Inside SiriusQT: A Seriously Skilled RAT

By Riccardo Di Pietro

Published: 2026-10-22 · Archived: 2026-11-22 02:01:08 UTC

Introduction

SiriusQT is advertised as a Windows RAT, but the analyzed sample in this article shows something more complex than a remote administration tool. Under the name, it offers several capabilities in a single implant: remote access, HVNC, browser automation, credential theft, cryptocurrency wallet collection, clipboard monitoring, keylogging, process injection, persistence, anti-analysis checks, and custom encrypted command-and-control traffic.

This article walks through SiriusQT's server- and client-side components, highlighting what makes it interesting from a reverse-engineering perspective. The goal is not only to list its features, but to show a complex cluster of affiliates, their campaigns and how these components work together and why SiriusQT should be treated as a serious RAT and stealer platform.

Key discoveries

- SiriusQT is a RAT found for sale on dark web forums since April 2026.
- Its author is named "Amonus", but the seller is "GhostKernel".
- At least 30 users bought a valid license.
- Licenses can be of different sizes: one month, three months, lifetime.
- Licenses can also be customized by the SiriusQT operator depending on customer requests. 14-day licenses, 45-day licenses, and also 99-day licenses were spotted during the analysis.
- Malicious campaigns were targeting, among others, the United Kingdom, Germany, France, Spain, Italy, and the Netherlands.
- Embeds Hidden Virtual Network Computing (HVNC) for victims' remote control.
- Strong stealing and keylogging capabilities are also oriented toward extracting data from cryptocurrency wallets and most-used programs.
- Multiple Process Injection Techniques.

Group-IB Threat Intelligence Portal:

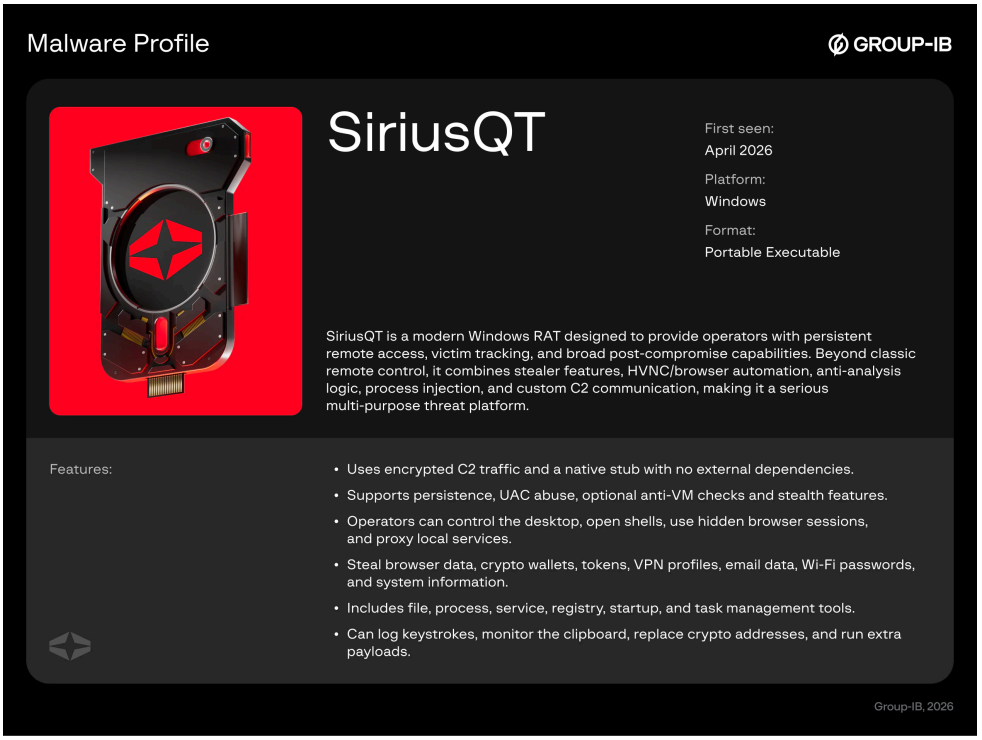
Group-IB customers can access our Threat Intelligence portal for more information about the threat actor and malware referenced in this blog:

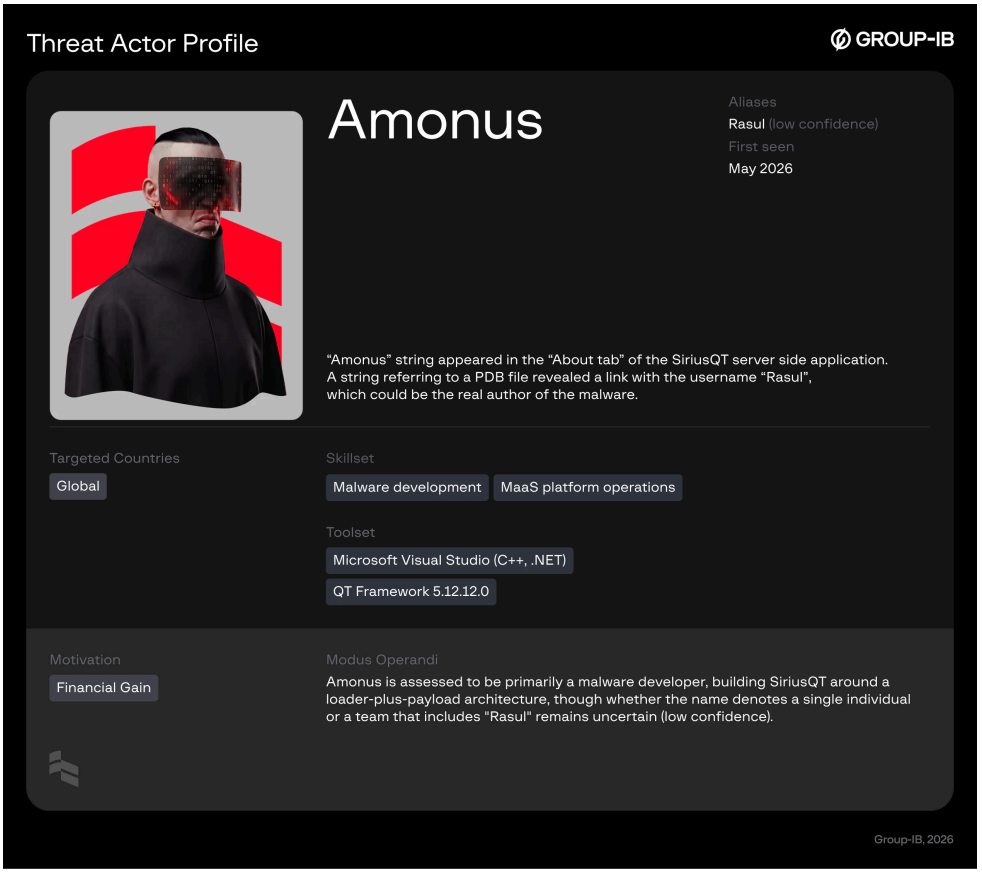
- SiriusQT Malware
- Amonus Threat Actor
- UpCrypter Malware

Public Malware Detonations

Access free detonation reports linked to SiriusQT and watch malware operate in real time through detailed insights from the Group-IB Malware Detonation Platform:

- SiriusQT Detonation Report





Reseller profile

Despite “Amonus” being referenced as the alleged author of SiriusQT RAT in the about form of the RAT server-side component, no direct evidence linking “Amonus” to the development, promotion, or distribution of the tool was found. The only clearly visible actor connected to the SiriusQT distribution is GhostKernel, a group active in both underground forums and Telegram.

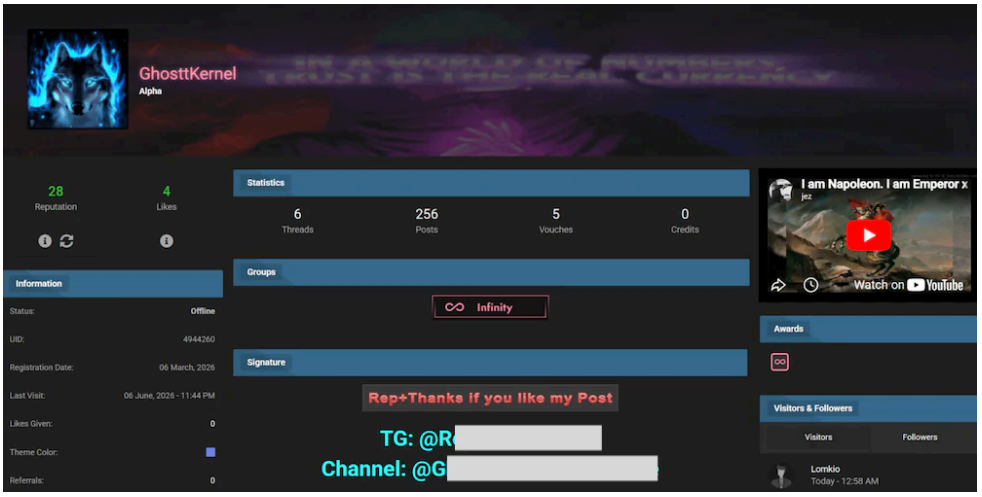


Figure 1. A reseller named GhostKernel is observed to be active on both underground forums and Telegram.

GhostKernel appears to be interested in stolen data, like credential lists, stealer logs, streaming accounts, and email databases. Its activities on Telegram started in February 2026, mainly for advertising the **BTMob v4.0**, an Android RAT with banking-related features. This suggests GhostKernel is a reseller interested in market diversification through various Windows and mobile malware.

The observed business model appears to be Malware-as-a-Service, with subscription access, customer support, and regular forum promotion, promising hassle-free setup for buyers.

Dark web distribution

Monitoring of the dark web messages shows an active distribution ecosystem around SiriusQT RAT. Most activity occurred on cracked[.]to and crax[.]pro, with a peak between April and May 2026.

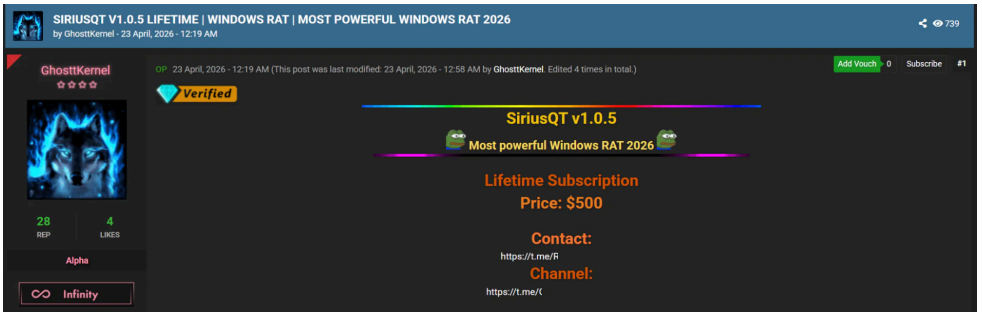


Figure 2. Example of a SiriusQT promotional forum post by GhostKernel.

In one observed forum post, GhostKernel advertises SiriusQT v1.0.5 as a Windows RAT and promotes it as a lifetime product priced at US$500. The actor also heavily promotes its Telegram accounts which are used for sales, support, and product updates.

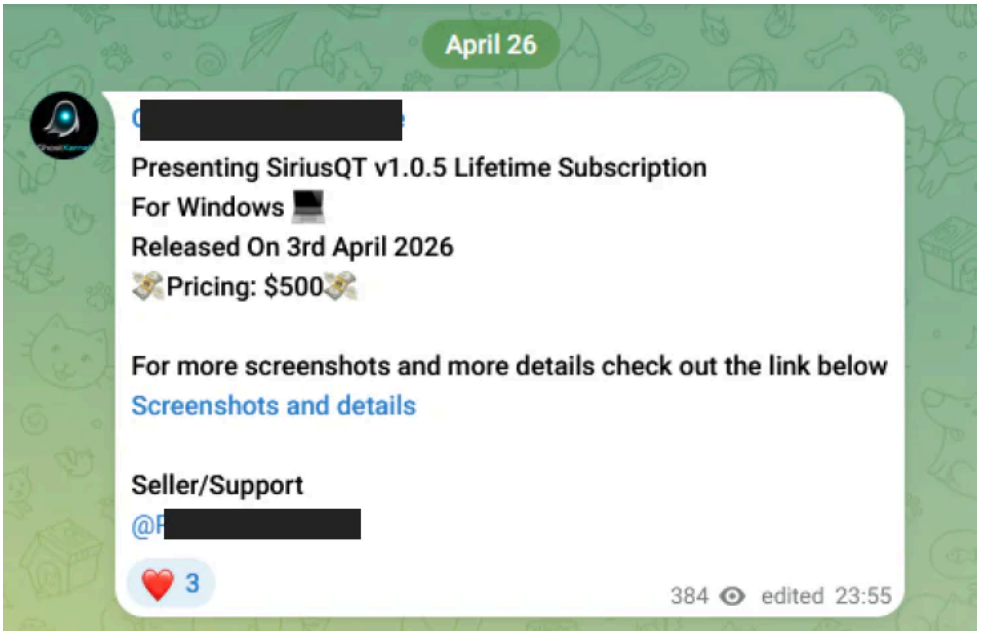


Figure 3. Example of a SiriusQT promotional Telegram post by GhostKernel.

In April 2026, the tool became even more popular, partly because secondary actors were observed to be sharing a cracked version for free.

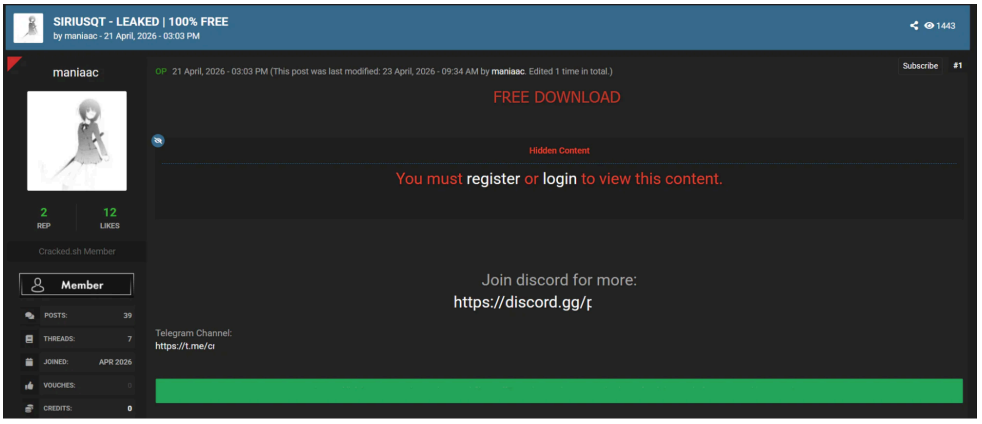


Figure 4. Example of SiriusQT being offered as cracked software in the wild.

On dark web forums and even GitHub repositories, many distributors of the SiriusQT cracked version were spotted.

This created two parallel markets for the same malware family: one for buyers who want the paid version with support and updates, and the other for lower-skilled or opportunistic users who want free cracked builds (sometimes also enriched with additional malware).

Even though GhostKernel has a dedicated Telegram channel for SiriusQT, and appears to be the only actor offering a paid version for sale, there is no evidence to qualify them as the original distributor. This view is further reinforced given that, on the forums where the sale of this RAT is promoted, all screenshots of the RAT running shows the window title “Sirius Cracked by Claude”.

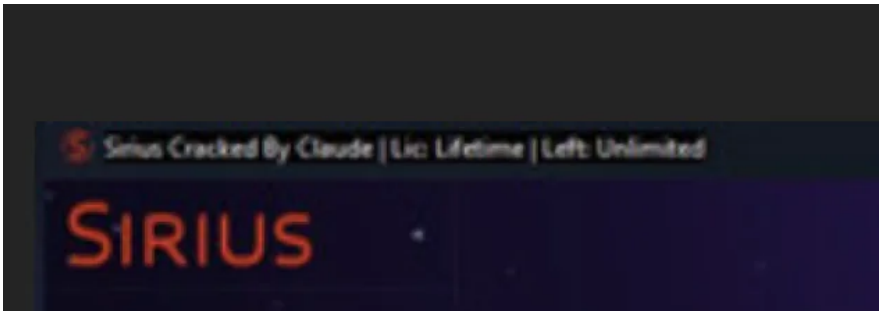


Figure 5. GhostKernel promotes SiriusQT by using cracked versions in screenshots.

License monitoring

Most non-legit versions came with a Python script that acts as a key generator or a lifetime key generated with the Python script.

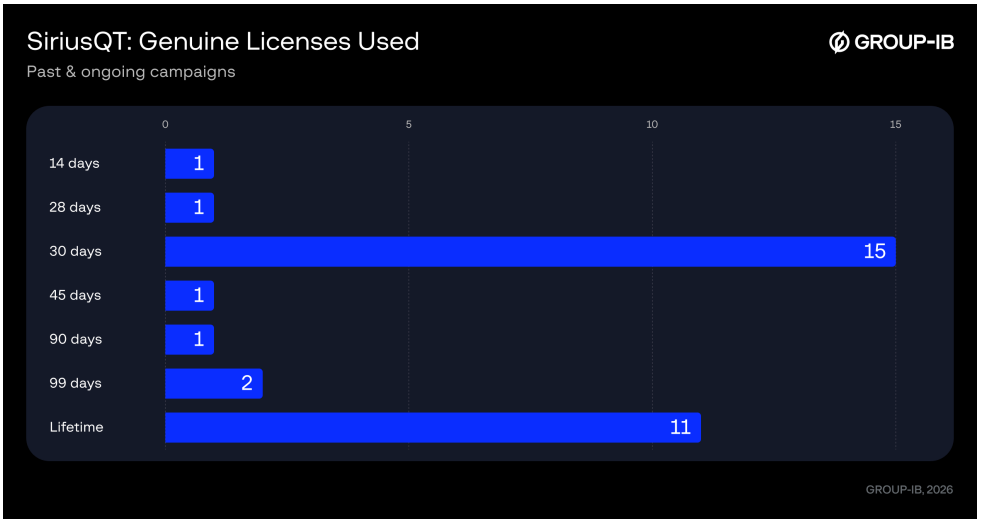
Later in this article, we will discuss how this malware stores all the information needed to work properly inside the configuration stub. At this stage, the key point is that the malware configuration includes the license used to generate the SiriusQT client-side component.

The SiriusQT server-side component requires a license in order to work. The license is made up of a 64-character array grouped by two, resulting in a 32-byte token. **This field is useful for clustering samples and tracking how the builder was used across different campaigns and operators.**



Figur 6. SiriusQT license structure.

In our dataset, we extracted 146 unique licenses. Out of these, 110 were identified as leaked licenses, which represents almost 75% of the observed license pool. The valid licenses show different durations, with 30-day and lifetime licenses being the most visible commercial tiers.



Figur 7. Distribution of SiriusQT license types currently in use.

The license durations are also interesting. Some tiers match the public commercial model, such as 30-day, 90-day, and lifetime licenses. However, other durations are less common, including 14, 28, 45, and 99 days. These unusual license

lengths may indicate private agreements between the threat actor and individual buyers.

The relevance of the additional data field is also supported by operator behavior observed in the wild. When an operator used a genuine license and that license expired, they did not simply continue with a keygen-generated license. Instead, they purchased a new genuine license.

One example is an actor who ran at least eight different campaigns between late March 2026 and early June 2026 using two different genuine licenses. After the first 30-day license expired, the actor purchased a new lifetime license approximately one week later:

SAMPLE FIRST SEEN	LICENSE	C2_ENDPOINT	GROUP TAG	LICENSE CREATED	LICENSE EXPIRY	COMM_KEY
2026-03-23 10:04:46	c23d****cfff	server[.]activeworkshops[.]com:16614	China – Justified Tools	2026-03-16 14:06:55 UTC	2026-04-13 14:06:55 UTC	0
2026-06-09 4:31:07	2a3f****c962	server[.]activeworkshops[.]com:16614	Mini Stub Mexico	2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-05-25 1:52:58	2a3f****c962	server[.]activeworkshops[.]com:16614		2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-05-21 15:34:35	2a3f****c962	server[.]activeworkshops[.]com:16614	Whatsapp	2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-05-19 11:37:26	2a3f****c962	server[.]activeworkshops[.]com:16614	UAE	2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-05-21 10:12:39	2a3f****c962	server[.]activeworkshops[.]com:16614	Whatsapp	2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-05-21 15:40:06	2a3f****c962	server[.]activeworkshops[.]com:16614		2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-05-19 14:33:32	2a3f****c962	server[.]activeworkshops[.]com:16614	EAST	2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-04-24 13:51:43	2a3f****c962	server[.]activeworkshops[.]com:31166	China + Taiwan	2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-05-21 15:40:01	2a3f****c962	server[.]activeworkshops[.]com:16614		2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-05-21	2a3f****c962	server[.]activeworkshops[.]com:16614	Whatsapp	2026-04-20	LIFETIME	a1a3****74e1

10:11:28				11:25:18 UTC		
2026-05-25 1:53:00	2a3f****c962	server[.]activeworkshops[.]com:16614	UAE	2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1
2026-04-24 13:44:15	2a3f****c962	server[.]activeworkshops[.]com:31166	China + Taiwan	2026-04-20 11:25:18 UTC	LIFETIME	a1a3****74e1

Two additional operators who purchased genuine licenses also appear to have been satisfied with the product, as both upgraded to lifetime licenses while their existing genuine licenses were still valid.

