RisePro Malware Analysis: New Version's C2 Communication

By ANY.RUN

Archived: 2026-04-05 19:17:38 UTC

RisePro is a malware-as-a-service info-stealer, first identified in 2022. Recently, we've detected a spike in it's activity and decided to conduct an investigation, which led to interesting findings.

RisePro is a well-documented malware, but we quickly realized that the network traffic patterns of our samples did not match the existing literature. It seemed like we had a new version on our hands.

Further analysis revealed that RisePro changed the way it communicates with C2 and that it has gained new capabilities — in particular, remote-control functions, making it capable of operating as a RAT.

This article will focus on this malware's new network communication patterns, but first, a quick refresher about what RisePro malware is.

What is RisePro malware?

RisePro, an information-stealing malware, was first detected by cybersecurity firms Flashpoint and Sekoia. It is distributed through fake cracks sites operated by the PrivateLoader pay-per-install (PPI) malware distribution service. It is designed to steal credit cards, passwords, and crypto wallets from infected devices.

RisePro is potentially based on the Vidar password-stealing malware and it employs a system of embedded DLL dependencies. RisePro's modus operandi includes fingerprinting the compromised system, writing stolen data to a text file, taking screenshots, and then bundling and sending this data to the attacker's server.

The PrivateLoader service, which distributes RisePro, is known for disguising malware as software cracks, key generators, and game modifications. It was first spotted by Intel471 in February 2022. Sekoia's findings indicate that RisePro shares significant code overlaps with PrivateLoader, suggesting a deeper connection between the two.

Like we said earlier, our analysis focuses on the recent changes in RisePro's C2 communication and network traffic patterns of its latest version, which differ drastically from previous iterations.

Traffic analysis of the new RisePro malware sample

There's a big change to highlight right of the bat. **Our sample uses custom protocol over TCP for communication.** This indicates a complete overhaul of the communication method, which previously transmitted instructions over HTTP.

Let's start our deep dive into this variant's communication patterns. Here's a screenshot of a network packet from ANY.RUN online malware sandbox, which was the starting point of our investigation:

Comparing encrypted (left) and decrypted (right) packet content

Upon examining the packet bytes (right column), it's evident that the traffic is encrypted, making it indecipherable. The first task, then, was to decrypt it.

Sekoia researchers have already cracked this encryption, so, to start, we decided to try and apply their decryption algorithm. Surprisingly, it successfully decrypted the data. This means the same encryption is still used.

The encryption algorithm is a basic substitution cipher followed by XOR with key 0x36. By Testing it with different ports we were able to find multiple keys. For example, the key for port 50500 is 0x36, and for port 50505 it is 0x79. Interestingly, opcodes take on different meanings depending on the port. In this article we will provide examples for port 50500.

Diving deeper in the packet analysis

But let's get back to the traffic analysis. Since we decrypted the TCP stream, we can begin to understand the structure of each packet.

Each packet has 3 blocks that follow a set pattern

In the image above, we see several packets (the first being the initialization packet). Three distinct blocks are noticeable, following a clear pattern. We can represent this structure as follows:

Packet structure

- The first 4 bytes, labeled as **magic**, are always repeated and determine the beginning of the packet.
- The next 4 bytes define the length of the data attached to the packet, labeled as **payload_len**.
- And, as you can see from the screen above, immediately following is **packet_type**.

During the analysis, we discovered the following **packet_types,** which represent various opcodes:

Packet type	Value	Payload	Description
SERVER_PING	0x2710	(OPTIONAL) text string	Default response, Keep-Alive (heartbeat)
CLIENT_PING	0x2711		Keep-Alive (heartbeat)
SERVER_INIT	0x2712	24 bytes string	Server Hello
SET_TIMEOUT	0x2713	Number string	Server/client timeout for action (e.g. upload)
CLIENT_REQUEST_FILE	0x2714	File name (string)	Request file from server

