Operation TunnelSnake

By Mark Lechtik

Published: 2021-05-06 · Archived: 2026-04-02 10:48:52 UTC

Windows rootkits, especially those operating in kernel space, are pieces of malware infamous for their near absolute power in the operating system. Usually deployed as drivers, such implants have high privileges in the system, allowing them to intercept and potentially tamper with core I/O operations conducted by the underlying OS, like reading or writing to files or processing incoming and outgoing network packets. The capability to blend into the fabric of the operating system itself, much like security products do, is the quality that earns rootkits their notoriety for stealth and evasion.

Having said that, the successful deployment and execution of a rootkit component in Windows has become a difficult task over the years. With Microsoft's introduction of Driver Signature Enforcement, it has become harder (though not impossible) to load and run new code in kernel space. Even then, other mechanisms such as Kernel Patch Protection (also known as PatchGuard) make it hard to tamper with the system, with every change in a core system structure potentially invoking the infamous Blue Screen of Death.

Consequently, the number of Windows rootkits in the wild has decreased dramatically, with the bulk of those still active often being leveraged in high profile APT attacks. One such example came to our attention during an investigation last year, in which we uncovered a formerly unknown Windows rootkit and its underlying cluster of activity. We observed this rootkit and other tools by the threat actor behind it being used as part of a campaign we dubbed 'TunnelSnake', conducted against several prominent organizations in Asia and Africa.

In this blog post we will focus on the following key findings that came up in our investigation:

- A newly discovered rootkit that we dub 'Moriya' is used by an unknown actor to deploy passive backdoors on public facing servers, facilitating the creation of a covert C&C communication channel through which they can be silently controlled;
- The rootkit was found on networks of regional diplomatic organizations in Asia and Africa, detected on several instances dating back to October 2019 and May 2020, where the infection persisted in the targeted networks for several months after each deployment of the malware;
- We observed an additional victim in South Asia, where the threat actor deployed a broad toolset for lateral movement along with the rootkit, including a tool that was formerly used by APT1. Based on the detection timestamps of that toolset, we assess that the attacker had a foothold in the network from as early as 2018;
- A couple of other tools that have significant code overlaps with Moriya were found as well. These contain a user mode version of the malware and another driver-based utility used to defeat AV software.

We provided information about this operation in our threat intelligence portal in August 2020. More details and analysis are available to customers of our private APT reporting service. For more details contact: intelreports@kaspersky.com.

What is the Moriya rootkit and how does it work?

Our investigation into the TunnelSnake campaign started from a set of alerts from our product on a detection of a unique rootkit within the targeted networks. Based on string artefacts within the malware's binaries, we named this rootkit Moriya. This tool is a passive backdoor which allows attackers to inspect all incoming traffic to the infected machine, filter out packets that are marked as designated for the malware and respond to them. This forms a covert channel over which attackers are able to issue shell commands and receive back their outputs.

The rootkit has two traits that make it particularly evasive. The packet inspection happens in kernel mode with the use of a Windows driver, allowing attackers to drop the packets of interest before they are processed by the network stack, thus ensuring they are not detected by security solutions. Secondly, the fact that the rootkit waits for incoming traffic rather than initiating a connection to a server itself, avoids the need to incorporate a C&C address in the malware's binary or to maintain a steady C&C infrastructure. This hinders analysis and makes it difficult to trace the attacker's footprints.

The figure below illustrates the structure of the rootkit's components. They consist of a kernel mode driver and a user mode agent that deploys and controls it. In the following sections we will break down each of these components and describe how they operate to achieve the goal of tapping into the target's network communication and blending in its traffic.

