has been removed in the latest versions of the Redline backend.

Interestingly, the Builder class also contained dead code to generate a malware sample from a stub executable. We believe this was a leftover from the method used to generate previous versions of RedLine Stealer. The latest version of the Redline backend does not contain the code anymore.

Another handler, named OnSignFile, allows affiliates with lifetime licenses to sign arbitrary files with a certificate located on the backend server. We were unable to collect this certificate, but have reason to believe that it may be the same certificate used to sign RedLine panels, since it has also been used to sign a large number of RedLine Stealer and other malware samples. Another likely candidate is the certificate with the thumbprint 28F9A8E7601F5338BF6E194151A718608C0124A8, issued to Hangil IT Co., Ltd. This, likely stolen, certificate has been used to sign many RedLine Stealer samples and other malicious files. It has already been revoked.

RedLine.MainServer

The RedLine.MainServer module combines some of the functionality of the two previous modules with a GUI that allows the admin to easily manage affiliate accounts and advertisement data. Despite this added functionality, multiple factors indicate that this version of MainServer is probably an ancestor of LoadBalancer and DbController rather than a successor:

- MainServer samples are compiled with an older version of the .NET framework (4.6.2 vs 4.8).
- The assembly's copyright year is 2020 instead of 2021.
- Some MainServer samples do not contain functionality for advertisements.
- The main WCF service's contract (MainPanelService) does not include the IsAlive handler that is present in the same class as LoadBalancer. This handler is used by all the most recent RedLine panels we have analyzed.

Note that in later versions of the RedLine backend, the GUI administration panel has been split off into its own module also named MainServer.

The examined version of the GUI gave us interesting insight into RedLine management. It provides a form to create and edit affiliate account data, as shown in Figure 14. The fields correspond to the ClientData class described previously.

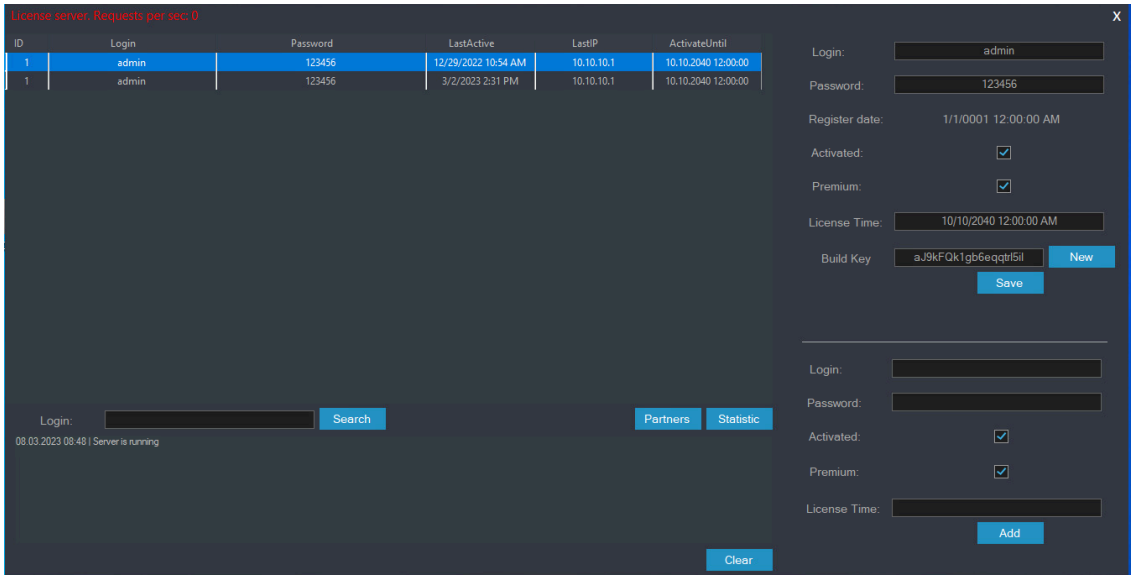


Figure 14. RedLine MainServer *affiliate account management interface*

A similar form, in Figure 15, exists to manage advertisements.

