# A Full Analysis of the Pure Malware Family: Unique and Growing Threat

By khr0x and Jane

Published: 2024-01-16 · Archived: 2026-04-05 22:30:36 UTC

In this article, we're analyzing one of the most unusual crypters— **PureCrypter**, and a multifunctional stealer — **PureLogs**. We'll look at several examples and identify patterns among Pure-malware families, and also explain how to detect PureCrypter and PureLogs.

## Why did we decide to undertake this analysis?

While analyzing Public Submissions, we came across several interesting samples. We were intrigued by unusual traffic that showed signs of encryption operations on executable files with short keys, as well as TCP connections with high entropy in the connections.

Inside, all samples looked different from other malware and were very similar to each other. Through network analysis, we found a couple of articles dedicated to PureCrypter and the family, which shed light on this group, but we wanted to add our insights and combine all the information in one place.

Our objectives were:

- Study the distribution system
- Investigate the distinctive features of PureCrypter and PureLogs
- Develop detection methods for PureCrypter and PureLogs
- Examine the traffic

The distribution of PureCoder products began in March 2021, according to information provided by the developer on the the malware's old website.

Information about the service on the old website

On the main page of Pure's current website, there is a message stating that the software is used for educational and penetration testing purposes.

The website lies about educational and pentesting nature of the software

However, it's worth noting that we observe a trend where the code sold is actually being used for malicious purposes. Here are examples on services that tell us about the distribution of these products along with other malware:

- ANY.RUN Submissions (PureCrypter, PureLogs)
- Abuse.ch Bazaar

Telegram update

Pure's update notes tell us that since March 2023, it is also sold through a Telegram bot. Telegram bots make purchasing malware more automated and anonymous. Bot usage shows that the author is developing the service, exploring new channels, and scaling up.

Here are all the products that this group distributes under the guise of "educational purposes":

Pure products

Despite the claim that these products are distributed for educational purposes, the presence of silent miners, botnets, and hidden HVNC seems odd.

Comments and ratings on Pure's website reveal high demand — every month there are at least couple of purchases.

Comments and reviews on Pure's website

We attempted to follow the purchase flow and found that users are to make a cryptocurrency payment In Bitcoin. The payment page offers several Bitcoin wallets. These wallets are likely part of a Bitcoin mixer. The activity in these wallets started between **May 19-26, 2023**, and as of the writing of this article, one of them already had **250 transactions amounting to $32,000** (see it on Blockhain.com).

Details about the Cryptocurrency wallet

So far, we've established that there is a wide range of Pure malware, it is popular, and it has existed for several years. Now, let's move on to the technical analysis of the Pure family.

## Staged and Stage-less loader

The deployment of products from the Pure family usually begins with a loader that includes both Staged and Stage-less payloads. Let's analyze the behavior using **PureCrypter** as an example.

### What is PureCrypter?

PureCrypter is a crypter (or obfuscator), as its name suggests, that has a set of algorithms for data obfuscation and encryption. In combination, they hide malware from antivirus programs and also make it difficult to analyze for analysts.

Behaviour flow of PureCrypter

As seen in the diagram above, the loader has two stages: Staged and Stage-less payload. The decrypted resources contain libraries such as **Protobuf-net** and **Costura**. Using **Protobuf-net**, data are deserialized, forming a configuration with the compressed malware. Ultimately, after decompression, the malware is launched with parameters from the configuration in a new process.

Let’s examine each variant separately.

### Staged Loader

| | |
|---|---|
| SHA256 | 3ACD90196DCF53DD6E265DC9C89B3CB0C47648A3B7AC8F226C6B4B98F39F2FC8<br>View the task |

The static analysis of the sample under examination reveals that it’s written in .NET:

Information from DIE

After analyzing the sample in the ANY.RUN sandbox, we determined that a file with an .mp4 extension is downloaded:

Payload download

Additionally, we found examples downloading payloads with extensions like .vdf, .mp3, etc. **This is another characteristic of the loader — to download files with legitimate extensions.**

In the image above, the downloaded file is not an actual .mp4 file, but an encrypted payload. It’s challenging to determine the encryption method right away, so we will analyze its internals.

Code analysis in DnSpy allowed us to establish that the downloaded **payload is encrypted with an XOR operation**, with a key length of 3 bytes. You can see the overall scheme of payload downloading and decryption below:

Source code

To decrypt the file, we will use our CyberChef recipe.