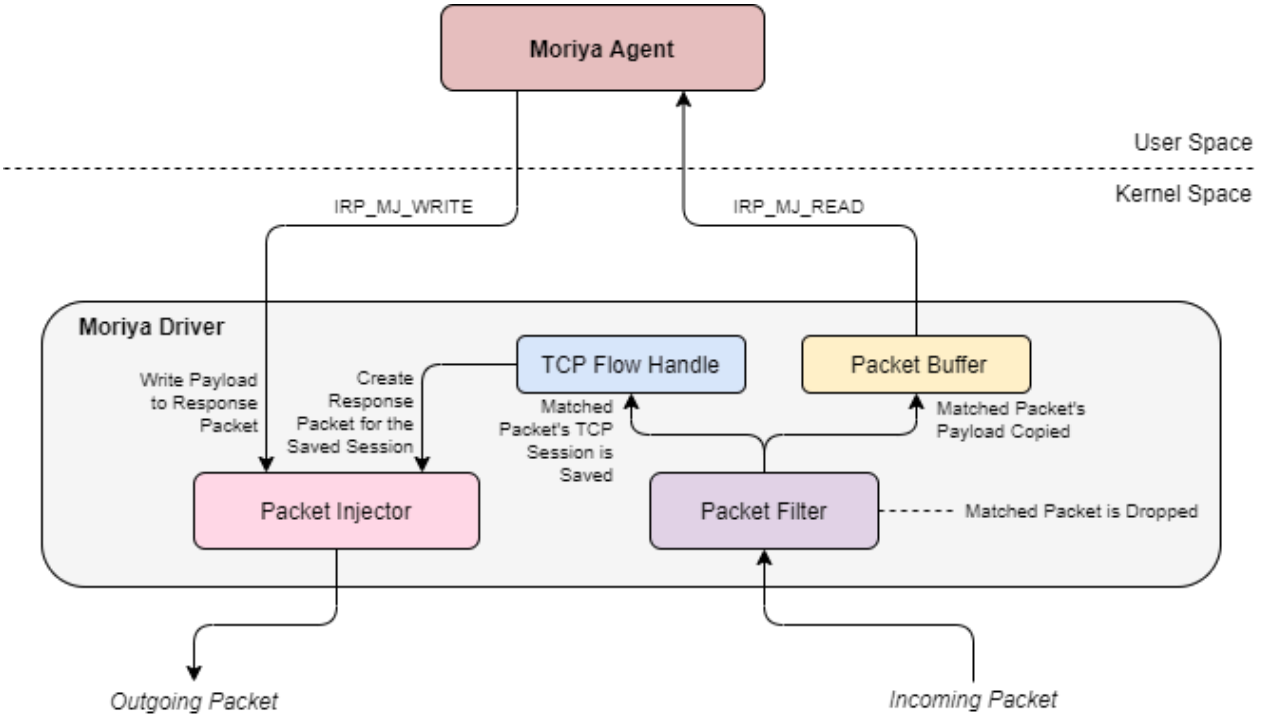

Fig. 1. The architecture of the Moriya rootkit

User mode agent analysis

The user mode component of the Moriya rootkit has two purposes. One is to deploy the kernel mode component of the malware on the machine and the other is to leverage the covert communication channel created by it to read shell commands sent from the C&C server to the compromised machine and to respond to them. Since Moriya is a

passive backdoor intended to be deployed on a server accessible from the internet, it contains no hardcoded C&C address and relies solely on the driver to provide it with packets filtered from the machine's overall incoming traffic.

The first order of business for the attacker when using Moriya is to gain persistence on the targeted computer. For this purpose, the user mode agent's DLL contains an export function named Install, which is intended to create a service named ZzNetSvc with the description 'Network Services Manager' and start it. In turn, the path to the user mode agent's image is set to the registry key HKLM\System\CurrentControlSet\Services\ZzNetSvc\Parameters\ServiceDll so that it will be invoked from its ServiceMain export each time the service is initiated.

Next, after the service is started, the agent will attempt to load the rootkit's driver into the system. Its binary is bundled as two driver images within the DLL's resource section, corresponding to 32- and 64-bit architectures, while in reality only one of them is written to disk. In the cases we analyzed, the agent DLLs were compiled for 64-bit systems, dropping a 64-bit driver to the drivers directory in the system path, under the name MoriyaStreamWatchmen.sys, hence the rootkit's name.

```
OutputDebugStringW(L"zzZ...");
Sleep(60000u);
moriya_stream_watchman_sys_path = 0;
memset(buf, 0, sizeof(buf));
GetSystemDirectoryW(&moriya_stream_watchman_sys_path, 0x104u);
wcscat_s(&moriya_stream_watchman_sys_path, 0x104ui64, L"\\drivers\\MoriyaStreamWatchmen.sys");
OutputDebugStringW(&moriya_stream_watchman_sys_path);
if ( write_pe_resource_to_file(v2, v1, &moriya_stream_watchman_sys_path) )
```

Fig. 2. Code that writes the Moriya driver to disk

The agent uses a known technique whereby the VirtualBox driver (VBoxDrv.sys) is leveraged to bypass the Driver Signature Enforcement mechanism in Windows and load Moriya's unsigned driver. DSE is an integrity mechanism mandating that drivers are properly signed with digital signatures in order for them to be loaded, which was introduced for all versions of Windows starting from Vista 64-bit. The technique used to bypass it was seen in use by other threat actors like Turla, Lamberts and Equation.

Moriya's user mode agent bypasses this protection with the use of an open-source code[1] named DSEFIX v1.0. The user agent dumps an embedded VBoxDrv.sys image of version 1.6.2 to disk and loads it, which is then used by the aforementioned code to map Moriya's unsigned driver to kernel memory space and execute it from its entry point. These actions are made possible through IOCTLs implemented in VBoxDrv.sys that allow writing to kernel address space and executing code from it. Throughout this process, the bypass code is used to locate and modify a flag in kernel space named g_CiOptions, which controls the mode of enforcement.

After the unsigned driver is loaded, the agent registers a special keyword that is used as a magic value, which will be sought in the first bytes of every incoming packet passed on the covert channel. This allows the rootkit to filter marked packets and block them for any application on the system other than the user mode agent. The registration of the value is done through a special IOCTL with the code 0x222004 sent to the driver, where a typical magic string is pass12.

```
if ( !DeviceIoControl(
        g_h_moriya_driver,
        0x222004u,
        g_pass12_packet_magic,              // "pass12"
        6u,
        0i64,
        0,
        &BytesReturned,
        0i64) )
{
  err_msg = L"DeviceIoControl fails!";
  goto end;
}
```

Fig. 3. Registration of the packet magic value using a designated IOCTL

Except for its covert channel communication feature, Moriya is capable of establishing a reverse shell session using an overt channel. For this purpose, it waits for a special packet that consists of a message with the structure connect <c2_address> <c2_port>. The address and port are parsed and used by the agent to start a new connection to the given server, while creating a new cmd.exe process and redirecting its I/O to the connection's socket. The handles for the newly created process and its main thread are destroyed to avoid detection.

In any other case, the agent attempts to read the incoming TCP payload from the driver, which will be retrieved as soon as a designated packet with a magic number and shell command is received. An attempt is made to read the data with a plain ReadFile API function as a blocking operation, i.e., reading is accomplished only once the buffer in kernel mode is populated with data from a Moriya-related packet.