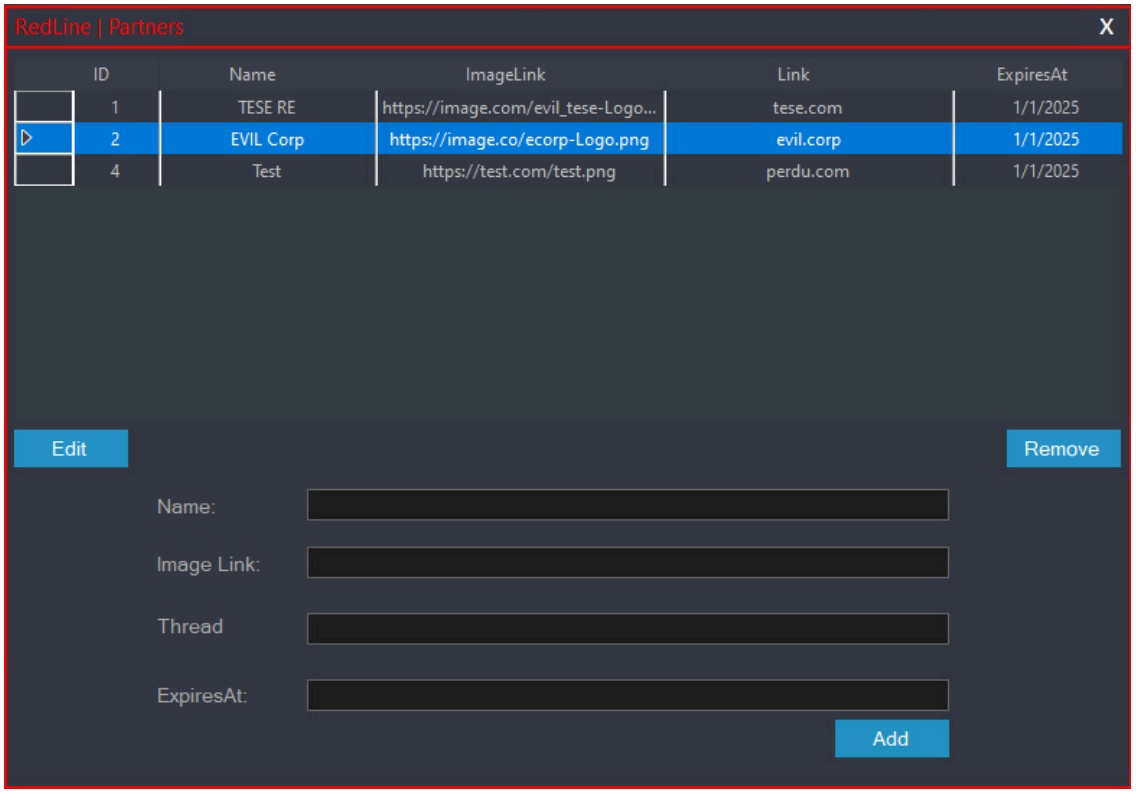


Figure 15. RedLine MainServer *advertisement management interface*

Finally, the rudimentary dashboard in Figure 16 gives the operator an overview of license sales.

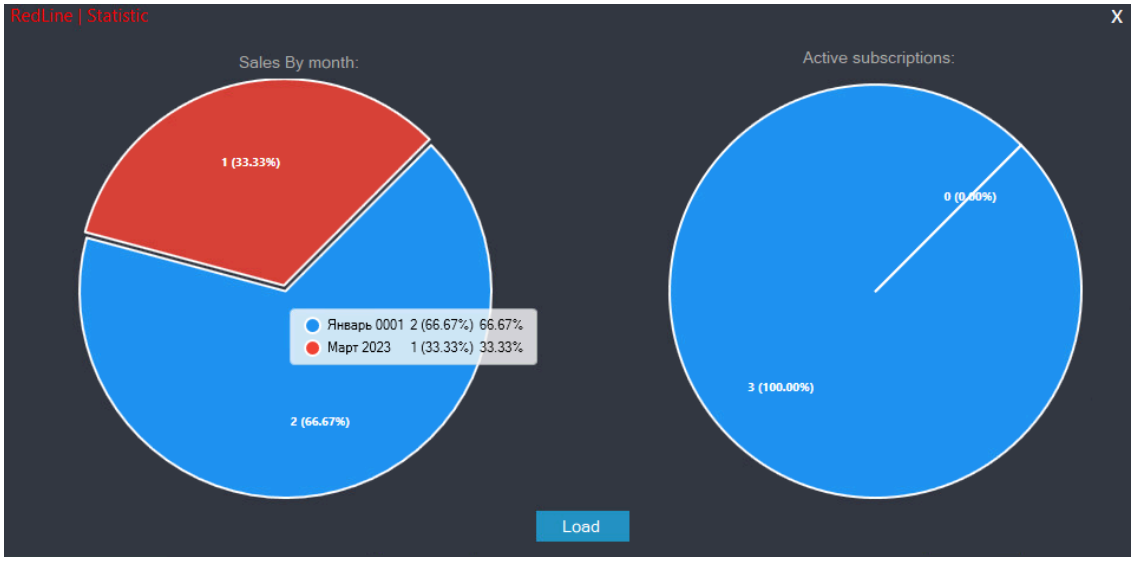


Figure 16. Statistic *dashboard of the* MainServer

Nodes.Api

Analysis of samples shared by Dutch law enforcement shows that in the latest RedLine versions, the functionalities of the LoadBalancer and DbController modules have been combined into a new one named Nodes.Api. This backend module is packaged as a single-file .NET application and uses the WebApplication class from ASP.NET to provide the REST API used by the panels. As seen in Figure 17, the module handles requests

sent to both RedLine (fivto[.]online) and META (spasshik[.]xyz) servers, showing that the two infostealers share the same operators.

```
UseExtensions.Use(webApplication, delegate(HttpContext context, Func<Task> next)
{
    string[] array = new string[] { "::1", "localhost", "spasshik.xyz", "fivto.online" };
    if (!Enumerable.Contains<string>(array, context.Request.Host.Value.ToLower()))
```