XOR with the key ”335” in Cyberchef

As evident from the screenshot above, the downloaded and decrypted file is an executable or a library.

Additionally, this type of encryption is not the only possible method and can be substituted with reverse encryption and others.

Now, let’s move on to the analysis of the stage-less loader.

### Stage-less Loader

| | |
|---|---|
| SHA256 | 5030BC30C14139D9C48DC4CD175DE6C966E83A9059035D18AF33DDA06F2541AB<br>View the task |

Unlike the Staged payload, the examined Stage-less payload is protected by SmartAssembly, which you can see in DIE:

Information from DIE

In the stage-less loader, the payload is stored in a resource in an encrypted form:

Resource on board

First, the resource is decrypted using AES with embedded keys (KEY: dd2e7fe3fd9cb1b2f91a16460c8acb5b and IV: 80f3f9712e01f98fab92ab84ec40a8e5) and then decompressed:

Rijndael Algorithm

We will use a CyberChef recipe for decryption.

Cyberchef – **AES+decompress**

As a result, we get a .NET Assembly without executable code but with an encrypted resource, which is also decrypted and loaded into modules:

**Nchya** resource

The decryption of the second resource is done using 3DES with the key KEY “68433890991609093ead30a9d75c39db” and IV “4A64DD85048433D7” (CBC model).

Let’s use a CyberChef recipe to decrypt this resource:

Cyberchef – **TripleDES**

After decryption, we obtain an executable file or library, similar to the case with the Staged Loader.

Now, let’s move on to the analysis of the obtained files.

## PureCrypter

Comparing the entry points of Staged and stage-less **PureCrypter**, we see that they are identical. From this, we can conclude that they are essentially the same.

Here’s EntryPoint of **PureCrypter**:

EntryPoint of PureCrypter

**PureCrypter** can carry two types of payloads – 3rd party malware or its own proprietary product, **PureLogs**. Let's examine each option separately.

### 3rd party malware (AgentTesla)

The program begins decrypting and loading the .NET Assembly resource, similar to the stage-less process. This happens in an identical manner – using **AES (Rijndael) encryption**.

Resource decryption with AES (Rijndael)

We can use this CyberChef recipe for decryption.

CyberChef **AES+inflate**

After decrypting with the AES algorithm, the program takes this resource and proceeds to the second stage of its decryption.

The first action the program performs is parsing the data of the header. The first 30 bytes are the length of the data, and the next 10 bytes are the XOR key:

The header

From the header, a license and key are obtained. The data can be decrypted using this CyberChef recipe. A similar string was already seen in **ZGRat**.

XOR

After calculating the data, the program selects a method of decryption. In our case, it is AES.

Encryption selection

In AES, **IV** is used, which is the XOR key 7C685406ED380C74532A9488BA58083D, and the KEY is the last 32 bytes in the decrypted header b2912dfe705af74a11e7d2bf3786103116adee71727185419bf7c4d7f986bd4c.

We can decrypt the data using a CyberChef recipe.

As a result of the decryption, we obtain a .NET Assembly with a set of resources and encrypted strings before the MZ header.

Resources after decryption

The header looks as follows:

- 4 bytes at the beginning and end (for calculation purposes)
- 1 byte for the message size highlighted in red
- And the message itself afterwards.

The header

The messages can be decrypted using the following CyberChef recipe:

Encoding and XOR

The resource zJSLu is decrypted in the same way — using AES and decompression. It contains strings that will be used later (if the flag is set in the configuration).

The zJSLu resource, once decrypted (subject to serialization)

The malware uses **protobuf** for deserializing data, which is taken from the Issal resource after prior decompression.

Deserialization

At this point, having decrypted all the resources, we arrive at the program's main function. But before starting the analysis of the main function, let's take a look at what the **PureCrypter** builder interface looks like:

PureCrypter builder

From the screenshot, we can see a large set of functions, including anti-debugging, anti-deletion, and others. Let's go through the main code and see what techniques are used in this version of the crypter.

Code with configuration analysis and checks.

In the version under study, the check for the presence of a virtual machine is disabled, but the functionality is present in the code of the program. It includes:

- CheckRemoteDebuggerPresent to find a debugger;
- Checks for the presence of the Sandboxie virtual environment by searching for the loaded library sbiedll.dll;
- Executes a WMI query "select * from Win32_BIOS" to check the BIOS version;
- Executes a WMI query "select * from Win32_ComputerSystem" and looks for one of the substrings "Microsoft|VMWare|Virtual" in the results;
- Checks the width and height of the monitor screen, which should be more than 1024 and 768 pixels, respectively;
- Checks whether the program is running on a 64-bit OS, as most modern operating systems are 64-bit;
- Compares the current username with one from a list.