Upon an incoming packet event, the agent creates a new cmd.exe process and redirects its I/O using named pipes. One pipe is used to read the retrieved shell command from the covert channel and the other is used to write the shell's output (obtained from the stdout and stderr streams) back to it after execution. To write any data back, the agent uses the WriteFile API function with the driver's handle.

All traffic passed on the channel is encoded with a simple encryption scheme. Every sent byte has its payload, following the magic string, XORed with the value 0x05 and then negated. Following the same logic, to decode the incoming traffic's payload, every byte of it should be first negated and then XORed with 0x05.

```
curr_len = 0;
if ( len > 0 && len >= 0x10 )
{
  pattern_1 = _mm_load_si128(&g_pattern_1);   // 0x05050505050505050505050505050505
  pattern_2 = _mm_load_si128(&g_pattern_2);   // 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  do
  {
    i = curr_len;
    curr_len += 16;
    *&packet_buff[i] = _mm_andnot_si128(_mm_xor_si128(_mm_loadu_si128(&packet_buff[i]), pattern_1), pattern_2);
  }
  while ( curr_len < len - len % 16 );
}
if ( curr_len < len )
{
  packet_buff_chunk = &packet_buff[curr_len];
  residue = len - curr_len;
  do
  {
    packet_buff_chunk_chr = *packet_buff_chunk++;
    *(packet_buff_chunk - 1) = ~(packet_buff_chunk_chr ^ 5);
    --residue;
  }
  while ( residue );
}
```

Fig. 4. Code used for packet encoding

Kernel mode driver analysis

The Moriya rootkit's driver component makes use of the Windows Filtering Platform (WFP) to facilitate the covert channel between the compromised host and the C&C server. WFP provides a kernel space API that allows driver code to intercept packets in transit and intervene in their processing by the Windows TCP/IP network stack. This makes it possible to write a driver that can filter out distinct packet streams, based on developer-chosen criteria, and designate them for consumption by a specific user mode application, as is the case in Moriya.

The driver fetches the distinct Moriya-related traffic using a filtering engine. This is the kernel mode mechanism used to inspect traffic according to rules that can be applied on various fields across several layers of a packet (namely data link, IP and transport), making it possible to handle matching packets with unique handlers. Such handlers are referred to as callout functions.

In the case of Moriya, the filtering engine is configured to intercept TCP packets, sent over IPv4 from a remote address. Each packet with these criteria will be inspected by a callout function that checks if its first six bytes correspond to the previously registered magic value, and if so, copies the packet contents into a special buffer that can be later read by the user mode agent. The matching packet will then be blocked in order to hide its presence from the system, while any other packet is permitted to be processed as intended by the network stack.

To allow the crafting of a response back to the server, the callout function saves a special value in a global variable that identifies the received TCP stream. This value is called a flowHandle, and is taken from the packet's corresponding FWPS_INCOMING_METADATA_VALUES0 struct. When the user issues a response to the server via the driver, the latter would craft a new packet using the FwpsAllocateNetBufferAndNetBufferList0 function and insert the response data and target server based on the saved flowHandle to it, using the function FwpsStreamInjectAsync0.