Figure 17. Code to set up the listener for requests sent to both RedLine and META servers

Table 2 lists the endpoints exposed by the REST API along with the corresponding WCF handler from previous versions.

Table 2. RedLine endpoints

Endpoint	Corresponding WCF Handler	Description
/api/test	N/A	Probably used to test connectivity. This simply returns the value OK.
/secure-api/sign-in	OnSignInInternal	Handles logins from the panel.
/secure-api/getBanners	OnGetPartners	Returns the ads to be displayed in the panel.
/secure-api/createFile	OnCreateLastBuild	Creates an instance of the stealer malware.
/secure-api/checkConnect	OnCheckConnect	Pings the specified address and port to check whether it is externally reachable.
/secure-api/updateDb	N/A	Makes the backend reload affiliate data. This is only done if the request comes from a loopback address.
/edk92hd/createRandom	N/A	Creates a new affiliate entry with a random username and password.
/edk92hd/renew	N/A	Extends the license validity period of the specified affiliate.
/Panel.zip	N/A	Returns the Panel.zip file from disk. This password-protected archive contains the latest version of the panel.

Unlike the 2023 versions of the backend we originally analyzed, lifetime licenses no longer appear in the code of the new module. Since licenses of this type were still being sold, we believe they were likely handled by simply setting their expiration to a date far in the future. Functionality related to code signing has also been completely removed.

Samples created by the Nodes.Api module are obfuscated using .NET Reactor if it is present at the hardcoded path C:\Program Files (x86)\Eziriz\.NET Reactor\dotNET_Reactor.exe. This corresponds to its default installation

path.

Beyond the aforementioned changes, features that were present in older versions behave much as they did before. However, a couple of interesting functionalities have been added.

Affiliate management

Three endpoints used to manage affiliate data are exposed. These appear to be meant for use only by the operator, but this is handled in two very different ways. The /secure-api/updateDb endpoint causes the server to reload user data from disk. Figure 18 shows the update method invoked by the MainServer module whenever a new entry is created or modified. The operation is only performed by the Nodes.Api module if the request comes from a loopback address, which prevents Panel users from using it.

```
public static string RequestUpdates()
{
    return new MyWC
    {
        Headers = {
        {
            HttpRequestHeader.UserAgent,
            "RestSharp"
        } }
    }.DownloadString("https://localhost/secure-api/updateDb");
```

Figure 18. Method invoked by the MainServer *module*

The other two endpoints, /edk92hd/createRandom and /edk92hd/renew, use a different method of mitigating unauthorized access. As seen in Figure 19, they can only be invoked if the request contains an sko3s header set to a seemingly random hardcoded value.

```
if (!context.Request.Headers.TryGetValue("sko3s", out stringValues) || !
  (Enumerable.FirstOrDefault<string>(stringValues) == "sl03j'SDKdskjd=!3dfg"))
{
    return Results.BadRequest(null);
```

