Deep Analysis: FormBook New Variant Delivered in Phishing Campaign – Part II

fortinet.com/blog/threat-research/deep-analysis-formbook-new-variant-delivered-phishing-campaign-part-ii

April 21, 2021

FortiGuard Labs Threat Research Report

Affected platforms:	Microsoft Windows
Impacted parties:	Windows Users
Impact:	Collect Sensitive Information from Victim's Devices
Severity level:	Critical

This is part II of a threat analysis series examining a phishing campaign that FortiGuard Labs captured in our SPAM monitoring system. The sample we captured was attempting to deliver "FormBook" malware through a PowePoint document attached to an email. FormBook is a malware designed to steal sensitive information from a victim's device as well as to receive control commands to perform additional malicious tasks on that device.

In the Part I of my analysis, I explained how the VBA code in the PowerPoint file was used to download a PowerShell file, how it extracts a .Net framework file, and how the FormBook payload file is processed through three .Net modules.

In this second part, we will examine what anti-analysis techniques FormBook performs, what Windows processes it focuses on, and how the FormBook malware running inside AddInProcess32.exe injects itself into a randomly-picked Window process. Furthermore, we will see how FormBook injects itself into a number of target processes through the Windows process.

Payload File Runs in “AddInProcess32.exe”

As mentioned in part I of this analysis, a FormBook payload is injected into a newly-created process, “AddInProcess32.exe”, and the relevant registers are set to point to the entry of the injected FormBook. After that, the entry point is called after executing the API ResumeThread() by the AMe8 module—which is the point that I will start from in this post.

The payload file of the FormBook malware is a 32Bit Native Code PE file (an EXE file), not a .Net module. Figure 1.1 is a screenshot of the entry point function of FormBook.

Figure 1.1 - The entry point function of the FormBook malware

Configuration Object and Anti-Analysis Techniques

Configuration Object:

Before discussing the FormBook main module, I need to introduce a global configuration object (“ConfigObj”) or configuration block, which is frequently read and written throughout the entire FormBook malware. It occupies 0xC9C bytes containing many configuration options, such as:

- the base addresses of FormBook, and many Dll modules (ntdll.dll, kernel32.dll, advapi32.dll, etc.)
- encrypted Dll names like “kernel32.dll” and “advapi32.dll”, etc.
- a “flag group” revealing whether FormBook is running in an analysis device
- many encrypted blocks with string hash codes for retrieving APIs
- flags revealing if it is on a 32-bit or 64-bit platform
- many API addresses (ExitProcess(), CreateProcessInternalW(), etc.)
- and so on

Figure 2.1 is a screenshot of part of the ConfigObj in a memory view that had just been initialized.

Figure 2.1 - Memory view of part of ConfigObj

Anti-analysis Techniques Used by FormBook:

1. Imported APIs are hidden:

Figure 2.2 – Example of hidden APIs with their hash codes

All APIs are hidden from analysts in FormBook. They are retrieved by a special function with the APIs' name hash code. Some hash codes are given by constant value and some are decrypted from ConfigObj.

As you can see, the code segment shown in Figure 2.2 is an example of obtaining two APIs via the renamed function get_API_by_name_hashcode() with their name's hash codes, which are 0C84882B8h => CoCreateInstance() and 44E954F9h => CreateDirectoryW().

2. Duplicating the ntdll.dll module:

ntdll.dll is the kernel layer DLL on Windows that provides NT kernel APIs. FormBook deploys a duplicated ntdll.dll in its memory offering kernel APIs function instead of the original one that prevents researchers from identifying the APIs. Figure 2.3 shows the duplicated ntdll.dll (0x810000) at the upper and the originally-loaded ntdll.dll (0x77140000) at the bottom.

Figure 2.3 – The duplicated ntdll.dll in Memory Map view

3. Detecting whether it is running in an analysis environment:

FormBook compares predefined hash codes in a list that is decrypted from ConfigObj with running process name's hash code. It calls the API ZwQuerySystemInformation() with the parameter SystemProcessInformation to gather information about all running processes into a link structure.