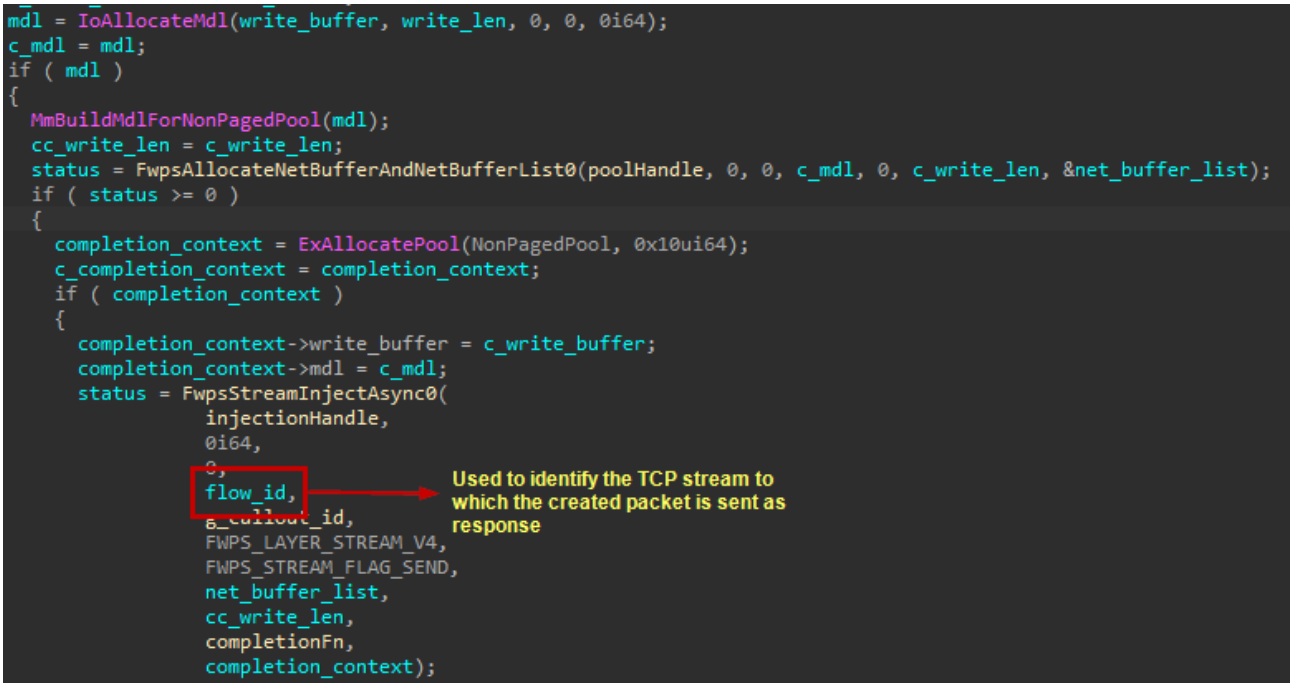

Fig. 5. Code that creates a new packet, designates it for the flow of the corresponding incoming TCP packet and injects data written from user space into it

As formerly mentioned, the driver registers several functions that are exposed to the user mode agent in order to interact with it:

- **IRP_MJ_READ**: used to allow the user mode agent to read the body of a Moriya TCP packet from a special buffer to which it is copied upon receipt. The function itself waits on an event that gets signaled once such a packet is obtained, thus turning the ReadFile function called by the user mode agent into a blocking operation that will wait until the packet is picked up by the driver.
- **IRP_MJ_WRITE**: injects user-crafted data into a newly created TCP packet that is sent as a response to an incoming Moriya packet from the server.
- **IRP_MJ_DEVICE_CONTROL**: used to register the keyword to check the beginning of every incoming TCP packet in order to identify Moriya-related traffic. The passed magic is anticipated to be six characters long.

```
current_stack_location = irp->Tail.Overlay.CurrentStackLocation;
status = 0;
c_irp = irp;
if ( current_stack_location->Parameters.DeviceIoControl.IoControlCode == 0x222004 )
{
  if ( current_stack_location->Parameters.DeviceIoControl.InputBufferLength == 6 )
  {
    ioctl_tcp_magic_buffer = irp->AssociatedIrp.SystemBuffer;
    *g_tcp_packet_magic_value = *ioctl_tcp_magic_buffer;
    *&g_tcp_packet_magic_value[4] = *(ioctl_tcp_magic_buffer + 2);
  }
```

Fig. 6. Code used for registering the packet magic value from the driver side

How were targeted servers initially infected?

Inspecting the systems targeted by the rootkit, we tried to understand how they got infected in the first place. As previously mentioned, Moriya was seen deployed mostly on public-facing servers within the victim organizations. In one case, we saw the attacker infect an organizational mail server with the China Chopper webshell, using it to map the victim's

network and then deploy other tools in it. Moriya's user mode agent was explicitly installed using a command line executed on the targeted server this way. This command and examples of others run on the victim machine via the webshell can be seen below.

```
"cmd" /c cd /d C:\inetpub\wwwroot\&ipconfig -all

"cmd" /c cd /d C:\inetpub\wwwroot\&reg query

HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest

"cmd" /c cd /d C:\inetpub\wwwroot\&$public\acmsetup.exe

"cmd" /c cd /d C:\inetpub\wwwroot\&query user

"cmd" /c cd /d C:\inetpub\wwwroot\&ipconfig/all

"cmd" /c cd /d C:\inetpub\wwwroot\&ping google.com

"cmd" /c cd /d C:\inetpub\wwwroot\&netstat -anp tcp

"cmd" /c cd /d C:\inetpub\wwwroot\&tasklist /v

"cmd" /c cd /d C:\inetpub\wwwroot\&whoami

"cmd" /c cd /d C:\inetpub\wwwroot\&cd $windir\web\

"cmd" /c cd /d $windir\Web\&rundll32 MoriyaServiceX64.dll, Install

"cmd" /c cd /d C:\inetpub\wwwroot\&ipconfig/all

"cmd" /c cd /d C:\inetpub\wwwroot\&time /t

...
```

In general, we assess that the group's modus-operandi involves infiltrating organizations

through vulnerable web servers in their networks. For example, an older variant of Moriya named IISSpy (described below) targets IIS web servers. Our telemetry shows that it was likely deployed by exploiting CVE-2017-7269 to let the attackers gain an initial foothold on a server prior to running the malware.

Post exploitation toolset

During our investigation we found a target in South Asia that enabled us to get a glimpse into some of the other tools that we assess were in use by the same attacker. The toolset includes programs used to scan hosts in the local network, find new targets, perform lateral movement to spread to them and exfiltrate files. While most of the tools seem custom made and tailored for the attackers' activities, we could also observe some open-source malware frequently leveraged by Chinese-speaking actors. Following is an outline of these tools based on their purpose in the infection chain.

- **Network Discovery**: custom built programs used to scan the internal network and detect vulnerable services.
    - **HTTP scanner**: command-line tool, found under the name '8.tmp', which discovers web servers through banner grabbing. This is done by issuing a malformed HTTP packet to a given address, where no headers are included and the request is succeeded with multiple null bytes.

```
00000000  47 45 54 20 2f 20 48 54  54 50 2f 31 2e 30 20 0d GET / HT TP/1.0 .
00000010  0a 0d 0a 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
    00000000  48 54 54 50 2f 31 2e 31  20 34 30 30 20 42 61 64 HTTP/1.1  400 Bad
    00000010  20 52 65 71 75 65 73 74  0d 0a 44 61 74 65 3a 20  Request ..Date:
```

Fig. 7. Malformed packet generated by HTTP scanner

If the server responds, the output will be displayed in the console, as shown below.

```
C:\>8.tmp 10.0.0.1 80
target ip :10.0.0.1
target port :   80

 connect success

 Recive Data num:481

 Recive Data: HTTP/1.1 400 Bad Request
Date:
Server: Apache/2.4.7 (Ubuntu)
Content-Length: 300
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
```

Fig. 8. Console output with a server response displayed upon discovery of a new server in the network

- **DCOM Scanner:** another command-line utility that attempts to connect to a remote host on TCP port 135 (RPC), and use the DCOM IOxidResolver interface to resolve addresses assigned to all network interfaces available on the remote system.