Figure 19. Code for checking whether request contains the correct sko3s *header*

Backups

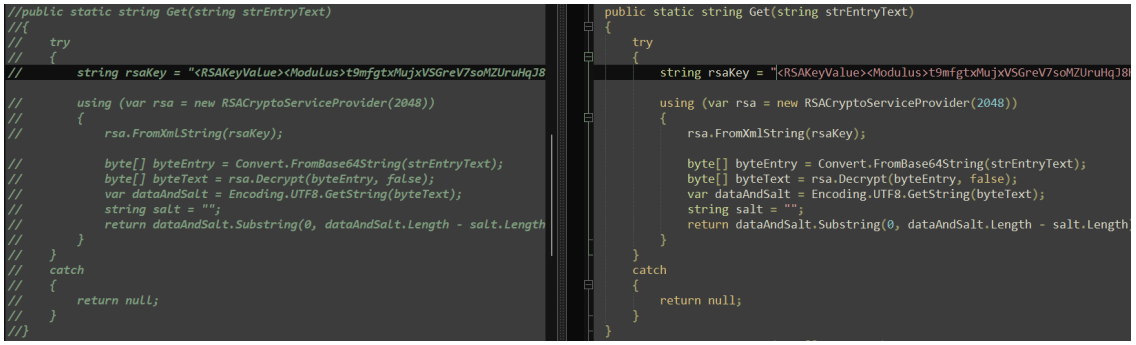
The other interesting feature is the ability to back up affiliate data, functionality that was missing in the older versions of the code. Since this data is stored in a series of files on disk, the backup is a ZIP archive of the directory that contains the files. This archive is then sent via Telegram using a hardcoded Chat ID shown in Figure 20.

```
db.InvokeLockedTransaction(delegate(DbController<User, int, JsonDbSeriallizer<User>> x)
{
    ZipFile.CreateFromDirectory(db.DataBaseDir, zipName + ".zip");
});
InputOnlineFile inputOnlineFile = new InputOnlineFile(new MemoryStream(File.ReadAllBytes(zipName + ".zip")), z
TelegramBotClient telegramBotClient = new TelegramBotClient("7246210644:AAEvpWwij5wRZ48mC7qKQvMESZrZCgjw5bk",
telegramBotClient.StartReceiving(null, default(CancellationToken));
telegramBotClient.SendDocumentAsync("-1002239081293", inputOnlineFile, null, ParseMode.Default, null, false, f
```

Figure 20. ZIP archive creation

Links with META Stealer

META Stealer is the other infostealer disrupted alongside RedLine Stealer. According to an article by Kela, META Stealer was first announced on cybercrime forums in March 2022. The author claimed that it used the same code as RedLine Stealer and provided the same functionality and panel. META Stealer hasn't been investigated as thoroughly as RedLine Stealer, but our research indicates that the claims it initially made are accurate. Based on the source code, the two infostealers are most probably made by the same person. While there are some differences, most of the code is the same with instances of the string RedLine replaced by Meta. As can be seen in Figure 21, the code that is commented out to be inactive in RedLine is present in META.



```
//public static string Get(string strEntryText)
//{
//    try
//    {
//        string rsaKey = "<RSAKeyValue><Modulus>t9mfgtxMujxVSGreV7soMZUruHqJ8
//
//        using (var rsa = new RSACryptoServiceProvider(2048))
//        {
//            rsa.FromXmlString(rsaKey);
//
//            byte[] byteEntry = Convert.FromBase64String(strEntryText);
//            byte[] byteText = rsa.Decrypt(byteEntry, false);
//            var dataAndSalt = Encoding.UTF8.GetString(byteText);
//            string salt = "";
//            return dataAndSalt.Substring(0, dataAndSalt.Length - salt.Length
//        }
//    }
//    catch
//    {
//        return null;
//    }
//}
```

```
public static string Get(string strEntryText)
{
    try
    {
        string rsaKey = "<RSAKeyValue><Modulus>t9mfgtxMujxVSGreV7soMZUruHqJ8
        using (var rsa = new RSACryptoServiceProvider(2048))
        {
            rsa.FromXmlString(rsaKey);

            byte[] byteEntry = Convert.FromBase64String(strEntryText);
            byte[] byteText = rsa.Decrypt(byteEntry, false);
            var dataAndSalt = Encoding.UTF8.GetString(byteText);
            string salt = "";
            return dataAndSalt.Substring(0, dataAndSalt.Length - salt.Length
        }
    }
    catch
    {
        return null;
    }
}
```

Figure 21. RedLine Stealer (left), and META Stealer (right) source code comparison

As we already mentioned, another piece of evidence pointing towards RedLine and META having the same operators is seen in the code of the Nodes.Api module, which handles requests sent to both RedLine and META servers, as shown in Figure 17.

We also found two samples of the META panel signed with a certificate that was also used to sign samples of the RedLine panel. This panel used the same dead-drop resolver schemes, going as far as using the same AES and RSA keys, only with a different GitHub repository. Additionally, as Figure 22 shows, a comparison of the panels used by META and RedLine reveals only minor cosmetic differences.