Packet type	Value	Payload	Description
SERVER_SEND_FILE	0x2715	File name, compressed file (zlib)	Used by server to send additional libraries
CLIENT_CONFIRM_IP	0x2716	Response string	IP receive confirmation
SERVER_SEND_MARKS	0x2717	JSON string	List of marks configs
CLIENT_CONFIRM_MARKS	0x2718	Response string	Marks receive confirmation
SERVER_SEND_GRAB_CONFIG	0x2719	JSON string	Settings and grabbers
CLIENT_CONFIRM_GRAB_CONFIG	0x271A	Response string	Settings receive confirmation
SERVER_SEND_LOADER_CONFIG	0x271B	JSON string	List of loader configs, includes urls and execution conditions
CLIENT_CONFIRM_LOADER_CONFIG	0x271C	Response string	Loader configs receive confirmation
SERVER_SET_FILE_FILTER	0x271D	JSON string	List of file filtration rules
CLIENT_CONFIRM_LOADER_EXECUTION	0x271E	Name from loader config	Confirmation of execution load target from particular config
CLIENT_SEND_FILE	0x271F	File name, response string, build id, compressed file (zip)	Exfiltrated files in archive with name representing geolocation and IP address
CLIENT_INIT	0x2720	(OPTIONAL) text string	Client Hello, optional authentication in format “{HWID}|{response string}”
SERVER_SEND_IP	0x2721	IP string	Used by server to send client’s public IP
CLIENT_SEND_UNKNOWN	0x2722		Mentioned in code, not used
SERVER_SEND_UNKNOWN	0x2723		Mentioned in code, not used
SERVER_SEND_HWID	0x2724	HWID string	Used by server to send HWID as step of HVNC maintenance
SERVER_SEND_FORCE_QUIT	0x272B		Force client to call ExitProcess(0)

It is evident that this is a client-confirmed protocol, as most messages include a CONFIRM response. From the table above we can see that the protocol supports functionalities like loading configuration settings, sending files, and more.

Examining various packets reveals that the payload is typically an encrypted UTF-8 encoded string. However, it’s worth noting that the payload length can be zero.

Moreover, there are two distinct packet types that deviate from the usual string payload: **CLIENT_SEND_FILE** and **SERVER_SEND_FILE**.

Packet_type **0x271F** (**CLIENT_SEND_FILE**) has this payload structure, represented here:

packet_type 0x271F (CLIENT_SEND_FILE)

And here's representation of **packet_type 0x2715** (**SERVER_SEND_FILE**):

packet_type 0x2715 (SERVER_SEND_FILE)

As you can see from the images above, these packets contain substructures in place of strings to handle file data.

Packet order

Having established the packet structure, we can now observe the typical sequence in which they arrive. If we were to illustrate the entire communication sequence in a flowchart, it would be represented as follows:

Communication flow of RisePro illustrated in a flow chart

The communication protocol with the Command and Control (C2) server is broken down into three main stages:

- **Initialization**: This is the first step where the client establishes a connection with the server and initializes the communication session.
- **Getting the configuration**: In this stage, the client retrieves configuration details from the server, which may include commands, operational parameters, or target information.
- **Performing stealer and loader functions**: Here, the client executes its intended malicious activities such as stealing data (stealer function) and confirming receipt of payloads (loader function).

There's also an optional 4th Stage – HVNC launch: it involves the initiation of Hidden Virtual Network Computing (HVNC), allowing for remote control without detection.

Let's delve into each stage one by one for a detailed understanding.

Stage 1: Initialization

The default initialization flow for the communication with the C2 server is as follows, with the dotted line indicating an optional packet:

Initialization flow

1. Communication begins with a **SERVER_INIT** packet following the establishment of the connection.
2. The client may send a **CLIENT_INIT** packet right after connecting, before the server sends its packet. If the client initiates with **CLIENT_INIT**, the server responds with a **SERVER_PING** by default.
3. The **SERVER_INIT** packet includes a session token, which is used to uniquely identify the session.
4. Subsequently, the server sends the public IP address of the victim to the client.
5. The client acknowledges the IP address by sending back a confirmation along with an additional string in its response.

With these steps, the connection initialization between the client and the server is completed.

Stage 2: Getting the configuration

The configuration stage involves the server sending configurations in a particular order, and the client sending back confirmations with additional payload.

Getting the configuration

The server sends the **marks_config**, **grab_config**, and **loader_config** in a strict sequence to set the malware's behavior. Having received the configurations, we can now examine what they entail.

A deeper look at the config

The first thing that comes from the server is marks config, shown below:

Screenshot of the marks_config

The configuration we're looking at likely dictates how the domain-related data, as presented, will be color-highlighted. This seems to correspond to the color coding of data within the admin panel. It's an unusual feature — the purpose of which is

not completely clear for the client.

