RED TEAMER AND SECURITY ADDICT

ENIGMA0X3

« DEFEATING DEVICE GUARD: A LOOK INTO CVE-2017-0007

BYPASSING AMSI VIA COM SERVER HIJACKING »

PHISHING AGAINST PROTECTED VIEW

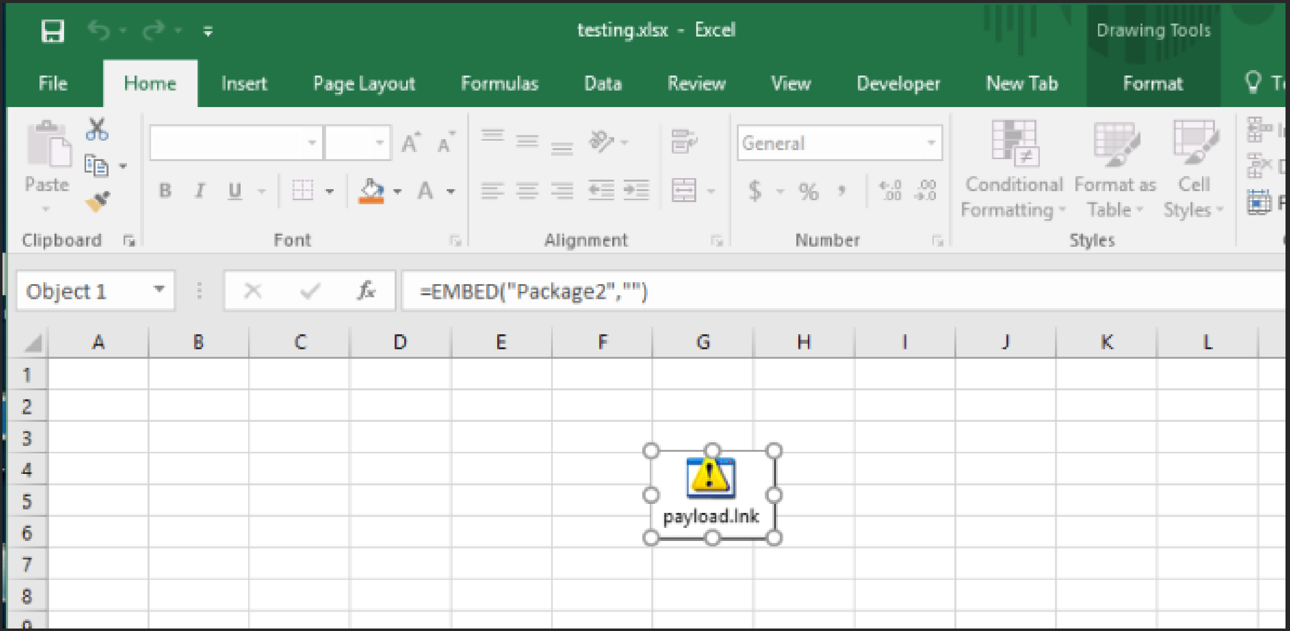
July 13, 2017 by enigma0x3

Microsoft Office has a security feature called Protected View. This feature opens an Office document that originates from the internet in a restricted manner. The idea is that it will prevent automatic exploitation of things such as OLE, Flash and ActiveX by restricting Office components that are allowed to execute. In 2016, Microsoft Patched a bug in Protected View around Excel Add-in files via CVE-2016-4117. @HaifeiLi has done some great research in this area, which you can read about here. MWR Labs also has a great white paper on understanding the Protected View Sandbox, which you can read about here. In this post, I will highlight some techniques you can employ to circumvent Protected View while still having access to the techniques us red teamers have grown to know and love.

In my experience, end users are less likely to exit Protected View than they are to click through an Office dialogue box. I believe the reason for this is that they can access the document's content while in Protected View, which is all they really need. When phishing, reducing the number of clicks for a user is always helpful. Protected View presents one additional click; if we can get rid of it, the better off we will be.