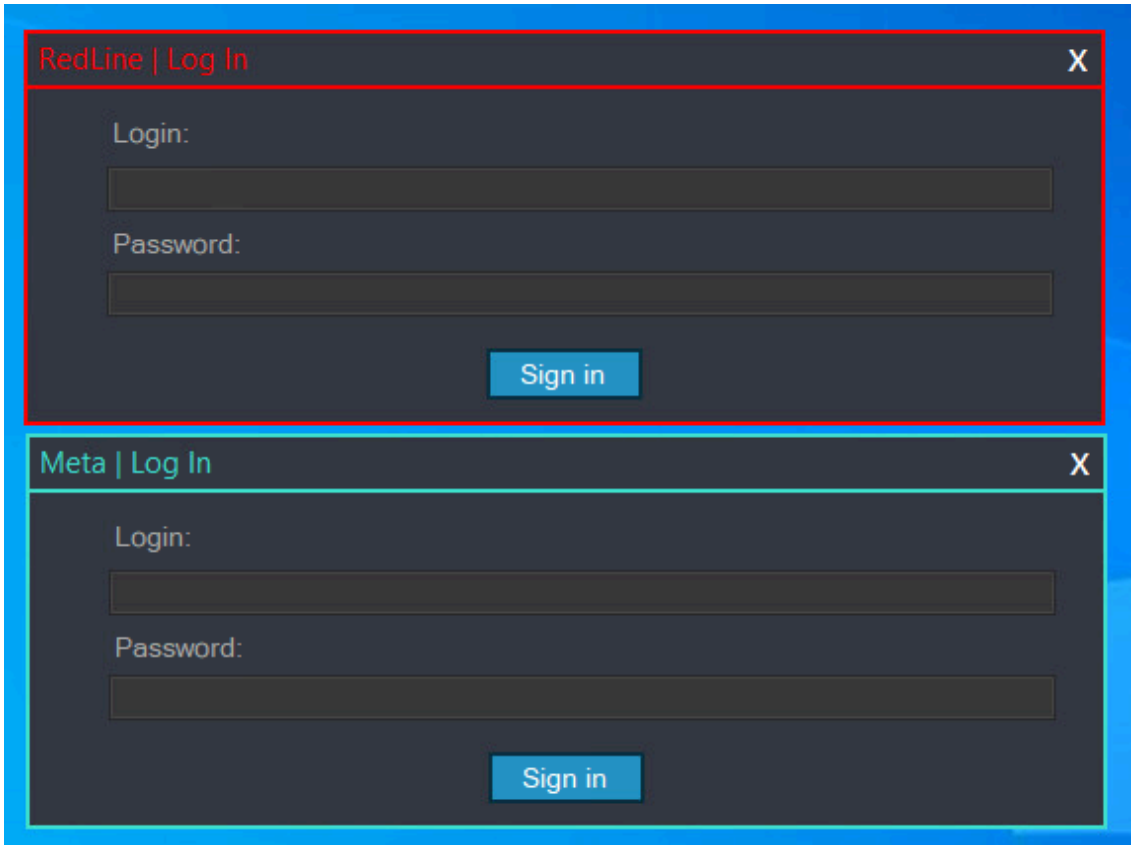


Figure 22. Panel login prompts for RedLine (top) and META (below)

META Stealer uses the same combination of DNGuard and BoxedApp to protect its panel from analysis. However, META's authentication process could not be completed when run against our instance of the RedLine backend from 2023, so it seems to have been modified from that of RedLine.

Network infrastructure

By parsing samples of RedLine that we detected between November 30th, 2022 and March 23rd, 2023, we were able to identify over 1,000 unique IP addresses used to host RedLine panels. Figure 23 shows the geographical distribution of these hosted panels. Russia, Germany, and the Netherlands each account for about 20% of the total, while Finland and the United States each represent around 10%. Even though this data only comes from samples targeting our customers, we believe it paints a fairly accurate picture with regards to the overall distribution of the malware.

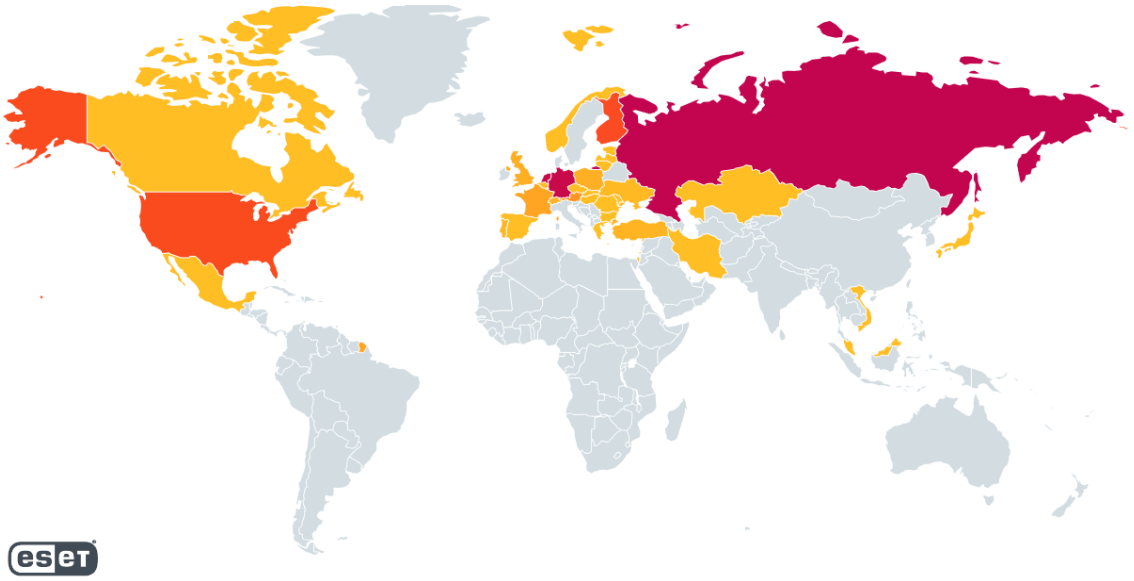


Figure 23. Heatmap showing the geographical distribution of hosted RedLine panels

