Infostealer Malware FormBook Spread via Phishing Campaign – Part I

April 22, 2025

Article Contents

By Xiaopeng Zhang | April 22, 2025

Fortinet’s FortiGuard Labs observed a phishing campaign in the wild that delivered a malicious Word document as an attachment. This document contained crafted data designed to exploit the vulnerability CVE-2017-11882. After conducting an in-depth analysis, I discovered that the campaign was spreading a new variant of Formbook.

Formbook is information-stealing malware targeting Windows users. It steals sensitive data from compromised systems, including stored credentials from popular software, the victim’s keystrokes, screenshots, and system clipboard data.

[2025 Global Threat Landscape Report](#)

[Use this report to understand the latest attacker tactics, assess your exposure, and prioritize action before the next exploit hits your environment.](#)

I will present my research into this malware in a series of analysis blogs. This first one provides insights into how the phishing email tricks the recipient into opening the attached Word document, how it exploits the vulnerability CVE-2017-11882 with crafted equation data, how it downloads and decrypts the fileless FormBook executable, and how it ultimately executes the FormBook malware in a selected target process via process hollowing.

Phishing Email Initialization

Figure 1: Example of the phishing email
The phishing campaign starts with an email disguised as a sales order urging the recipient to open the attached Word document. As shown in Figure 1, FortiMail has flagged the email as "[virus detected]" in the Subject line to warn the recipient.

Figure 2: Inner view of the Word document
The attached Word document, order0087.docx, is saved in OOXML (Office Open XML) format and compressed as a ZIP archive. Figure 2 reveals the relevant files and file contents inside the Word document.

When the recipient opens the Word document, the "<w:altChunk>" node within document.xml is parsed, automatically loading an external file, "Algeria.rtf," as shown in Figure 2.

Exploiting CVE-2017-11882

The RTF file ("Algeria.rtf") is obfuscated with a large amount of junk data. After de-obfuscating the RTF file, two embedded binary objects (leading by "\objdata" tag) can be found within it.

The first binary object is a package containing a 64-bit DLL file, as shown in Figure 3. The DLL file, named "AdobeID.pdf," is extracted to the system %temp% folder when the RTF is opened in Microsoft Word.

Figure 3: A 64-bit DLL file contained in the RTF file
The second binary object is in OLE format and contains crafted equation data intended for Microsoft Equation Editor 3.0. When Word parses the RTF file, this exploits the CVE-2017-11882 vulnerability.

CVE-2017-11882 is a known remote code execution (RCE) vulnerability in Microsoft Equation Editor (EQNEDT32.EXE).

Figure 4: The extracted equation data from the RTF file
Parsing the crafted equation data in EQNEDT32.EXE causes a buffer overflow, and the return address of the vulnerable function is overwritten with 0x430C12, as seen in Figure 4.

The instruction at 0x430C12 is "call ds:WinExec()," which is called after the buffer overflow occurs and the vulnerable function returns to this address. Figure 5 shows a screenshot of EQNEDT32.exe calling this API at 0x430C12.

Figure 5: WinExec() API about to be called
The command-line parameter to WinExec() API is "CmD.exe /C rundll32 %tmp%\AdobeID.pdf,IEX A". This command is copied by the vulnerable function from the crafted equation data, as shown in Figure 4. As a result, the extracted 64-bit DLL file,

AdobeID.pdf, is executed by rundll32.exe.

Dissecting the Extracted 64-bit Dll File

Based on my analysis, the 64-bit DLL acts as a downloader and installer for Formbook. Let's examine how it works.

Persistence Mechanism

It adds a key into "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run," with the following settings:

Key name: "RtkAudUService"
Key value: "C:\Windows\System32\rundll32.exe C:\Users\win-10\AppData\Roaming\Templates\AdobeID.pdf, IEX"

This ensures the malicious command-line command executes automatically at the system's startup. Figure 6 shows the added auto-run item in the system registry.

Figure 6: Registry entry for persistence
Correspondingly, it:

1> Creates a folder named "Templates" under the %appdata% folder.