SAMPLE FIRST SEEN	LICENSE	C2_ENDPOINT	GROUP TAG	LICENSE CREATED	LICENSE EXPIRY	COMM_KEY
2026-06-06 5:04:06	ca8a****6285	2[.]27[.]62[.]140:4449	Chizzy	2026-05-24 20:40:32 UTC	LIFETIME	0d44****3d70
2026-05-21 11:03:26	c2e7****0369	2[.]27[.]62[.]140:4449	Spy	2026-05-20 21:24:11 UTC	2026-06-19 21:24:11 UTC	0d44****3d70
2026-06-03 12:30:25	ca8a****6285	2[.]27[.]62[.]140:4449	Chizzy	2026-05-24 20:40:32 UTC	LIFETIME	0d44****3d70

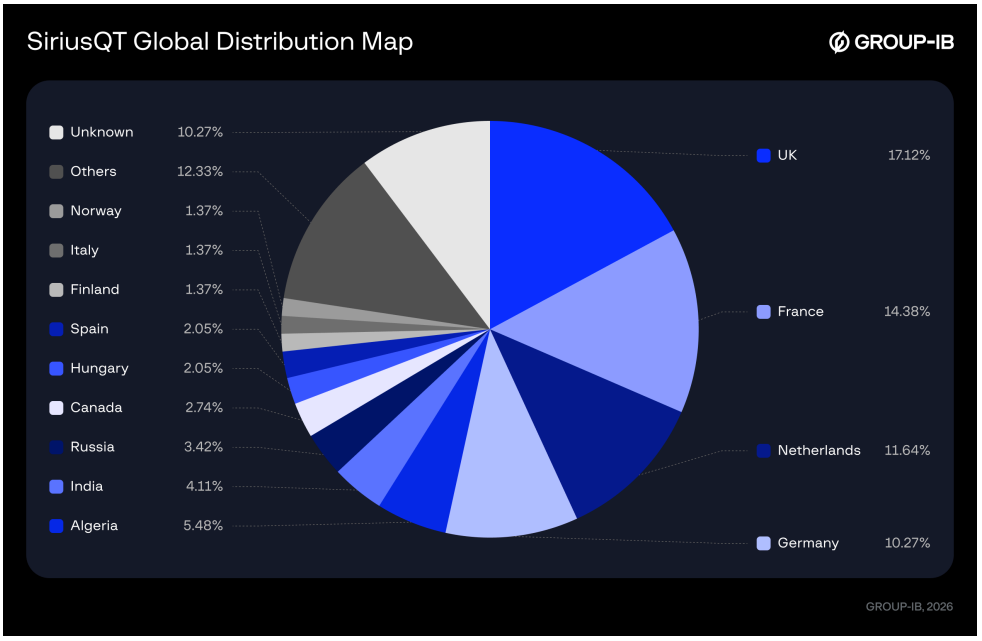
SAMPLE FIRST SEEN	LICENSE	C2_ENDPOINT	GROUP TAG	LICENSE CREATED	LICENSE EXPIRY	COMM_KEY
2026-05-01 20:14:24	8c32****5f96	serios[.]ydns[.]eu:4449	April-30_2026	2026-04-30 06:03:52 UTC	LIFETIME	eec4****ecc5
2026-03-27 9:49:25	f34a****e6b6	serios[.]ydns[.]eu:4449	Mar-27_2026	2026-03-26 05:16:22 UTC	2026-07-03 05:16:22 UTC	0

Moreover, the three different operators were using the same client-server traffic encryption key (COMM_KEY) for all their own campaigns, except for the very first campaigns where maybe no channel encryption was available, so this means that most likely they were using a non-leaked version of SiriusQT.

Worldwide distribution

By looking at publicly available samples, more than 280 SiriusQT clients related to various different campaigns were discovered.

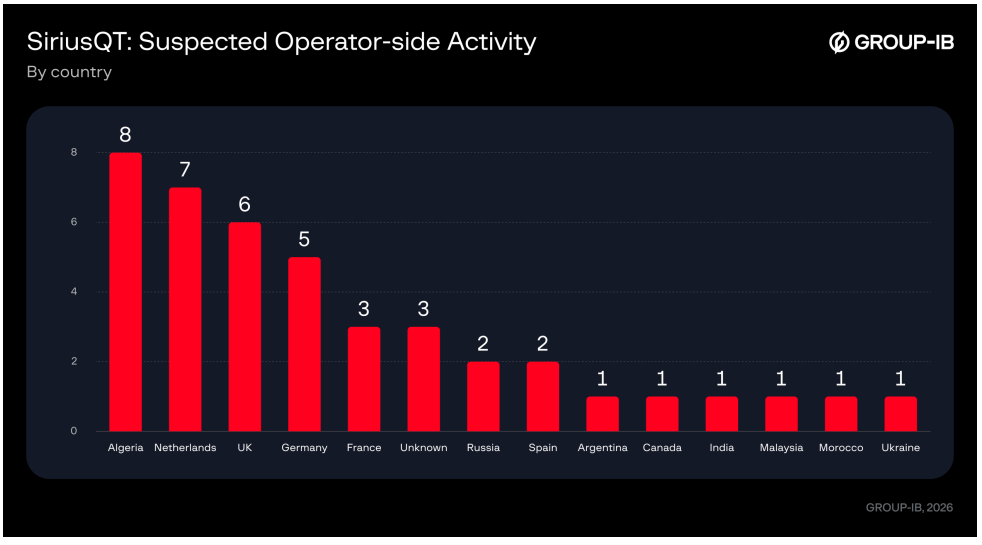
This distribution suggests that SiriusQT was not limited to a single regional market or operator group. By looking at the VirusTotal submissions, the UK, France, Germany, and the Netherlands represent the largest share of the dataset, with more than 50% of the observed locations.



Figur 8. Distribution of SiriusQT clients by VT public submissions.

This geolocation includes real compromised hosts, but also operator-side testing, sandbox executions, and infrastructure used to validate detection rates. This distinction matters because the analyzed SiriusQT builds have a relatively high detection rate. SiriusQT appears to be deployed as a last-stage payload, after the UpCrypter loader with very low detection or claimed FUD capability.

Unlike the previous geolocation chart, the following view is closer to operator telemetry than victim telemetry, because the related analyzed clients contain private IP addresses in their configuration.



Figur 9. Country distribution of SiriusQT samples related to operator tests.

The private IP address in the SiriusQT client configuration suggests that this sample was probably used for internal testing. The operator was likely checking that the implant worked correctly in a controlled local environment before using it against real targets.

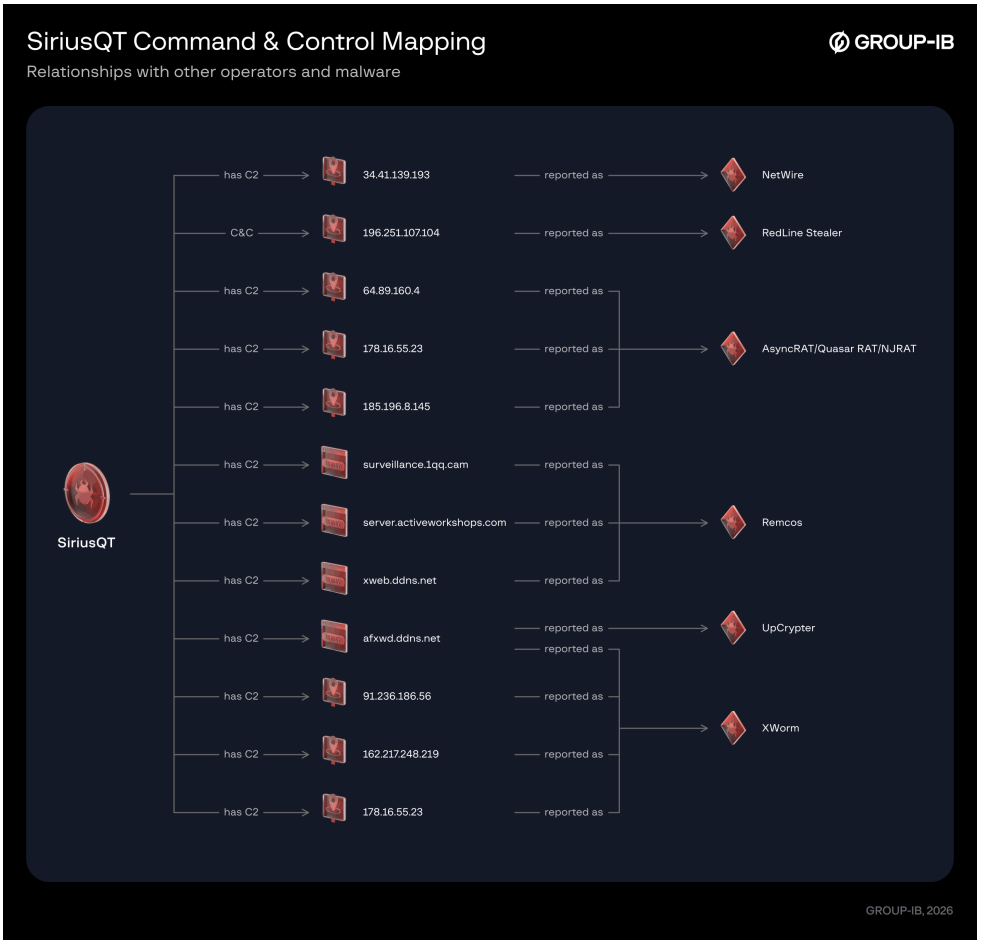
By looking at the name of the campaign ID field in the configuration, repeated values such as, “**Whatsapp**”, “**May bots**”, and “**Chizzy**” confirm that multiple RAT clients were generated from the same leaked version of the SiriusQT server.

Other values are more descriptive, like “**Mini Stub Mexico**”, “**UAE**”, “**China + Taiwan**”, and “**China – Justified Tools**”, which may indicate targeting, lure themes, operator notes, or internal campaign names, but they also could be an indicator of the operator’s intentions.

Date-based values such as “**Mar-27_2026**” and “**April-30_2026**” also suggest that the same operator used the campaign id to track a brief campaign. In this case, the same actor used at least two different genuine licenses and obtained the second license (which is lifetime) after having the first one already valid for 99 days. This could mean that the actor saw the potential of this RAT and decided to invest a bit more.

SiriusQT has been observed in relation to other malware families, such as **Remcos**, **XWorm**, **AsyncRAT** and **QuasarRAT/NJRAT**, since SiriusQT operators may rely on shared or reused infrastructure. One of the command&control found was previously linked with **RedLine Stealer**. **SiriusQT** was distributed as final-stage malware using the **UpCrypter** loader in May 2026 and the same infrastructure was used one year earlier to host another RAT campaign starting with the UpCrypter loader, Fortinet states.

The following graph shows the infrastructure links observed around SiriusQT analyzed samples.



Figur 10. SiriusQT Command&Control relationships with other operators and malware.

Among the command&control servers found in the analyzed SiriusQT samples, an operator is found abusing legitimate Google Cloud infrastructure, this makes detection harder since traffic to Google IP ranges is typically trusted and rarely blocked by corporate firewalls. The IP 34.41[.]139.193 belonging to Google Cloud resources has also been observed in past **NetWire** campaigns.

The server-side component

The server-side component of SiriusQT is SiriusQT.exe. When double-clicked for the first time, a legal disclaimer form appears, but many users don’t seem to heed the warning 😉 .

The main SiriusQT interface is organized as follows:

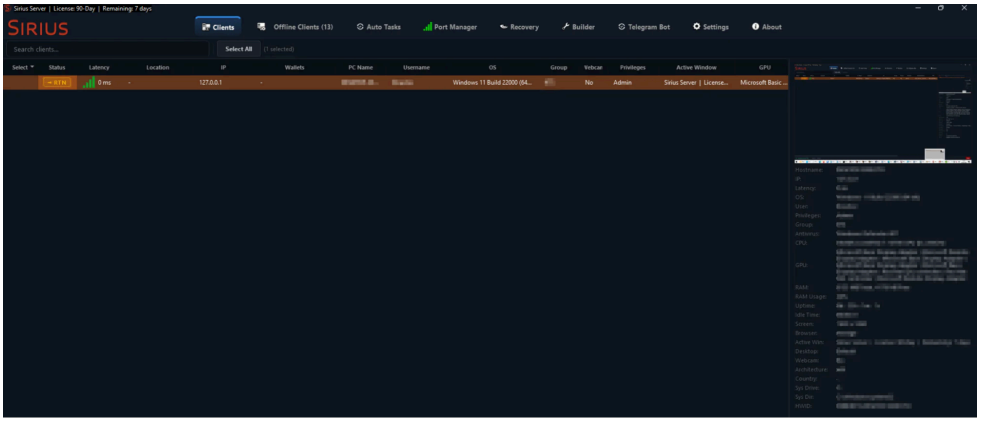


Figure 11. Evidence of the server-side component of SiriusQT.

IT comprises several main sections: Clients, Offline Clients, Auto Tasks, Port Manager, Recovery, Builder, Telegram Bot, Settings, and About.

The Clients tab shows the active infected hosts connected to the server. Each client entry includes fields with some fields as summary, such as status, latency, IP address, PC name, username, and operating system. Once the client entry is focused (after a mouse click), the right-side information panel displays additional host details, including hostname, OS, user, privileges, antivirus, CPU, GPU, RAM, uptime, idle time, screen, browser, active window, desktop, architecture, system drive, system directory, and HWID.

For each client entry, the same context menu exposes the following sections:

- The **Surveillance** menù entry groups features related to Remote Desktop, HVNC, Credential recovery, and traffic redirection.
- The **Managers** section is designed to inspect and manage files, registry keys, services, processes, scheduled tasks, network state, windows, clipboard content, policies, firewall settings, and storage resources on the remote machine.
- The **Miscellaneous** menu covers system information collection, command execution, file searching, Windows Defender exclusion management, webpage opening, script execution, and user-interaction features.
- The **Shell** submenu offers direct command execution through Remote CMD and Remote PowerShell. The Client submenu includes Client Folder, Power, Connection, and Client Details, which are used for client-side management and connection-related actions.

When a client connects for the first time, SiriusQT shows past clients collected. This information is shown in the Offline Clients tab:

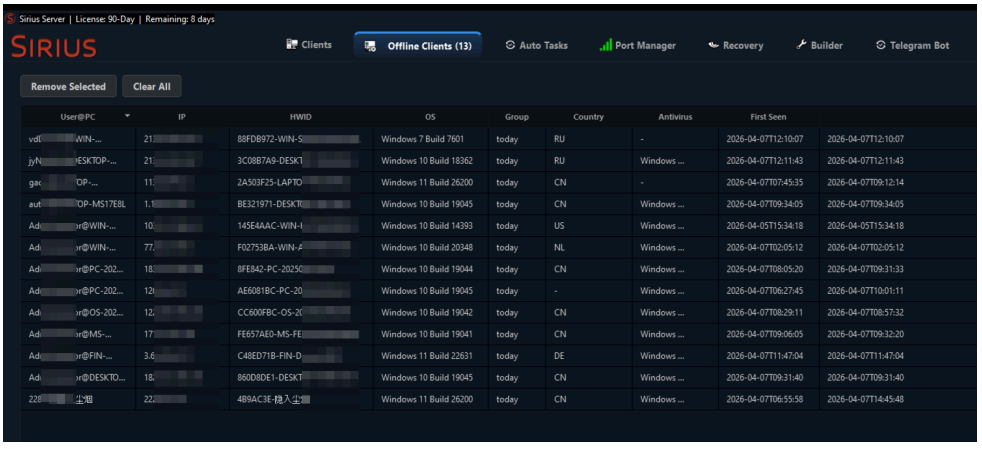


Figure 12. Offline client view in the server-side component of SiriusQT.

The past clients' information is retrieved from the **appdata.dat** file, inside the Tool-Data folder in the SiriusQT server package. The above evidence means that the actor who shared the leaked version of SiriusQT was conducting an active campaign with the group-ID named as "today," and different victims were from **Russia**, **China**, the **Netherlands**, the **USA,** and **Germany**.

This information is not in clear text inside the **appdata.dat** file, since the file structure follows this template:

```
+0x00 53 44 42 01                magic = "SDB\x01"
+0x04 00 00 00 00                4-bytes unused field
+0x08 ...                        XOR-obfuscated JSON payload
```

The following is evidence of the **appdata.dat** file in the leaked SiriusQT server-side component:

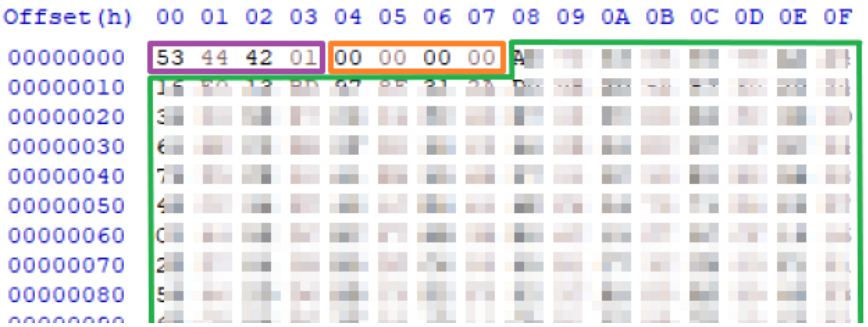


Figure 13. SiriusQT appdata.dat structure evidence.

SiriusQT uses a fixed 256-byte keystream for decrypting the **appdata.dat,** starting from mixing the following constants:

```
0xBF58476D1CE4E5B9
0x94D049BB133111EB
0x7BFFE77DDDAF7E93
0xF3FEAFBCEDDBFFAF
```

In the appendix, you'll find the script useful for decrypting **appdata.dat.**

After deobfuscation, a JSON is obtained containing the shown information and more:

```
{
"license": "...",
"enckey": "...",
"blocked": [],
"offlineClients": [],
"autoTasks": [],
"starredHwids": [],
"knownHwids": []
}
```

Among the others, the JSON contains juicy information like the license value, the client-server communication key, and the past connected clients, as in the previous evidence.

Another tab that is pretty interesting is the **Auto Tasks** tab that allows the operator to define automated tasks that can run when a client connects, run one time during the current server session, or run repeatedly based on an interval.

The **Recovery tab** shows a direct and fast view of the recovered data, like passwords, cookies, wallets, and files:

The **Builder** tab is used by the operator to configure a new SiriusQT client payload. Like many other malware, it supports endpoint fallback and has a lot of additional and optional features, like Anti-VM Detection, blocking execution if the locale belongs to CIS Regions (Armenia, Azerbaijan, Belarus, Kazakhstan, Kyrgyzstan, Moldova, Russia, Tajikistan, Uzbekistan, and Turkmenistan), encrypting the client, and also building an x86/64 .NET Loader option.

One important feature is the dropdown panel in the Builder tab labeled "Output Format". The SiriusQT Server-side component is able to build three different client formats: the x64 C++ (native), the .NET Loader x86 and the .NET Loader x64.

In the following paragraphs, they will be discussed.

Overall, this component is a lightweight .NET loader designed to extract an embedded payload from the file overlay and execute it directly from memory, avoiding the need to drop the decoded payload as a separate file on disk.

As the last tab, the **Telegram Bot** view provides bot-based management options. It includes fields for a bot token and notification channel ID, a polling interval setting, an option to enable the bot controller, and an option to auto-start the bot

when SiriusQT is being launched.

The client-side component

The client side of SiriusQT is pretty much not obfuscated; this also explains why there are so many detections from online security engines.

During malware execution and static analysis, several capabilities were identified. This section provides a structured overview of the observed behaviors and summarizes the main findings from the investigation.

