

DPRK Malware Targeting Security Researchers

norfolkinfosec.com/dprk-malware-targeting-security-researchers/

norfolk

January 26, 2021

Earlier today, Adam Weidemann from Google's Threat Analysis Group (TAG) published research regarding a threat actor targeting security analysts following a social engineering campaign. Google attributes this activity to DPRK threat actors. This blog has no evidence to corroborate or refute this claim, but considers Google to be a reputable source of information.

According to the published research, the threat actors would engage in a social engineering effort in which they would attempt to collaborate with security analysts on a Visual Studio project, ultimately leading to them delivering a malicious DLL that the researcher would unknowingly launch.

This post examines that DLL and parts of its second-stage workflow.

Technical Analysis

MD5: 56018500f73e3f6cf179d3b853c27912

SHA-1: a3060a3efb9ac3da444ef8abc99143293076fe32

SHA-256: 4c3499f3cc4a4fdc7e67417e055891c78540282dccc57e37a01167dfe351b244

This file is a DLL that expects to be executed under the following conditions to initiate the malicious workflow:

- The operating system must be 64-bit
- The correct export must be called
- Exactly two additional command line arguments must be supplied alongside this export

Although Google provided multiple hashes, the file above was selected as a starting point because its exports matched the export shown in an image in Google's post (CMS_dataFinal). The post used this image how the malware would execute in normal circumstances, which in turn allows us to supply two additional critical parameters to the file:

- Bx9yb37GEcJNK6bt
- 4901

Under the attacker's workflow, these would have been supplied through a PowerShell command initiated through a Visual Studio Build Event, but these can also be supplied through a normal command line.

Once executed under the specified conditions, the malware will move an encoded set of strings into memory and decode them.

The screenshot displays a debugger window with the following components:

