



# PassiveNeuron: a sophisticated campaign targeting servers of high-profile organizations

Georgy Kucherin : : 10/21/2025

---



## Authors

-  Georgy Kucherin
-  Saurabh Sharma

## Introduction

Back in 2024, we [gave a brief description](#) of a complex cyberespionage campaign that we dubbed “PassiveNeuron”. This campaign involved compromising the servers of government organizations with previously unknown APT implants, named “Neursite” and “NeuralExecutor”. However, since its discovery, the PassiveNeuron campaign has been shrouded in mystery. For instance, it remained unclear how the implants in question were deployed or what actor was behind them.

After we detected this campaign and prevented its spreading back in June 2024, we did not see any further malware deployments linked to PassiveNeuron for quite a long time, about six months. However, since December 2024, we have observed a new wave of infections related to PassiveNeuron, with the latest ones dating back to August 2025. These infections targeted government, financial and industrial organizations

located in Asia, Africa, and Latin America. Since identifying these infections, we have been able to shed light on many previously unknown aspects of this campaign. Thus, we managed to discover details about the initial infection and gather clues on attribution.

## SQL servers under attack

While investigating PassiveNeuron infections both in 2024 and 2025, we found that a vast majority of targeted machines were running Windows Server. Specifically, in one particular infection case, we observed attackers gain initial remote command execution capabilities on the compromised server through the Microsoft SQL software. While we do not have clear visibility into how attackers were able to abuse the SQL software, it is worth noting that SQL servers typically get compromised through:

- Exploitation of vulnerabilities in the server software itself
- Exploitation of SQL injection vulnerabilities present in the applications running on the server
- Getting access to the database administration account (e.g. by brute-forcing the password) and using it to [execute malicious SQL queries](#)

After obtaining the code execution capabilities with the help of the SQL software, attackers deployed an ASPX web shell for basic malicious command execution on the compromised machine. However, at this stage, things did not go as planned for the adversary. The Kaspersky solution installed on the machine was preventing the web shell deployment efforts, and the process of installing the web shell ended up being quite noisy.

In attempts to evade detection of the web shell, attackers performed its installation in the following manner:

1. They dropped a file containing the Base64-encoded web shell on the system.
2. They dropped a PowerShell script responsible for Base64-decoding the web shell file.
3. They launched the PowerShell script in an attempt to write the decoded web shell payload to the filesystem.

As Kaspersky solutions were preventing the web shell installation, we observed attackers to repeat the steps above several times with minor adjustments, such as:

- Using hexadecimal encoding of the web shell instead of Base64
- Using a VBS script instead of a PowerShell script to perform decoding
- Writing the script contents in a line-by-line manner

Having failed to deploy the web shell, attackers decided to use more advanced malicious implants to continue the compromise process.

## Malicious implants

Over the last two years, we have observed three implants used over the course of PassiveNeuron infections, which are:

- Neursite, a custom C++ modular backdoor used for cyberespionage activities
- NeuralExecutor, a custom .NET implant used for running additional .NET payloads
- the Cobalt Strike framework, a commercial tool for red teaming

While we saw different combinations of these implants deployed on targeted machines, we observed that in the vast majority of cases, they were loaded through a chain of DLL loaders. The first-stage loader in the chain is a DLL file placed in the system directory. Some of these DLL file paths are:

- C:\Windows\System32\wlbsctrl.dll
- C:\Windows\System32\TSMSISrv.dll
- C:\Windows\System32\loci.dll

Storing DLLs under these paths has been beneficial to attackers, as placing libraries with these names inside the System32 folder makes it possible to automatically ensure persistence. If present on the file system, these DLLs get automatically loaded on startup (the first two DLLs are loaded into the `svchost.exe` process, while the latter is loaded into `msdtc.exe`) due to the employed [Phantom DLL Hijacking technique](#).

It also should be noted that these DLLs are more than 100 MB in size — their size is artificially inflated by attackers by adding junk overlay bytes. Usually, this is done to make malicious implants more difficult to detect by security solutions.