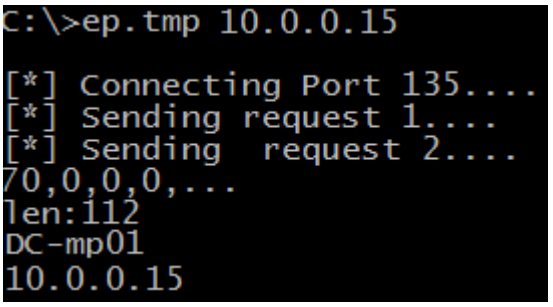

Fig. 9. Output of the DCOM scanner utility

- **Lateral Movement**: tools used to spread to other hosts in the targeted networks.
  - **BOUNCER**: malware that was first described by Mandiant in their 2013[2] report on APT1. This tool is another passive backdoor that waits for incoming connections on a specific port and provides different features, as outlined below, that can be used to control a remote host and facilitate lateral movement from it.

1	0x01: Proxy Init Connection
2	0x02: Proxy Send Packet
3	0x03: Proxy Close Connection
4	0x07: Execute Shellcode
5	0x0A: Kill Bot
6	0x0C: Reverse Shell CMD
7	0x0D: Delete File
8	0x0E: Execute local program
9	0x0F: Enumerate Servers In Domain and save output in gw.dat
10	0x10: Enumerate SQL Servers and save output in sql.dat
11	0x12: Reverse Shell CreateProcess
12	0x16: Upload File - Write Data
13	0x17: Download File - Finish
14	0x1E: Download File - Start
15	0x1F: Upload File - Start

16	0x2D: Enumerate Servers
17	0x2E: Enumerate SQL Server
18	0x2F: Enumerate Servers Verbose
19	0x30: Enumerate Users
20	0x32: Do nothing

The BOUNCER sample that we observed contained a string that indicates which command-line arguments it anticipates:

	usage:%s IP port [proxip] [port] [key]

However, the backdoor is configured to accept only the port number on which it will listen.

We saw two versions of this backdoor, initiated by two different launchers. The first one is an executable file named nw.tmp that decrypts an embedded payload using the RC4 algorithm and injects it into a newly spawned svchost.exe process. The injected payload is similar to one described by Mandiant in 2013, which is yet another intermediate loader that decrypts and loads an embedded BOUNCER DLL. The last stage is started by invoking the DLL's dump export with the arguments passed via the command line.

The other version was stored with the name rasauto.dll in the system directory, impersonating the Windows Remote Access Auto Connection Manager library. Like the other version, it decrypts an embedded DLL using RC4, but this time uses no intermediate stage, instead directly calling the DLL's dump export without arguments. The decrypted library is a slightly modified BOUNCER variant that always listens on the hardcoded port 1437.

```
dump            proc near               ; DATA XREF: .rdata:c

WSAData         = WSAData ptr -190h

                sub     esp, 190h
                push    offset unk_1002B180 ; void *
                call    c_memset
                pop     ecx
                lea     eax, [esp+190h+WSAData]
                push    eax             ; lpWSAData
                push    202h            ; wVersionRequested
                call    ds:WSAStartup
                mov     eax, 1437
                push    eax             ; Port
                mov     Port_1437, eax
                call    Core
```

Fig. 10. Code from the second BOUNCER variant that uses the hardcoded port 1437 to listen for new packets

Based on compilation timestamps of all BOUNCER-related executables, as shown below, we assess that the attacker reused old samples of the malware rather than compiled new versions of it:

> nw.tmp – stage 0 - launcher - 08-03-2017 03:56:24
>
> nw.tmp – stage 1 - embedded loader - 26-08-2014 04:49:58
>
> nw.tmp – stage 2 - embedded BOUNCER backdoor - 28-05-2012 13:44:37
>
> rasauto.dll - stage 0 – loader 26-08-2013 09:37:08
>
> rasauto.dll - stage 1 - embedded BOUNCER backdoor - 26-08-2013 09:36:27

- **Custom PSExec**: the attacker deployed a tool to execute commands remotely on compromised machines. Like the original PSExec tool, this one consists of two components – a client named tmp and a service named pv.tmp. In order to use the tool, the attacker has to execute it via a command line with the parameters specified below.