- Assembly View:** Shows a series of instructions pushing data to the stack. The instructions are:


```

      0FBE05 04200100 movzx eax,byte ptr ds:[7FEEA075470]
      48:8D8D 31210000 lea rcx,qword ptr ss:[rbp+2130],xmm1
      0F1005 D61F0100 movups xmm0,xmmword ptr ds:[7FEEA075450]
      0F100D DF1F0100 movups xmm1,xmmword ptr ds:[7FEEA075460]
      3302 xor edx,edx
      41:88 FF030000 mov r8d,3FF
      C785 00200000 10027100 mov dword ptr ss:[rbp+2000],D8710210
      OF1185 00210000 movups xmmword ptr ss:[rbp+2100],xmm0
      0F118D 00210000 movups xmmword ptr ss:[rbp+2110],xmm1
      C785 04200000 05628700 mov dword ptr ss:[rbp+2004],68876205
      C785 08200000 4D440800 mov dword ptr ss:[rbp+2008],5A08444D
      C785 0C200000 AC5B5700 mov dword ptr ss:[rbp+200C],73B5E5AC
      66:C785 10200000 CDA: mov word ptr ss:[rbp+2010],A3CD
      C785 90200000 2F033300 mov dword ptr ss:[rbp+2090],BE33032F
      C785 94200000 2057DD00 mov dword ptr ss:[rbp+2094],32D05720
      C785 98200000 6E6E1C00 mov dword ptr ss:[rbp+2098],631C6E6E
      C785 9C200000 85F20400 mov dword ptr ss:[rbp+209C],68D024E
      C785 A0200000 CA567F00 mov dword ptr ss:[rbp+20A0],CA7F56CA
      C785 A4200000 82E58F00 mov dword ptr ss:[rbp+20A4],108F5E82
      C785 A8200000 BF0B5300 mov dword ptr ss:[rbp+20A8],C4530BBF
      C785 AC200000 30BF8300 mov dword ptr ss:[rbp+20AC],ACB3BF30
      C785 B0200000 F1A8E400 mov dword ptr ss:[rbp+20B0],F3E4A8F1
      C785 B4200000 A70A4C00 mov dword ptr ss:[rbp+20B4],484C0AA7
      C785 B8200000 4C693400 mov dword ptr ss:[rbp+20B8],F134694C
      C685 C0200000 A2: mov byte ptr ss:[rbp+20C0],A2
      C785 18200000 E0F44000 mov dword ptr ss:[rbp+2018],B14A0F0E
      C785 1C200000 2C5FC600 mov dword ptr ss:[rbp+201C],39C65F2C
      C785 20200000 79713400 mov dword ptr ss:[rbp+2020],76347179
      C785 24200000 8BC0E700 mov dword ptr ss:[rbp+2024],A5E7C08B
      C785 28200000 102C7100 mov dword ptr ss:[rbp+2028],87712C10
      C785 2C200000 85F20400 mov dword ptr ss:[rbp+202C],68D024E
      C785 30200000 3A483D00 mov dword ptr ss:[rbp+2030],2430483A
      C785 34200000 D7C48B00 mov dword ptr ss:[rbp+2034],288BC4D7