2> Copies the extracted 64-bit DLL into the new folder by executing the following command-line command:

```
"cmd.exe /c copy C:\Users\%username%\AppData\Local\Temp\AdobeID.pdf
C:\Users\%username%\AppData\Roaming\Templates"
```

Formbook Payload Download

The Formbook payload is disguised as a picture file (PNG) to be downloaded. The URL to the PNG file is:

"hxxps://www2[.]0zz0[.]com/2025/02/02/10/709869215.png"

To retrieve the payload, it uses multiple Windows API functions, including:

WinHttpCrackUrl(),
WinHttpOpen(),
WinHttpConnect(),
WinHttpOpenRequest(),
WinHttpSendRequest(),
WinHttpQueryOption(),

WinHttpReceiveResponse(),
WinHttpQueryDataAvailable()
WinHttpReadData()

Payload Analysis

I manually downloaded the PNG file to analyze it. Its size is 0x47000 (284KB). I then observed that its content is encrypted rather than a legitimate PNG (Figure 7).

Figure 7: The encrypted content of the downloaded “PNG” file
The malware then calls a function to decrypt the PNG file into the FormBook executable binary, as seen in Figure 8. The decrypted FormBook can be seen in the memory at the bottom.

The decryption key is generated from a hardcoded string, “H1OX2WsqMLPKvGkQ.”

Figure 8: Decrypted FormBook executable in memory

Performing Process Hollowing

To evade detection by endpoint security software, the FormBook payload executable is kept entirely in the memory. Since it does not write the decrypted Formbook to a local file to execute, this is a fileless variant of FormBook.

Target Process

This variant of FormBook chooses “C:\Program Files (x86)\Windows Photo Viewer\ImagingDevices.exe” as a target process to run the decrypted FormBook via process hollowing.

The full path of the target process is hardcoded in the variant, which is passed to CreateProcessInternalW() API.

In Figure 9 below, we see that the process is about to call the CreateProcessInternalW() API, whose second parameter (RDX holds) is a string of the full path of the target process. The sixth parameter ([rsp+30]) represents the CreationFlag.

Figure 9: About to create a suspended target process
The value for the CreationFlag is 0x808040C, which is a combination of the following flags:

CREATE_SUSPENDED
DETACHED_PROCESS
CREATE_UNICODE_ENVIRONMENT
CREATE_NO_WINDOW

This ultimately creates a suspended “ImagingDevices.exe” process.

Mapping FormBook to Target Process

The malware employs sophisticated process injection techniques:

It creates a temporary and invisible file by calling two APIs:

GetTempFileNameW()
NtCreateFile()

It then calls NtWriteFile() to write the decrypted FormBook executable into the invisible temporary file.

Figure 10: Decrypted FormBook written to a hidden temporary file
Next, it calls NtCreateSection() API to create a memory section object from the temporary file, whose file handle is passed to the API. It then maps the section object into the target process (ImagingDevices.exe) by calling the Windows native API NtMapViewOfSection(). It also returns the address of the mapped FormBook inside the target process.

At that point, the decrypted FormBook has been copied into the target process.

Running FormBook

Next, it calls the Wow64GetThreadContext() API to retrieve the thread context (the CPU register value/state) of the suspended target process. It then modifies the value of some registers so the target process points to the mapped FormBook.

The corresponding API, Wow64SetThreadContext(), is then called to apply the modified registers to the target process.

Figure 11: FormBook about to call the Wow64SetThreadContext() API
Figure 11 shows the WOW64_CONTEXT data in memory. It only modifies two register values, EAX to 0x6E1550 and EBX to 0x2D0D0.

Why does it only modify those two registers rather than EIP? The selective modification of only EAX and EBX becomes clear when examining the target process's suspended state. The process has been intercepted at the execution of RtlUserThreadStart(), where EAX holds the thread function address and EBX contains the parameter to be passed to this thread function.

Now, the value of EAX holds the entry point of FormBook inside the target process, where FormBook’s base address is 0x6E0000.