Other capabilities include:

1. A feature to reset the network interface by executing the command “cmd /c ipconfig /release,” presumably to prevent security tools or antivirus software from communicating with their servers. In the sample analyzed, this feature is disabled.

2. Ability to use a mutex to prevent the launch of a duplicate copy. In this instance, the mutex is named “Gjrstoo,” but this option is not active.

3. A function to check if it is running with administrator privileges and to restart with the necessary permissions if needed. Moreover, the malware uses the command “set-mppreference -exclusionpath ” to add the entire “C:” drive to the antivirus exclusions. This function is also disabled.

4. A delay execution feature, where the delay occurs N times for 1 second each.

5. Capability to execute an arbitrary PowerShell command passed in Base64 via the “-enc” parameter. This function is also disabled.

6. Displaying a fake error message, but this feature is also turned off.

7. Ability to establish persistence in the system through Run registry keys or the Startup directory.

The purpose of using “ipconfig /renew” is unclear, but the functionality exists (it allows the release of all dynamic IP addresses assigned to the computer using a DHCP server).

Subsequently, the payload is loaded, which can be downloaded from the internet (as indicated by the HTTP Client), or from a resource (as in our example).

Injection options

1. Ordinary library loading
2. Decryption of the resource followed by loading
3. Unclear — possibly, decryption is executed here.

Subsequently, the malware reverses bytes and is decompressed using the same GZIP, the recipe for which can be found here.

Cyberchef – Gunzip

After decompression, PureCrypter creates a new process:

And injects code into this process.

This is how malware is commonly distributed, often including stealers and RATs. Now let’s move on to analyzing PureLogs, which can distribute PureCrypter.

## PureLogs Loader

**PureLogs** malware is typically distributed by a loader covered by the NET Reactor protector. **PureLogs** is a small library that is involved in data theft. Usually, the library is loaded by the loader from a C2 server.

Analysis of the loading traffic revealed that in the first connection an encrypted message is sent, and an encrypted response is received. All of this occurs within the loader.

First connection in the loader

Both messages within the first connection are encrypted in the same way, but the response has an additional layer of serialization and undergoes re-encryption with byte reversal.

First, the data is compressed and then encrypted using 3DES with a key (which is stored in the loader's resources, along with the IP data and client ID). However, the key itself is encrypted using md5Crypto, resulting in the hash '9F4D71CF2393253FB5324C6731B962F8'.

After encryption, the program sends this message to the server. The process begins with sending 4 bytes indicating the size of the message, followed by the message itself.

A complete decryption is presented here:

Decryption key

Now let's consider the received message. After decryption, the data undergoes deserialization and is then re-encrypted again (3DES+GZIP), similar to the initial process. However, at the end, a Reverse bytes operation is applied.

TripleDES+GZIP (you can immediately remove the first 4 bytes of length or use DROP)

After deserialization: TripleDES+GZIP+Reverse

As a result, we obtain our library, which is responsible for stealing data and then sending it onward. Let's take a closer look at it.

## PureLogs

**PureLogs** is a multi-functional stealer. Like **PureCrypter**, **PureLogs** has obfuscation and obfuscation methods that complicate its analysis. But what's really interesting is its network traffic, which we will discuss in this section.

Like other samples of the Pure family, PureLogs is sometimes confused with ZGRat — we are going to clear up this misunderstanding in this article.

Let's get to the analysis.

Looking at the class library, we immediately see a class called PlgCore (We assume this stands for PureLogsCore).

ClassLibrary1

Serialized data enters the library as an argument from a resource, which the loader has loaded from the C2 server.

Configuration data

Inside, they are deserialized and stored as configuration.

Configuration

Next, the library iterates through a vast number of functions and collects data from the system:

- Browser data, including extensions
- Data about Crypto Wallets
- Complete information about the user
- Full information about the PC configuration

Below is an example of some of the system data.

System data

Next, all the collected data is serialized. Following the same principle, the data is encrypted before transmission: compression and 3DES encryption using a pre-existing key. Now, all the data ready for transmission is transformed, and the final connection is made to send the gathered data.