      66:C785 38200000 E0A: mov word ptr ss:[rbp+2038],AAE0
      C785 40200000 17033300 mov dword ptr ss:[rbp+2040],BE330317
      C785 44200000 2050C600 mov dword ptr ss:[rbp+2044],33C65020
      C785 48200000 6D730A00 mov dword ptr ss:[rbp+2048],7D0A736D
      C785 4C200000 A6CF7700 mov dword ptr ss:[rbp+204C],70F7CFA6
      C785 50200000 E57E4C00 mov dword ptr ss:[rbp+2050],EA4C7E5
      C785 54200000 C686A100 mov dword ptr ss:[rbp+2054],29A1B6C6
      C685 58200000 1A: mov byte ptr ss:[rbp+2058],1A
      C785 60200000 29033300 mov dword ptr ss:[rbp+2060],BE330329
      C785 64200000 2050C600 mov dword ptr ss:[rbp+2064],33C65020
      C785 68200000 6D730A00 mov dword ptr ss:[rbp+2068],7D0A736D
      C785 6C200000 A6CF7700 mov dword ptr ss:[rbp+206C],70F7CFA6
      C785 70200000 E57E4C00 mov dword ptr ss:[rbp+2070],EA4C7E5
      C785 74200000 C686A100 mov dword ptr ss:[rbp+2074],29A1B6C6
      C785 78200000 91224B00 mov dword ptr ss:[rbp+2078],CD4B2291
      C785 7C200000 32B2F400 mov dword ptr ss:[rbp+207C],83F4B232
      C785 80200000 AABDF800 mov dword ptr ss:[rbp+2080],F7F8BDA
      C785 84200000 F34A1100 mov dword ptr ss:[rbp+2084],6114AF3
      66:C785 88200000 527: mov word ptr ss:[rbp+2088],7152
      C685 8C200000 E3: mov byte ptr ss:[rbp+208A],E3
      8B85 20210000 mov byte ptr ss:[rbp+2130],E1
      40:8B85 30210000 mov byte ptr ss:[rbp+2130],E1
      E8 67F10000 call cms2.7FEEA065500
      48:8D8D 00200000 lea rcx,qword ptr ss:[rbp+2000]
      E8 CBFAFFFF call cms2.7FEEA063140
      48:8D8D 90200000 lea rcx,qword ptr ss:[rbp+2090]
      E8 BFAFFFF call cms2.7FEEA063140
      48:8D8D 18200000 lea rcx,qword ptr ss:[rbp+2018]
      E8 B3FAFFFF call cms2.7FEEA063140
      48:8D8D 28200000 lea rcx,qword ptr ss:[rbp+2028]
      E8 A7FAFFFF call cms2.7FEEA063140
      48:8D8D 40200000 lea rcx,qword ptr ss:[rbp+2040]
      E8 9BFAFFFF call cms2.7FEEA063140
      48:8D8D 60200000 lea rcx,qword ptr ss:[rbp+2060]
      E8 BFAFFFF call cms2.7FEEA063140
      48:8D8D 40200000 lea rcx,qword ptr ss:[rbp+2040]
      3302 xor edx,edx
      
```
- Memory Dump:** Shows the decoded strings stored in memory. The strings include:

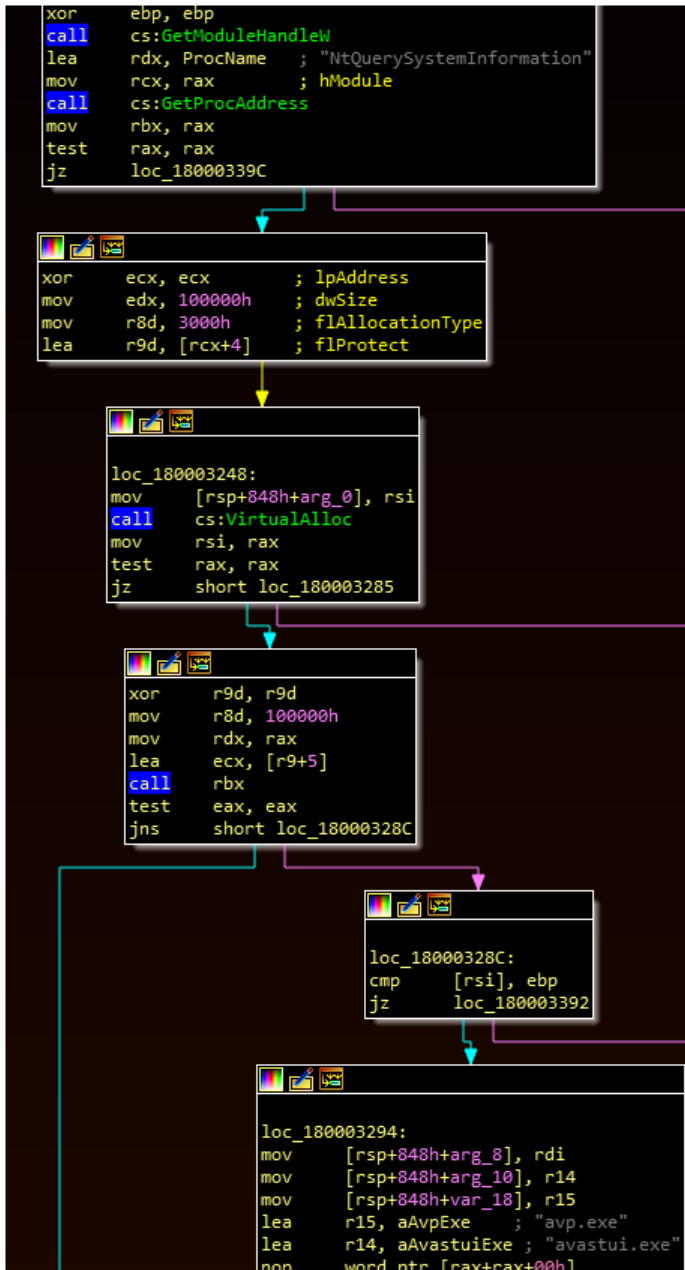

```

      71 DB 05 62 87 68 40 44 08 5A AC E5 B5 73 :.q0.b.kMD.Z-dus
      00 00 00 00 00 00 0E 0F 4A B1 2C 5F C6 39 :If.....)=,_49
      34 76 8B C0 E7 A5 10 2C 71 B7 15 35 F7 06 :yq4v.Acv.q.5+.
      3D 24 D7 C4 B8 28 E0 AA 00 00 00 00 00 00 :H=5xA(a*.....
      33 BE 20 50 C6 33 6D 73 0A 7D A6 CF F7 70 :.3% P43ms.)I+p
      4C EA C6 B6 A1 29 1A 6E 28 77 00 00 00 00 :a-L43(i).o+w...
      33 BE 20 50 C6 33 6D 73 0A 7D A6 CF F7 70 :.3% P43ms.)I+p
      EA 26 8A A1 29 91 22 48 0D 32 B2 E4 B3 :+d-B43(i).*KZ...
      F8 F7 F3 44 11 06 52 71 E3 00 00 00 00 00 :*sb-0j..Rq4...
      33 BE 20 57 0D 32 6E 06 1C 63 BE F2 D0 68 :.3% wY2nn.c%0dh
      7F CA 82 E5 8F 10 8F 0B 53 C4 30 BF B3 AC :EV.E.a..z.SA0.*-
      E4 F3 A7 0A 4C 48 1E 6C 41 98 4C 69 34 F1 :h a0$.LH.1A.L14h
      0F 00 00 00 00 00 90 7E 30 00 00 00 00 00 :e.....0.....
      4E 00 00 00 00 00 30 B1 30 00 00 00 00 00 :.AF.....0.....
      30 00 00 00 00 00 04 00 00 00 00 00 00 00 :.0.....1..0p...
      4E 00 00 00 00 00 CD AE 9D EB FE 07 00 00 00 :.AF.....0.....
      6A 76 43 64 25 23 6D 55 53 2E 61 7D 21 43 :A_jvcdkamUS.a}IC
      35 61 5E 4C 6B 38 6E 2F 74 2F 3C 36 24 3D :q75a\Lh8n/t/<65=
      00 00 00 00 00 00 00 00 30 00 00 00 00 00 :.....0.....
      00 00 00 00 00 00 44 00 00 00 00 00 00 00 :.....D.....
      3D 00 00 00 00 00 AE 78 4E 77 00 00 00 00 00 :E.0.....8{Nw...
      00 00 00 00 00 00 44 00 00 00 00 00 00 00 :.....D.....
      39 00 02 83 30 00 00 00 00 00 00 00 00 00 :.9.....0.....
      01 00 00 00 00 00 01 00 00 00 00 00 00 00 :.....*..0p...
      00 00 00 00 00 00 13 BA 8D FB FE 07 00 00 00 :.....0p...
      4E 00 00 00 00 00 70 00 00 00 00 00 00 00 :.AF.....0.....
      00 00 00 00 00 00 70 FE 1E 00 00 00 00 00 00 :.....p.....
      
```