> Usage: psexec <hostname > psserve_path exefilename ServerName[option]\n

The service component is a tiny program that uses the CreateProcessA API to start a program specified as an argument. The client component uses the Service Control Manager (SCM) API to create a service on the target machine. If the ServerName argument is not specified, the service will be named Server%c%c where %c is a random lower case character. The exefilename argument is then passed to the StartServiceA function in order to initiate the command execution.

```
v5 = rand() % 26 + 97;
v6 = rand() % 26 + 97;
if ( strlen((const char *)a4_ServiceName) )
  sprintf(ServiceName, "%s", (const char *)a4_ServiceName);
else
  sprintf(ServiceName, "Server%c%c", v5, v6);
printf("Server Name: %s\n", ServiceName);
hRSvc = OpenSCManagerA(a1_Hostname, 0, SC_MANAGER_ALL_ACCESS);
hSCObject_2 = hRSvc;
hSCObject = hRSvc;
if ( !hRSvc )
  return fprintf((FILE *const)&Stream, "ERROR! Cannot open service manager on host %s.\n", a1_Hostname);
hService = CreateServiceA(
             hRSvc,
             ServiceName,
             ServiceName,
             SERVICE_ALL_ACCESS,
             SERVICE_WIN32_OWN_PROCESS,
             SERVICE_DEMAND_START,
             SERVICE_ERROR_IGNORE,
             a3_PsSVCPath,
             0,
             0,
             0,
             0,
             0);
if ( hService )
{
  ServiceArgVectors[0] = a3_PsSVCPath;
  ServiceArgVectors[1] = (LPCSTR)a2_ExeFilename;
  if ( StartServiceA(hService, 2u, ServiceArgVectors) )
```

Fig. 11. Code used to create and start the service on targeted host

It is worth noting that the program has some limitations. Compared with the original PSExec, it is not able to copy the service binary (i.e., pv.tmp, which has its path specified in the psserve_path argument) to a remote machine, but rather assumes it is already present on it. Besides, it cannot handle network credentials, limiting the ability to execute commands as other users, nor does it support pipes, which means it does not receive the output of the commands it issues.

- **Exfiltration**: multi-platform utilities commonly used to establish connections with remote hosts and conduct file system operations on them, including file upload and download.
    - **Earthworm and Termite**: well-known command-line utilities developed to facilitate intrusion into intranet networks. These programs are multiplatform and can be deployed on various architectures. Earthworm is used to create tunnels between compromised hosts and transfer data.

```
VERSION : free 1.0
 ./xxx ([-options] [values])*
 options :
 Eg: ./xxx -s ssocksd -h
 -s state setup the function.You can pick one from the
 following options:
 ssocksd , rcsocks , rssocks ,
 lcx_listen , lcx_tran , lcx_slave
 -l listenport open a port for the service startup.
 -d refhost set the reflection host address.
 -e refport set the reflection port.
 -f connhost set the connect host address .
 -g connport set the connect port.
 -h help show the help text, By adding the -s parameter,
 you can also see the more detailed help.
 -a about show the about pages
 -v version show the version.
 -t usectime set the milliseconds for timeout. The default
 value is 1000
 ......
```

Fig. 12. Earthworm help message

Termite provides additional features to download and upload files between the compromised hosts, as well as a way to spawn a remote shell to control the targeted machine.

