01/02/2023

Riccardo Ancarani

Jojo O'Gorman

Introduction

OneNote is a software part of the Office suite, commonly used within most organisations for note-keeping, task management and more. In the last year, OneNote gained more attention from a security perspective, mostly thanks to the research paper published by Emeric Nasi. In fact, OneNote's .one file presented various characteristics that are potentially interesting from an attacker's perspective. Specifically, the ability to attach files in existing OneNote notebooks that victims can execute with minimal warnings was of interest. This was particularly relevant as attackers were shifting their techniques towards a heavy containerization of their initial access payloads, to either avoid security features such as Mark of the Web (MOTW) or evade sandboxing.

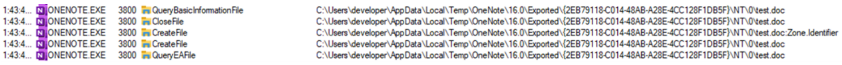
Moreover, Emeric Nasi found in his research that OneNote did not propagate MOTW on its attachments. This meant that attackers could, for example, embed unsigned executables and not being subject to SmartScreen or even better, attach macro-enabled documents and avoid the restriction that Microsoft recently put in place.

This post will discuss various abuse cases associated with the OneNote file formats and their respective detection strategies. The examples shown were re-created by WithSecure and detonated in a controlled environment with various monitoring tools, such as ProcMon and Sysmon configured with Florian Roth's recommended settings

Silent Patch and Mark Of The Web

As of 18/01/2023, WithSecure observed that the ability to bypass MOTW for attachments within .one notebooks was patched, and therefore all the attachments that OneNote will write on disk will have MOTW by default, even if the .one file does not come from the internet.

The figure below is taken from ProcMon and shows the OneNote.exe process appending the MOTW on the created file:

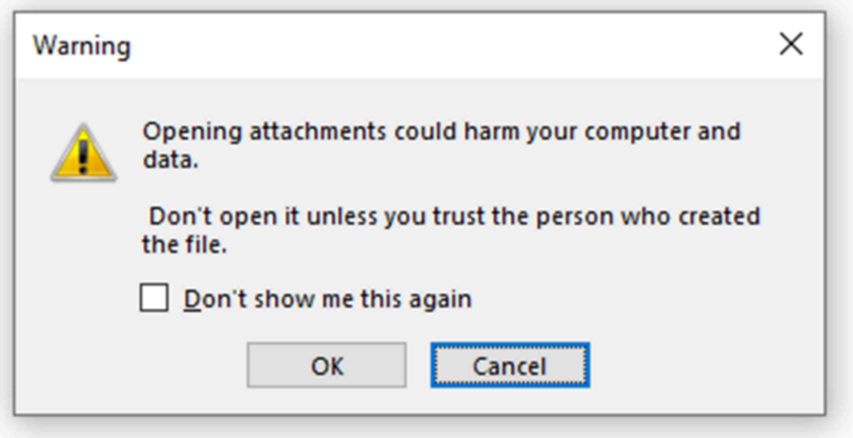


```
1:43:4...  ONENOTE.EXE  3800  QueryBasicInformationFile   C:\Users\developer\AppData\Local\Temp\OneNote\16.0\Exported\{2EB79118-C014-48AB-A28E-4CC128F1DB5F}\NT\0\test.doc
1:43:4...  ONENOTE.EXE  3800  CloseFile                   C:\Users\developer\AppData\Local\Temp\OneNote\16.0\Exported\{2EB79118-C014-48AB-A28E-4CC128F1DB5F}\NT\0\test.doc
1:43:4...  ONENOTE.EXE  3800  CreateFile                  C:\Users\developer\AppData\Local\Temp\OneNote\16.0\Exported\{2EB79118-C014-48AB-A28E-4CC128F1DB5F}\NT\0\test.doc:Zone.Identifier
1:43:4...  ONENOTE.EXE  3800  CreateFile                  C:\Users\developer\AppData\Local\Temp\OneNote\16.0\Exported\{2EB79118-C014-48AB-A28E-4CC128F1DB5F}\NT\0\test.doc
1:43:4...  ONENOTE.EXE  3800  QueryEAFile                 C:\Users\developer\AppData\Local\Temp\OneNote\16.0\Exported\{2EB79118-C014-48AB-A28E-4CC128F1DB5F}\NT\0\test.doc
```

This greatly decreases the risk associated with .one files, but does not completely eliminate it. The rest of the post will describe various abuse-cases and possible detection opportunities.