On startup, the first-stage DLLs iterate through a list of installed network adapters, calculating a 32-bit hash of each adapter's MAC address. If neither of the MAC addresses is equal to the value specified in the loader configuration, the loader exits. This MAC address check is designed to ensure that the DLLs get solely launched on the intended victim machine, in order to hinder execution in a sandbox environment. Such detailed narrowing down of victims implies the adversary's interest towards specific organizations and once again underscores the targeted nature of this threat.

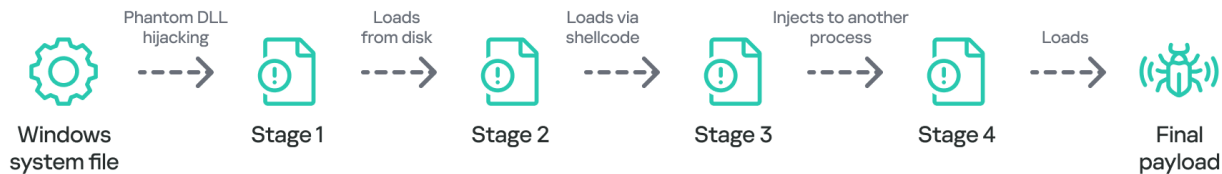
Having checked that it is operating on a target machine, the loader continues execution by loading a second-stage loader DLL that is stored on disk. The paths where the second-stage DLLs were stored as well as their names (examples include `el_scorewmyc.dll` and `wellgwlserejzuai.dll`) differed between machines. We observed the second-stage DLLs to also have an artificially inflated file size (in excess of 60 MB), and the malicious goal was to open a text file containing a Base64-encoded and AES-encrypted third-stage loader, and subsequently launch it.

```
353608|QThUHJ2FHx4gljzGntjEw2B8ldjq/gbGjui+KpmeUzLQQA5LND0lNJcx4y3Li
923464|0Z8DRfIj3IoH94KDV7sV+SvB5aX1RIkYUwjRSZRMAtcWJ15Uf1CEuxl0cP57Z
```

Snippet of the payload file contents

This payload is a DLL as well, responsible for launching a fourth-stage shellcode loader inside another process (e.g. `WmiPrvSE.exe` or `msiexec.exe`) which is created in suspended mode. In turn, this shellcode loads the final payload: a PE file converted to a custom executable format.

In summary, the process of loading the final payload can be represented with the following graph:



### Final payload loading

It is also notable that attackers attempted to use slightly different variants of the loading scheme for some of the target organizations. For example, we have seen cases without payload injection into another process, or with DLL obfuscation on disk with VMProtect.

## The Neursite backdoor

Among the three final payload implants that we mentioned above, the Neursite backdoor is the most potent one. We dubbed it so because we observed the following source code path inside the discovered samples: `E:\pro\code\Neursite\client_server\nonspec\mbedtls\library\ssl_srv.c`. The configuration of this implant contains the following parameters:

- List of C2 servers and their ports
- List of HTTP proxies that can be used to connect to C2 servers
- List of HTTP headers used while connecting to HTTP-based C2 servers
- A relative URL used while communicating with HTTP-based C2 servers
- A range of wait time between two consecutive C2 server connections
- A byte array of hours and days of the week when the backdoor is operable
- An optional port that should be opened for listening to incoming connections

The Neursite implant can use the TCP, SSL, HTTP and HTTPS protocols for C2 communications. As follows from the configuration, Neursite can connect to the C2 server directly or wait for another machine to start communicating through a specified port. In cases we observed, Neursite samples were configured to use either external servers or compromised internal infrastructure for C2 communications.

The default range of commands implemented inside this backdoor allows attackers to:

- Retrieve system information.
- Manage running processes.
- Proxy traffic through other machines infected with the Neursite implant, in order to facilitate lateral movement.