It then calculates the process name's hash code one by one and compares it against those predefined hash codes. The process names corresponding to predefined hash codes are about VMware, Virtual Box, Sandboxie, Parallels Desktop, and other analysis tools for monitoring files, processes, and network and system registry events. Following is a list of those processes.

vmwareuser.exe, vmwareservice.exe, VBoxService.exe, VBoxTray.exe, sandboxiedcomlaunch.exe, sandboxierpcss.exe, procmon.exe, filemon.exe, wireshark.exe, NetMon.exe, prl_tools_service.exe, prl_cc.exe, vmtoolsd.exe, vmsrvc.exe, vmusrvc.exe, python.exe, perl.exe, regmon.exe

It also records any match results as a flag in the "flag group" of ConfigObj.

4. Detecting file names, user names, path:

It then calculates the hash codes of strings that are retrieved from current process names (the process name may be renamed by researchers), user names (famous sandboxes use fixed user name), and the list of split strings of loaded modules' path string (analysis tool's modules). It then checks these hash codes with the predefined hash codes in FormBook. The result affects the "flag group".

5. Detect any debuggers:

Next, it retrieves the API ZwQueryInformationProcess() from the duplicated ntdll.dll and calls it with different parameters to obtain SystemKernelDebuggerInformation data. This is used to check the kernel debugger and ProcessDebugPort data to identify a ring3 debugger. Figure 2.4 displays the two parameters used to obtain debugger information.

Figure 2.4 – Detecting the Kernel and Ring3 debugger

6. Detect the time gap from executing instructions:

This detection has been disabled in this variant. It is used to determine if FormBook is being debugged by comparing the gap time of executing ASM instructions > 300h (should be less than 300h). It has hardcoded the value to 50h to disable this detection. It also records the result in the "flag group" in ConfigObj.

7. Encrypted key functions:

There are five segments of key functions that are encrypted and decrypted before injecting into the target processes. The five segments are identified by five magic codes, which are 48909090h, 49909090h, 4A909090h, 4B909090h, and 4C909090h.

8. Using many undocumented APIs:

FormBook uses many low level undocumented APIs, such as LdrGetProcedureAddress(), LdrLoadDll(), ZwOpenProcessToken(), ZwAdjustPrivilegesToken(), NtOpenProcessToken(), ZwReadVirtualMemory(), RtlQueryEnvironmentVariable(), RtlDosPathNameToNtPathName_U(), ZwDelayException(), ZwQueueApcThread(), and so on.

The so-called "Undocumented API" simply means the API is hidden to Windows users. You are unable to gain any official description for the API from MSDN.

There is a special function to check the result in "flag group" that is set in some detections. Once one detection is triggered, it returns 0, otherwise it returns 1. Below is the pseudocode of this function, whose parameter is the ConfigObj. The flag group occupies the bytes from offset 40 (0x28) to 55 (0x37).

Int __cdecl **sub_407D50**(unsigned char* pConfigObj)

```
{

return !*(pConfigObj + 41)  && *(pConfigObj + 42) && *(pConfigObj + 43)

&& !*(pConfigObj + 44)  && !*(pConfigObj + 45) && *(pConfigObj + 46)

&& !*(pConfigObj + 47)  && *(pConfigObj + 48)  && !*(pConfigObj + 49)

&& *(pConfigObj + 50) && !*(pConfigObj + 51);

}
```

If the result of the function is 0, it then exits the process without doing anything.

You cannot simply change the result (from 0 to 1) here to ignore detection and change the code flow. The reason is that in the next step, the “flag group” (10H long) will be an RC4 seed to generate RC4 keys to finally decrypt other data, like module names such as “kernel32.dll” and “advapi32.dll”. It could also fail to load these modules if the “flag group” is wrong.