Case 1 - Executable Attachment

Attackers can still embed executables in OneNote sections and lure users to execute them with various pretexts. When double clicking an attachment, OneNote will warn the user with the following message:

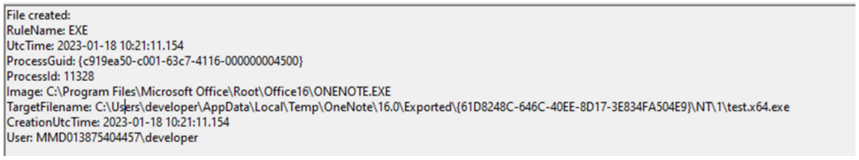


Opening attachments could harm your computer and data.

Don't open it unless you trust the person who created the file.

☐ Don't show me this again

If the user confirms, the executable will be written on disk and then executed. We can observe it from Sysmon event ID 11:

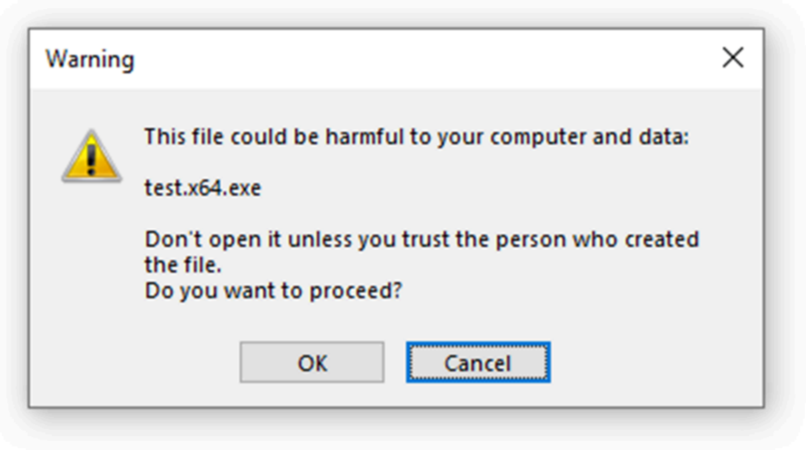


```
File created:
RuleName: EXE
UtcTime: 2023-01-18 10:21:11.154
ProcessGuid: {c919ea50-c001-63c7-4116-000000004500}
ProcessId: 11328
Image: C:\Program Files\Microsoft Office\Root\Office16\ONENOTE.EXE
TargetFilename: C:\Users\developer\AppData\Local\Temp\OneNote\16.0\Exported\{61D8248C-646C-40EE-8D17-3E834FA504E9}\NT\1\test.x64.exe
CreationUtcTime: 2023-01-18 10:21:11.154
User: MMD013875404457\developer
```

As we can see from the image above, the file is written in OneNote's temporary directory, %TEMP%\OneNote\16.0\Exported.

Additionally, if the user ticks or has ticked previously the box "Don't show me this again" the following registry key will be created:

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Office\16.0\OneNote\Options\EmbeddedFileOpenOptions with a DWORD key named EmbeddedFileOpenWarningDisabled set to 1. A value of zero would revert it back to the normal setting. If the value is 1, however, OneNote will still display the warning but with a slightly different message:



This file could be harmful to your computer and data:

test.x64.exe

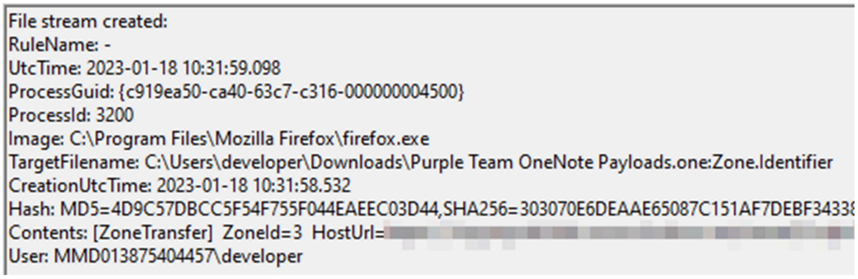
Don't open it unless you trust the person who created the file.
Do you want to proceed?

It must be noted that this setting applies to all the notebooks opened from the machine were the registry key is present. The message above will appear when executing potentially dangerous files, but will automatically open the attachment in case of “safer” content such as office documents.

Detection

From a detection perspective, it would be possible to look for file write events where the target path matches the OneNote directory mentioned above. A first approach would be to look only for specific file extensions as it could easily lead to a concerning amount of false positives.