Configuration

The resource named **Caffee_Config** contains the SiriusQT client configuration and is being loaded in the main function of the RAT using the FindResource and LoadResource APIs:

```
FUN_14019fac0(local_3c4c,0,0x3c00);
local_48c8 = 0x2e302e302e373231;
local_48c0 = 0x31;
local_3fbe = 0;
local_3ebd[0] = '\0';
local_3dbd[0] = '\0';
local_66e8 = (HMODULE)0x0;
GetModuleHandleExA(6,(LPCSTR)FUN_140100700,&local_66e8);
if ((((local_66e8 == (HMODULE)0x0) ||
     (hResInfo = FindResourceA(local_66e8,&DAT_0000007b,"Caffee_Config"), hResInfo == (HRSRC)0x0))
    || ((hResData = LoadResource(local_66e8,hResInfo), hResData == (HGLOBAL)0x0 ||
        ((pvVar11 = LockResource(hResData), pvVar11 == (LPVOID)0x0 ||
         (DVar3 = SizeofResource(local_66e8,hResInfo), DVar3 < 6)))))) ||
   (iVar2 = FUN_140100700(pvVar11,DVar3,&local_48c8), iVar2 == 0)) {
```

Figure 14. Reading the Caffee_Config resource in SiriusQT client.

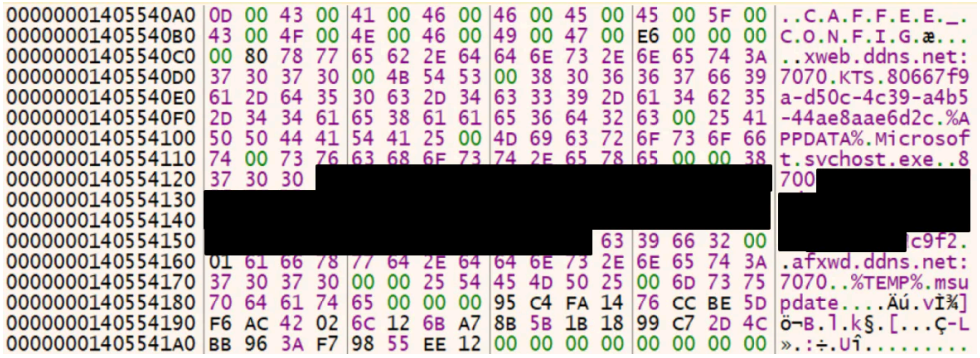


Figure 15. Inside the Caffee_Config resource in SiriusQT client.

If the external configuration resource cannot be loaded, the stub falls back to a 4096-byte embedded configuration blob identified by the marker `<<SIRIUS_STUB_CONFIG>>`. The same parser is used for both the internal resource and the fallback blob:

```
FUN_14019fac0(local_3c4c,0,0x3c00);
local_48c8 = 0x2e302e302e373231;
local_48c0 = 0x31;
local_3fbe = 0;
local_3ebd[0] = '\0';
local_3dbd[0] = '\0';
export_name_rva = uVar28;
do {
  *(char *)((longlong)local_58c8 + export_name_rva) =
       s_<<SIRIUS_STUB_CONFIG>>_1401f1460[export_name_rva];
  export_name_rva = export_name_rva + 1;
} while ((longlong)export_name_rva < 0x1000);
iVar2 = FUN_140100700(local_58c8,0x1000,&local_48c8);
if (iVar2 == 0) {
  return 0;
}
```

Figure 16. Alternate SiriusQT marker for loading the configuration.

During the reverse engineering activities has been possible to obtain the meaning of most of the configuration fields:

Index	Field	Name	Description
1	xweb[.]ddns[.]net:7070	PRIMARY_C2	Main Command&Control endpoint
2	KTS	GROUP_TAG	Campaign ID or Actor ID
3	80667f9a-d50c-4c39-a4b5-44ae8aae6d2c	MUTEX_NAME	Mutex name
4	%APPDATA%	INSTALL_BASE	Base setup directory
5	Microsoft	INSTALL_FOLDER	Base subfolder name
6	svchost.exe	INSTALL_FILENAME	Base file name
7	–	SCHEDULED_TASK_NAME	Scheduled Task name (if enabled)
8	8700***c9f2	LICENSE	License key
9	1	FALLBACK_COUNT	Number of fallback endpoints
10	afxxwd[.]ddns[.]net:7070	FALLBACK C2	Variable list of endpoints n-times in size the number in the previous field
11	–	STARTUP_FLAGS	Bitwise mask for persistence
12	%TEMP%	FALLBACK_BASE	Base setup directory as fallback
13	msupdate	FALLBACK_FOLDER	Base subfolder name as fallback
14	–	SHORTCUT_NAME	Shortcut name in Startup Folder
15	–	ENTRY_NAME	Entry name for Run Key / Logon Script / persistency
16	0x95, 0xC4 … 0xEE, 0x12	COMM_KEY	Symmetric client-server communication key

Mutex

Once the configuration is parsed, the malware ensures that no other copies of the RAT are running on the system by using the mutex. Mutex name “80667f9a-d50c-4c39-a4b5-44ae8aae6d2c” is got from the configuration resource and has been randomly generated by the SiriusQT server at the time of building:

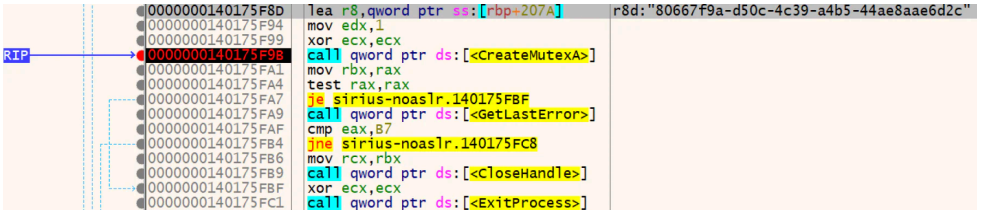


Figure 17. Mutex setup during execution of client version of SiriusQT.

Persistence

The SiriusQT RAT can be configured to start using different startup methods, including:

- a Registry Run key
- a Scheduled Task
- a shortcut in the Startup folder
- a logon script
- the BootExecute registry value (but requires the malware to have administrator privileges), which runs before normal user logon
- Windows Default User profile

These options are up to the operator to be activated during the stub building phase. They allow the SiriusQT to adapt to different environments and privilege levels.

These options are up to the operator to be activated during the stub building phase. They allow the SiriusQT to adapt to different environments and privilege levels.

1. If the **registry run key option** is enabled, the sample implements user-level persistence through the `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` registry. If no custom value name is provided, it uses the Windows Update name. This allows the payload to be launched automatically when the user logs in, without requiring any administrative privileges.
2. When the **schedule task option** is enabled the SiriusQT client side component builds an XML task definition and creates it using `schtasks.exe`. The generated task is configured with a logon trigger, intended to run when the user logs in. If no custom task name is provided, the malware uses the name `Windows Update.`
3. When the **Startup Folder option** is enabled, the RAT initializes COM interface and creates a Shell Link object. It sets the shortcut target to the configured payload path, assigns the description `Windows Update`, and then saves the shortcut as a .lnk file, using `WindowsUpdate` .lnk as the default name.
4. The RAT implements persistence through the `UserInitMprLogonScript` mechanism. When the **Logon Script option** is enabled, the RAT creates a batch script named `logon.cmd`, in the directory provided in the configuration stub. The script contains a command that starts the SiriusQT client side component. After creating the script, the malware opens `HKCU\Environment` and sets the `UserInitMprLogonScript` value to the path of the generated `logon.cmd` file. This causes the script, and therefore the RAT, to be executed when the user logs in. This is a user-level persistence technique and does not require administrative privileges.
5. When the **Boot Execute option** is enabled, the malware attempts to use an administrator-level persistence and more stealth method than normal user-level startup persistence. It first checks whether it is running with administrative privileges. If so, it opens the Session Manager registry key `HKLM\SYSTEM\CurrentControlSet\Control\Session Manager` and queries the existing `BootExecute` value. By appending an additional entry to it, the SiriusQT RAT can attempt to execute code during system startup.
6. When the SiriusQT has the **Default User persistence option** it poisons the Windows default User Profile. This is because Windows uses `C:\Users\Default` as a template when creating new user profiles. The file `C:\Users\Default\NTUSER.DAT` contains the registry hive that will be used as the base for the new user's HKEY_CURRENT_USER hive. The SiriusQT abuses this behavior by loading the Default User registry hive with RegLoadKeyA, modifying it, and then unloading it with RegUnLoadKeyA. While the hive is loaded, the malware writes an autorun-style registry value. If no custom value name is found in the configuration, the value is named Windows Update. The data written by the RAT uses %APPDATA%, this means the RAT attempts to seed persistence both in the current user profile and into future user accounts created on the same machine.

Dynamic API resolution

Once the SiriusQT client ends setting the persistence mechanism, it starts walking the PEB structure, enumerating module exports, and resolving the API using a custom 32bit FNV-1 hash algorithm.

```
do {
  module_base = puVar23[6];
  export_directory =
       (ulonglong)*(uint *)((longlong)*(int *)(module_base + 0x3c) + 0x88 + module_base) +
       module_base;
  if ((export_directory != module_base) &&
     (uVar21 = (ulonglong)*(uint *)(export_directory + 0x18),
     *(uint *)(export_directory + 0x18) != 0)) {
    do {
      uVar27 = (int)uVar21 - 1;
      uVar21 = (ulonglong)uVar27;
      seed = 0x7158b20f;
      export_name_rva =
           (ulonglong)
           *(uint *)(module_base + uVar21 * 4 + (ulonglong)*(uint *)(export_directory + 0x20));
      current_char = *(char *)(export_name_rva + module_base);
      export_name_ptr = (char *)(export_name_rva + 1 + module_base);
      if (current_char != '\0') {
        do {
          uVar20 = (uint)current_char;
          current_char = *export_name_ptr;
          export_name_ptr = export_name_ptr + 1;
          seed = (uVar20 ^ seed) * 0x1000193;
        } while (current_char != '\0');
        if (seed == 0x62a6c3f7) {
          (*(code *)((ulonglong)
                     *(uint *)((ulonglong)*(uint *)(export_directory + 0x1c) + module_base +
                               (ulonglong)
                               *(ushort *)
                                ((ulonglong)*(uint *)(export_directory + 0x24) + module_base +
                                uVar21 * 2) * 4) + module_base))
                    (0,0,FUN_140156f90,0,(ulonglong)pcVar13 & 0xffffffff00000000,0);
          goto LAB_140177a00;
        }
```

Figure 18. FNV-1 hash algorithm in SiriusQT client side component.

In the above evidence, the seed is 0x7158B20F and the target hash is 0x62A6C3F7, which resolves to CreateThread API. Another function was found using the same algorithm but with the seed 0x29f1276c and resolving the 0x706ECA3E hash, corresponding to LoadLibraryA.

A python script is provided for fast hash computation:

```
def sirius_api_hash(api_name: str) -> int:

h = 0x7158B20F

fnv_prime = 0x01000193




for b in api_name.encode("ascii"):

h ^= b

h = (h * fnv_prime) & 0xFFFFFFFF

return h

target = "CreateThread"

hash_value = sirius_api_hash(target)

print(f"{target} -> 0x{hash_value:08X}")
```

The client dynamically resolves LoadLibraryA and GetProcAddress through PEB, using the same FNV-1a like hashing algorithm, loads iphlpapi.dll, and obtains the address of NotifyAddrChange API. This allows the main C2 server to be automatically reconnected after the network stops working or the internal IP address changes.

Payload execution

SiriusQT also includes multiple ways of doing process injection and in-memory payload execution. It references APIs and techniques such as WriteProcessMemory and VirtualProtectEx. In addition, it resolves dynamically API like NtQueryInformationProcess and NtQueueApcThread.

The SiriusQT server-side component let choose between Process Hollowing and different injection techniques like QueueUserAPC / Early-Bird Injection and Remote Thread Injection using NtCreateTreadEx or CreateRemoteThread APIs, designed to run both EXE and DLL x32/x64., available on the client-side component.

The payload execution form also allows choosing the running target process on the client:

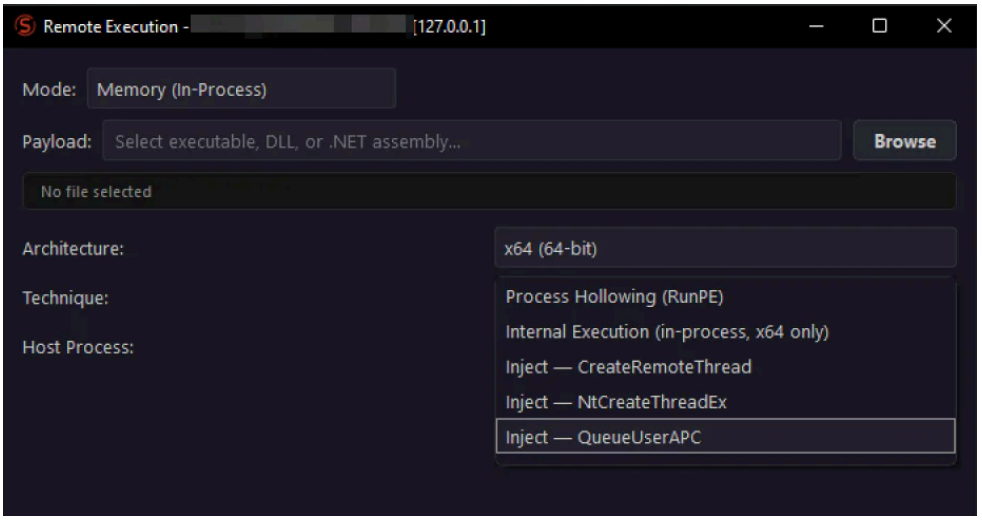


Figure 19. Remote execution option panel in SiriusQT server-side component.

HVNC

Like many modern RATs, SiriusQT RAT provides an HVNC feature. HVNC, or Hidden VNC, gives the attacker a remote graphical session inside a hidden desktop that the victim does not normally see.

The HVNC protocol between the client and the server is discussed in the Command and Control section.

Before that HVNC session starts, the SiriusQT server-side component delivers an additional DLL module to the SiriusQT client, named **HVNCInjection.x64.dll**. It supports the hidden session by installing hooks, redirecting selected file-system paths, and injecting itself into child processes created during the HVNC session.

The **HVNCInjection.x64.dll** module has the following SHA-256 hash:

- c206fa613257110b0522b60b402ca0b35da2cb126113e47b67543ba3ff93c53c

This DLL contains a reflective loading mechanism. This means that the **HVNCInjection.x64.dll** could be loaded directly from memory instead of being loaded from disk by the Windows loader.

The reflective loader locates the DLL image in memory, validates the PE headers, allocates memory for the new image, copies headers and sections, resolves imports, applies relocations, and finally calls the DLL entry point.

Inside the reflective loader stage, the loader dynamically resolves Windows APIs. It walks the loaded module list from the PEB and computes a ROR13-based hash over module names. For example, the comparison with the hash value 0x6A4ABC5B identifies **kernel32.dll** library.

```
for (PEB->Ldr = *(undefined8 **)(*(longlong *)((longlong)ProcessEnvironmentBlock + 0x18) + 0x20);
    PEB->Ldr != (undefined8 *)0x0; PEB->Ldr = (undefined8 *)*PEB->Ldr) {
  sVar12 = 1;
  hash = 0;
  pbVar9 = (byte *)PEB->Ldr[10];
  Blink = *(short *)(PEB->Ldr + 9);
  do {
    hash = hash >> 0xd | hash << 0x13;
    if (0x60 < *pbVar9) {
      hash = hash - 0x20;
    }
    hash = hash + *pbVar9;
    pbVar9 = pbVar9 + 1;
    Blink = Blink + -1;
  } while (Blink != 0);
  if (hash == 0x6a4abc5b) {
    lVar20 = PEB->Ldr[4];
```

Figure 20. ROR13 based hash algorithm in HVNC helper module named HVNCInjection.x64.dll.

The **HVNCInjection.x64.dll** DLL uses MinHook to install userland hooks. MinHook is a small user-mode API hooking library for Windows. It allows the SiriusQT client-side component to intercept API calls, forcing the execution first to the DLL's custom handler. The handler can inspect and modify the parameters, then call the original API through a trampoline created by MinHook.

When the DLL is loaded the following APIs are hooked:

- NtCreateFile
- NtOpenFile
- NtDeleteFile
- NtSetInformationFile
- NtQueryAttributesFile
- NtQueryFullAttributesFile

This means the DLL can redirect file creation, file opening, file deletion, rename or move operations, and file attribute queries.

The DLL also installs hooks on:

- NtQueryDirectoryFile
- NtQueryDirectoryFileEx

However, for these two last APIs the handler does not modify the returned directory listing, it simply calls the original function, so the handler acts like a wrapper.

The DLL also provides a way to do filesystem path redirection based on two environment variables RDI_SEARCH_PATH and RDI_REPLACE_PATH.

When a process tries to access a file path, the hook checks whether that path matches the value configured in RDI_SEARCH_PATH. If it does, the DLL changes the path by using the value configured in RDI_REPLACE_PATH. The process believes it is accessing the original path, but the real Windows API receives the replaced path.

For example, if the two environment variables are configured like the following:

```
RDI_SEARCH_PATH = C:\Users\Admin\Desktop
RDI_REPLACE_PATH = C:\Users\Admin\AppData\Local\Temp\HiddenDesktop
```

then a request for accessing the `"C:\Users\Admin\Desktop\file.txt"`, can be redirected to the `"C:\Users\Admin\AppData\Local\Temp\HiddenDesktop\file.txt"`.

This mechanism is transparent to the process that is trying to access the `"C:\Users\Admin\Desktop\file.txt"`, but the real API call uses the redirected path once the handler modified it.

Using HVNC, the SiriusQT operator can interact with applications installed on the client machine, while selected file paths are transparently redirected to a separate location. This can help isolate the HVNC environment from the victim's real desktop or profile paths, depending if the two path variables are enabled and how they are configured.

The SiriusQT client embeds the code for launching a Chromium-based browser instance for the HVNC module:

```
FUN_1400dc720(local_1008,0x1000,0xffffffffffffffff,
              "\"%s\" --user-data-dir=\"%s\" \"data:text/html,<center><title>Welcome to HVNC !</ti
              tle><br><br><br><h1><font color=\'white\'>Welcome to HVNC !</font></h1></center>\" -
              -no-sandbox --allow-no-sandbox-job --disable-3d-apis --disable-accelerated-layers --
              disable-accelerated-plugins --disable-audio --disable-gpu --disable-d3d11 --disable-
              accelerated-2d-canvas --disable-deadline-scheduling --disable-ui-deadline-scheduling
               --aura-no-shadows --mute-audio"
              ,pCVar3,local_1d78);
local_1e98 = 0;
local_1ea8 = (HANDLE)0x0;
pvStack_1ea0 = (HANDLE)0x0;
```

Figure 21. Command line for launching the chromium based browser in the SiriusQT client.

It builds a dedicated browser profile path, checks whether the profile directory exists, and prepares it if needed. The browser is then started with a custom `“--user-data-dir”` and an inline `“data:text/html”` page displaying `“Welcome to HVNC !”`. The command line also is intended for disabling sandboxing, GPU acceleration, audio, plugins, 3D APIs, and other rendering features, just like for doing as little noise as possible to run a HVNC environment.

When the operator executes the browser app (for instance Edge) in the HVNC session, he can also clone the browser profile resulting in complete client session stealing:

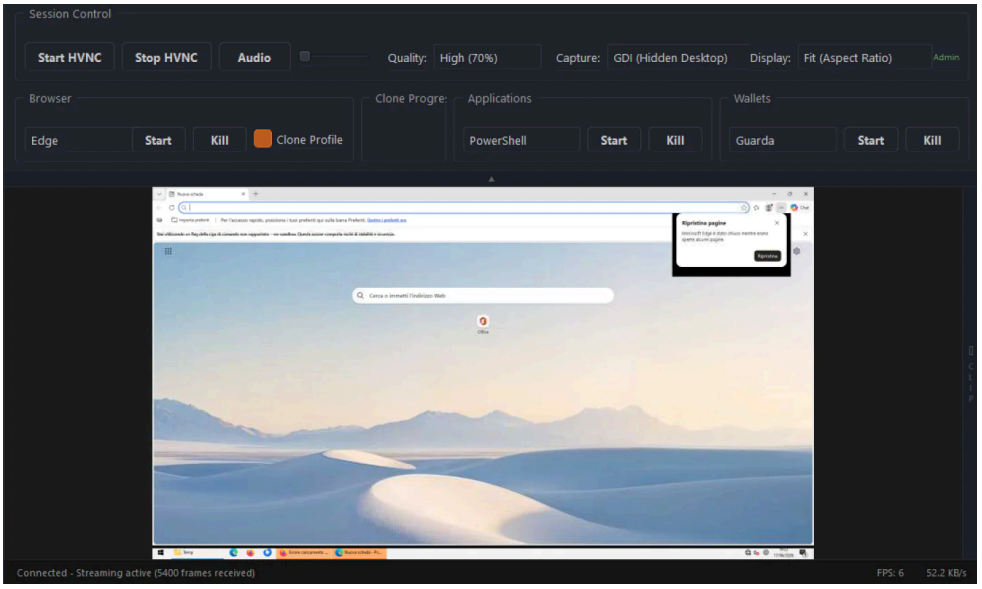


Figure 22. HVNC panel in SiriusQT server-side component.

```
uVar1 = FUN_18000155c(param_2,param_3);
if (uVar1 == 0) {
  string = "[ChildInject] FAIL: ReflectiveLoader export not found in DLL";
}
else {
  lpBaseAddress = VirtualAllocEx(param_1,(LPVOID)0x0,(ulonglong)param_3,0x3000,0x40);
  if (lpBaseAddress == (LPVOID)0x0) {
    string = "[ChildInject] FAIL: VirtualAllocEx failed";
  }
  else {
    BVar2 = WriteProcessMemory(param_1,lpBaseAddress,param_2,(ulonglong)param_3,&local_res20);
    if (BVar2 == 0) {
      VirtualFreeEx(param_1,lpBaseAddress,0,0x8000);
      string = "[ChildInject] FAIL: WriteProcessMemory failed";
    }
    else {
      hHandle = CreateRemoteThread(param_1,(LPSECURITY_ATTRIBUTES)0x0,0x100000,
                                   (LPTHREAD_START_ROUTINE)
                                   ((longlong)lpBaseAddress + (ulonglong)uVar1),(LPVOID)0x0,0,
                                   (LPDWORD)0x0);
      if (hHandle != (HANDLE)0x0) {
        DVar3 = WaitForSingleObject(hHandle,30000);
        string = "[ChildInject] WARN: loader thread timed out (30s)";
        if (DVar3 != 0x102) {
          string = "[ChildInject] OK: child injection completed";
        }
        FUN_180001498(string);
        CloseHandle(hHandle);
        return 1;
      }
      string = "[ChildInject] FAIL: CreateRemoteThread failed";
    }
  }
}
```

Figure 23. Child injection in the HVNC helper module named HVNCInjection.x64.dll.