The Outline of FormBook’s Tasks

Figure 3.1 – Outline of what FormBook does on a victim’s device

Figure 3.1, above, outlines most FormBook actions that are performed on a victim’s device. FormBook’s AddInProcess32.exe executable injects itself into a newly-created Windows process (like ipconfig.exe) that is created through Explorer.exe (**steps 1, 2, and 3**). Then, once FormBook is inside the Windows process, it injects malicious code into target processes (FormBook focuses on 92 different target processes in total, including “iexplorer.exe”, “chrome.exe”, “skype.exe”, “outlook.exe”, “whatsapp.exe”, and so on) from which it steals victim inputs and clipboard data from time to time (**step 4**).

It also uses a large, shared memory section for storing stolen data gathered from the FormBook instance running inside target processes and the FormBook instance running in a Windows process (**step 5**).

The stolen data is then sent to its C2 server via the FormBook instance running inside “Explorer.exe” (**step 6**).

I will elaborate on how it performs these actions in the rest of this blog.

Deploy FormBook Into a Windows Process via Explorer.exe

The FormBook payload running inside AddInProcess32.exe looks for Explorer.exe by comparing the hash codes of running processes’ names, which it obtains by calling the API ZwQuerySystemInformation() with the parameter 0x5 (SystemProcessInformation).

The hash code of explorer.exe is 19996921h. As you can see in Figure 4.1, it is an ASM code snippet showing you how FormBook finds explorer.exe by comparing its hash code with the hash code of other processes through a function that I call match_hashcode().

Once explorer.exe is matched, the function returns 1 and FormBook proceeds to the next step. Otherwise, it retrieves the next running process name to match in a loop.

Figure 4.1 – Code snippet comparing the explorer.exe hash code

Next, FormBook opens the process handle of Explorer.exe, allocates memory to it, and then copies the entire FormBook payload into that Explorer.exe memory. It then proceeds to execute FormBook from the different entry point within a newly-started thread of Explorer.exe.

To do this, it calls a number of APIs, including ZwOpenProcess(), ZwCreateSection(), ZwMapViewOfSection(), ZwOpenThread(), ZwSuspendThread(), ZwGetContextThread(), ZwSetContextThread(), and ZwResumeThread().

The logic and features of the FormBook instance injected into Explorer.exe is very clear and simple. It is to run a randomly selected Windows process (that locates at %Windir%\system32\) in suspended mode and return with the process status and information.

The Windows process name list is encrypted within ConfigObj (starting at offset +6Bh) and is picked by its index. It has thirty-eight such Windows process names in total (the string index range is from 0x3 to 0x29), which are decrypted and listed below:

"svchost.exe", "msiexec.exe", "wuauclt.exe", "lsass.exe", "wlanext.exe", "msg.exe", "lsm.exe", "dwm.exe", "help.exe", "chkdsk.exe", "cmmon32.exe", "nbtstat.exe", "spoolsv.exe", "rdpclip.exe", "control.exe", "taskhost.exe", "rundll32.exe", "systray.exe", "audiodg.exe", "wininit.exe", "services.exe", "autochk.exe", "autoconv.exe", "autofmt.exe", "cmstp.exe", "colorcpl.exe", "cscript.exe", "explorer.exe", "WWAHost.exe", "ipconfig.exe", "msdt.exe", "mstsc.exe", "NAPSTAT.EXE", "netsh.exe", "NETSTAT.EXE", "raserver.exe", "wscript.exe", "wuapp.exe", "cmd.exe".

Figure 4.2 – Display of one decrypted Windows process in Explorer.exe

Figure 4.2 shows its random function and decryption function, as well as a just-decrypted Windows process named “ipconfig.exe”, with the random string index 0x20. I will use ipconfig.exe to explain how FormBook works with a Windows process.

It first calls the API CreateProcessInternalW() with the ipconfig.exe full path and dwCreationFlags parameter of 0x800000C, which means “CREATE_NO_WINDOW|CREATE_SUSPENDED|DETACHED_PROCESS”. This will then start ipconfig.exe with no window and in suspended mode.