Data transmission (first 4 bytes indicate size, followed by the message)

Then, there are three transmissions: the first and last are hashes of the data, and the second one contains the actual data.

To decrypt the traffic, we can use this CyberChef recipe.

Decryption of the first message (the last one is identical)

Decryption of the data

And with this, we have dissected the traffic and the operation of PureLogs. Now, let's consider another malware variant from the Pure family — a miner.

## PureMiner

While examining samples on other services with detections for **PureLogs** and **PureCrypter**, we came across several samples that didn't appear to be like either but exhibited a strikingly similar signature.

**Firstly, the traffic they generated followed an identical pattern** (first 4 bytes for length, followed by the remaining bytes for data). What's even more intriguing is that they were encrypted in the same fashion (using **3DES** encryption with a key that was similarly encrypted through **MD5Crypto**).

**Secondly, there were similarities in code behavior and module loading**, such as the use of the **proto-buf** module for processing configuration data.

Thirdly, the code structure and its resemblance to PureCrypter and PureLogs code were notable.

Lastly, the configurations showed similarities in their structure.

The diagram below shows this resemblance to PureLogs. It involves the transmission and reception of data from the C2 server (with the data being encrypted using 3DES).

Based on all of this information, we have decided to conduct an investigation into the discovered sample:

| | |
|---|---|
| SHA256 | A20F2623022BC0D5BDC49B235736CC791A3392198D7A601B2478C1974D5D9F17<br>View the task |

The first thing we noticed during our analysis was the sample's behavior when executed with administrative privileges (we will see why we executed the sample with admin privileges later on).

The process of restarting using cmd

After the restart, the program creates a scheduled task in Task Scheduler to launch its copy at **%APPDATA%/HResult/TypeId.exe**. Following this, there is an injection into a new legitimate process.

The process is launched under Task Scheduler

After analyzing the behavior, we proceeded with an in-depth analysis of the sample. We discovered its configuration, which utilizes **proto-buf**.

Configuration

Additionally, there is an executable file that appears to be responsible for executing commands from the C2 server. In the strings, there is a mention of the XMRIG miner. We suspect that this is a distributor of the miner, which runs it quietly.

Strings from the miner

We decided to decrypt the traffic to understand what information is being transmitted and what we receive in response. We managed to obtain the decryption key. It was no surprise that the traffic was encrypted using the same 3DES encryption, and the first 4 bytes represent the length, which we remove:

Decrypted traffic

The decryption recipe is available here. And here's the link to a server response.

Response from the server

And with this, we have found another malware from the Pure family — a miner. PureMiner collects information about the system and sends it to C2. After this, it receives a response with mining instructions.

## Let’s wrap up the analysis

To summarize — this was one of the most comprehensive investigations we’ve done so far in ANY.RUN. We analyzed a widely popular and rapidly spreading malware family — Pure — and even uncovered a new variant — a miner.

Pure tools masquerade as legitimate software created for “educational purposes”. But analysis of the code clearly shows that it is a powerful malicious tool. Recently, the creators began distributing it through a telegram bot, which indicates that they are scaling an operation. Currently, Pure receives at least a couple of orders every month, but it’s highly likely that its popularity will start skyrocketing in the near future.

We hope that this analysis helped you better understand how to reverse and detect malware from the Pure family, so if that happens, you’ll be well prepared.

If you have any information to add about Pure — we’d love to hear it. Let’s discuss in the comments below. And as always, make sure to share your thoughts about the article.

## About ANY.RUN

ANY.RUN is an interactive malware analysis sandbox that streamlines the work of SOC and DFIR teams. Our service is trusted by 300,000 professionals worldwide who use it to investigate both emerging and persistent threats.

Request a free trial of ANY.RUN for 14 days to explore all the features we offer.