Moving on, the server always sends a **grab_config**, which is illustrated in the screenshot below:

Screenshot of the grab_config

The grab_config specifies the data collection scope, the destination for the collected information, and the functions the malware will utilize.

For instance, it enables the malware to configure a proxy server on the victim's computer, initiate HVNC, and transmit data to Telegram (with **tg_ids** specifying the recipients of the message and **tg_token** being the bot token within Telegram). Additionally, the malware is capable of capturing a screenshot at the time of execution (**grab_screen**) and exfiltrating data from applications like Telegram and Discord.

Following this, we have the **loader_config**, as seen below:

Screenshot of the loader_config

Here are some noteworthy details from the configuration:

- **ld_geo**: This setting likely activates a geographical filter. If set, it probably checks for a specific country code, allowing the loader to execute only if there's a match.
- **ld_marks**: These are additional conditions that determine when the loader should be activated.
- **ld_name**: This is the identifier for the specific configuration.
- **ld_url**: This specifies the source URL from which the payload will be downloaded.

This configuration structure differs in new and old samples of this malware. It is noteworthy, that when the server is updated, older versions of the malware, such as earlier iterations of RisePro, will continue to function. However, they might ignore some of the new data or configuration values introduced in the updates.

Stage 3: Performing stealer and loader functions

At this stage in the process, the server issues a command specifying the data to be collected, and in response, the client compiles and sends back a .zip archive containing all the stolen data.

Network communication for performing stealer and loader functions

The server essentially sets the type of data to be collected.

Setting exfiltration scope and stealing data

Here’s an example of the rules for data exfiltration:

An example of data exfiltration rules

Here are some key aspects to note in these rules:

- **rule_collect_recursv**: This indicates that the malware will search through folders recursively, delving into subfolders to locate files.
- **rule_exceptions**: This defines specific locations or files that the malware should avoid.
- **rule_files**: This is a pattern or set of file extensions that the malware targets for theft.
- **rule_folder**: This specifies the path from which files, as defined by the environment configurations, will be extracted.
- **rule_name**: This is the internal identifier for the rule. There can be multiple such rules, as observed.
- **rule_size_kb**: This likely sets a maximum file size limit. Files larger than this specified value will not be collected.

Exfiltrated data

Upon receiving the configuration, the client steals specific data and sends it back in a zip archive. In our case, the contents of this archive were as displayed on the screenshot below:

Contents of the archive containing exfiltrated data

Packets that transmit this data have a set structure, which we can express as follows:

Structure of packets that transmit data

The structure of the packet for sending stolen data includes the country code, followed by an underscore (_), then the IP address, and finally the .zip extension. For instance, "**DE_127.0.0.1.zip**".

An additional name, formatted as described, accompanies the archive. This includes the response code and the build identifier, which specify which client is to process or merge the data.

This stage involves actions that are contingent on the specified configuration, like loader functions.

Performing loader functions is optional

If a loader config is provided, the client will download a file and execute it using scheduled tasks (schtasks). This indicates that the malware has loader functions.

Further details are encompassed in the "**CLIENT_CONFIRM_LOADER_EXECUTION**" packet. Following the execution, the client sends a confirmation back to the server, including the value of "**ld_name**." Above is an example illustrating how the client communicates with the server to download additional malicious code and the corresponding server response.

Referring to the flowchart above, the first packet contains the number **9**. This corresponds to **LD-name**, which is the identifier for the first loader configuration.

Stage 4: Optional HVNC launch

This new version of RisePro also possesses remote control capabilities, which means it can now function as a Remote Access Trojan. The ability to enable HVNC is included in the **grab_config**, as shown in the screenshot provided.

Use of HVNC is set to true in the grab_config

If HVNC is enabled, RisePro initiates another instance of itself, specifically to download a DLL and run a server for the remote control functionality.

Multiple TCP streams as seen in ANY.RUN

The screenshot above reveals an interesting aspect of the malware’s operation: communication occurs across multiple TCP streams.

Network communication involved in HVNC

- **First connection (process 2600)**: This includes all the previously discussed stages, such as initialization, configuration, and data exfiltration.
- **Two connections from process 2612**: These represent two distinct activities:

The first connection is for receiving a DLL module.