The FormBook instance in Explorer.exe will continue to collect the process information of ipconfig.exe (such as its full path, the process ID, the thread ID, loaded base address, etc.) and return them to FormBook in AddInProcess32.exe. At this point, the work of FormBook inside Explorer.exe is done.

Why doesn't it run the Windows process directly, rather than through Explorer.exe? In some analysis tools, doing it this way shows that the Windows process (ipconfig.exe) was started from Explorer.exe, the same as normal processes started by the victim. This helps hide itself from analysts as well as the victim. Another trick it uses is that the processes are all Windows default processes, which makes it less likely for users and analysts to connect it to a malware. As you can see in Figure 4.3 taken from the Explorer process, ipconfig.exe is recognized under explorer.exe, which is the same as other processes, such as "notepad.exe" and "calc.exe", which I opened by double clicking their icons.

Figure 4.3 – ipconfig.exe is recognized under explorer.exe

FormBook in AddInProcess32.exe then obtains the process information of the suspended ipconfig.exe that is returned from Explorer.exe. It is then able to copy the FormBook payload file into ipconfig.exe and modify its main thread's entry point code to the new entry point of the injected FormBook. It eventually calls the API ZwResumeThread() to resume ipconfig.exe in order to execute FormBook's malicious code. At the same time, it calls ExitProcess() to terminate the lifetime of the FormBook instance injected into AddInProcess32.exe.

FormBook in Windows Process is Injected Into Target Processes

The injected FormBook instance running in a Windows process, like ipconfig.exe, takes the control of maintaining its life on the victim's device.

First, it initializes its own ConfigObj and performs the detections that I explained earlier in the anti-analysis section.

It is then time for FormBook to decrypt five key functions that will be called within target processes after FormBook has been injected into them. It has a magic code for each of these functions, which are 48909090h, 49909090h, 4A909090h, 4B909090h, and 4C909090h.

FormBook finds the encrypted code by searching the magic codes in the entire code section and then decrypts them using an RC4 algorithm. The RC4 decryption key is generated from data in ConfigObj. Figure 5.1 shows the encrypted code for a key function for magic code 48909090h on the left side, and the decrypted code on the right side.

Figure 5.1 – Display of both encrypted and decrypted code for a key function

The five decrypted key functions are used to perform C2 relevant work, like decrypting the C2 host strings, loading network APIs, and communicating with C2 servers, etc.

A function in FormBook focuses on filtering target processes from gathered current processes by calling the API ZwQuerySystemInformation() with the SystemProcessInformation parameter by comparing a process name's hash code with the predefined hash codes in FormBook that are saved in ConfigObj. Once a process's hash code is matched with its predefined hash code, it initiates a function to inject FormBook into the matched process and then executes code from different entry points set inline hooks for stealing data.

Figure 5.2 shows a pseudocode of the code flow structure of how FormBook filters a target process and calls a function to inject FormBook into that process once the process name matches a predefined hash code. FormBook performs this check every five seconds to better cover newly opened target processes.

Figure 5.2 – Pseudocode of the code-flow finding target processes

There are a total of 92 predefined target processes in this variant of FormBook, which has an encrypted hash code list of process names saved in ConfigObj, starting at offset + 444h. I haven't yet defined all of the target process names by their hash codes. However, through my analysis, I have identified most of the products the target processes belong to. They can be divided into several categories based on their features, as shown below:

Web browsers:

Google Chrome, Microsoft IE and Edge, Mozilla Firefox, Opera Browser, Apple Safari, Torch Browser, Maxthon Browser, SeaMonkey Browser, Avant Browser, Comodo Dragon and IceDragon, K-Meleon Browser, BlackHawk Browser, Cyberfox Browser, Vivaldi Browser, Lunascape Browser, Epic Browser, Midori Browser, Pale Moon Browser, QtWeb Browser, Falkon Browser, UCBrowser, Waterfox Browser, and so on.