We were also able to identify multiple distinct backend servers. It's likely that there were more in the dead-drop resolvers that we couldn't decrypt. Based on their geographical distribution, shown in Figure 24, the servers are mainly located in Russia (about a third of them) while the UK, the Netherlands, and the Czech Republic each represent around 15% of the servers we identified.

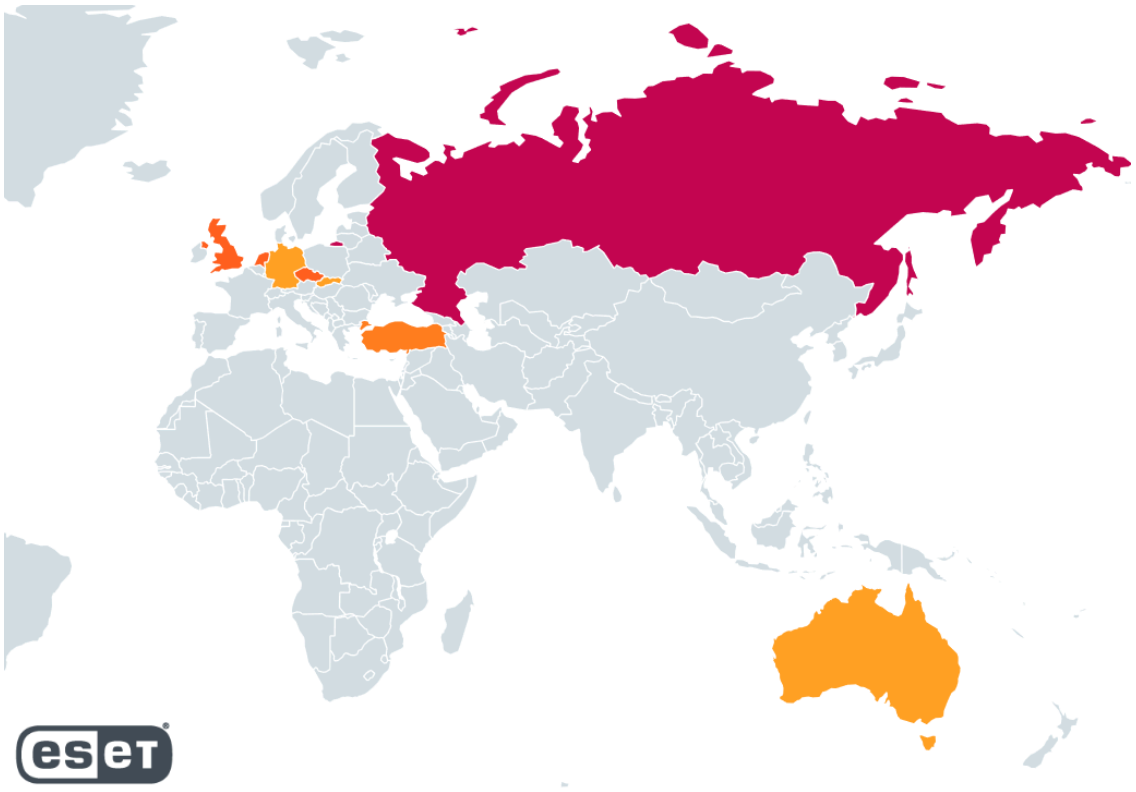


Figure 24. Heatmap showing the geographical distribution of RedLine backend servers

Dead-drop resolvers

As we already mentioned, the 2023 versions of the RedLine panel used GitHub repositories as dead-drop resolvers for its authentication servers. These repositories all contained a file with an encrypted list of server addresses. The file was encrypted using a custom module simply named RSA. In one version of this module, shown in Figure 25, the list was encrypted using AES-CBC with a hardcoded key and IV, and saved to a file named nodesUpdate.config.

```
public static string EncryptHosts(string input)
{
    string text;
    using (Aes aes = Aes.Create())
    {
        aes.Key = Convert.FromBase64String("mL2                                    FV7s=");
        aes.IV = Convert.FromBase64String("7lV7                    SiIQ==");
        ICryptoTransform cryptoTransform = aes.CreateEncryptor(aes.Key, aes.IV);
        using (MemoryStream memoryStream = new MemoryStream())
        {
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream, cryptoTransform, CryptoStreamMode.Write))
            {
                using (StreamWriter streamWriter = new StreamWriter(cryptoStream))
                {
                    streamWriter.Write(input);
                }
                text = Convert.ToBase64String(memoryStream.ToArray());
            }
        }
    }
    return text;
}
```