The second connection is for maintaining an HVNC server, which facilitates remote connections.

Stage 4.1: Requesting HVNC module

To understand how the HVNC connection is established, let’s examine the process as it occurs in the second TCP stream. This will provide insights into the steps and communications involved in initiating an HVNC connection. Using a flowchart, the process can be described as follows:

Initial stage of HVNC launch

Let’s explain what actually takes place step-by-step:

- **Client’s file request**: The client sends a request for a DLL file, including a string that specifies the file name.
- **Server’s response and file transmission**: The server acknowledges the request, sends a token for the session, and then transmits the requested file.

Having established the sequence, let’s examine the structure of these packets in pseudocode:

Packet structure in the HVNC communication sequence

Stage 4.2: Third connection

In the third connection, if the server initiates the communication, the process generally unfolds as follows:

Data transmission during the second stage of HVNC launch and maintaining conneciton

During the third connection, the communication involving HVNC is characterized by two main stages:

1. **Data transmission from server**: Initially, the server sends specific data related to the HVNC operation.
2. **Cyclic pinging**: Subsequently, to maintain the connection, the server periodically sends ping messages to the client.

Unfortunately, we weren't able to analyze the packet structure when someone connects to the victim using this system, so we can't provide specific details about that aspect of the communication process.

Data exfiltration

Having explored network communication patterns of RisePro, we can move on to examine the contents of the files sent by the malware. This will help us understand what data the malware is designed to collect and transmit.

We'll examine a file called information.txt first, shown below:

Information.txt file

This file contains various details. Here are some of the higlights:

- **Malware version**: Specifies the version of the malware.
- **Launch date**: The date when the malware was activated.
- **GUID**: Likely used to uniquely identify the computer.
- **Hardware ID**: A unique identifier for the hardware of the infected system.
- **Launch path**: The file path from where the malware was executed.
- **Temporary data storage folder**: A folder created by the malware to temporarily store stolen data.
- **Victim's computer data**: Information like IP address, locale, system details, and other typical computer specifications.
- **Hardware information**: Details about the video card, processor, RAM, etc.
- **Running processes**: Names and IDs of system processes, likely used to check if any antivirus software is active.
- **Registered software**: Lists software registered in the machine's registry.

In addition, the malware sends out stolen passwords in a separate file named passwords.txt. It is formatted rather elaborately:

passwords.txt file

Immediately noticeable is a conspicuous link to a Telegram support group associated with the malware's operation, likely provided for further assistance or instructions. The file also lists passwords that have been extracted from databases of browsers, email clients, and other software.

For each set of credentials, the following details are included:

1. URL of the Site: The web address for which the credentials are used.
2. Login: The username or login ID.
3. Password: The corresponding password.

Wrapping up: a look at the known versions

There are numerous versions of RisePro, and we have only analyzed one specific variant. Consequently, the details may vary across different versions.

As of November 22, 2023, the current version is labeled as 1.0. It appears that the versioning was reset to the beginning when the communication protocol underwent significant changes.

Additionally, it is noted on the malware's Telegram support channel that there are two main versions of this stealer: one written in C# and another in C++. The C++ version of the stealer is usually protected with VMProtect and is obfuscated to evade detection and analysis.

C++ version of the stealer is usually protected with VMProtect

This C# malware is obfuscated, potentially using Confuser.Core. You can see the C# version of RisePro in this sample.

C# version of RisePro is obfuscated, potentially using Confuser.Core

C++ version of RisePro can inject into processes. This behavior is evident in this task.

Injection behaviour

As usual, we'll leave you with some essential resources for detecting this malware and IOCs we've collected during our research:

IOCs

RisePro v0.9, C++ build, HVNC

Sample: https://app.any.run/tasks/01a74cc5-b571-4879-9104-e3f2383ba391/

SHA256: e95d8c7cf98dc1ed3ec0528b05df7c79bae2421ba2ad2b671d54d8088238f205

Files:

C:\Users\admin\AppData\Local\MaxLoonaFest1\MaxLoonaFest1.exe	e95d8c7cf98dc1ed3ec0528b05df7c79bae2421ba2ad2b671d54d808823