Additionally, this implant is equipped with a component that allows loading supplementary plugins. We observed attackers deploy plugins with the following capabilities:

- Shell command execution
- File system management
- TCP socket operations

## **The NeuralExecutor loader**

NeuralExecutor is another custom implant deployed over the course of the PassiveNeuron campaign. This implant is .NET based, and we found that it employed the open-source ConfuserEx obfuscator for protection against analysis. It implements multiple methods of network communication, namely TCP, HTTP/HTTPS, named pipes, and WebSockets. Upon establishing a communication channel with the C2 server, the backdoor can receive commands allowing it to load .NET assemblies. As such, the main capability of this backdoor is to receive additional .NET payloads from the network and execute them.

## **Tricky attribution**

Both Neursite and NeuralExecutor, the two custom implants we found to be used in the PassiveNeuron campaign, have never been observed in any previous cyberattacks. We had to look for clues that could hint at the threat actor behind PassiveNeuron.

Back when we started investigating PassiveNeuron back in 2024, we spotted one such blatantly obvious clue:

```

Cyber\0020обфускатор_2q(Cyber\0020обфускатор_DR
Cyber\0020обфускатор_aP(Cyber\0020обфускатор_AO
Cyber\0020обфускатор_co(Cyber\0020обфускатор_DR
Cyber\0020обфускатор_eO(Cyber\0020обфускатор_AC
Cyber\0020обфускатор_EQ() : byte[] @06000113
Cyber\0020обфускатор_fp(Cyber\0020обфускатор_qM
Cyber\0020обфускатор_Gm(object) : void @060000FE
Cyber\0020обфускатор_Hm(Cyber\0020обфускатор_AC
Cyber\0020обфускатор_Jp(Cyber\0020обфускатор_AO
Cyber\0020обфускатор_km(Cyber\0020обфускатор_rK
Cyber\0020обфускатор_KR() : string @06000112
Cyber\0020обфускатор_IP(byte[], out byte[]) : void @060
Cyber\0020обфускатор_mQ(Cyber\0020обфускатор_AC
Cyber\0020обфускатор_O(byte[], byte[]) : void @060001C
Cyber\0020обфускатор_Oq() : string @06000111
Cyber\0020обфускатор_Pm(Cyber\0020обфускатор_AC
Cyber\0020обфускатор_R(byte[]) : void @0600010F

```

#### Function names found inside NeuralExecutor

In the code of the NeuralExecutor samples we observed in 2024, the names of all functions had been replaced with strings prefixed with “Cyber обфускатор”, the Russian for “Super obfuscator”. It is important to note, however, that this string was deliberately introduced by the attackers while using the ConfuserEx obfuscator. When it comes to strings that are inserted into malware on purpose, they should be assessed carefully during attribution. That is because threat actors may insert strings in languages they do not speak, in order to create false flags intended to confuse researchers and incident responders and prompt them to make an error of judgement when trying to attribute the threat. For that reason, we attached little evidential weight to the presence of the “Cyber обфускатор” string back in 2024.

After examining the NeuralExecutor samples used in 2025, we found that the Russian-language string had disappeared. However, this year we noticed another peculiar clue related to this implant. While the 2024 samples were designed to retrieve the C2 server addresses straight from the configuration, the 2025 ones did so by using [the Dead Drop Resolver technique](#). Specifically, the new NeuralExecutor samples that we



found were designed to retrieve the contents of a file stored in a GitHub repository, and extract a string from it:

## helloworld

```
wtyyvZQYWwRjgyxD3PqUR/3z1GGDgBQc5GTbdrwQbON/  
U535940gIn0D7mkhsSqQ05Z5g1cJhp+UIAIKdC60ndkKpXIHBYwt7qIjyeDmiPv2yUYKFSqMC8zuedXdZvV2KhAPP  
eZO6kBDGyc9xgUjzw4IIBj8FBqFFokWGvxfFkKjuT5WIMQrNeBw825WcA7ek+jcgC9HFI6rUoPYgzYvVAHQPLP0Ag  
==stU7BU0R
```