The code works in the following steps:

1. finds the **ReflectiveLoader** export inside the DLL blob
2. allocates memory in the child process with VirtualAllocEx
3. writes the DLL blob into the child process with WriteProcessMemory
4. starts a remote thread with CreateRemoteThread passing the remote address of **ReflectiveLoader** as the thread start address (v11 variable)

Despite the presence of such described implants, during our analysis no evidence of hooking or path redirection was seen. **The HVNCInjection.x64.dll appears to be loaded but not used.**

Stealing activities

The Recovery window on the SiriusQT server-side component is a credential collection panel. It allows the operator to select specific data sources to target, such as cryptocurrency wallets, messengers, social applications, browsers, VPN clients, games, password managers, cloud storage, developer tools, RDP, and mail clients, among more than 100 different checkboxes:

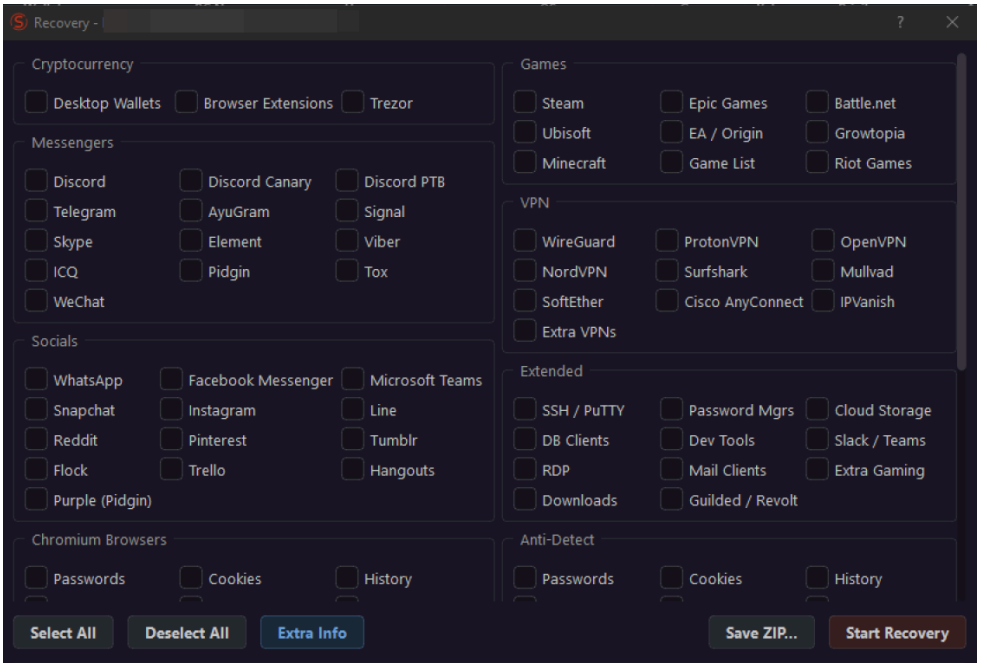


Figure 24. Credential recovery interface in SiriusQT server-side component.

The credential-recovery functionality can handle modern Chromium secrets, including AES-GCM-protected passwords. For example the following snipped is able to restore encrypted Chromium/Electron Discord tokens marked by the "`dQw4w9WgXcQ:`" prefix:

```
NVar9 = BCryptOpenAlgorithmProvider(&local_e50,L"AES",(LPCWSTR)0x0,0);
if (NVar9 == 0) {
  NVar9 = BCryptSetProperty(local_e50,L"ChainingMode",(PUCHAR)L"ChainingModeGCM",
                            0x20,0);
  if ((NVar9 == 0) &&
     (NVar9 = BCryptGenerateSymmetricKey
                        (local_e50,local_e48,(PUCHAR)0x0,0,local_res20,param_5,0),
      NVar9 == 0)) {
    local_e58[0] = 0;
    UVar1 = DVar15 - 0x1c;
    local_de8 = 0;
    pUStack_e10 = pbBinary + (longlong)(int)UVar1 + 0xc;
    local_e18 = 0;
    uStack_e20 = 0;
    uStack_e00 = 0;
    local_df8 = 0;
    uStack_df0 = 0;
    local_e38 = 0x100000058;
    local_e28 = 0xc;
    local_e08 = 0x10;
    pBStack_e30 = pbBinary;
    NVar9 = BCryptDecrypt(local_e48[0],pbBinary + 0xc,UVar1,&local_e38,(PUCHAR)0x0
                          ,0,local_438,UVar1,local_e58,0);
    if ((NVar9 == 0) && ((local_e58[0] != 0 && ((int)local_e58[0] < 0x200)))) {
      uVar14 = (ulonglong)local_e58[0];
      FUN_14019fe80(local_a38,local_438,local_e58[0]);
```

Figure 25. Code snippet to handle AES-GCM (Advanced Encryption Standard in Galois/Counter Mode) using ChainingMode.

The SiriusQT client doesn't just steal but also checks whether the stolen tokens are still valid. It's the case for Discord related credentials: the RAT has a custom function to verify stolen credentials by sending requests directly to a Discord endpoint using a custom "Mozilla/5.0" User-Agent:

```
uVar12 = (undefined4)((ulonglong)in_stack_fffffffffffd958 >> 0x20);
uVar11 = (uint)((ulonglong)in_stack_fffffffffffd948 >> 0x20);
FUN_14019fac0(param_2,0,0x2e2);
strncpy(param_2,param_1,0xff);
iVar9 = 0;
iVar10 = 0;
param_2[0x2e1] = '\0';
lVar3 = WinHttpOpen(L"Mozilla/5.0",0,0,0,(ulonglong)uVar11 << 0x20);
if (lVar3 == 0) {
  return 0;
}
lVar4 = WinHttpConnect(lVar3,L"discord.com",0x1bb,0);
if (lVar4 == 0) {
  uVar5 = WinHttpCloseHandle(lVar3);
}
else {
  uVar11 = 0;
  lVar6 = WinHttpOpenRequest(lVar4,&DAT_1401c2660,L"/api/v9/users/@me",0,0,0,
                             CONCAT44(uVar12,0x800000));
  if (lVar6 != 0) {
    iVar2 = MultiByteToWideChar(0xfde9,0,param_1,-1,(LPWSTR)0x0,0);
    if (iVar2 - 1U < 299) {
      lpWideCharStr = local_2688;
      MultiByteToWideChar(0xfde9,0,param_1,-1,lpWideCharStr,300);
      wsprintfW(local_2428,L"Authorization: %s");
      iVar2 = WinHttpSendRequest(lVar6,local_2428,0xffffffff,0,
                                 (ulonglong)lpWideCharStr & 0xffffffff00000000,
                                 (ulonglong)uVar11 << 0x20,0);
      if ((iVar2 != 0) && (iVar2 = WinHttpReceiveResponse(lVar6,0), iVar2 != 0)) {
```

Figure 26. Code snippet for verifying leaked Discord accounts by SiriusQT client.

Clipboard monitoring and keylogging

Clipboard monitoring logic periodically opens the Windows clipboard and checks whether Unicode text is available. If clipboard text is present, the routine locks the clipboard handle, converts the wide-character string to UTF-8 using code page 65001, closes the clipboard, and passes the resulting string to a pattern-matching routine.

```
if (((14999 < (uint)(param_2 - *(int *)(param_1 + 0x30))) &&
    (1999 < (uint)(param_2 - _DAT_140284f8c))) &&
   (_DAT_140284f8c = param_2, BVar5 = OpenClipboard((HWND)0x0), BVar5 != 0)) {
  BVar5 = IsClipboardFormatAvailable(0xd);
  if (((BVar5 == 0) || (pvVar7 = GetClipboardData(0xd), pvVar7 == (HANDLE)0x0)) ||
     (lpWideCharStr = GlobalLock(pvVar7), lpWideCharStr == (LPCWSTR)0x0)) {
    CloseClipboard();
    return;
  }
  lVar10 = -1;
  iVar6 = WideCharToMultiByte(0xfde9,0,lpWideCharStr,-1,local_818,0x7ff,(LPCSTR)0x0,(LPBOOL)0x0);
  GlobalUnlock(pvVar7);
  CloseClipboard();
  if (1 < iVar6) {
    puVar3 = *(undefined1 **)(param_1 + 0x18);
    uVar2 = *(undefined4 *)(param_1 + 0x20);
    local_818[iVar6] = '\0';
    puVar9 = &DAT_1401ad4cd;
    if (puVar3 != (undefined1 *)0x0) {
      puVar9 = puVar3;
    }
    lVar8 = FUN_1400fc1f0(local_818,puVar9,uVar2);
    if (lVar8 != 0) {
      FUN_1400dc660(local_a18,0x200,"Clipboard content matched pattern \'%s\'",lVar8);
      if ((DAT_1401f2480 != -1) && (DAT_14048e9f8 != (code *)0x0)) {
```

Figure 27. Clipboard pattern matching functionality in SiriusQT client.

A user activity monitoring loop for a specified window is also available. It installs a low-level keyboard hook using SetWindowsHookExA with hook type WH_KEYBOARD_LL (0xD), then keeps a Windows message loop active through

PeekMessageW, TranslateMessage, and DispatchMessageW APIs.

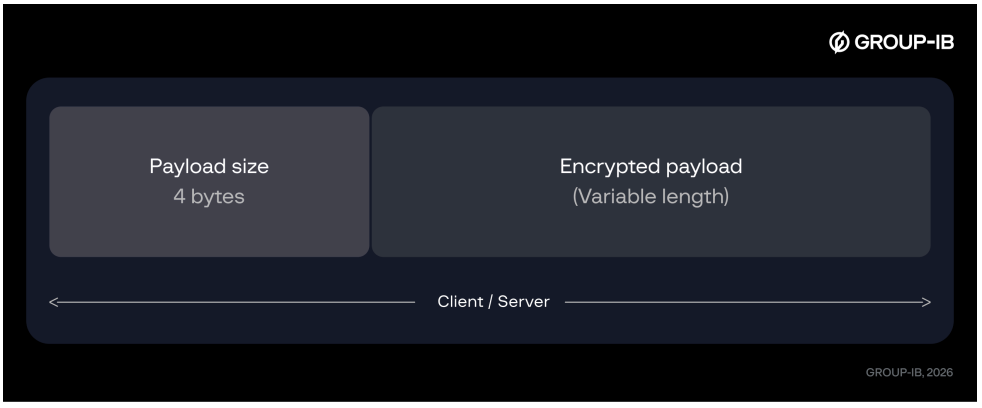
The routine periodically retrieves the current Windows foreground window and stores its title.

SiriusQT is also able to match the active window title against configured patterns. Once the logging activity is done, it removes the hook.

As well as many others RAT it also pretty prints special buttons for proper keylogging activity.

Command & Control

The SiriusQT RAT communication channel is encrypted at the payload level. The protocol uses a simple TCP framing format where each message starts with a 4-byte little-endian payload length. This length field is transmitted in cleartext and is used only to determine how many bytes must be read from the TCP stream. The following payload bytes are encrypted.



Figur 28. SiriusQT Client – Server encrypted communication channel format.

The same frame structure is used in both directions. Only the payload is encrypted.

SiriusQT does not appear to use a standard encryption algorithm such as ChaCha20 or Salsa20 for its network traffic, even though the binary contains the string “expand 32-byte k”.

The SiriusQT communication is encrypted using a custom stream-cipher-like routine for payload transport and receiving data.

While looking at the SiriusQT client-side component, the function at 0x1400D82C0 receives a cipher context, a buffer, and a buffer size, then modifies the buffer in place.

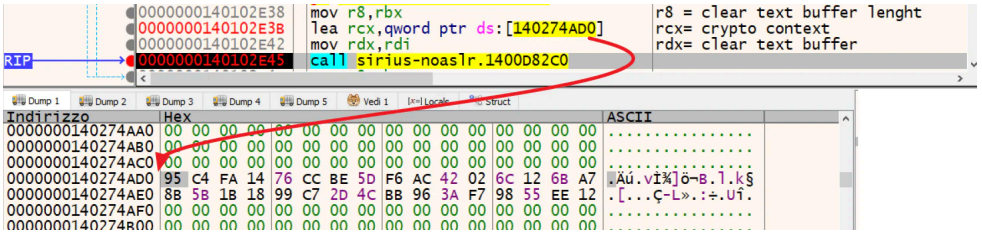


Figure 29. SiriusQT Client – Server encrypting clear-text buffer.

The cipher context is made up of:

- the 32-byte communication key stored in the configuration
- the current 64-bit packet counter

For each payload, the 64-bit counter is read from the context and then incremented:

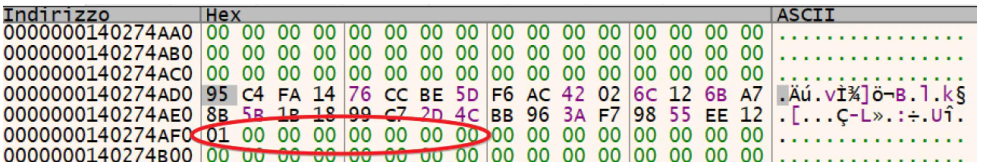


Figure 30. Cipher context for clear-text buffer encryption.

The encryption routine combines the counter, the current payload offset, and values from the cipher context to derive temporary 64-bit values.

These values are then mixed using two constants associated with the **MurmurHash3** fmix64 finalizer algorithm:

```
0xFF51AFD7ED558CCD
0xC4CEB9FE1A85EC53
```

The resulting mixed values are used to build a 16-byte keystream block. This keystream block is then XORed with the cleartext buffer.

Below is the Python implementation for decrypting messages between the SiriusQT client and server:

```
import re

MASK64 = 0xFFFFFFFFFFFFFFFF
C1 = 0xFF51AFD7ED558CCD
C2 = 0xC4CEB9FE1A85EC53

CTX_HEX = """
95 C4 FA 14 76 CC BE 5D F6 AC 42 02 6C 12 6B A7
8B 5B 1B 18 99 C7 2D 4C BB 96 3A F7 98 55 EE 12
"""

PAYLOAD_HEX = """
AA BB CC ...
"""

COUNTER = 0
SKIP_CLEAR = 0      # put 8 if the first 8 bytes are not to be transformed


def hx(s):
    return bytes(int(x, 16) for x in re.findall(r'(?> 33) ^ x) * C1) & MASK64
    y = (((x >> 33) ^ x) * C2) & MASK64
    z = ((y >> 33) ^ y) & MASK64
    return z, y


def init_v8(counter, k0):
    hi = u64(k0[8:16])

    v = (((counter >> 56) & 0xff) ^ k0[7]) & 0xff
    for shift, kb, hshift in [
        (48, 6, 48),
        (40, 5, 40),
        (32, 4, 32),
        (24, 3, 24),
        (16, 2, 16),
        (8,  1, 8),
    ]:
        v = ((v << 8) | ((((counter >> shift) & 0xff) ^ k0[kb]) | ((hi >> hshift) & 0xff))) & MASK64

    v = (
        (v << 8)
        | ((counter & 0xff) ^ k0[0])
        | (hi & 0xff00000000000000)
        | (hi & 0xff)
    ) & MASK64

    return v


def init_v9(k1):
    lo = u64(k1[:8])
    hi = u64(k1[8:16])
    return (((hi | lo) & 0xffffffffffffff00) | (hi & 0xff) | (lo & 0xff)) & MASK64
```

```
def transform(data, ctx, counter, skip_clear=0):
    k0 = ctx[:16]
    k1 = ctx[16:32]

    v2 = counter & MASK64
    v8 = init_v8(v2, k0)
    v9 = init_v9(k1)

    out = bytearray(data)
    off = 0
    size = len(out) - skip_clear

    while off < size:
        v8 = (((off + v2) & MASK64) ^ v9 ^ v8) & MASK64
        v8, t1 = fmix(v8)

        block = bytearray(16)
        block[0:4] = (v8 & 0xffffffff).to_bytes(4, "little")
        block[4:8] = ((t1 >> 32) & 0xffffffff).to_bytes(4, "little")

        v9, t2 = fmix((v9 ^ v8) & MASK64)
        block[8:12] = (v9 & 0xffffffff).to_bytes(4, "little")
        block[12:16] = ((t2 >> 32) & 0xffffffff).to_bytes(4, "little")

        n = min(16, size - off)

        for i in range(n):
            out[skip_clear + off + i] ^= block[i]

        off += n

    return bytes(out)


def decrypt_payload_hex(payload_hex, ctx_hex, counter, skip_clear=0):
    return transform(
        hx(payload_hex),
        hx(ctx_hex),
        counter,
        skip_clear
    )

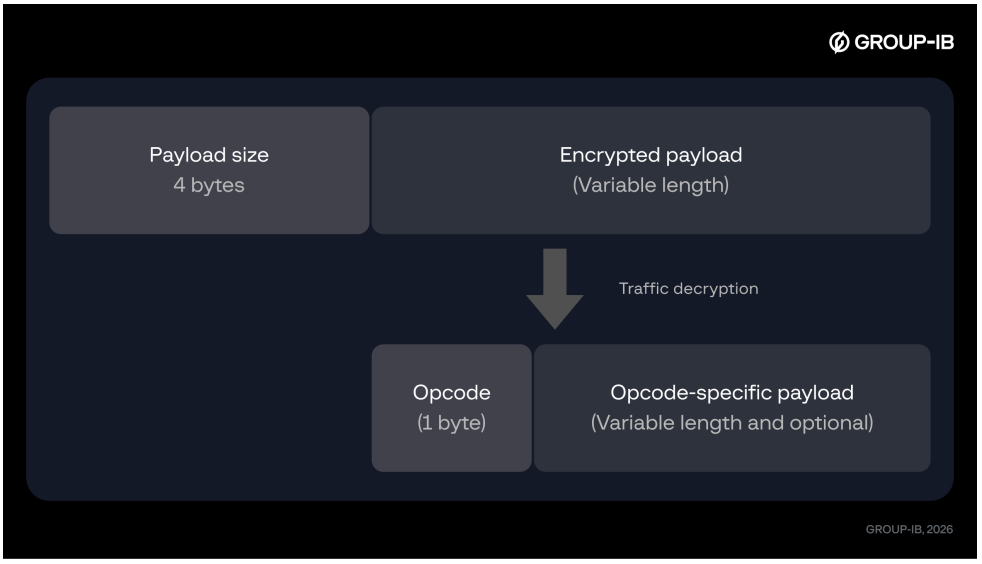

if __name__ == "__main__":
    plain = decrypt_payload_hex(
        PAYLOAD_HEX,
        CTX_HEX,
        COUNTER,
        SKIP_CLEAR
    )

    print("HEX:")
    print(plain.hex(" "))

    print("\nTEXT:")
    print(plain.decode("utf-8", errors="replace"))
```

The embedded communication key is also useful for tracking and clustering SiriusQT samples. Because the key is stored inside the generated client configuration and also resides in the SiriusQT server-side component folder, in the "**connection.key**" file, samples using the same key can often be linked to the same leaked server and so even the builder instance. Reused keys across otherwise different samples may indicate non-legitimate SiriusQT server usage.