IP: 194[.]169.175.128

URL: http://91[.]92.245.23/download/k/KL.exe

RisePro v0.7, C++ build, loader

Sample: https://app.any.run/tasks/992ee8b9-b53a-489f-a97a-49798b125183/

SHA256: 973867150fd46e2de4b3d375d9c2d59eeda808a9dd1d137bd020b2f15c155ede

Files:

C:\Users\admin\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\K78MRVB5\KL[1].exe	f327c2b5ab1d98f0382a35cd78f694d487c74a7290f1ff7be53f42e23021e599

IP: 194[.]169.175.123

URL: http://91[.]92.245.23/download/k/KL.exe

RisePro v0.6, C# build

Sample: https://app.any.run/tasks/88f133ad-338b-43bb-a2fd-e093616219d5

SHA256: ba7f4474a334d79dd16cfb8a082987000764ff24c8a882c696e4c214b0e5e9cf

Files:

C:\Users\admin\AppData\Local\Temp\tempAVS1DYR2zldnwaG\sqlite3.dll	0c7cd52abdb6eb3e556d81caac398a127495e4a251ef600e6505a8

IP: 194[.]169.175.128

RisePro v0.9, C++ build, C# injector

Sample: https://app.any.run/tasks/d34ad531-7b30-46cb-922a-718e4bd6a9d8/

SHA256: D440EEB8FD204EF2B3845894FE4E256E6505796B75FE5201CFFA7F5453C2FB5F

Files:

C:\Users\admin\AppData\Local\LegalHelper130\LegalHelper130.exe	D440EEB8FD204EF2B3845894FE4E256E6505796B75FE5201CFFA7

IP: 194[.]49.94.53

RisePro botnet version, communication over TCP:50505

Sample: https://app.any.run/tasks/f841e850-d97a-4395-93cb-c2dff7e7bf7e/

SHA256: 4435DA81D8BC840408AFED9E993B3F0CC1AA08FF1CD03BBEC609379517EC1379

Files:

C:\ProgramData\WinTrackerSP\WinTrackerSP.exe	7F17D3D47F053498A3EFECAB532932DCC8018E3EE0DA60FB090BE
C:\Users\admin\AppData\Local\Temp\tmpSTLpopstart\stlmapfrog	(encrypted json, contains start timestamp and IP)
C:\Users\admin\AppData\Local\Temp\tmpSTLpopstart\todelete	(json with file paths)

IP: 194[.]169.175.128

SIGMA

```
 title: RisePro Rule

id: aba15bdd-657f-422a-bab3-ac2d2a0d6f1c

status: experimental

description: Detects RisePro malware

author: ANY.RUN

date: 2023/11/17

tags:

    - windows

    - RisePro

logsource:

    category: file_event

    product: windows

detection:

    selection:

              TargetFilename|regex:

           - "(?i)\\\\AppData\\\\Local\\\\Temp\\\\.*\\\\passwords\\.txt$"

           - "(?i)\\\\AppData\\\\Local\\\\Temp\\\\.*\\\\information\\.txt$"

    condition: selection

level: medium
```

YARA

We've created a YARA rule to detect these updated versions of RisePro. You can find it in our GitHub.

TCP stream decoder (python script)

For further investigation, we've prepared for you a script, that can be used to decrypt and parse the TCP stream to a JSON file. This allows for easier visualization and processing of RisePro communication. The script can be found in our GitHub

SURICATA Rule structure

After detecting RisePro traffic in our sandbox environment, we shared our insights on network rule configurations with the Emergency Threats community. You can view the thread discussing these network rules with the ET community here.

The Suricata rules are defined by multiple conditions:

Conditions in the rule	Value	Description
tcp $HOME_NET any -> $EXTERNAL_NET ![80,443,445,5938]	tcp	TCP protocol
	$EXTERNAL_NET	Direction to external network
	![80,443,445,5938]	Unused port exceptions
dsize:>1100;	1100	TCP packet payload size
content:"|00 1F 27 00 00|"; offset:7; depth:5	00	Limit uploaded file length values to three bytes
	1F 27 00 00;	Packet type CLIENT_SEND_FILE

Suricata IDS rules for detecting RisePro are available at Emerging Threats — Suricata Rules. Relevant rule IDs include 2046267, 2046269, 2046268, 2046266, 2046270, and 2049060.