It finally calls the Windows native API, NtResumeThread(), to resume the target process, and the RtlUserThreadStart() API is invoked to run the FormBook payload in a newly created thread.

Summary

In this first part of the blog series on the FormBook malware, I presented the entire process, from the phishing email to how the FormBook payload is downloaded, decrypted, and deployed in a targeted process.

Figure 12: Workflow diagram of this FormBook campaign
To start, I presented the phishing email we discovered that was designed to trick recipients into opening a fake sales order attachment. Once the attached Word document is opened on the targeted device, it extracts a 64-bit DLL file, disguised as "AdobeID.pdf," into the system's temporary folder. At the same time, it exploits the CVE-2017-11882 vulnerability in Microsoft Equation Editor 3.0 to execute the extracted DLL.

The DLL file is launched via rundll32.exe, establishing persistence on the victim's system by adding an auto-run item to the system registry. Additionally, it downloads and decrypts the FormBook executable file.

I then provided a detailed explanation of how the decrypted FormBook payload is deployed in a target process (C:\Program Files (x86)\Windows Photo Viewer\ImagingDevices.exe) through process hollowing, as well as the Windows-native APIs it calls to accomplish this task.

In Part II of this series, I will present the anti-analysis techniques used in this variant of FormBook and how FormBook leverages the Heaven's Gate technique to prevent analysis and detection. I'll also cover how it collects sensitive data from the compromised device, its communications with its command and control (C2) server, and the specific C&C commands it uses to control the victim's system. Stay tuned.

Fortinet Protections

Fortinet customers are already protected from this campaign with FortiGuard's AntiSPAM, Web Filtering, IPS, and AntiVirus services as follows:

The FortiGuard's Anti-Botnet Service has blocked the DNS requests for downloading the FormBook variant.

The relevant URL downloading the PNG file is rated as "**Malicious Websites**" by the FortiGuard Web Filtering service.

FortiMail recognizes the phishing email as “virus detected.” In addition, real-time anti-phishing provided by FortiSandbox embedded in Fortinet’s FortiMail, web filtering, and antivirus solutions provides advanced protection against both known and unknown phishing attempts.

FortiGuard IPS service detects the vulnerability exploit against CVE-2017-11882 with the signature “**MS.Office.EQNEDT32.EXE.Equation.Parsing.Memory.Corruption**”.

FortiGuard Antivirus service detects the malicious Word document, the embedded RTF file, the extracted 64-bit Dll file as well as the decrypted FormBook with the following AV signatures.

MSWord/Formbook.9184!tr
RTF/CVE_2017_11882.FB!exploit
W64/Formbook.RT!tr
W32/Formbook.AA!tr

FortiGate, FortiMail, FortiClient, and FortiEDR support the FortiGuard AntiVirus service. The FortiGuard AntiVirus engine is part of each solution. As a result, customers who have these products with up-to-date protections are already protected.

To stay informed of new and emerging threats, you can sign up to receive future alerts.

We also suggest our readers go through the free NSE training: NSE 1 – Information Security Awareness, a module on Internet threats designed to help end users learn how to identify and protect themselves from phishing attacks.

If you believe this or any other cybersecurity threat has impacted your organization, please contact our Global FortiGuard Incident Response Team.

IOCs

URLs

hxxps://www2[.]0zz0[.]com/2025/02/02/10/709869215.png

Relevant Sample SHA-256

[order0087.docx]
93CF566C0997D5DCD1129384420E4CE59764BD86FDABAAA8B74CAF5318BA9184

[Algeria.rtf]
7C66E3156BBE88EC56294CD2CA15416DD2B18432DEEDC024116EA8FBB226D23B

[AdobeID.pdf]
2E73B32D2180FD06F5142F68E741DA1CFF1C5E96387CEBD489AD78DE18840A56

[Decrypted FormBook from PNG file]
6AC778712DFFCE48B51850AC34A846DA357BE07328B00D0B629EC9B2F1C37ECE