To even refine the detection, it would be possible to look for executable content dropped by OneNote.exe after a .one file is downloaded from the internet. This is possible by using Sysmon event ID 15 (file stream created). As shown below, we can see that an ADS associated with MOTW is created after the .one file is downloaded form an external source:



```
File stream created:
RuleName: -
UtcTime: 2023-01-18 10:31:59.098
ProcessGuid: {c919ea50-ca40-63c7-c316-000000004500}
ProcessId: 3200
Image: C:\Program Files\Mozilla Firefox\firefox.exe
TargetFilename: C:\Users\developer\Downloads\Purple Team OneNote Payloads.one:Zone.Identifier
CreationUtcTime: 2023-01-18 10:31:58.532
Hash: MD5=4D9C57DBCC5F54F755F044EAEEC03D44,SHA256=303070E6DEAAE65087C151AF7DEBF3433
Contents: [ZoneTransfer]  ZoneId=3  HostUrl=[redacted]
User: MMD013875404457\developer
```

One thing to note (no pun intended) is that the integer after the NT part of the path is an incremental value that changes depending on how many attachments the user clicks from the same OneNote file; therefore that value should be considered non-static for detection engineering purposes.

Prevention

This type of abuse can be prevented using different strategies:

- By deploying application control and therefore blocking unsigned EXEs from being executed
- By enabling Attack Surface Reduction with the rule “Blocking Office applications from creating child processes”

Both solution present problems, for instance application control can be easily bypassed if not properly configured and some LOLBins will almost certainly be allowed.

The ASR configuration might be impractical to enforce across an estate of a considerable size, but most importantly there are LOLBins that will bypass this rule, such as explorer.

An experiment was also made by leveraging Sysmon's 14.0's FileBlockExecutable rule, so that the OneNote.exe process cannot write executable content on disk. A snippet of a Sysmon configuration file that implements the prevention mechanism can be found below:

```
<RuleGroup name="ImageBlock"
groupRelation="or">

     <FileBlockExecutable
onmatch="include">

             <Image
condition="image">onenote.exe</Image>


</FileBlockExecutable>

</RuleGroup>
```

However, due to unknown reasons, the rule above never triggered when OneNote dropped executable files on disk. This might berelated to some of the known limitations of this capability. It is therefore recommended not to rely on this to protect against this particular use-case.

Case 2 - Living-off-the-Land Binaries and LNKs

Another common abuse case that was observed being used in the wild consisted in attaching content to a OneNote section that, by default, will be executed with a

LOLBin. Examples of this can be HTA, CHM, CPL, XLL or LNK files.

For this scenario, we will start by detonating an XLL file that was attached in a .one container and obtain the following:

By enabling the add-in, we would then observe the following events:

- Process creation of a process with the current directory being set to the OneNote temporary directory and a command line with the same folder in it. The parent process would be OneNote.exe

- Module Load event where the Image path contains “OneNote\16.0\Exported”

Detection

Considering the variety of LOLBins and the huge difference in their behaviour, it seems that the most reliable strategy is to look for OneNote.exe spawning child processes that are also LOLbins. This obviously presents a challenge as many LOLBin-like programs (such as all the Office products) are likely being used legitimately most of the time. Analysis of the command line is therefore needed to cover edge cases, such as the one presented by the XLL file.

Case 3 - RTLO Spoofing

Another interesting edge-case is when the attackers spoof the extension of the file using the Right-to-Left Override (RTLO) technique (https://blog.sevagas.com/?Bypass-Defender-and-other-thoughts-on-Unicode-RTLO-attacks). In this case, despite the process-tree will remain the same, the file name found in the file-write event might not reflect the actual content of the dropped file.

The example below shows the creation of an HTA file with a spoofed extension to make it look like a PNG:

Detection

Despite there are no trivial ways of detecting this specific behaviour, the parent-child process relationship will remain the same and therefore the considerations made in the previous sections should still apply.

Case 4 - Maldoc Attachment

Despite the MOTW patch now prevents documents that were embedded in a .one section to have unsigned macros, it is possible to imagine that attackers will:

- Sign their macro
- Find a MOTW bypass

It is therefore recommended to pay particular attention to module load events of VBE7.dll from files that were written in the OneNote temporary folder

Other Detection Methods

Static Scanning

From empirical experiments, it appeared that OneNote files are not being parsed correctly by various security solutions. That, combined with the fact that .one files are not part of the Outlook attachment blacklist are making .one files appealing for attackers.

From what it was possible to observe, without digging into the OneNote file format, all the embedded attachments are simply appended within the .one file.

The figure below shows the header of an embedded EXE that was attached for testing:

All the YARA-based detection should still apply and be effective against .one files. If OneNote files were part of some allow-list, it would be recommended to remove it. A simple test consists in embedding a known-malicious payload, such as Mimikatz, and applying the yara rule against the .one file:

```
PS C:\Users\developer\Desktop>
.\Tools\yara64.exe -s
..\Downloads\mimi.yar '.\Purple Team
OneNote Payloads.one'

mimikatz .\Purple Team OneNote Payloads.one

0x1bb40c:$exe_x64_1: 33 FF 41 89 37 4C 8B
F3 45 85 C0 74

0x1bb41c:$exe_x64_1: 33 FF 45 89 37 48 8B
F3 45 85 C9 74

0x1bb42c:$exe_x64_1: 33 FF 41 89 37 4C 8B
F3 45 85 C9 74

0x1bb3cc:$exe_x64_2: 4C 8B DF 49 C1 E3 04
48 8B CB 4C 03 D8
```

Despite not realistic, this shows that the attachments within OneNote files have no built-in compression or obfuscation.