After decryption, the payload becomes:



Figur 31. SiriusQT Client – Server command structure after traffic decryption.

All the commands sent by the SiriusQT server-side component are one byte in size and for many of the commands there is no additional payload. More complex commands use 1 byte opcode as well but also use an opcode-specific body format for sending custom structures or embedded PE files.

For example, the simplest packet with one byte payload size is the one with the opcode 0x1, which is a kind of heartbeat or keepalive.

When a new client connects to the SiriusQT server-side component, the 0x2 opcode is used to ask the client to send the server data for victim host registration:

```
Indirizzo         Hex                                             ASCII
00000000009D9F80  02 17 00 [illegible]                            ...[illegible]
00000000009D9F90  [illegible]                                     [illegible]
00000000009D9FA0  [illegible]                                     [illegible]
00000000009D9FB0  20 42 75 69 6C 64 20 31 39 30 35 30 28 00 49 6E  Build 19050(.In
00000000009D9FC0  74 65 6C 28 52 29 20 43 6F 72 65 28 54 4D 29 20  tel(R) Core(TM)
00000000009D9FD0  69 37 2D 31 30 37 30 30 20 43 50 55 20 40 20 32  i7-10700 CPU @ 2
00000000009D9FE0  2E 39 30 47 48 7A 89 00 4D 69 63 72 6F 73 6F 66  .90GHz..Microsof
00000000009D9FF0  74 20 42 61 73 69 63 20 44 69 73 70 6C 61 79 20  t Basic Display
00000000009DA000  41 64 61 70 74 65 72 20 7C 20 52 65 64 20 48 61  Adapter | Red Ha
00000000009DA010  74 20 56 69 72 74 49 4F 20 47 50 55 20 44 4F 44  t VirtIO GPU DOD
00000000009DA020  20 63 6F 6E 74 72 6F 6C 6C 65 72 20 7C 20 4D 69   controller | Mi
00000000009DA030  63 72 6F 73 6F 66 74 20 52 65 6D 6F 74 65 20 44  crosoft Remote D
00000000009DA040  69 73 70 6C 61 79 20 41 64 61 70 74 65 72 20 7C  isplay Adapter |
00000000009DA050  20 4D 69 63 72 6F 73 6F 66 74 20 52 65 6D 6F 74   Microsoft Remot
00000000009DA060  65 20 44 69 73 70 6C 61 79 20 41 64 61 70 74 65  e Display Adapte
00000000009DA070  72 1B 00 38 31 38 35 20 4D 42 20 54 6F 74 61 6C  r..8185 MB Total
00000000009DA080  2C 20 35 36 38 30 20 4D 42 20 46 72 65 65 03 00  , 5680 MB Free..
00000000009DA090  4B 54 53 05 00 41 64 6D 69 6E 07 00 44 65 66 61  KTS..Admin..Defa
00000000009DA0A0  75 6C 74 6A 00 73 69 72 69 75 73 2D 6E 6F 61 73  ultj.sirius-noas
```

Figure 32. Host registration information structure in SiriusQT client.

The SiriusQT client outbound message after receiving the 0x2 opcode from the server, contains a full host registration profile used to give the threat actor a summary of the compromised host.

It includes the following information:

- 4 bytes HWID generated by the RAT
- Machine name
- Username
- operating system version and build
- CPU model
- display adapters
- memory information
- Group-Identification
- privilege level,
- current process,
- active window context
- hostname,

- current system32 directory
- screen resolution
- Architecture
- detected security product
- uptime values

Also the 0x2 opcode is an update poll or frame request that is sent to the client every time the operator at the SiriusQT server-side component switches the focus to that client.

After receiving this 0x02 packet, the client replies with a client-to-server packet using opcode 0x03, that carries the client-side update data and JPEG screenshot blob.

Client-to-server traffic uses the same framing model but different opcode semantics.

Below is a list of the most important server-to-client opcodes found:

Opcode	Message format	Functionality	Description
0x01	Opcode only	Heartbeat	The client replies with 0x01 opcode
0x02	Opcode only	Update poll / frame request	The client replies with 0x03 opcode, carrying a JPEG update blob
0x10	Opcode + payload	Remote Desktop configuration / start / update	Payload starts with GDI\x00 followed by configuration bytes
0x11	Opcode only	Remote Desktop stop/control or monitor-control	Observed as a one-byte command during Remote Desktop activity
0x12	Opcode only	Remote Desktop init / reset	Observed during Remote Desktop setup sequence
0x17	Opcode only	Storage Manager	Open the Storage Manager module
0x20	Opcode + payload	Remote Shell start	Payload contains the interpreter name, such as cmd or powershell ASCII string
0x21	Opcode + payload	Remote Shell input	Payload contains command input sent to the remote shell, such as “whoami\n”
0x22	Opcode only	Remote Shell stop	Close the remote shell session
0x26	Opcode only	Firewall Manager	Open the Firewall Manager module
0x30	Opcode only	Clipboard Manager	Open the Clipboard Manager module
0x33	Opcode only	Remote Clipper start	Open the remote clipper component
0x34	Opcode only	Remote Clipper stop	Close the remote clipper component
0x38	Opcode + payload	Credential Recovery	Payload is a 16-byte bitmask based on the selected programs
0x39	Opcode only	Policy Manager	Open the Policy Manager module
0x47	Opcode + payload	HVNC module transfer	Payload is embedded PE file starting
0x4C	Opcode only	Remote Camera	Open the Remote Camera module
0x4F	Opcode only	Remote Microphone	Open the Remote Microphone module
0x52	Opcode only	Remote Speaker	Open the Remote Speaker module
0x60	Opcode only	File Manager	Open the File Manager module
0x80	Opcode + payload	Registry Manager	Payload has 2 bytes set to zero when started
0x90	Opcode only	Startup Manager	Open the Startup Manager module

0x93	Opcode only	Services Manager	Open the Services Manager module
0x9B	Opcode only	Installed Programs Manager	Open the Installed Programs module
0xA0	Opcode only	Process Manager	Open the Process Manager module
0xA8	Opcode only	Task Manager	Open the task-management module
0xB0	Opcode only	Network Manager	Open the Network Manager module
0xC2	Opcode only	Activity Monitor	Open the Activity Monitor module
0xD4	Opcode + payload	PE execution module	The payload contains a target path length, optional target path, and embedded PE payload
0xD5	Opcode only	Window Manager	Open the Window Manager module
0xE0	Opcode only	HVNC initialization	Open the HVNC module
0xE1	Opcode only	HVNC stop	Close the HVNC module
0xE2	Opcode +payload	HVNC operator runtime command	Payload contains an ASCII string containing application commands
0xEB	Opcode only	HVNC start	Sent after HVNC module initialization
0xF5	Opcode only	Reverse Proxy start	Open the reverse proxy component
0xF6	Opcode only	Reverse Proxy stop	Close the reverse proxy component
0xFA	Opcode + payload	Hidden RDP start	Payload contains configuration data as ASCII NUL-separated strings
0xFB	Opcode only	Hidden RDP stop	Close the Hidden RDP component.

As can be seen from the table some opcodes use a structured body instead of a single-byte command.

The Remote Shell module uses three related opcodes:

1. Opcode 0x20 starts a shell process and carries the interpreter name as an ASCII string, such as cmd or powershell. So payload size can be 4 or 11 bytes:

```
+0x00  20                opcode = 0x20
+0x01  63 6D 64          "cmd"
```

```
+0x00  20    opcode = 0x20
+0x01  70 6F 77 65 72 73 68 65 6C 6C  "powershell"
```

2. Opcode 0x21 sends input to the active remote shell. The body contains the command text as ASCII. For example, the command whoami\n is encoded as:

```
+0x00  21                    opcode = 0x21
+0x01  77 68 6F 61 6D 69 0A  "whoami\n"
```

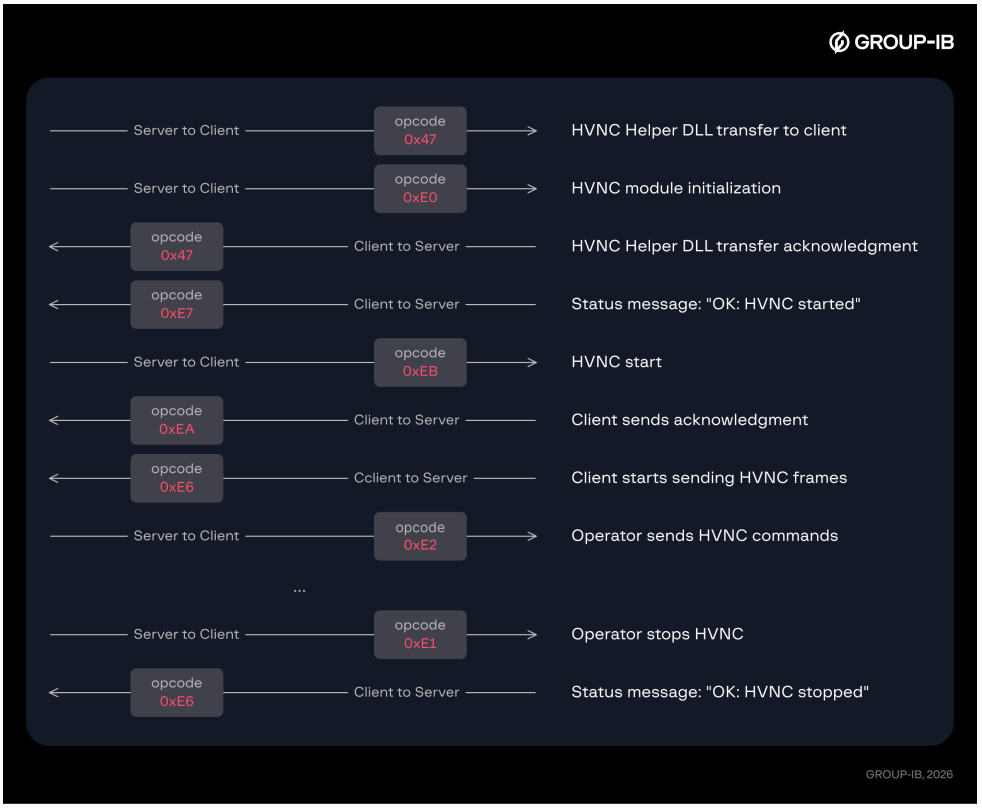
3. Opcode 0x22 closes the remote shell session, without carrying any additional body

The Password Recovery module uses opcode 0x38. Its body is a 16-byte little-endian bitmask. Each bit corresponds to a specific target application or credential source selected in the controller UI. For example, consider the following:

Target	Payload
WhatsApp	**38** 00 00 00 00 00 00 00 00 00 00 **20** 00 00 00 00 00
Facebook Messenger	**38** 00 00 00 00 00 00 00 00 00 00 **40** 00 00 00 00 00
Microsoft Teams	**38** 00 00 00 00 00 00 00 00 00 00 **80** 00 00 00 00 00
Snapchat	**38** 00 00 00 00 00 00 00 00 00 00 **01** 00 00 00 00

Instagram	**38** 00 00 00 00 00 00 00 00 00 00 00 **02** 00 00 00 00
Line	**38** 00 00 00 00 00 00 00 00 00 00 00 **04** 00 00 00 00
Reddit	**38** 00 00 00 00 00 00 00 00 00 00 00 **08** 00 00 00 00
Pinterest	**38** 00 00 00 00 00 00 00 00 00 00 00 **10** 00 00 00 00
Tumblr	**38** 00 00 00 00 00 00 00 00 00 00 00 **20** 00 00 00 00
Flock	**38** 00 00 00 00 00 00 00 00 00 00 00 **40** 00 00 00 00
Trello	**38** 00 00 00 00 00 00 00 00 00 00 00 **80** 00 00 00 00
Hangouts	**38** 00 00 00 00 00 00 00 00 00 00 00 **01** 00 00 00
Purple / Pidgin	**38** 00 00 00 00 00 00 00 00 00 00 00 **02** 00 00 00

The HVNC module uses a dedicated opcode flow rather than a single standalone command. The observed sequence is the following:



Figur 33. SiriusQT client-server dialogue during HVNC session.

The HVNC session starts with the server sending the opcode 0x47 with the HVNC PE/module payload. Immediately after that, the server sends the opcode 0xE0 with the value 0x46 as a payload

```
+0x00  E0                          opcode = 0xE0
+0x01  46                          payload = 0x46
```

After receiving the HVNC module and initialization command, the client sends the server the 0x47 acknowledgement opcode. Again the client sends the server the 0xE7 opcode with the ASCII message “OK: HVNC started”:

```
+0x00  E7                          opcode = 0xE0
+0x01  10 00                       string length = 0x0010
+0x03  4F 4B 3A 20 48 56 ...       "OK: HVNC started"
```

Once the module is ready, the server sends the opcode 0xEB.without additional payload to act as the HVNC start command.

The client replies with opcode 0xEA with one byte 01 acknowledgement:

```
+0x00  EA                           opcode = 0xEA
+0x01  01                           payload = 0x01
```

After this point, the client begins sending packets starting with opcode 0xE6.

The 0xE6 opcode is used by the client to send visual frame updates to the server. The packet contains a tile-based update structure. The header includes the screen resolution, the number of updated tiles, and the size of the following encoded data stream.

```
+0x00  E6                           opcode = 0xE6
+0x01  10                           update_type = 0x10
+0x02  80 07                        screen_width = 0x0780 = 1920
+0x04  38 04                        screen_height = 0x0438 = 1080
+0x06  15 00                        tile_count = 0x0015 = 21
+0x08  C4 72 00 00                  tile data size (29380 bytes)
+0x0C  C0 05 40 00                  tile[0]  x=0x05C0, y=0x0040
+0x10  00 06 40 00                  tile[1]  x=0x0600, y=0x0040
+0x14  40 06 40 00                  tile[2]  x=0x0640, y=0x0040
...
+0x5C  40 07 C0 00                  tile[20] x=0x0740, y=0x00C0
+0x60  76 B0 10 EE ...              encoded tile data
```

A **tile** is a portion of the client HVNC screen. The client does not always send the entire screen. This depends on the update_type byte, value 0x11 means full screen update, while 0x10 means partial screen update. In this HVNC session, the screen is divided into a grid of fixed-size tiles and the client sends only the tiles that changed. Since the tile table in the payload contains pairs of X-Y coordinates that are aligned to 0x40 (which is 64), that means the frame is built on a **64×64 tile grid**.

The remaining bytes contain the encoded frame data for those tiles.

Operator runtime interaction with the HVNC session is performed through the 0xE2 opcode. This opcode requires a payload variable in size, encoded like ASCII string not NUL terminated and containing command and arguments splitted by the “*” character:

```
***...
```

For example when the operator moves the mouse on the remote HVNC, the resulting command would be the following one:

```
+0x00  E2                           opcode = 0xE2
+0x01  38 2A 36 31 33 2A 31 30 2A 30  "8*613*10*0"
```

This would mean the server is telling the client “Move the hidden desktop mouse pointer to coordinate 613, 10”

Other HVNC events related to server operator mouse activity are the following:

```
2*x*y = mouse button down on coordinate_x, coordinate_y
4*x*y = mouse button up on coordinate_x, coordinate_y
```

To stop the HVNC session, the server uses opcode 0xE1. After the server sends the client a one byte packet with 0xE1 opcode, the client replies with another 0xE7 opcode starting message, with the following payload “OK: HVNC stopped”

This indicates that opcode 0xE7 is used by the client to report HVNC status messages back to the SiriusQT server side component.

Another important action that the threat actor could do is to **execute a payload on the client**. This is accomplished by sending the 0xD4 opcode. The payload starts with a 16-bit little-endian target path length. If the length is non-zero, it is followed by the target process path. The remaining bytes contain an embedded PE file.

```
+0x00 D4                            opcode = 0xD4
+0x01 39 00                         target_path_length = 57 bytes
```

```
+0x03 43 3A 5C 57 69 6E 64 6F ...       “powershell.exe” path
+0x3C 4D 5A 90 00 ...                   embedded_PE_payload
```

If the execution is fine, the client replies with the 0xD4 opcode with the message “OK: PE injected successfully” as a payload, confirming successful PE injection.

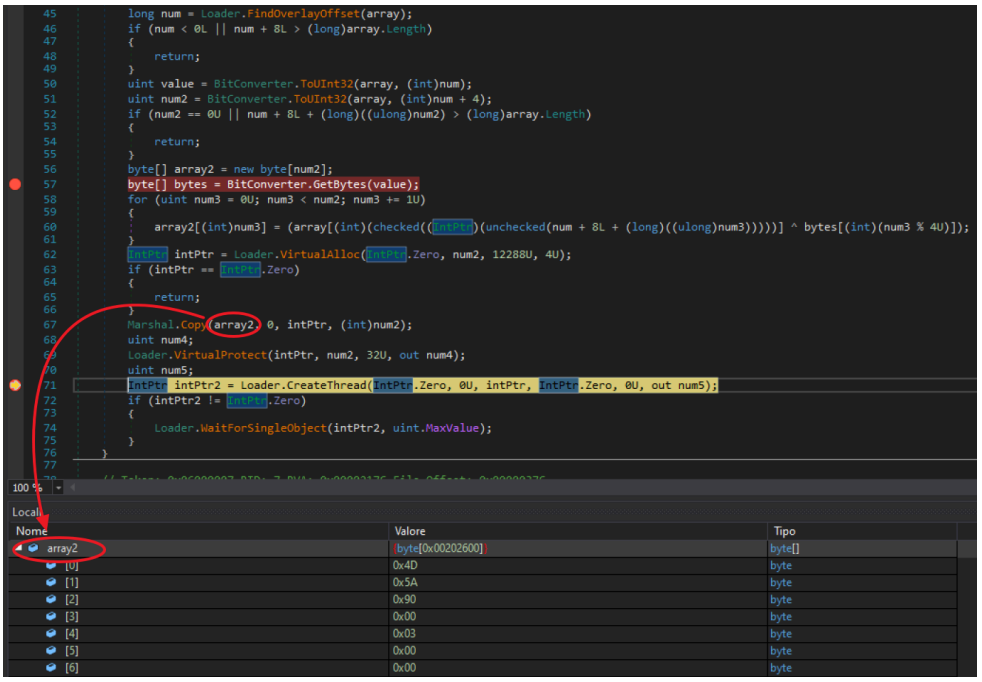
Different client formats

When dealing with the **Builder** view of SiriusQT, three different output formats were available to the operator to be used.

Besides the 64-byte C++ payload discussed in this blog, also the .NET Loaders, both 32 and 64 bytes, are worth mentioning.

When an operator chooses to produce a SiriusQT client compatible with .NET structure, the client is still x64 based, but packed.

In fact the .Net loader reads its own executable from disk using Assembly.GetExecutingAssembly().Location, then locates the appended overlay data with FindOverlayOffset(). The overlay starts with an 8-byte header containing a 32-bit XOR key followed by a 32-bit encoded payload size. The remaining overlay bytes contain the encoded payload.



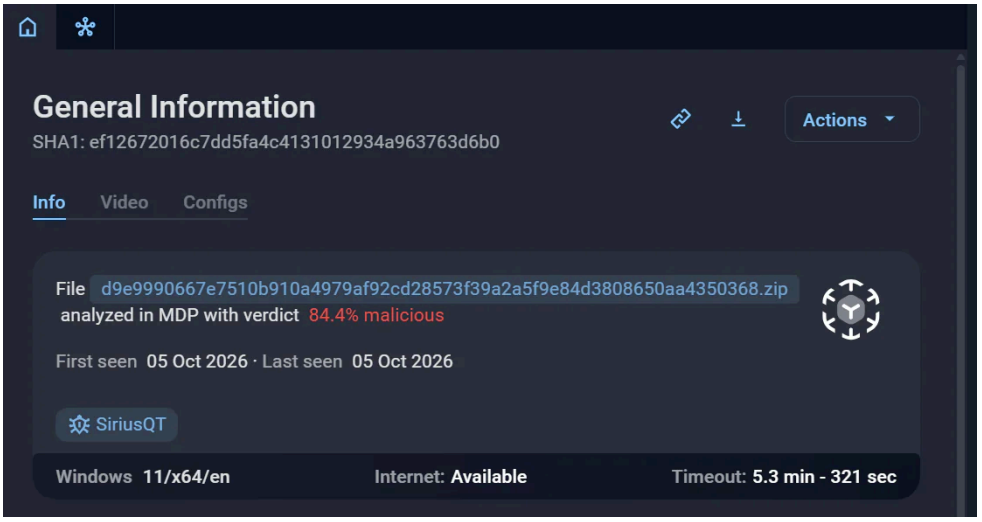
Figur 34. SiriusQT .NET Loader just after final payload decryption.

The SiriusQT client payload is decoded by XORing each byte with the 4-byte key repeated across the encoded payload.