Figure 25. EncryptHosts *function with hardcoded AES key and IV*

More recent versions, which were in use at least until mid 2024, employ RSA encryption instead (see Figure 26), with the output written to a file named nodes.config. In this case, the key is read from a file rather than being hardcoded in the executable. However, the class used to perform RSA encryption is also present in the LoadBalancer module, with hardcoded default values for the public and private keys. Note the use of “nodes” again to refer to the backend servers.

```
public static string Encrypt(string strText, string strPublicKey = "<RSAKeyValue><Modulus>t9mfgtxMujxVSGreV7soMZUruHqJ8Hj13V4L++/9Oke
  +v65Mg8vebywwYt4aLdvMuiZO9SJqFSE1E6rRfc4H+hNh7/2APgF5Q3vw9FJDBNo33H65Hsbr5U9725HpSfRiKP3NQteYjEg0ZuK2g+8
  +YPbtvdAVkHY594iyUq5jv6DlY5rcridLVfawtisYfxeKGvop2RL7JAZrcajJHuqKSrrXJAbD9wM+Y9zhOZkm6WFI2o1pkj3eilZ/
  MSOR4gXcngunjzl1wXTMlOVASHw9kkUIz1xg8DCRIXQGwzSDADa+SHp2ICKXkBP52rvzi/ybqP9wL3pc20BP9BYlNlcNYQ==</Modulus><Exponent>AQAB</Exponent></
  RSAKeyValue>")
public static string Decrypt(string strEntryText, string strPrivateKey = "<RSAKeyValue><Modulus>4kPe5YKpQJOG6n
  +ksH1hb541xRFlWdXKE3Gg09YunDKa8mBgOumKttbEM3R/53/RwNO8sqnhEaa4lTAJKR/4bm5TLk5reZ9kuj0zaLV4LcjiHWYz0XKEBmiNdApe2EYrkRybT0YNLu/
  xEoBrlpmLQOME9h03rnx9JF8i1d7FkKoJTqOGAQtMNjeStKaYO3j/3etOPT25oCi2MT/k3n/pRtGsqrZi86vZ6sNipszRYTNdXrO896FdW0n/
  hi7Sd6zdiNGUcpFbC6cUetZWOvXeuuDkIpEnkEY19+ZNWfZp4MrBBvK3CQfWl9IbVYZqF/xSnNs/Kf8B/abZ0ooWb93AQQ==</Modulus><Exponent>AQAB</
  Exponent><P>
```

Figure 26. RSA encryption and decryption functions with hardcoded default keys

We have observed GitHub repositories that contain at least one such encrypted file. Along with the keys shown above, we were able to extract an RSA private key from a sample of the panel. This allowed us to obtain lists of authentication servers, which we shared with law enforcement agencies.

Conclusion

Before Operation Magnus, RedLine was among the most widespread of infostealer malware with a very large number of affiliates using its control panel. However, the malware-as-a-service enterprise seems to be orchestrated by only a handful of people, some of whom have been identified by law enforcement.

META Stealer, RedLine Stealer’s clone and most probably created by the same threat actors, made its entry into the field in 2022. It does not appear to be a successor to RedLine, since the development of both families has continued in parallel. It was taken down alongside RedLine Stealer.

Our in-depth analysis of the behind-the-scenes parts of RedLine – its control panel and backend – hopefully provides a more holistic understanding of this threat.

> *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) and samples can be found in our GitHub repository.

Files

SHA-1	Filename	Detection	Description
1AD92153B56FC0B39F8F CEC949241EC42C22FA54	Nodes.Api.exe	N/A	RedLine backend single-file application.
8A0CAFE86C0774F1D9C7 F198505AE15D04447DD6	MainServer.exe	N/A	RedLine backend user and advertisement management module (2024 version).
607DBA5F630A1DBFF0E1 3EEBA2730AB9AB2FB253	Nodes.Api.dll	N/A	RedLine backend main module.
FB3ABAC1FAC852AE6D22 B7C4843A04CE75B65663	Panel.exe	MSIL/Spy.RedLine.O	META stealer panel (2024 version).
EE153B3F9B190B1492DE FBB1C70830A28F7C41B2	RedLine.MainPanel.exe	MSIL/Spy.RedLine.H	RedLine stealer panel (2024 version).
1AB006B1C5403BA46480 59DF93B6DAEB0E3EC43F	Panel.exe	MSIL/Spy.RedLine.O	RedLine stealer panel (2024 version).
DC3A236245AE8C4D5D07 9E429ED6B77A5B5245C2	RedLine.MainServer.exe	N/A	RedLine backend licensing server GUI.