Sequential Containerisation

As it became obvious from most of the recent initial-access campaigns spotted in the wild, attackers tend to pack their malware in multiple containers, such as ZIP files or ISOs. Now that OneNote files are gaining traction, it is expected to see TAs packaging .one within other containers or vice-versa, creating endless – and sometime useless – chains that someone will eventually end up posting on Twitter.

OneNote Packages (.onepkg)

OneNote offers the ability to export an entire notebook in a packaged file with the extension of .onepkg. The figure below shows the export of an entire notebook in the .onepkg format:

When the file is double-clicked, the user will be presented a file browser windows to decide where to store the packaged .one files. Once the file is saved, OneNote will open it automatically and display its content.

From an attacker’s point of view, despite .onepkg files are not extremely common to see being shared, they could potentially evade some static scanning detections. This is due to the fact that the embedded .one files are not simply appended similarly to what we saw before, but packaged in CAB file.

If we inspect the header of the .onepkg, we can see that the magic bytes “MSCF” correspond to a CAB file:

If we extract its content with the “expand” utility:

```
expand "Open Sections.onepkg" one

Microsoft (R) File Expansion Utility

Copyright (c) Microsoft Corporation. All
rights reserved.

Adding one\open sections.onepkg to
Extraction Queue

Expanding Files ....

Expanding Files Complete ...
```

We can then see that another onepkg file was extracted, this time displaying the entire content of its attachments as shown with the classic .one files.

From a defense perspective the previous recommendations apply, it is however recommended to block the download of .onepkg files as well.

Abuse in the Wild

As mentioned before, OneNote is being abused more frequently by threat actors. An example of campaign that leveraged the RTLO technique described in this post was analysed in great details by the researchers at SpiderLabs in their post Trojanized OneNote Document Leads to Formbook Malware.

Other, and more recent, evidence of abuse of OneNote were also highlighted from the Perception Point Attack

Trends researchers:

The analysed sample, alongside other similar one downloaded from VirusTotal, showed that at the moment the preferred technique is to embed some sort of script, either a VBS or an HTA file. This is useful for defenders as it give a priority to what techniques to look for.

For example, on the sample mentioned above, by leveraging the OneNoteAnalyzer tool by knight0x07 we can extract the HTA attachment:

In this case the sample was using the RTLO technique as well, and we can see that the execution method chosen by the attacker was using PowerShell launched via WMI:

The sample was also downloading a decoy .one from the internet, probably to show some content to the victim without raising suspicion. Note that the available analysers cannot parse .onepkg files yet, so they might have to be manually unpacked first.

Another, apparently common, technique that is being used in the wild consists in utilising images in front of attachments to avoid user suspicion.

In a nutshell, if you put an image in front an attachment, and the victim clicks on the image, OneNote will attempt to start the attachment in the background.

This strategy is not something new, and was initially used in the sample shared by SpiderLabs. A more recent sample was shared on Twitter. WithSecure analysed it in their malware analysis lab and confirmed that the same technique was used, as it can be seen in the image below:

Identifying this doesn't really require a lot of effort, as it requires only to move the image and that would reveal all the attachments. Interestingly enough, the attachments are "duplicated" to cover the entire image, doing so would trigger the attachment open regardless of where the user clicks on the lure image.

Despite the analysed techniques are relatively straightforward, it is expected that complexity will be introduced by adversaries in these types of payloads.

Conclusions

In this post we analysed the various aspects of the abuse of the OneNote format, how it can be used from an attacker perspective and what trace it leaves on systems when a successful exploitation is achieved. The following key points summarise the recommended remedial actions:

- If possible, block direct download of one and onepkg files at the proxy level
- If possible, block .one and .onepkg mail attachments
- Monitor the operations of the OneNote.exe process, especially when a .one file is downloaded from the internet
- Pay particular attention to process creation events associated with common LOLBins

- File write operations should also be monitored closely

Blocking the .one and .onepkg files from being downloaded will not be sufficient, as attackers can utilise techniques such as HTML smuggling and other containerisation mechanisms to avoid perimeter controls.

WithSecure™ Newsletter

Vulnerability Disclosure Policy

WithSecure™ Labs Publications