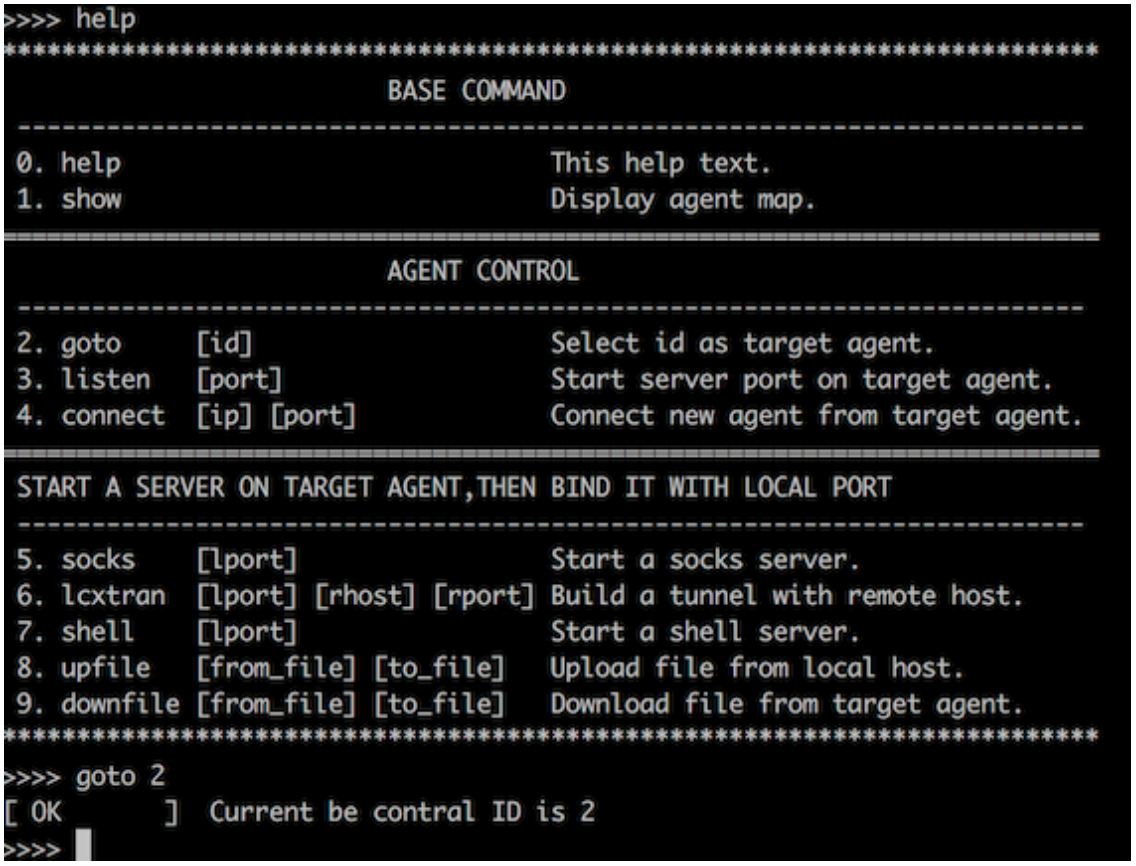

Fig. 13. Termite help message

- **TRAN:** another tool that we detected under the filename tmp that was used to transfer data between compromised hosts. The binary we saw operated as a loader that embodies a tiny web server encrypted with the RC4 algorithm within it. This server is later injected into a newly created legitimate schtask.exe process and usually listens on port 49158. It is used for managing files uploaded by the attacker into an in-memory virtual file system maintained by the malware.By default the file system includes a tiny program named client.exe, which can be downloaded by any host using a standard HTTP GET request to the path /client.exe. This file is a command-line utility that can be used to control the virtual file system managed by the server, through one of several available commands outlined below.

Fig. 14. Client.exe help message

IISSpy: tracing Moriya back to a user-mode rootkit

IISSpy is an older user-mode version of the Moriya rootkit that we were able to pinpoint in our telemetry. It is used to target IIS servers for establishing a backdoor in their underlying websites. It was detected on a machine in 2018, unrelated to any of the attacks in the current operation. This suggests the threat actor has been active since at least that year.

The malware, which comes as a DLL, achieves its goals by enumerating running IIS processes on the server (i.e., those that are executed from the image w3wp.exe), and injecting the malware's DLL into them to alter their behavior. The executed code in the IIS processes will then set inline hooks for several functions, most notably CreateFileW.

The corresponding CreateFileW hook function checks if the filename argument contains the directory '\MORIYA\' or '\moriya\' in its path, and if so, infers that the attacker has sent a specially crafted HTTP request to the web server. In this request, the Moriya path in the URL is followed by an encoded command. After the command is decoded and processed, it is passed via a mailslot (\\.\mailslot\slot) to a separate thread, while signaling an event called Global\CommandEvent.

```
OutputDebugStringW(L"ProxyCreateFileW\n");
OutputDebugStringW(lpFileName);
s_IISINFO_HTM_path[0] = 0;
memset(&s_IISINFO_HTM_path[1], 0, 0x206ui64);
if ( lpFileName )
{
  s_MORIYA_position = wcsstr(lpFileName, L"\\MORIYA\\");
  s_moriya_position = wcsstr(lpFileName, L"\\moriya\\");
```

Fig. 15. Code of the CreateFileW hook function that looks for the 'MORIYA' \ 'moriya' directory in a request path

Should the currently handled file contain the Moriya path, the very same hook function will generate a special file on the web server to which command execution output will be written. This file's path is created by finding the position of the '\MORIYA\' or '\moriya\' strings in the inspected filename argument, and replacing it with the string '\IISINFO.HTM'. This will then be appended to the command data passed on the mailslot, following a ' > ' character.

The other thread waiting on the command event mentioned above is in charge of processing attacker data fetched from the mailslot. Any such command will be read and parsed to find the ' > ' character and the file path that follows it, in this case the one corresponding to 'IISINFO.HTML'. After executing the command via cmd.exe, the output will be written to the file in this path, allowing the attacker to read it by issuing a corresponding HTTP request where the URL path leads to this file on the server.

Other functions that are hooked in the IIS process are CreateProcessAsUserW and CreateProcessW. These are used to detect if the current process spawns a new server instance, which will in turn be injected with the malware's DLL. Apart from this, IISSpy will also create a monitoring thread that will periodically look for newly created httpd.exe processes, corresponding to the Apache server. If detected, the malware will be injected to them as well.

Although it is evident from both the functionality and use of the Moriya keyword by the malware that IISSpy and the Moriya rootkit are related, further evidence in the code substantiates the connection:

- The older variant is capable of creating a reverse shell transmitted through an overt channel in exactly the same way as the more recent version of the malware, i.e., it identifies a connect request followed by a C&C server address and port, connects to it and redirects the IO of a new exe process to the underlying socket.

- Both variants use the same packet encoding and decoding algorithm, whereby each clear-text byte is XORed with 0x5 and negated, and vice-versa.

```
do
{
  encoded_chr = buffer[++j];
  buffer[j] = ~(encoded_chr ^ 5);
}
while ( j < buffer_len );
```

Fig. 16. Packet decoding loop that follows the same logic as that used in Moriya

- In both cases the developers left a trail of unique debug messages, issued to the OutputDebugString API function. An example of such a string used in identical code in the two variants is shown below.

```
OutputDebugStringW(L"The cmd process has not created or has terminated!\n");
OutputDebugStringW(L"Let's create one!\n");
if ( g_h_cmd_exe )
{
  CloseHandle(g_h_cmd_exe);
  g_h_cmd_exe = 0i64;
  g_pid_cmd_exe = 0;
}
memset(&startup_info.lpReserved, 0, 0x60ui64);
process_information.hThread = 0i64;
*&process_information.dwProcessId = 0i64;
startup_info.hStdInput = h_cmd_exe_input_read;
process_information.hProcess = 0i64;
startup_info.cb = 104;
startup_info.hStdError = h_cmd_exe_output_write;
startup_info.hStdOutput = h_cmd_exe_output_write;
startup_info.wShowWindow = 0;
startup_info.dwFlags = 257;
cmd_exe_path = 0;
memset(v62, 0, sizeof(v62));
GetSystemDirectoryW(&cmd_exe_path, 0x104u);
wcscat_s(&cmd_exe_path, 0x104ui64, L"\\cmd.exe");
if ( !CreateProcessW(
        &cmd_exe_path,
        0i64,
        0i64,
        0i64,
        1,
        0x20u,
        0i64,
        0i64,
        &startup_info,
        &process_information) )
{
  err_msg = L"Create shell fails!\n";
```

Fig. 17. Code used in both variants to spawn a new shell, while printing unique debug messages

- Both implants are deployed by invoking an export function named Install that creates a service that allows persistent execution, with the malware's logic residing in the ServiceMain Moreover, the Install functions are highly similar to one another.