Email clients:

Microsoft Outlook, Pocomail, Opera Mail, Tencent Foxmail, IncMail, Mozilla Thunderbird, Google Gmail Notifier Pro, and so on.

IM clients:

Yahoo Messenger, ICQ, Pidgin, Trillian, Microsoft Skype, FaceBook WhatsApp, and so on.

FTP clients:

Estsoft ALFTP, NCH Classic FTP, Core Ftp, FAR Manager, FileZilla, FlashFXP, NCH Fling, FTP Voyager, WinSCP, and so on.

Others:

Windows Notepad and “Explorer.exe”.

The detailed target processes are

"iexplore.exe", "firefox.exe", "chrome.exe", "microsoftedgecp.exe", "opera.exe", "safari.exe", "torch.exe", "Maxthon.exe", "seamonkey.exe", "avant.exe", "dragon.exe", "icedragon.exe", "kmeleon.exe", "blackhawk.exe", "Cyberfox.exe", "Vivaldi.exe", "luna.exe", "Epic.exe", "Midori.exe", "palemoon.exe", "QtWeb.exe", "qupzilla.exe", "UCBrowser.exe", "Waterfox.exe", "notepad.exe", "explorer.exe", "outlook.exe", ";poco.exe", "operamail.exe", "foxmail.exe", "incmail.exe", "thunderbird.exe", "Barca.exe", "gmailNotifierPro.exe", "yahoomessenger.exe", "icq.exe", "pidgin.exe", "Trillian.exe", "skype.exe", "WhatsApp.exe", "alftp.exe", "classicftp.exe", "coreftp.exe", "Far.exe", "filezilla.exe", "FlashFXP.exe", "fling.exe", "FTPVoyager.exe", "WinSCP.exe".

It calls the API ZwOpenProcess() and then ZwCreateSection() to open a target process and then create a section of memory in it. Next, it transfers the entire FormBook along with the decrypted five key functions into the section. It then executes it from a new entry points within a newly-created thread in the target process.

Conclusion on Phishing Campaign

In this part II, I started my analysis from the point where the FormBook payload file is injected into the AddInProcess32.exe process. At first, I introduced an important data structure—Configuration Object—which holds the key configuration data that is used throughout FormBook for whatever it is injected into. I then elaborated on the anti-analysis techniques that FormBook performs, how it then selects a process from the thirty-nine Windows processes (like ipconfig.exe) it looks for, and then injects FormBook using Explorer.exe as a middle process. And finally, through my research on the hash codes of the process name, I was able to recover most of the target processes that FormBook is interested in.

In the final part of this analysis, I will explain how FormBook establishes inline hooks on some APIs inside target processes, what kind of data it can steal from a victim’s device, how the stolen data is sent to the C2 server, what its control commands are able to do on a victim’s machine, as well as the strategy used to have various FormBook instances work together across the Windows processes (ipconfig.exe), Explorer.exe, and target processes.

Fortinet Protections

Fortinet customers are already protected from this FormBook variant with FortiGuard’s Web Filtering and AntiVirus services, as follow:

The download URL launched from the PowerPoint sample is rated as "**Malicious Websites**" by the FortiGuard Web Filtering service.

The attached PowerPoint file is detected as “**VBA/FormBook.C393!tr**” and the “item3.jpg” file is detected as "**MSIL/FormBook.ZXL!tr**" and blocked by the FortiGuard AntiVirus service.

The FortiGuard AntiVirus service is supported by FortiGate, FortiMail, FortiClient, and FortiEDR. The Fortinet AntiVirus engine is a part of each of those solutions as well. As a result, customers who have these products with up-to-date protections are protected.

Besides, FortiSandbox is able to detect the PowerPoint sample as malicious.

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

Learn more about FortiGuard Labs threat research and the FortiGuard Security Subscriptions and Services portfolio.

Learn more about Fortinet’s free cybersecurity training initiative or about the Fortinet NSE Training program, Security Academy program, and Veterans program.