SHA-1	Filename	Detection	Description
06A2A900561C122F4508 8A5EAE9146F7675C63F6	rsa.exe	N/A	Tool to encrypt the list of C&C servers.
1626F2666782710FC28D 4AFE607C7BE54F1FC67F	RedLine.Nodes .LoadBalancer.exe	N/A	RedLine backend server module.
37D1221CE6BB82E7AD08 FD22BD13592815A23468	RedLine.SharedModels .dll	MSIL/Spy.RedLine.K	RedLine WCF models and contracts definitions.
66C0E7E74C593196E092 5A7B654E09258E3B1FB7	Panel.exe	Win32/GenCBL.ATC	RedLine panel (v22.4).
2E5D9F2ED82C81609F4C 49EA31642B1FB5FC11B5	RedLine.MainPanel.exe	MSIL/Spy.RedLine.H	RedLine panel (non-virtualized).
47B78A5698A289C73175 C5C69786DE40C7C93C12	RedLine.SharedModels .dll	MSIL/Spy.RedLine.J	RedLine models and contracts definitions.
49BE1D7C87AC919BB908 3FA87F7B907E5F2C9835	Panel.exe	MSIL/Spy.RedLine.H	META Stealer Panel.
4BF4D42EED7FCA8FD528 63B7020AC646EC6D97E9	RedLine.Nodes .DbController.exe	N/A	RedLine backend server user and advertisement management module.
27BD472729439D5B8814 D4A8A464AF9832198894	Panel.exe	MSIL/Spy.RedLine.H	RedLine panel (v26).
A154DFAEDC237C047F41 9EB6884DAB1EF4E2A17D	Panel.exe	MSIL/Spy.RedLine.H	RedLine Panel (leaked cracked version).

Network

Note that the domains in the table below have been seized by law enforcement. The other panel and server addresses that we collected were shared with law enforcement agencies on a regular basis to help in their actions and are no longer active.

IP	Domain	Hosting provider	First seen	Details
N/A	spasshik[.]xyz	N/A	2024-06-02	META backend REST server.
N/A	fivto[.]online	N/A	2024-08-03	RedLine backend REST server.

MITRE ATT&CK techniques

This table was built using version 15 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1583.003	Acquire Infrastructure: Virtual Private Server	Instances of the RedLine back end are hosted on leased virtual private servers.
	T1583.004	Acquire Infrastructure: Server	Instances of the RedLine back end are hosted on servers that appear to be exclusive to RedLine.
	T1587.001	Acquire Infrastructure: Web Services	Operators of RedLine have created multiple GitHub accounts and repositories.
	T1587.002	Develop Capabilities: Malware	Operators of RedLine have developed their own malware families, control panels, and back-end servers.
	T1588.003	Develop Capabilities: Code Signing Certificates	The RedLine back end automatically generates self-signed certificates when creating samples.
	T1608.002	Obtain Capabilities: Code Signing Certificates	RedLine panels are signed with valid certificates issued to AMCERT,LLC.
	T1608.001	Stage Capabilities: Upload Malware	Back-end components of RedLine are uploaded to private servers.
Defense Evasion	T1622	Debugger Evasion	The RedLine panel automatically terminates itself if it detects a debugger or analysis tools.
	T1027.002	Obfuscated Files or Information: Software Packing	Samples of the RedLine panel are packed using DNGuard and BoxedApp.
Command and Control	T1132.001	Data Encoding: Standard Encoding	RedLine makes extensive use of base64 encoding in its network communications.

Tactic	ID	Name	Description
			Network communication uses the standard binary encoder of the WCF framework.
	T1573.001	Encrypted Channel: Symmetric Cryptography	Communications between the panel and back-end server use AES encryption. In some cases, dead-drop resolver content is encrypted with AES-CBC.
	T1573.002	Encrypted Channel: Asymmetric Cryptography	Communications between the panel and back-end server use RSA encryption. In some cases, dead-drop resolver content is encrypted with RSA.
	T1071.001	Application Layer Protocol: Web Protocols	Network communication in recent versions is done via a REST API over HTTPS.
	T1095	Non-Application Layer Protocol	Network communication is done with the WCF Framework over TCP.
	T1102.001	Web Service: Dead Drop Resolver	The RedLine panel uses GitHub repositories as dead-drop resolvers to obtain the address of back-end servers.
	T1571	Non-Standard Port	By default, the RedLine panel’s Guest Links functionality runs an HTTP server on port 7766.

Source: https://www.welivesecurity.com/en/eset-research/life-crooked-redline-analyzing-infamous-infostealers-backend/