Contents of the configuration file stored on GitHub

The malware locates this string by searching for two delimiters, wtyyvZQY and stU7BU0R, that mark the start and the end of the configuration data. The bytes of this string are then Base64-decoded and decrypted with AES to obtain the C2 server address.

```
public static string string_0 = "https://github.com/[REDACTED]/helloworld";  
public static string string_1 = "1200";  
public static string string_2 = "wtyyvZQY";  
public static string string_3 = "stU7BU0R";
```

Snippet of the implant configuration

It is notable that this exact method of obtaining C2 server addresses from GitHub, using a string containing delimiter sequences, is quite popular among Chinese-speaking threat actors. For instance, we frequently [observed it being used in the EastWind campaign](#), which we previously connected to the APT31 and APT27 Chinese-speaking threat actors.

Furthermore, during our investigation, we learned one more interesting fact that could be useful in attribution. We observed numerous attempts to deploy the PassiveNeuron loader in one particular organization. After discovering yet another failed deployment, we have detected a malicious DLL named `imjp14k.dll`. An analysis of this DLL revealed that it had the PDB path `G:\Bee\Tree(pmr)\Src\DII_3F_imjp14k\Release\DII.pdb`. This PDB string was referenced in a [report by Cisco Talos](#) on activities likely associated with the threat actor APT41. Moreover, we identified that the discovered DLL exhibits the same malicious behavior as described in the Cisco Talos report. However, it remains unclear why this DLL was uploaded to the target machine. Possible explanations could be that the attackers deployed it as a replacement for the PassiveNeuron-related implants, or that it was used by another actor who compromised the organization simultaneously with the attackers behind PassiveNeuron.

When dealing with attribution of cyberattacks that are known to involve false flags, it is difficult to understand which attribution indicators to trust, or whether to trust any at all. However, the overall TTPs of the

PassiveNeuron campaign most resemble the ones commonly employed by Chinese-speaking threat actors. Since TTPs are usually harder to fake than indicators like strings, we are, as of now, attributing the PassiveNeuron campaign to a Chinese-speaking threat actor, albeit with a low level of confidence.

## Conclusion

The PassiveNeuron campaign has been distinctive in the way that it primarily targets server machines. These servers, especially the ones exposed to the internet, are usually lucrative targets for APTs, as they can serve as entry points into target organizations. It is thus crucial to pay close attention to the [protection of server machines](#). Wherever possible, the attack surface associated with these servers should be reduced to a minimum, and all server applications should be monitored to prevent emerging infections in a timely manner. Specific attention should be paid to protecting applications against SQL injections, which are commonly exploited by threat actors to obtain initial access. Another thing to focus on is protection against web shells, which are deployed to facilitate compromise of servers.

## Indicators of compromise

### PassiveNeuron-related loader files

[12ec42446db8039e2a2d8c22d7fd2946](#)  
[406db41215f7d333db2f2c9d60c3958b](#)  
[44a64331ec1c937a8385dfeeee6678fd](#)  
[8dcf258f66fa0cec1e4a800fa1f6c2a2](#)  
[d587724ade76218aa58c78523f6fa14e](#)  
[f806083c919e49aca3f301d082815b30](#)

### Malicious imjp14k.dll DLL



[751f47a688ae075bba11cf0235f4f6ee](#)

- [Microsoft SQL](#)
- [Windows Server](#)
- [GitHub](#)
- [VBS](#)
- [web shell](#)
- [DLL](#)
- [CobaltStrike](#)
- [Malware Technologies](#)
- [DLL hijacking](#)
- [PowerShell](#)
- [.NET](#)
- [Encryption](#)
- [Backdoor](#)
- [Malware](#)



- [Malware Descriptions](#)

#### Authors

-  [Georgy Kucherin](#)
-  [Saurabh Sharma](#)

PassiveNeuron: a sophisticated campaign targeting servers of high-profile organizations

Your email address will not be published. Required fields are marked \*

[Cancel](#)

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)