Then the .NET loader allocates memory with VirtualAlloc, copies the decoded payload into the allocated region and starts execution using CreateThread API, avoiding the need to drop the decoded payload as a separate file on disk.

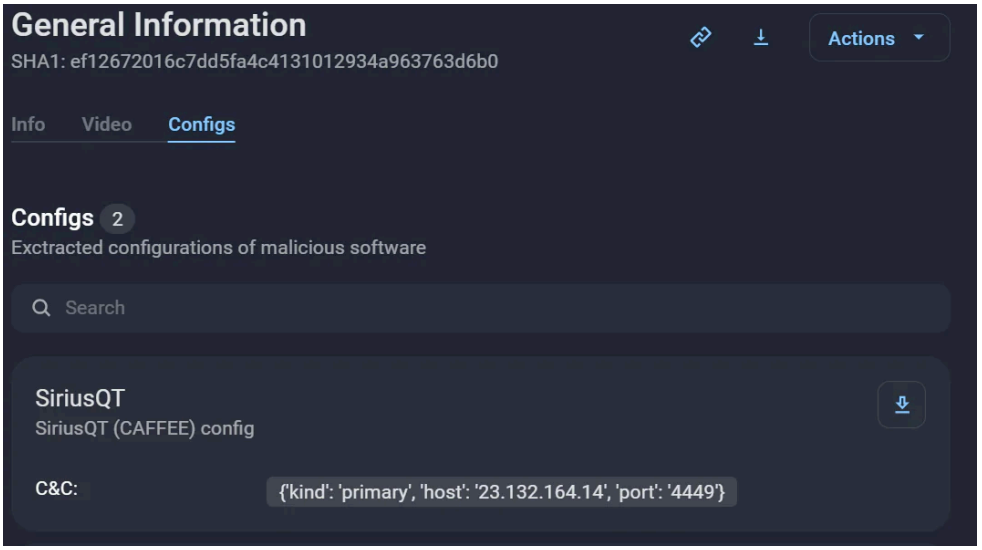
From sample to configuration: automated analysis on Group-IB MXDR

A recent SiriusQT sample shows how Group-IB MXDR works in practice. The stub file was detonated in our analysis environment on a Windows 11 x64 system and confirmed the malicious verdict.



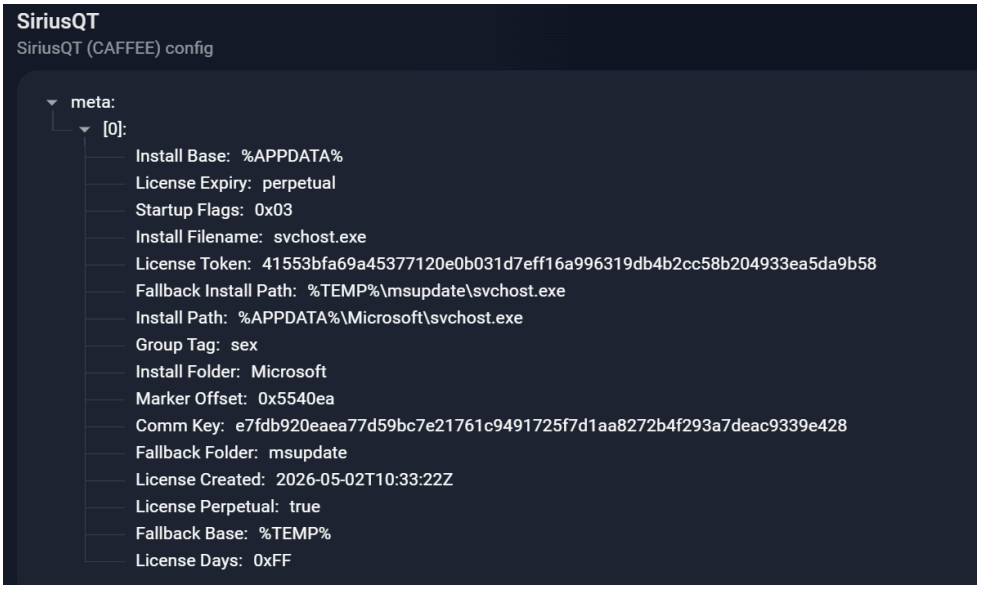
Figur 35. SiriusQT malicious verdict from Group-IB MXDR.

The configuration gives a clear view of the attacker's setup. The primary command-and-control server is 23.132.164[.]14 on the default port 4449. The malware installs itself as %APPDATA%\Microsoft\svchost.exe, to look like a legitimate Windows process. If that fails, it falls back to %TEMP%\msupdate\svchost.exe. The configuration also contains a group tag and a perpetual license created on May 2, 2026, which are useful for tracking the operator over time.



Figur 36. SiriusQT compact configuration available on Group-IB MXDR.

For defenders, this means fast and practical results. The C2 address, install paths, and file names can be used right away to block the threat and to hunt for earlier infections in the customer's environment. This shortens the time between detection and response.



Figur 37. SiriusQT fully extracted configuration on Group-IB MXDR.

Readers can access this public detonation report on Group-IB MXDR and review the complete findings, including the process tree, triggered signatures, and fully extracted configuration.

Conclusion

SiriusQT illustrates how modern commodity RATs continue to evolve around the operational needs of financially motivated threat actors. Operators are not only looking for basic remote access capabilities, but for tools that are stable, commercially supported, feature-rich, and capable of maintaining access while reducing detection opportunities.

At the same time, the analyzed client shows high detection rates, which limits its effectiveness when deployed directly as a standalone binary. This suggests that successful operators would likely pair SiriusQT with an external loader, packer, or fileless execution chain to reduce exposure during initial execution and improve the chances of establishing a working infection.

The HVNC implementation is particularly relevant because it could provide operators with an isolated hidden desktop environment. HVNC is not an auxiliary feature, but a core capability intended to support stealthy interactive activity on infected systems. This type of functionality is increasingly valuable in the MaaS RAT ecosystem, where operators look for tools that enable hidden browser-based fraud rather than opportunistic credential theft.

Recommendations

- Stay alert to any request asking you to download, open, or execute files, especially when the request comes from emails, messaging platforms, fake support pages, cracked software websites. Always verify the source before proceeding to reduce the risk of exposing your device, credentials, or organization to malware.
- Enforce Multi-Factor Authentication (MFA) across all critical services and monitor for suspicious login activity. While session hijacking through stolen cookies or cloned browser profiles can bypass password-based authentication, MFA still increases the difficulty for attackers attempting to gain or maintain unauthorized access.
- Keep operating systems, browsers, password managers, and security tools regularly updated. Browser security features, such as Chrome’s App-Bound Encryption, are designed to reduce unauthorized access to stored secrets, but their effectiveness depends on timely updates and proper endpoint protection.
- Maintain visibility into endpoint and network activity. Even commodity RATs can provide operators with remote access, credential theft, file management, proxying, and hidden desktop capabilities. Monitoring unusual outbound connections, suspicious browser activity, and unexpected execution behavior can help identify compromise earlier.
- Strengthen your security posture with reliable threat intelligence like Group-IB’s Threat Intelligence. Access to timely intelligence on emerging malware families, attacker infrastructure, delivery methods, and social engineering trends can help you identify risks sooner and respond more proactively.

MITRE ATT&CK

Execution	T1059.001	PowerShell

	T1059.003	Windows Command Shell
	T1055	Process Injection
	T1055.012	Process Hollowing
	T1055.002	Portable Executable Injection
	T1055.001	Dynamic-link Library Injection
	T1055.004	Asynchronous Procedure Call
	T1569.002	System Services: Service Execution
Persistence	T1053.005	Scheduled Task
	T1547.001	Registry Run Keys / Startup Folder
	T1548.002	Abuse Elevation Control Mechanism: Bypass User Account Control
	T1037.001	Logon Script
Privilege Escalation	T1134	Access Token Manipulation
Defense Evasion	T1027	Obfuscated Files or Information
	T1140	Deobfuscate/Decode Files or Information
	T1564.003	Hidden Window
	T1112	Modify Registry
	T1497.001	System Checks
	T1686.003	Disable or Modify System Firewall: Windows Host Firewall
	T1070.004	Indicator Removal: File Deletion
	T1036.004	Masquerading: Masquerade Task or Service
Credential Access	T1555	Credentials from Password Stores
	T1555.003	Credentials from Web Browsers
	T1555.005	Password Managers
	T1539	Steal Web Session Cookie
	T1056.001	Keylogging
	T1552.001	Credentials In Files
Discovery	T1082	System Information Discovery
	T1057	Process Discovery
	T1120	Peripheral Device Discovery
	T1518	Software Discovery
	T1518.001	Security Software Discovery
	T1083	File and Directory Discovery
	T1012	Query Registry
	T1010	Application Window Discovery
	T1033	System Owner/User Discovery
	T1087	Account Discovery
	T1016	System Network Configuration Discovery
	T1614	System Location Discovery

	T1049	System Network Connections Discovery
	T1007	System Service Discovery
Collection	T1005	Data from Local System
	T1119	Automated Collection
	T1560	Archive Collected Data
	T1115	Clipboard Data
	T1113	Screen Capture
	T1123	Audio Capture
	T1125	Video Capture
Command and Control	T1095	Non-Application Layer Protocol
	T1021.001	Remote Services: Remote Desktop Protocol
	T1021.005	Remote Services: VNC
	T1573.001	Encrypted Channel: Symmetric Cryptography
	T1001	Data Obfuscation
	T1571	Non-Standard Port
	T1105	Ingress Tool Transfer
	T1090	Proxy
Exfiltration	T1041	Exfiltration Over C2 Channel
	T1020	Automated Exfiltration
Impact	T1529	System Shutdown/Reboot
	T1491.001	Defacement: Internal Defacement, if confirmed

YARA

Yara rule for SiriusQT server

```
rule siriusQT_server
{
  meta:
    author = "Riccardo Di Pietro"
    company = "Group-IB"
    sample = "4fbbb632e08fde2aed8401ffcae9e45627d87084335ac88eb4c25088f0708b79" // SiriusQT.exe
    severity = 5
    family = "siriusQT"
    company = "Group-IB"
    description = "SiriusQT server"
    ti_malware_id = "08a69dd233ac6f4b83aff34cfcb2984590e2670a"
    date = "2026-06-24"


  strings:
    $key1 = { 93 7E AF DD 7D E7 FF 7B} //mov rsi,7BFFE77DDDAF7E93
    $key2 = { AF FF DB ED BC AF FE F3} //mov r10,F3FEAFBCEDDBFFAF
    $key3 = { B9 E5 E4 1C 6D 47 58 BF } //mov rbp,BF58476D1CE4E5B9
    $key4 = { EB 11 31 13 BB 49 D0 94} //mov r15,94D049BB133111EB
    $amonus = { 80 75 C9 38 80 75 CA 39 80 75 CB 3A 80 75 CC 3B 80 75 CD 3C 80 75 CE 3D C6 45 C8 53 80 75 CF
    $sirius = { 80 75 C1 38 34 3D 80 75 C2 39 80 75 C3 3A 88 4D C4 48 8D 4D C0 88 45 C6 C6 45 C0 53 } //sirius
    $license = { 6C 69 63 65 6E 73 65 4B 65 79 48 65 78} //licenseKeyHex


  condition:
```

```
    2 of ($key*) and ($amonus or $sirius or $license) and filesize > 10000KB
}
```

Yara rule for SiriusQT native client

```
rule siriusQT_client
{
  meta:
    author = "Riccardo Di Pietro"
    sample = "2eced97bf1395c1560fce05971a3400e0718a3bb70aadb5c2323121cdf6d19a2" // 9iz8076.exe
    severity = 5
    family = "siriusQT"
    company = "Group-IB"
    description = "SiriusQT client"
    ti_malware_id = "08a69dd233ac6f4b83aff34cfcb2984590e2670a"
    date = "2026-06-24"


  strings:
    $hash = { 41 B9 0F B2 58 71} //mov r9d,7158B20F
    $fnv = { 93 01 00 01  } //1000193
    $sirius = { 73 69 72 69 75 73 } //sirius
    $caffee = {43 61 66 66 65 65 5F 43 6F 6E 66 69 67} //Caffee_Config
    $caffeetwo = {43 00 41 00 46 00 46 00 45 00 45 00 5F 00 43 00 4F 00 4E 00 46 00 49 00 47} //C.A.F.F.E.E._


  condition:
    ($hash or $fnv) and $sirius and ($caffee or $caffeetwo) and filesize < 6000KB
}
```

Yara rule for SiriusQT Loader client

```
rule siriusQT_net_client
{
  meta:
    author = "Riccardo Di Pietro"
    company = "Group-IB"
    sample = "903fdebedf8976c8f6f937ab02e86dd73e5386120b6ec257bc9b2ff57a653cee" // NetLoader.exe
    severity = 5
    family = "siriusQT"
    company = "Group-IB"
    description = "SiriusQT .NET Loader client"
    ti_malware_id = "08a69dd233ac6f4b83aff34cfcb2984590e2670a"
    date = "2026-06-24"


  strings:
    $offset = { 46 69 6E 64 4F 76 65 72 6C 61 79 4F 66 66 73 65 74} //FindOverlayOffset
    $rasul = {52 61 73 75 6C 5C} //Rasul
    $amonus = {41 6D 6F 6E 75 73} //Amonus
    $ct = {43 72 65 61 74 65 54 68 72 65 61 64} //CreateThread
    $nl = {4E 65 74 4C 6F 61 64 65 72} //NetLoader


  condition:
    all of them and filesize < 6000KB
}
```

Yara rule for HVNCInjection.x64.dll HVNC helper module

```
rule siriusQT_HVNC_helper_DLL
{
  meta:
    author = "Riccardo Di Pietro"
    sample = "c206fa613257110b0522b60b402ca0b35da2cb126113e47b67543ba3ff93c53c" // HVNCInjection.x64.dll
    severity = 5
```

```
    family = "siriusQT"
    company = "Group-IB"
    description = "SiriusQT RAT HVNC Helper"
    ti_malware_id = "08a69dd233ac6f4b83aff34cfcb2984590e2670a"
    date = "2026-06-24"


  strings:
    $hash = { 3D 5B BC 4A 6A } //cmp eax,6A4ABC5B
    $string = { 52 44 49 5F  } //RDI_
    $stringtwo = { 4D 69 6E 48 6F 6F 6B } //MinHook


  condition:
    all of them and uint16(0) == 0x5A4D
}
```

Indicators of Compromise (IOCs)

File Indicators:

SHA256	Filename
00c6341ecb3dfed3e3b2f1e8ad3464356738c07edc48368cc0a5c501a0a97fa3	SysConfig_00000000.ppkg
044fabf6c74ee4de19cee1e3abe4c2beb6a33a38cbed401c7a822c696fbaa33a	Spotify.exe
05447eb22f74a607391b3260291491c129ac92a286d044f53f07d45911c13bfd	/tmp/05447eb22f74a607391b3260291491c129ac92a286d044f53f
057348eeda75a2061a8314d3215a602f02d4d53ea16ca0c3c55d1189244da4c8	C:\Windows\81qbo0.exe
05ebce0211b25b4af2aca0cf74a175dc59330719026c4ebbfdb11c2b8af3e049	MasonRootkit.exe
07ac1da4b8580958ac5ab693fd7c34dbb05132b1df03ae99e8390fe6e34ceb1b	C:\Windows\lrb5vbc.exe
09387b8f33f17c02c096397557fb19d4ef076c47f58b9b2e62853b4064a0918f	MsKl0aA.exe
09b9aa8f61dcfcd1bb8b56f9af083c3eb0d2399eb5714fa6e4e97e37e280fb9e	vtaks.exe
0c29aab4656498cd6e847257d64404ca8f2619467502d84b16de5cb396f409ce	UsaShohdi.asu
0ecb89b09a2aeee9285bef8a54aef6410e3a3fedcc4ad4022a5f5d6d6a0c2043	RuntimeBroker.exe
0f4fba21d5b02928d2fabc7d3aa6c1d98dcb3ddebde8e76e31e02f0a6784825f	C:\Windows\c44n51sr.exe
10899bccfe2d03922935cdd1eab7caef8c309f00f30df9797e972371f33f38a5	payload/svchost.exe
11667a43a05871e91891001f7ad34b5044d0ff400608b183a90641e8ef03d1cf	C:\Users\user\AppData\Roaming\Microsoft\svchost.exe
12e2b046127093fcb01d402599239a723278e0023898677607149328 2c6c95da	NOTEPAD.EXE.MUI
13b4441f5d8851a7b5f64f676ed83e1081b579078a7113606733c68ddf422847	Built_Stub.exe
163ebd5c74e865827639ba521bb6de49d8d01eade7367e421cb04f5285096709	No meaningful names
16c6526e5780f26b162a6a9429b54d77e6253e5ede17c42704011ba88f04806a	bbss.exe
16d247a2613bd77ca0b5df938cf6caaceef6a322cd0f88a36fb09e356b65567a	svchost.exe
17dac97c559e95c6700f100c623a4b88bb9169a88aa74548b679d0bf48de9ba1	No meaningful names
192b3442249bd2f002231d606e1f7c50738fad3d1c24f7233d62a3d729e00b96	SysConfig_00000000.ppkg
1a3b49397268c7ce3ce222756d13886ec3f7724975c6eaf6c963a2a7e5483e04	MasonRootkit.exe
1d0862222b7236ba62cc2ca44acef718f3422ceda9c5eced96f799287abc1d9a	svchost.exe
1d35ce5483a2ee834ab05a08c08d482244e3e15d2d371a1308cb74d6428a4215	Built_Stubb.exe
1f02ac81f5474658899e349808a69dbc9e7557ae50b24bc92b32a306ef60aa15	No meaningful names
207bb60a0d128da71ba31c11666e2b67a7a6e9d98ee6f0a6788ad0b7912a9caf	C:\Recovery\Customizations\SysConfig_EAEAFCAC.ppkg

21387409e7ca85e74364e64658a309dcb92ff037730938f4317be6458bd05b70	No meaningful names
2305f90fd17b12eb9c07b9bb2eb0f56dd879168c15fcf503afbdfc65a1eecaa1	RuntimeBroker.exe
236a61e60210260b1fe90af8988f61a8439bbee28fb2d82efb56fdbdd7efba1c	Built_Stubnew.exe
23bea69eefe5f935f24eaf1a5cf8442f999bccc61c70fb18f9f19eae129d766c	C:\Windows\m7gph3b1f.exe
25c927e0154a8d32a38171d04adf6c5a8ba781b00ab8275801ba10ccd7751086	C:\Windows\q2mluf.exe
265e184ad90a609d4d512113ef6000f58e1bced6307f202b6762a2ef61da9ce1	No meaningful names
277bc12f823de0bec82a964cb585254a7458a16cd2b1c077af2ed372a60e3af9	C:\users\default\appdata\roaming\microsoft\svchost.exe
278302e6f5ea966862e8520f61f967a5e4b1753bcae8ec729dec68c66b1193b3	C:\Users\user\AppData\Roaming\Microsoft\svchost.exe
2b5d634186e54996f0e967990da7b11be43dc8c114441afcb88991ec78bf7d9d	chrome.exe
2bfa7e924704dcd8ba14b26717ce5129e26b58c76390663852233b39dd15400c	Office.exe
2c0976827d68210f31e03a202f890b603687bd21b1c6a792630aed4993531e89	Adobe Download Manager
2c535630ffc555afde2fde178f3c65b1b041c388bd9a062430c8053043315de7	svchost.exe
2d049f77d656b59ba0e80750d1b85bd6bfeb0a2e782282e8b34a2fe16dbadf74	C:\Windows\y43tsk.exe
2d4cb785d48a888c2542e90c51180c5279f306d15661f6ae72a14333e6f60de6	svchost.exe
2eced97bf1395c1560fce05971a3400e0718a3bb70aadb5c2323121cdf6d19a2	C:\Windows\9iz8076.exe
2f76c60aa75fbda1dc233d996e090504e08cb9e2ca3a2d793717bac87de5b2b3	SysConfig_00000000.ppkg
3036151f19d61fa7562adb0ca164d961f596bea92618314119ecc2da6606bc8f	Adobe Download Manager
303d043c9af5d002d2d6cb2f1ffc9335dd465b9234b8d0b36841061eb3b3cff7	C:\Recovery\Customizations\SysConfig_00000000.ppkg
30533736fd48054459e782833a3d19131f5929671411686e794217fcd2d06b35	C:\Windows\f3mgx9.exe
31ede244af84389364846467f5cb39d42eb4b0080373ee20ac4c87384616c35a	table.exe
32f5a4a664f0de5fb78fbccce335c5c80cb83f8346d1d5c9feddc392b665903e	output.exe
340c8b70d815d48133db2f032ca34ef75c06ae8b0e38d335339d6bcbffca63eb	SysConfig_00000000.ppkg
34432e63cd75727b64996206c257bb6733b154bb878271d8248687bc1b1a2e23	Built_Stub.exe
3504f87e5d63af63f38a68f1ad1fc10ad150592e461a508cc6a7462baba3a054	acrobat_sl.exe
35e37c663686f9e20cdfdfd836da1c6297b40bfd442f3924c9b7c77422f9c20e	C:\Windows\bvwhc.exe
3601d88ca6b4dd7ce12a4d6ad4ca0c68ed9121bc5b979ed87ed174695e7be42b	Built_Stub.exe
37bd32bd2caf568b91abc87999ca74edc69ff80e0dd5aa0fe307a67777e19bd7	C:\Recovery\Customizations\SysConfig_08CFF1B0.ppkg
3896cabd97d77404939850fd4d4108405b6781e01970fcf172bfefbd20a7a107	C:\Windows\yb0jdi.exe
392fb6e6380f80d16faa339e0feef4d738f9bae78825aac4872d75e7af8194db	SysConfig_00000000.ppkg
3b263ef0b8449b871f3c05560afe3974dc96c0f6e588300d641c7a978bf4ecf5	svchost.exe
3b26fa170ebd9fa9e553c89216bcd3fe3e393f1e63cc004be28647f534aab91a	BuiltStub.bin
3b4efa5cdb9d815faf485a49abf11454cc66e5242b4a30f0a6ed9d1228dbb657	svchost.exe
3be3f0993b2f2facf1d8e7ea576ddeac6785c78fba411876ccf3302c0737b617	Discord.exe
3c64b7eb74b87b4a0b1a6f501eb3b2d32665022e955dd025a35c827029d00061	svchost.exe
3c9d769a7aeb326739f706051bd0d4c49658a3a8190ae9c4eb6fd439599d182e	UsaShohdi.asu
3dfa54af1b7c139f91eef19c83ee9e519069ad732381c93476567f86a0f857c6	RuntimeBroker.exe
3f708f06b7483d35525f159e871cc1c7443b8ae2b205fca360bfa0436fe1f50e	Built_Stb.exe
40a2ae026e88ef5770d78573a700143c9c01adf1e4e7e5701c474298fb11d363	NOTEPAD.EXE.MUI