Fig. 18. Comparison of Install export function CFGs between IISSpy and Moriya

The ProcessKiller rootkit vs. security products

Another interesting artefact found in our telemetry that could be tied to the developers of Moriya is a malware named ProcessKiller. As its name suggests, it is intended to eliminate execution of processes, with the use of a kernel mode driver. Ultimately, this tool is used to shut down and block initiation of AV processes from kernel space, thus allowing other attack tools to run without being detected.

This malware operates through the following stages:

- An attacker calls the malware's DLL from an export named Kill, passing it a list of process names it would like to shut down and block as a command-line argument.
- The malware writes a driver that is embedded as a resource within it, impersonating a Kaspersky driver under the path %SYSTEM%\drivers\kavp.sys.

- There is an attempt to load the driver using the Service Control Manager. However, since it is not signed and loading is prone to fail on Windows versions above Vista 64-bit, the malware uses the same DSEFix code to bypass Digital Signature Enforcement as witnessed in Moriya's user mode agent.
- The malware parses the process names passed as arguments and creates a vector of 'blacklisted processes' out of them.
- For each process in the list, the malware detects its PID and issues it through an IOCTL with code 0x22200C to the driver which is in charge of shutting it down from kernel space. The shutdown is carried out by locating the process object with the function PsLookupProcessByProcessId and then terminating it with ZwTerminateProcess.
- The list of processes is then passed via another IOCTL with the code 0x222004 to the driver, which inserts each member of it to a linked list in kernel space. When the driver is bootstrapped, it registers a callback for newly created processes through the PsSetCreateProcessNotifyRoutineEx function, which inspects the image name of the created process and compares it against those found in the linked list. If a match is found, the process creation status in the PPS_CREATE_NOTIFY_INFO structure will be set to STATUS_UNSUCCESSFUL, signaling the user space API function that process creation failed.
- At this point any other malware can theoretically operate without being detected.
- If the attacker wishes to disable blacklisting, it can be done by issuing an IOCTL with the code 0x222008, which would destroy the linked list of blacklisted processes.

Once again, the connection to Moriya is based on several observations:

- Distinct debug error messages, as the one presented below.

```
if ( h_driver == (HANDLE)-1i64 )
{
  GetLastError();
  FreeResource(loaded_resource);
  debuge_message(L"ResourcesToFile fails!");
  return result;
}
```

Fig. 19. Unique debug message that appears in ProcessKiller and Moriya

- Filename of the same structure, i.e., Moriya's agent is internally named 'MoriyaServiceX64.dll', and ProcessKiller's DLL is named 'ProcessKillerX64.dll'
- Usage of the exact same DSEFix code to load an unsigned driver.

What do we know about the threat actor?

Unfortunately, we are not able to attribute the attack to any particular known actor, but based on the TTPs used throughout the campaign, we suppose it is a Chinese-speaking one. We base this on the fact that the targeted entities were attacked in the past by Chinese-speaking actors, and are generally located in countries that are usually targeted by such an actor profile. Moreover, the tools leveraged by the attackers, such as China Chopper, BOUNCER, Termite and Earthworm, are an additional indicator supporting our hypothesis as they have previously been used in campaigns attributed to well-known Chinese-speaking groups.

Who were the targets?

Based on our telemetry the attacks were highly targeted and delivered to less than 10 victims around the world. The most prominent victims are two large regional diplomatic organizations in South-East Asia and Africa, while all the others were victims in South Asia.

Conclusion

The TunnelSnake campaign demonstrates the activity of a sophisticated actor that invests significant resources in designing an evasive toolset and infiltrating networks of high-profile organizations. By leveraging Windows drivers, covert communications channels and proprietary malware, the group behind it maintains a considerable level of stealth. That said, some of its TTPs, like the usage of a commodity webshell and open-source legacy code for loading unsigned drivers, may get detected and in fact were flagged by our product, giving us visibility into the group's operation.

Still, with activity dating back to at least 2018, the threat actor behind this campaign has shown that it is able to evolve and tailor its toolset to target environments. This indicates the group conducting these attacks may well still be active and retooling for additional operations in the area of interest outlined in this publication, as well as other regions. With that in mind, we continue to track this attacker and look for signs of its reappearance in the wild. Any findings and updates will be made available to customers of our Threat Intelligence Portal.

For more information about operation TunnelSnake and the underlying threat actor, contact us at: intelreports@kaspersky.com.
To learn more on reverse engineering and malware analysis from Kaspersky GReAT experts, check out the website https://xtraining.kaspersky.com.

IOCs

[1] Today a copy of the original code can be found here: http://www.m5home.com/bbs/thread-8043-1-1.html

[2] https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf

---