Request demo →

## MITRE ATT&CK Matrix

| Tactics | Techniques | Description |
|---|---|---|
| Defense Evasion | T1140 - Deobfuscate/Decode Files or Information | Deobfuscate/Decode resourses and files |
| Discovery | T1082 - System Information Discovery | Pure-malware discovery system information |
| | T1083 - File and Directory Discovery | PureLogs discoveries files for stealing |
| Collection | T1119 - Automated Collection | Collect information |
| | T1005 - Data from Local System | Search local system sources for stealing |
| Command and Control | T1071.001 - Application Layer Protocol:Web Protocols | Connection and delivery |

## IOCs

### PureCrypter

#### FIle

0f60f086665fd4d442821851c878c21b

| | |
|---|---|
| MD5 | 0f60f086665fd4d442821851c878c21b |
| SHA256 | 3acd90196dcf53dd6e265dc9c89b3cb0c47648a3b7ac8f226c6b4b98f39f2fc8 |
| SHA1 | a4d4f31fb794bbf59be542f493aea9f9e3857d4 |

**Dropped file**

| | |
|---|---|
| Path | C:\Users\admin\AppData\Roaming\ydVSL\ydVSL.exe |
| SHA256 | 3acd90196dcf53dd6e265dc9c89b3cb0c47648a3b7ac8f226c6b4b98f39f2fc8 |

**Connections**

- 5[.]181.80.126

**URLs**

- Http[x]://5.181.80.126/Hjysa.mp4

**FIle**

QUOTATION_NOVQTRFA00541·PDF.scr

**FIle**

0f60f086665fd4d442821851c878c21b

| | |
|---|---|
| MD5 | 83999a2ce0109ea4adbecb3a96744e8c |
| SHA256 | 5030bc30c14139d9c48dc4cd175de6c966e83a9059035d18af33dda06f2541ab |
| SHA1 | 4b94f4b23b157c7ae2df54e251cd4d22c683134d |

**Domain**

- gator3220.hostgator[.]com

## PureLogs

**FIle**

RH2023-17.exe

| | |
|---|---|
| MD5 | a7c14a39a5ee93ca25ab793be06c1478 |
| SHA256 | e5b27dc1672088a5a584467511a02844d45f4eb6af92a96373c803fd3dc5e6b7 |
| SHA1 | c9eb61977fa0fd1bf1c9e7175a0088289e6b9bbd |

**Dropped file**

| | |
|---|---|
| Path | C:\Users\admin\AppData\Roaming\Xokmrjn.exe |
| SHA256 | e5b27dc1672088a5a584467511a02844d45f4eb6af92a96373c803fd3dc5e6b7 |
| Path | C:\Users\admin\AppData\Local\Temp\Costura\1485B29524EF63EB83DF771D39CCA767\64\sqlite.interop.dll |
| SHA256 | 5f0e72e1839db4aa41f560e0a68c7a95c9e1656bc2f4f4ff64803655d02e5272 |

**Connections**

- 91[.]92.120.119

**Domain**

- Teleturismo[.]it

### PureMiner

**FIle**

491310d10c0ea2d217c90a2403c20bea

| | |
|---|---|
| MD5 | 491310d10c0ea2d217c90a2403c20bea |
| SHA256 | a20f2623022bc0d5bdc49b235736cc791a3392198d7a601b2478c1974d5d9f17 |
| SHA1 | 5bd371ae2edc0c2cf926e1543e4cdd7d92c83577 |

**Dropped file**

| | |
|---|---|
| Path | C:\Users\admin\AppData\Roaming\HResult\TypeId.exe |
| SHA256 | a20f2623022bc0d5bdc49b235736cc791a3392198d7a601b2478c1974d5d9f17 |

**Connections**

- 91[.]92.240.95

**Domain**

- Farmjo[.]mine.nu

## More Submissions

https://app.any.run/tasks/f972efd3-c053-42c2-a2d4-eade0f40acfb/

https://app.any.run/tasks/c4344ee1-bcd6-438f-9aba-f13c1c3dcca9/

https://app.any.run/tasks/629232cd-67e4-4f3b-880d-34c3675931a0/

https://app.any.run/tasks/50294ac2-f3c1-43bd-9bec-0527fb1b8443/

ANY.RUN malware analyst

**khr0x**

I'm 21 years old and I work as a malware analyst for more than a year. I like finding out what kind of malware got on my computer. In my spare time I do sports and play video games.

ANY.RUN writer and network traffic analyst

**Jane**

I'm ANY.RUN ambassador and a real network traffic numismatist. I also love penguins and tortoises. My motto is to do good and throw it into the sea.

khr0x

Malware analyst at ANY.RUN

I'm 21 years old and I work as a malware analyst for more than a year. I like finding out what kind of malware got on my computer. In my spare time I do sports and play video games.

Jane

Leading network traffic analysis expert at ANY.RUN

I'm ANY.RUN ambassador and a real network traffic numismatist. I also love penguins and tortoises. My motto is to do good and throw it into the sea.

---