40c224ba76002280d07f65ebff5cb901147313245e12155b0addefd7a9de66f8	C:\Users\user\AppData\Roaming\Microsoft\svchost.exe
428a03e3e3a36a034230107e5bc39791c1bfacf50d2784702a69037342c37165	C:\Windows\62xa4gvp.exe
433b6b3184974b042c875cebee8d363fecef970832b59b33f5063651ed2df574	chrome.exe
435bceaf240dd7711542a723a6988b280948e77e9953614d981d653680de9f49	D:\auto_black_abuse\resources\unzipped\20260416_033939_20213\435bceaf240dd7711542a723a6988b280948e77e9953614d981
45c375eeb5e76edb785550d9ae0ac265be7dee1e83241ca27b0e2e181490e1f6	C:\Users\user\AppData\Local\Microsoft\Microsoft\svchost.exe
45ed7c215e6a258044968ca23509a9221194a6ca80f22905858cc4f11e70bb2e	C:\Windows\2228xopyk.exe
45ff8b8dc359d9d948defa5dc55286f684c4c3360730295e2e5b1994181d1570	svchost.exe
49c8fee089738afef6473f4a1eba3af654b994f4fc3a65e49b49b39656d2120a	No meaningful names
4a25ba8d66eeae6c5cbd9a0729288b704f5a07b1aa44816754c28654e79d9b2a	StartBuilt_Stub.exe
4a3272aec9b51533b7e79d7096ddb2198065c5258dd1e15e51603697043e0d11	No meaningful names
4aa693d6e973d03175c7a4403ab13d94215f5c778dcf066101b12d35506514dc	C:\Recovery\Customizations\SysConfig_00000000.ppkg
4b7c35770f05d45b42fe9c212a807dc156efa44d2f7b12242ca8bf1103f6f675	fdfdfd.exe
4c5a229fb67eab9da3eec9cc0ca412d88a0c084b9665a5326166c2300fc9e44f	payload/svchost.exe
4cbd5d8f38fae48d231f59a3cb3efc54e50314f655d549d70b898428e089ef17	Office.exe
4dcf241b28d04361ec7eaad8e78204a844ae73c78896971ba1002907f2a7fae8	Office.exe
5004325791883b8dd6daabe199238ce909fa7fbdd81c3cede32e68912c24a8d7	C:\Windows\pegk70.exe
500e217fc829d22b9bfbc29306e97e632720bafc96f6fa8b767f5cc4391c5e58	RuntimeBroker.exe
50c2d9ee37a518b7acb99c5bebcde12b87228d30d82daf95396d175a39cef71c	1_2M_Infected_and_34_7Terabits_a_second_UDP_botnet_panel
50ec939f488a2d6c2ceff802c88ebd286e8dc365cc55cf9179067d7e87148951	No meaningful names
510c410c073aaf00c9930b8785a53375ab2456040c60aa4ffea267aa482b0a7b	localfile~
514b3fa4a66295ba0f69d5efbf0c99a824aaeb32aac392394a4fb98bea26cb95	Office.exe
515f6f523cfee1e368fe579beeeaff54c457cb4c5fe8b7fbc449a6e77ad5ebc8	RuntimeBroker.exe
52580b839330562ee40dfee8525ac4d1aa83a50df256709bfcbb8ad2ed9a1fd9	/home/petik/ss/malware/2026-03-23_73b69a31d5d2b60fd671d9d87ab05de8_glassworm_icedid_lu
533d9a9dee32029c75115ff14704f8d7ba1f3ccbc27c28a68e6cc6268fea405e	MasonRootkit.exe
538139367a5a38916f165ab080043b831d28fc4c6ce9d9f8b36b9aa1c99bd7ab	svchost.exe
54ff34ff6782b964e71f18fb98a581a222f76834965c5c552e79c726ad613054	No meaningful names
553ed67d471555eed31223593302e0cc7951b1c07c96984fc47e78ea52764c6e	C:\Windows\k8m7re.exe
55afe5ad9092239bc9a64b85a16f4ad33dd9225fb14b9576f6897e904adb4742	RuntimeBroker.exe
58359849d455569a65378c7143cf0722d91cc76bbb82ad4225b6d3677400ebf4	python-3.14.4-amd64.exe
5856e9cb3b6ed9bc8a274bb3e349cb3038f4180096bf16aaaeee6ded297faed6	payload/svchost.exe
597aaaf263c6482944ce6aeb39cf20c98e579b22a973483816d4104d8772e4f1	Adobe Download Manager
597ca35933a2db1f014ea1eb94a8c6c5f6e92b7f171bc2787445fea5c90b4034	Minify-Setup-1.13.1.exe
5aaf5aae7babd494066bbe38626922db974571514a31f83e9ddf2a8610a76348	C:\Windows\sgkp9xk.exe
5b4ec46453e1c19fb31d899b6b6707b58a6a02ae323c87e8d3b8d51d10a32f5e	RuntimeBroker.exe
5c9e6d038e2f2cbb04c9fbfb76a20f230b5e5d9f0679260eed7d2a5203ba99f9	C:\Windows\otai5lch.exe
5f487a4f579a4f1d8ba23fb09879dda3143007ced38ebc8cd609626a9a5ca632	No meaningful names
5f5cd66c932f35ebd97a33ad49d4f487d4490cedc1862ad37aff2514806abf35	C:\Users\user\AppData\Local\Temp\Microsoft\svchost.exe

5f72f41ca6a6d9597ab1e0e586e14c3fa9c50617bb6be17068db28b6f0ac5dd8	/home/petik/ss/malware/2026-04-02_0918d60c330858d47c1d2ad83db7e8ed_icedid_luca-stealer_n
6012cea76f91f2f8104046c2bfaaec73536dd3dcef55418c0afa98e9276a9ab8	2026-07-17_e427b9b773c34a7e0d91abc901e2cace_glassworm_i stealer_njrat
60153d79d1e819d26f4ebd4bf2abd77cca97a90a26f3750ed1d5941bb2b59869	C:\Windows\hq1d4par.exepeexe
6060cf6277cc03e081cddfde7f8c58edc40596050d4d55f99e1c3a0cf413409b	Spotify.exe
61975ab66e57cdbc504b35bafaaf0657b561e64da08275e46be45e66c5e78f99	Spotify.exe
652278d492e94af9a0ff804cfc4ca35c2b26b6f0508dcf93b32d95ceecacad5b	C:\Windows\7g2a3a.exe
65d2a8f0ff42edf1f077f7fe2c47450a6ab97db4b21802b4063f66844d45414e	C:\Windows\wvf83t.exe
6652ff8838382270cb07c1441815fe7f651896e42d7ae2d52400984da7ef8f3a	No meaningful names
67638bd64812cb78f4173f05935a63ea80f8b3c7833cec37ace407b66cdaafa8	svchost.exe
6aadc2dabe1e3b97c7a5b560bdc26b1ad25323e89a636cf78b0614624fb5e6a2	Sirius.exe
6add46f5fff31aa1a3fe19c40e59981b29c5774c6f312cbd610bb07f3360d29d	Discord.exe
6c5540020b3530a2463408d1eff75e6989cb312e2d04176a9a1e01cf44351fa6	C:\Windows\ime699s4y.exe
6d7367c2c29fbd59fe2e55f2d299f7c145775604260a2f5ddbd682b34cdbef78	NOTEPAD.EXE.MUI
6e205588bb6ce8ec03c75905820a6e9c2f55b7d75be6f7ffdea56e4a7eda7dbf	Discord.exe
6f1cb10fa32ff451ac8383ec1e8caccf5a9d2db01b561aa7677c0206cd1c1569	svchost.exe
6f6f8e10e75258f5d7abc3f976b92c28264ca9d03be0c86a6bedebfc020974cd	SysConfig_00000000.ppkg
71ab6c33be63abb177bbb4a95ad1a34175c043a890ee5629e3c29b68a399c310	C:\Recovery\Customizations\SysConfig_00000000.ppkg
728e75ae848ac62aa705ed4e4c000dd3dfa2a4cf70a5adfbc56c5a9d2b017792	728e75ae848ac62aa705ed4e4c000dd3dfa2a4cf70a5adfbc56c5a9c
729682d4e9c559e33e8716ae845ebd8323da4636affeaf994259c63ab24807fa	CraxsRat.exe
7474ae4d9491df3152c75a14265d5311906feff4aaaa6fd4408c7e858c2e6b87	msedgewebview2.exe
7481e6dc93002b2151751557cdf4a9314410fd6874ad1f039ff92ddeea04e96a	svchost.exe
75b1e1a5c0f3f1a1230077a735f3df3278a283b7223c8d98be018368b8ff792a	C:\users\default\appdata\roaming\microsoft\svchost.exe
76e447ad7f9fb49168c95451c64e38add75eb980bce515134639fa4fe7dd0b8b	svchost.exe
778025ed905c788d16f87ae9a6ded8880a7bfdbb950963012f9d6b7fc8631ce8	No meaningful names
782c945effcea3779d19e1fb8e52bd637d04830cb6ba742883de255e7dbda977	C:\Windows\4l9ta1.exe
78e25e310b2e2263df80e1d0ef434811be52ab51b36bfac440df017d75144105	Qt5Core.dll
7a49ffa93d305f85369c72165adabd25c0f82ac78edd7d0787b8bc55b4a2ae7d	payload/svchost.exe
7a4fbd22085ffbc982719af76e89b0ab579b353f51a4c3f8a00d4ef169ca3e21	No meaningful names
7c36e0b9268defbea30674a6f8590cb53b92699bafbe7d231fa374bf0b41c35b	/tmp/7c36e0b9268defbea30674a6f8590cb53b92699bafbe7d231fa
7dfab4baca5926ec6561928caf994f45e62fa3f23596a24b4af0a8ef4ebddc4a	msedgewebview2.exe
7fd1fddc1e0fdb6d5aace5c90f44631831ac0d73ecaf7e9b13c0d2c4c96e0f8f	RuntimeBroker.exe
815e29156d4ad5c6249403773e4b7b2933fcab8f162e1a1c8e9e9708855d9a4f	svchost.exe
81cce6c045cd75224f458cf394d0f4cc9cb6c05404162bf29714308804427ef4	C:\Windows\q6cb4vn.exe
8360f37249851df11c598ba99b04caf6f0b4623778056592f73c741a3e942aeb	No meaningful names
847da0146fa3439ceb50c9dbdb1dcd06c42fb4e9ec152da6a123fb0e92bdf030	Office.exe
84edec85d97cf41f2445a46ea1c531288df06bb6eeb211c22b0b469c3f4ef557	/tmp/84edec85d97cf41f2445a46ea1c531288df06bb6eeb211c22b0
850ee74f8b9279452607941e0982be8dbf9b09d20f481507dba27b5317cb3f68	svchost.exe

85393ac678d3361fa57af38930b99364cf09449709b4d55e650f277889fb3eae	C:\Windows\6sba40.exe
855b6818adb18d576e0a92a6f2dcca439d3902a0175c08e6b8df5e26abd3710e	OneDrive.exe
865270b4f53c3177fed75a0126668d145a0ac540ec842158453ed86116c4889e	/home/petik/ss/malware/2026-04-12_e41692d49daecee9c4b387799f06d2e1_glassworm_icedid_luc
882b0855a5b578a31bd03b6a47351d29ad9eb819d4c2c63408c3931a83254c7a	worm1.exe
8838fe46c9aa4e8e60506e7320d34702a52539fc60cbcb03f0b8be6d8c1b4512	_8838fe46c9aa4e8e60506e7320d34702a52539fc60cbcb03f0b8be
889e5fe6fbe799fe1455232df86a9fe8eb8b949de385e5169ad0afd19b803aee	OneDrive.exepeexe
88d7c698084b725d00e8d514ca6f680617e83031a9e096318fb840bc2140a050	svchost.exe
8a3fdfed321d97a950afd445fd8fba095942fe95e1622be6cd77109b6b6ce8ce	C:\Windows\k6m54ti.exe
8be66b38f08714226f27c482e3c45b8d2b226e195766c3da6fca9b2a4bfc078c	C:\Windows\trg7es5wl.exe
8bf00e457a03bdbef068e8ab4969911ececf328784d978df966f3d4b81d83ae8	Office.exe
8c302b4c1abbb45abb67ea7c0eab040060d4d4da5f671273e6c7342e28fe536e	No meaningful names
8cba51d0546d62005e110a4c33c9c8fc4203d5812ac5ccd7c78a3b18b0240281	No meaningful names
8cbca7c6944ab0b3cbea2c06a3394f216dfb5db41591fedfafad4ad54c098907	Office.exe
8d221cad9d15e3852b1366d8675b3cc3a68741b399d7eeda8ab6de7592e144af	No meaningful names
8d85472d4aa05cddb26a3d1b8c13b829f3e3813a152e0a2c6165ef6ced54c957	sleestak_payload_1.exe
8e50201ad8bd4813be995ec2f8d504e1df57d866b3a8759214ea4c1f9d4a157a	Built_Stub.exe
92bfbbd8e6eafdeae7e22704b2a7212a4fd9f8e8f6a675800750d1dd30958f20	payload/svchost.exe
92f2c18f2271d404c306f6cf5a9a9b603435e48aa48df13115af6a45a2299f91	SysConfig_00000000.ppkg
931938d2edf11b236119bf724250e5811b14bbb52d5b40dd5a0d3bcd7b553775	C:\Windows\ulycmsig.exe
937f16e523594646a78a42031a84620ffb6d1abee8884c21bfc8baf4bd9bf31b	No meaningful names
938050157236523db559513d26223fd8370a4ab2e92590e2167c274fe06c5d1b	RuntimeBroker.exe
93c2d5b5d1bd9a9e88a988c1b78f645cee4afbebfd67024004ed035d52f4ea7c	No meaningful namespeexe
95c87d65e7f0e86936fd0f85bd3a19e35ae11a4f57c29e18bc1e798ab3f328ee	OneDrive.exe
95d3059ad7a0d69f9405ffcc4ef3ac5c060f8d6e088100dc740bd28e1c3947ad	C:\Windows\dw28vzjr.exe
9713e6ae5d7599379259cc79006ae9165e88a02c5cdca2b3eff89943ea92acb3	Office.exe
97609b45d5f2de9fc5111aa64ba27bdb0632957e64ceb3086403cc975515e009	Spotify.exe
97e23a372c9c789f2a16795b82155dafe7a2db1fdea2877c9e19fd4a7d1c0c13	svchost.exe
97fc2a47422f26f5cf7bbcb82de3ada3ae38e1250a2a6f93b0a88ea12d81faa9	svchost.exe
989c2c50666a7e0b8d8bf0c74020d29942124da2de28255109f822750a3a5de0	Myproduct.exe
98ec96eaf6bc35cdd62d449de03cc14d08e77b3834229052ba745e706d08c7cb	Built_Stub33.exe
998c2ce1c073fb82bfc76ce3dfce765869a9c67dd39a8fed9331f72ebe022365	C:\Windows\wffam.exe
99f25e86abe4a68ab75ded32762954b96eb1c3c5d96ec4310b139b9ba975ef78	C:\Users\Admin\Desktop\Built_Stub.exe
9b0ee65f05e4406648e0e6ff0d9edbb2c1b5c32da8167fbd697340b4a3fdd14f	tmpC24.tmp.exe
9bcf4609c6f162d7f05a8152cf960cf21a9e0b418ce9fce8491dff56b1e4dfce	SysConfig_00000000.ppkg
9c1b13babea4d84950d3436736d753c7c8b2a8294026712594658e91b3ea91ce	Xeno.exe
9c2f2c0496d70ebed73f04091da8f82a43aaae4c2706c164da142c59a04cd6b5	C:\Windows\v8fvti.exe
9dea024a4cfa28ebfc48c8416fccf5705f19ab49d4d24ae9742a573dd5d54926	Discord.exe
9e4a46d7b8eb94763419a6cfb077443a7f5310c52b79c1ebd5a45f643c325f54	C:\Windows\oodjc1.exepeexe