The malware uses these strings to call the CreateDirectory API at C:\ProgramData\VMWare and later to specify a filename at this location for a dropped payload (vmnat-update.bin).

These decoded strings also contain an export and an additional parameter for this dropped payload.

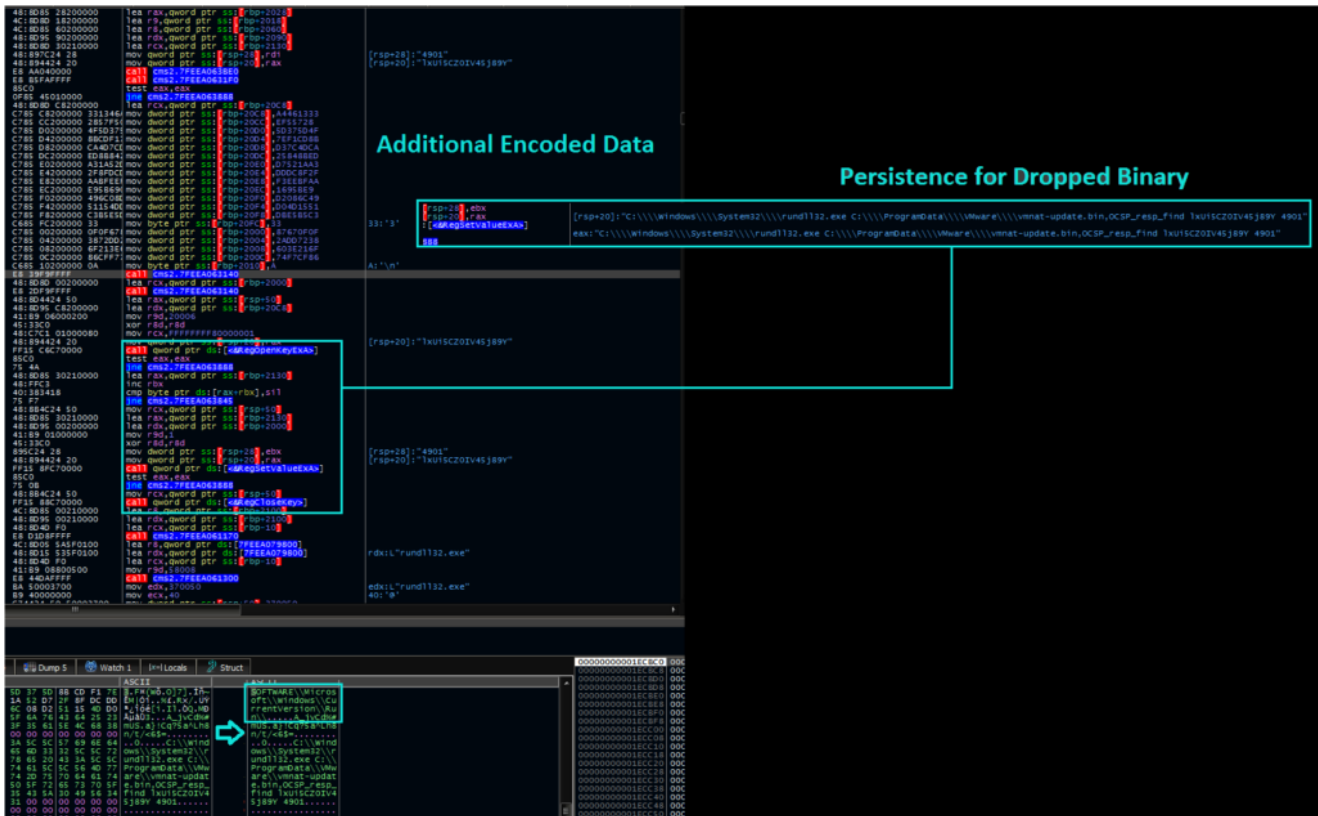
After this, the malware enumerates running processes via NtQuerySystemInformation and checks for the presence of avp.exe (Kaspersky) and avastui.exe (Avast). If either of these two processes are found, the malware will gracefully exit. If not, it continues with its workflow.



NtQuerySystemInformation Resolution (top)

and avp.exe and avastui.exe (bottom)

Following this step, the malware decodes a second set of strings. These are used to create persistence via the CurrentVersion\Run key under an entry named "OneDrive Update."



Once the malware has created persistence key, it writes a second-stage DLL to the “C:\ProgramData\VMWare\” directory and calls it using the previously decoded parameters.

Second Stage

Analysis of the second-stage payload is in-flight and additional details beyond what is listed below will be provided when available. It is possible (and perhaps likely, due to time constraints) that another researcher will complete this analysis before I do, in which case those details will be corroborated and added below for completeness, along with the appropriate credit.

- MD5 – f5475608c0126582081e29927424f338
- SHA-1 – 8e88fd82378794a17a4211bf2ee2506b9636b02
- SHA-256 – a75886b016d84c3eaacaf01a3c61e04953a7a3adf38acf77a4a2e3a8f544f855

The second-stage malware performs a similar command line check to verify that it is running with two supplied parameters. The first of these parameters, lxUi5CZ0IV45j89Y, is used as to create a mutex to ensure that only one copy of the malware is running at a time. If the mutex already exists, the malware will exit.

The malware then resolves a long list of API calls before jumping in to a section in memory. While this list is extensive, they indicate potential functionality, including C2 operations (HttpOpenRequest, HttpAddRequestHeaders, etc) and host-based operations (GetDesktopWindow, WriteFile).

Over the course of the last 24 hours, a lot of great research and analysis came to light from various parties. Most notably, I'd like to direct readers to three posts that offer additional context and demonstrate that the final action after this POST request is to download an additional payload onto the disk:

360 Threat Intelligence Center provides additional operational context for these attacks, including social engineering. This may be particularly valuable for threat hunters or threat intelligence practitioners. It also offers more details regarding the POST request and next-stage DLL in similar samples.

Qi'anxin Threat Intelligence Center identified similar activity (and malware) from this adversary in September 2020.

Anheng Threat Intelligence Center provides additional context regarding the social engineering and Visual Studio stages of this attack.