9f62e480b6aaa5c9840f975bb4d69804c18e8c99ab7ad5946e783a5cdb4a166b	C:\Windows\pkigkfyc6.exe
9faafb9f1a37a562c0f3d6e7f27d89558632462a011af6e8ac0e41f91e2a65ec	ngen.exe
a03209845d5b7cab32abbaff927d04d02ae113fa80cf19dc9ef8a867b1636020	c:\Program Files\Mozilla Firefox\uninstall\helper.exe
a08623407ae4780c1f3b0810dc79881daefefa038d24f8c650ceafc0c33a9eee	C:\Windows\s55c6o.exe
a0ab953a8c2eec84af932a54e47cd9264877f746378830770cff08791082336e	No meaningful names
a1451c391f67b4d489b8fb4a00db561e39d9675a2bbf9d18fad93f5b03501fbb	C:\Users\user\AppData\Roaming\Microsoft\svchost.exe
a3704b2426ee33f1cd546f814b4730f592a90a84133d863849aad0685e8ae920	C:\Windows\pfbh9x.exe
a3866f0d3cf0d369e50f097db35a03e428da055f5ed03b956fea277d4fa1838e	/tmp/a3866f0d3cf0d369e50f097db35a03e428da055f5ed03b956fe
a45a19ffeca3458db92a5b6dcd8aaf59ad23a14837532de128e397a619579852	Xeno.exe
a478f5057ca0f0706a61e4fc1a681090bb49a54ffe60567e5dabe1ae05d45880	D:\auto_black_abuse\resources\unzipped\20260607_044825_202 04\a478f5057ca0f0706a61e4fc1a681090bb49a54ffe60567e5dabe
a5c6bc0931f9dd6e7a832c97f76adf10322266e2b45b9f43e0b9513faaa79c68	C:\Windows\sd5md.exe
a626b8dd94f643be90884535721d674c7b3d2a45c9270dc3daee1eb04f304028	RuntimeBroker.exe
a6283338c800e30059cb43a6a5dcfc0ecdbcc52c3597ce86f2a116a0c0bd26d0	Office.exe
a702f1935de515b8b01ffb5558e5e774d92cd28653c6afb71370ca10cacdbf62	No meaningful names
a70f192a6512d4673f42aca3df19ad91af285838d778575af38950bb292ccd2e	Discord.exe
a912993eb858db7d367a0cf44a488d06f75f99e30ae2808156260276e6115574	C:\Recovery\Customizations\SysConfig_00000000.ppkg
a9a068e583e4c72f74f86ce124d0e6b7a291dddc33814d285e115f584eb361fc	2026-07-25_6b696cda4ff3b784a164e45176feb892_glassworm_ic stealer_njrat
aa9c1ce4aaa02f684f3497db566bc9111528f25499d77747692cf2da03648e62	C:\Windows\0wvlny61.exe
ac10d49654062b18c8e96e9d0a27fe073adeb388aeb3ba07ab038baf0034af39	No meaningful names
ac52999bcca032660f8938e83857ba72f4089c4105ec96560381b39d45146831	RuntimeBroker.exe
acba285daeae4ecb05deff8d7b14351c48a8ea2cd375d46053449f03575bf948	Spotify.exe
adbc63eeec9c02177851c29e4de474bf5765f6c367dd2a36418778ebca7ac486	zeuxdbvrgu.exe
ae25aa87fe54e8b3025e5885cfae16ad0aef7dc8e0eced1542ba70b3f7180343	msedgewebview2.exe
ae938385bece4d9f106ad94e9b5be5653881586cde507ef32d00f8f7e258486c	2026-04-24_e6ad565e8b5ed4564cd1d1d3a2ad9059_glassworm_ stealer_njrat
afe59d90e3467871c0bbabbf28fb8669a45922d83781662d1317a4a6f6021344	C:\Windows\irneexc86.exe
afec22878f7fc7ab96e1c4b1976383fbdfa00823434f4a459287e7a4f67a15d1	C:\Windows\7seavgmx.exe
b03c94d7bb24ebadf3f20eb4c93284df77a0ab9a31ce24677c98c28ceca238fd	BuiltStub.bin
b19b55867150ecb9a478424700e403856345dda44c5e1837ff9bfff8d0e80bed	C:\Windows\0vhoy.exe
b1c67eac6e0277c6e7b869c786111e184a62c4fc1b8386d85cf526bc51bbce8c	Office.exe
b2038ab2f49b88905a86f48b1d3d05410597e1af99c165be3fc68b84de14ef46	Built_Stub.exepeexe
b36da2538653d36eb4e15d984b7c3083a60fc2af333b9ce5caee108af90cf539	RuntimeBroker.exe
b4cfe995e00e7c3f1f522550b7d388cd3e07b4979bf43f56feb5677dac30bea1	No meaningful names
b737d546e33c0f0bc2d4409358c67fea35104aea47a915e199a5bf792573c555	C:\Windows\rh0dw.exe
b73f305230affcf4b09704ee72871d2ca7d2a0a574c061f4e77945a0051cb96d	C:\Windows\4hm1rrzk.exe
b7ad124e1e55a11a184b0e4de21f8ada3e6999b576fec7893fdf32a4b0e29c64	RuntimeBroker.exe
b8d1994ef15ca9ec1b8946356e9d6d5fe5cbfadb3af433b23e8741639c002dcf	C:\Windows\3kcdjn8w.exe
bac0fd7f51a4ecf930553b463ad3b9fe400aa19c4074564e6c4d4502239b20d3	/tmp/bac0fd7f51a4ecf930553b463ad3b9fe400aa19c4074564e6c4

bc01568885ff3c08414865922bc9cb0eb6f4d34eab6e7a95e4aa1f752f5c3152	payload/svchost.exe
bcda26f67e4577a7ae180bae67100eff0f89102d641eb874aa347f28e19bca07	svchost.exe
bead0690e96aafcf5c242d0ca0d30885584f42bcf2034a354b92f7e145ac23eb	xnx.dat
c0a1713198d3c3aa5535e8a930acc48ec423444ea019c939f2183f9b957b5b24	No meaningful names
c0fcd05db135001c6c14d86b6fc9666007ab98d19e8deda2302df0353cecee1c	Discord.exe
c1f95e580e28441f00985e15c558563bf2161386419b77472302bc7dcdf17533	Discord.exe
c288c64c579837eb8a7d2c91b238434b7239a07480f2c50842178289fc64d6ba	Built_Stub.exe
c44b7dbdc0e6763d0ef85ba7b76fb0f9f82f87a1279e0eaf18aab0711e37ea45	xc44b7dbdc0e6763d0ef85ba7b76fb0f9f82f87a1279e0eaf18aab07
c71d14d9e46cdb9897b8891798c788c42b8b1fb37cd930843f73f2fea16a8f4a	UpdaterSetup.exe
c85e46ba6c6f8c949f446f89ccab5a000de0fdb1b1a2b67ace12b31bd00e3b7c	C:\Windows\ozl693b.exe
c9e38c1c8a7a513ae235e84d0be5a22e542d6556c4f45db3b78a2863c67dc086	Office.exe
ca8080b429c34e4f31ce869c1eb46f758109e8de4ac51a9826759f39b583830e	OneDrive.exe
ca892ad72f9a773af1f3400db20fa09aeb3e6b9339bfb070dd9fc00c6ba3c009	Built_Stub.exe
cb56969655617d54343ad57e3a6029906c3427e73adc61811f1d99d34911351b	No meaningful names
cbe1cd1c3bbf77cc392e0bf88a04c894a7122c08ac5d875212d70ea8fc4aab39	Office.exe
cc26514389624bcf6f5236bbf046324bd1d6af1fede772efaf45c59963c5475e	Built_Stub_Mutex2.exe
ccf0a0252b3df9ded459207a717b76e21972f5d89f0eb663d65736d2ff6cfc9b	No meaningful names
cf17968d1248193dcf5a236679cb5dc493a852a4055ed8373bd1e2ed164e3113	C:\Windows\jfq66zh.exe
cfc81b850c8e5b41c77690d2a7a745489d04d5df6fe89282da38c6b6e25c9b87	%TEMP%\xrioq2my.lvj.exe
d00a3c90ec9e2528aa9f90a9f57f40680a316fc3e55a18550234280782b9ed25	C:\Windows\98lvl9.exe
d00cc4783faa95c50311be60b5e730c20de6b6b313a5ecada13ca948d1ff825a	Office.exe
d1a1e6a2c0adc45a4cf40128e683499cd984d5803ad2e3a1ce26768a612eaec2	123123.exe
d2be1b3b86f93a62fb4a5a8daf3c13471a1dcc70b3e998b7f3e1b034931d83a0	No meaningful names
d3401a2aecb32dd694686f912a06b7ee1c459626a6ade743eb01cb96e0badd0b	C:\Windows\1y29dfq.exe
d4dc0b77984b181008b2c7b7936ace4a00273e56743f3f91f1d0b3a818b15eda	C:\Windows\gcsvzy.exe
d558f0c20d4c27936af6168fd00fe9464955595ad02be93558bb64d82f81e92b	svchost.exe
d58de1bcb41688adbb8967b0c5ba83b5c5beae7de4c9fe1b24382db7902c19f1	C:\Windows\646cie1h.exe
d5b8b760ec4ed0cc640316ba853fa487d7a1968b63978e1f18b7f6bc46d77a3e	chrome.exe
d5ed8464f074e6673cbb5300533a9750bdcf5a92dcff27fc41aa7ccb4c863d24	svchost.exe
d5fcf89e289b87e201fe007f9c4efcc0261bc3c5842e012a2a6068f7d5467b76	C:\Windows\5868k10u0.exe
d6802fce5b918ed247ba85d6aa0545a8120cfbc3734b1c37794c4e381862d4ee	kolodial.dat
d6ec8d835b0028d1bdc9d092cab8eef4a69b3a1e11e930a5d39fcdddb4ceaffe	C:\Users\Admin\Desktop\Built_Stub.exe
d9e9990667e7510b910a4979af92cd28573f39a2a5f9e84d3808650aa4350368	svchost.exe
dad498e450436ddc76eac0427f253b06ee4b4eec497d8a2d2f01b1098b7dacd8	No meaningful names
dae4c1abc0a6136345e7a1f1a58b3981a44f0885dac256800b8576e91803b3a2	C:\Windows\qm9va.exe
db4e523318e5e9702dee4abce7b52da9a5fc2f4d0262ca54188b26af98a12d69	C:\Recovery\Customizations\SysConfig_00000000.ppkg
dbecf0a553ba4bb8bc928a6dd6727ef9ed2d4946f4579ff49b49ec59c6a87199	SysConfig_00000000.ppkg
dfaaac39723510727e4713439030f491a9bce3141f64bee0d4d1bc59ec195422	No meaningful names

e053d408047f2899c5d678ea774c2d4e8f3cb81fc590a729b292c6e10ba18ff1	No meaningful names
e1035bf33547756c7fbe3366fa7fe21c30b3914a99a61165ebe32576ccc3dfa5	Payload.bin
e1590649901ad23bc6dd6a3d0589deef5adbbe90191db7d8d1503808712d3e48	Office.exe
e1b4d44eeddc66967bf8937ae625a84040161f6238e72bb57f0bc8f812984e11	Built_Stub.exe
e56f83c1af5e624f41cd5eca35d14d1c786ed94f91395b20bcee9f07bbbdf57b	ms-teams.exe
e5a598b699bd3d192e3c099b14b12f68126b49b3260ba82004eaee87b4155463	ike.e
e6a52e739c17761af47d1cc3563ce89372135893daa1c909f15ab05c4a6aff1d	Built_Stub.bin
e8081eb041c09895b2d4a7f9471a4af658906c2f8e22df697e20213d4ea47a9c	chrome.exe
e892f4f332a0c1a67a2b014add3a6b83d9aec1ffc07b3ae2dc1ba87cdfb1126e	C:\Users\user\AppData\Roaming\Microsoft\svchost.exe
e8955d9f21f187d1dfe3b28b60fe561df011a762e10458699af5603079a10b54	Office.exe
e96625e8d2336bd8ab2fd45ab0f22bf3bb461c1d582fdcec94fe36b5948a3aba	No meaningful names
ea7125afc1baadf24910f6294e9d4837dccc3237569063865ff96fdf1243e689	C:\Windows\30w7rmw2t.exe
eca781982340a1d7cd12373b0b0b5de988dc75ad452aa3b2089dda15b6e48515	Office.exe
ed061270b4dee25b1f68b39584236a1786251c174c180e9eb5e1a4b2f37d6def	msedgewebview2.exe
edcae6d5c0b2e104043bf8b7dafb19d29d3e4341238a88b6606aaae2cc08b34f	/home/petik/ss/malware/2026-03-28_0a03d92960d568aaa75943ecbbd69472_glassworm_icedid_lu
ee92a3ed6975b7b772246a0678008f5afa5911bf993cf043f626a1efff081365	Spotify.exe
eed3c35c6dd610824f3069f1463de2bac1f7ab6e0d874f66df747cef33c81cea	MasonRootkit.exe
f00c656d44c324895c680900bf0ad0dd721becd5e7be5fbfbf2ede73ee89641b	C:\Windows\uz4ua.exe
f03d36360946d5d9627b50d258675397a99896ebe542bed44e557beb531de7a6	svchost.exe
f0b17572ad68d4d9f218e0f8cf62aee69653ed95fa10cc7b22500e28b4d01f7b	C:\Windows\1s3aw.exe
f0e60fc21f4517590d26d5bde37f61bbe0a2d95e76d16d5c85217bce563ce59e	2026-07-20_5846e76c2183b5e8b4f8a4f06720e1ce_glassworm_i stealer_njrat
f0ec0a569f05e154e29126e9c2222bf05659d2b15676b70e494b245e3a996ed2	c:\Program Files\Google\Chrome\Application\103.0.5060.114\notification_h
f1adcccf3ec87444ab989426b313c426a83dbbe774bcd834c249e079cda93c98	me.ex
f7a247c9d77002878054310ddb6895798eb2df591126231a4b3fafa2c51dbfd4	SysConfig_00000000.ppkg
f7f28ef35e6becde12c319b90acfc98ec25bfa9acb17b534cd68f8c5f732fde2	/1/f/7/f7f28ef35e6becde12c319b90acfc98ec25bfa9acb17b534cd6
f8c92b129da1f4ebec7ff2971fd7e4784e65537b7c25a0209032d16ed8d887d4	2026-07-25_7c0684fd9d32c4f4ce32b549f41061b6_glassworm_i stealer_njrat
fa32aba50f22d6f1c4634b7d3ffd9eee5cd573317b75a7cb6bf8402c59c983dc	UsaShohdi.asu
fa62de958ee2ace2b41728ff10c0911b81692c8d210c2f4e2d6d7c54b9a09f92	C:\Recovery\Customizations\SysConfig_00000000.ppkg
faa3697bfd7bf1790ae72bd3121dd7b2c02d7f78cb0cafeeb2605bd9561ed10c	rufus-4.14.exe
fadf8e151dc263bca251f53856f4670b9a7826e66be22516187879c479264dff	C:\Windows\me5or.exe
fd6215e78423780eb85043370b07d808631188b65438a5eb0ad9956fbcca5801	Discord.exe
fd83e49f81bdbd9cfb1c132ab79095e8329057a38425d45d8ef297d67219cbfd	C:\Windows\56imlfr.exe
fe6a726b412ddbe09dc85b2fa11974bb92ddbad97473f3de581eea068caba213	No meaningful names
c206fa613257110b0522b60b402ca0b35da2cb126113e47b67543ba3ff93c53c	HVNCInjection.x64.dll

Network indicators:

vmobqlunsw[.]localto[.]net:5044
196[.]251[.]107[.]104:4449
66[.]94[.]102[.]6:4449
tcp[.]niceports[.]shop:22887
103[.]89[.]12[.]159:4449
185[.]69[.]122[.]235:10015
vmobqlunsw[.]localto[.]net:8666
45[.]7[.]228[.]152:4449
newforsiruisadd[.]ydns[.]eu:4449
91[.]236[.]186[.]56:20797
91[.]92[.]120[.]61:4449
91[.]236[.]186[.]56:20797:4444
tcp[.]niceports[.]shop::22057
209[.]50[.]250[.]33:4449
188[.]215[.]229[.]24:4449
tcp[.]niceports[.]shop:26550
biesmanscars[.]biz:4489
sirius[.]surpasstools[.]com:8080
198[.]46[.]173[.]36:4449
185[.]238[.]191[.]37:4449
151[.]243[.]109[.]130:9672
195[.]226[.]92[.]129:28321
85[.]158[.]110[.]116:7766
alfa30[.]giize[.]com:8822
2[.]27[.]63[.]150:4449
bore[.]pub:59481
xweb[.]ddns[.]net:7070
45[.]154[.]98[.]251:5001
vmobqlunsw[.]localto[.]net:4358
217[.]160[.]43[.]170:80
185[.]227[.]69[.]72:4449
194[.]26[.]192[.]101:5010
vmobqlunsw[.]localto[.]net:5037
nanejaojmnas[.]duckdns[.]org:4449
144[.]172[.]116[.]47:4449
209[.]99[.]187[.]200:4449
23[.]132[.]164[.]14:4449
80[.]234[.]41[.]242:7895

45[.]153[.]34[.]168:4449
85[.]17[.]116[.]195:5007
chember4osc[.]ydns[.]eu:1966
tcp[.]niceports[.]shop:8062
107[.]172[.]135[.]18:4449
194[.]156[.]79[.]203:4449
kdxnrjbjlg[.]localto[.]net:3491
biesmanscars[.]biz:4484
vmobqlunsw[.]localto[.]net:2902
192[.]109[.]200[.]221:4449
172[.]245[.]246[.]86:4489
server[.]activeworkshops[.]com:16614
155[.]103[.]71[.]146:4040
2[.]27[.]62[.]140:4449
103[.]27[.]157[.]18:4449
46[.]16[.]34[.]62:4449
serios[.]ydns[.]eu:4449
tcp[.]niceports[.]shop:20986
w122411-64886[.]portmap[.]host:64886
194[.]156[.]79[.]168:4449
109[.]242[.]9[.]165:5552
md[.]kleenghjah[.]com:4449
194[.]59[.]30[.]175:4449
tcp[.]niceports[.]shop:21408
23[.]160[.]168[.]174:4449
84[.]54[.]33[.]26:7766
109[.]243[.]144[.]45:4449
135[.]136[.]141[.]43:4449
185[.]69[.]122[.]235:10006
venom5555-39989[.]portmap[.]host:39989
logbox[.]hopto[.]org:7408
144[.]31[.]48[.]132:33215
95[.]164[.]55[.]74:4449
siruvise[.]ydns[.]eu:9067
ngfjhrdtge-51083[.]portmap[.]host:51083
185[.]182[.]65[.]150:20597
155[.]117[.]183[.]31:4449
203[.]202[.]232[.]59:4484

194[.]156[.]79[.]151:4449
24[.]157[.]199[.]215:8303
changeducation[.]net:4401
185[.]243[.]112[.]227:7766
vmobqlunsw[.]localto[.]net:4665
193[.]161[.]193[.]99:8471
tcp[.]niceports[.]shop:21408:4444
tcp://Chujwdupepolicji123-40502[.]portmap[.]host:40502
sirius[.]gldbrt[.]com:4949
leanna[.]ydns[.]eu:55540
geyvklsxge[.]localto[.]net:2008
193[.]161[.]193[.]99:1194
directindustry[.]duckdns[.]org:4449
45[.]153[.]34[.]146:4449
185[.]73[.]126[.]54:8848:8848
133[.]18[.]107[.]191:4449
104[.]239[.]66[.]128:7171
tcp[.]cloudpub[.]ru:18623
vmobqlunsw[.]localto[.]net:6514
91[.]236[.]186[.]56:20797:4449
193[.]161[.]193[.]99:8000
tcp[.]niceports[.]shop:22057
38[.]240[.]51[.]143:5555
siruvise[.]ydns[.]eu:1967
31[.]40[.]204[.]71:7070
server[.]activeworkshops[.]com:31166
tcp[.]niceports[.]shop:23059
185[.]114[.]206[.]45:4449
95[.]31[.]217[.]8:1337
38[.]240[.]51[.]74:4449
8xzlyusic3[.]localto[.]net:8933
amandabennette[.]freeddns[.]org:4400
surveillance[.]1qq[.]cam:4449
ixcube[.]ddns[.]net:56545
213[.]49[.]31[.]253:4449
afxwd[.]ddns[.]net:7070
162[.]217[.]248[.]219:4449
160[.]191[.]50[.]145:3311

77[.]91[.]65[.]188:7004
64[.]89[.]160[.]4:1602
178[.]16[.]55[.]23:1605
178[.]16[.]55[.]23:1602
185[.]196[.]8[.]145:8443
64[.]89[.]160[.]4:56001
196[.]251[.]107[.]104:1912
34[.]41[.]139[.]193:5202
5[.]181[.]80[.]130:38241
190[.]115[.]16[.]13:776
190[.]115[.]12[.]16:776
5[.]181[.]82[.]130:38241
77[.]91[.]65[.]188:7004
64[.]89[.]160[.]4:1602
178[.]16[.]55[.]23:1605
178[.]16[.]55[.]23:1602
185[.]196[.]8[.]145:8443
64[.]89[.]160[.]4:56001
196[.]251[.]107[.]104:1912
34[.]41[.]139[.]193:5202
5[.]181[.]80[.]130:38241
190[.]115[.]16[.]13:776
190[.]115[.]12[.]16:776
5[.]181[.]82[.]130:38241
91[.]236[.]186[.]56:20797

APPENDIX A: Script for appdata.dat file decryption

The following python script is made for fast XOR based symmetric decryption of the appdata.dat file used in SiriusQT server component:

```
#!/usr/bin/env python3
import sys
import json
from pathlib import Path

KS_HEX = "d5525220f116f6c5658b609fadd46c06f2f9413b345b454016e905bae5a69903d46399aa91c61ffe059cf38b3db57311044
KS = bytes.fromhex(KS_HEX)

def decode_appdata(path):
    raw = Path(path).read_bytes()

    if raw[:4] != b"SDB\x01":
        raise SystemExit("Invalid appdata.dat magic")

    enc = raw[8:]
    return bytes(b ^ KS[i % len(KS)] for i, b in enumerate(enc))
```

```
if len(sys.argv) != 3:
    raise SystemExit(f"Usage: {sys.argv[0]} appdata.dat output.json")

decoded = decode_appdata(sys.argv[1])

try:
    obj = json.loads(decoded.decode("utf-8"))
    Path(sys.argv[2]).write_text(json.dumps(obj, indent=2), encoding="utf-8")
except Exception:
    Path(sys.argv[2]).write_bytes(decoded)
```

DISCLAIMER: All technical information, including malware analysis, indicators of compromise and infrastructure details provided in this publication, is shared solely for defensive cybersecurity and research purposes. Group-IB does not endorse or permit any unauthorized or offensive use of the information contained herein. The data and conclusions represent Group-IB's analytical assessment based on available evidence and are intended to help organizations detect, prevent, and respond to cyber threats.

Group-IB expressly disclaims liability for any misuse of the information provided. Organizations and readers are encouraged to apply this intelligence responsibly and in compliance with all applicable laws and regulations.

This blog may reference legitimate third-party services such as Telegram and others, solely to illustrate cases where threat actors have abused or misused these platforms.

This material is provided for informational purposes, prepared by Group-IB as part of its own analytical investigation, and reflects recently identified threat activity.

All trademarks referenced herein are the property of their respective owners and are used solely for informational purposes, without any implication of affiliation or sponsorship.

Source: https://www.group-ib.com/blog/siriusqt-rat-hvnc-stealer/