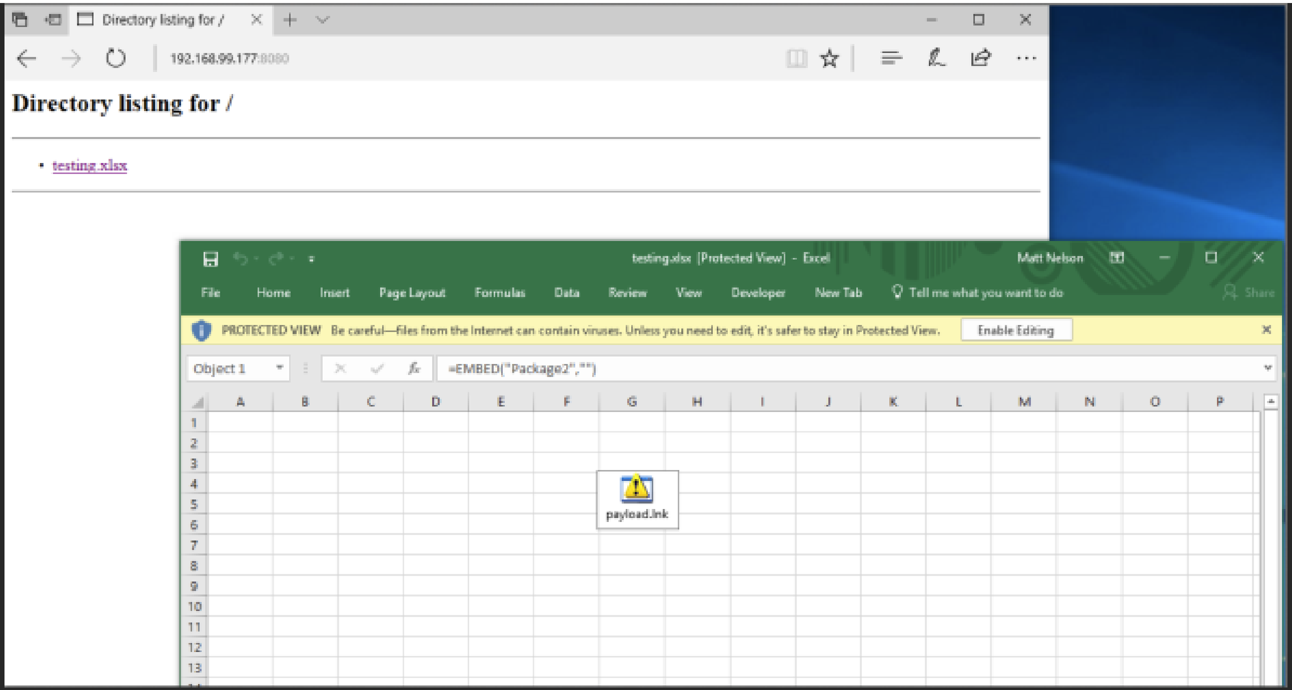
Full Disclosure: These were reported to MSRC on April 20th, 2017 and all of these have been deemed not a security issue. Features, not bugs 😉

Before I get into these techniques, it’s important to understand the normal behavior. Attackers often use a number of tricks to get code-execution on a target system. This often ranges from Office macros, to OLE objects and Excel formula injection via DDE. If we embed a LNK into an Excel document via OLE, we will see this locally:



=EMBED("Package2","")

Now, if we host the above document, Protected View will activate and the embedded OLE object will not be able to activate via a double-click until Protected View is exited:



Directory listing for /

- testing.xlsx

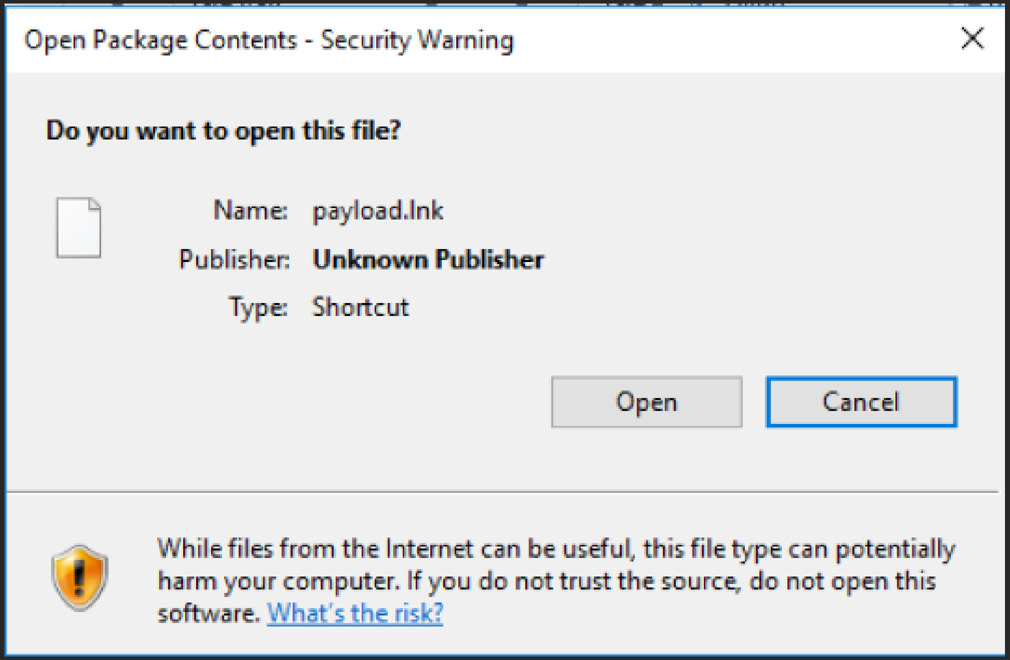
PROTECTED VIEW Be careful—files from the Internet can contain viruses. Unless you need to edit, it's safer to stay in Protected View.

=EMBED("Package2","")

This is what should happen when a document comes in from the internet; things such as OLE, ActiveX and DDE should be blocked until “Enable Editing” is clicked.

Now that we know what normal Protected View behavior looks like, we can dive into some ways around it. The first one I want to cover is executing a file via OLE from a Publisher file. Like Word and Excel, Microsoft Publisher often comes with Microsoft Office and includes similar functionality, such as OLE embedding. Attackers often use LNK files embedded via OLE, so we will do the same in this example. Publisher offers many features to make the OLE object enticing to the user. For simplicity, I will not go into these features.

For this example, we will use a LNK payload that simply executes: “C:\Windows\System32\cmd.exe /c calc.exe”. I won’t go into embedding OLE inside Publisher either as it’s nearly identical to the other Office formats. If we host the Publisher file with the OLE embedded LNK, you will notice that Protected View does not activate. Clicking on the OLE object displays 1 prompt to the user:



Open Package Contents - Security Warning

Do you want to open this file?

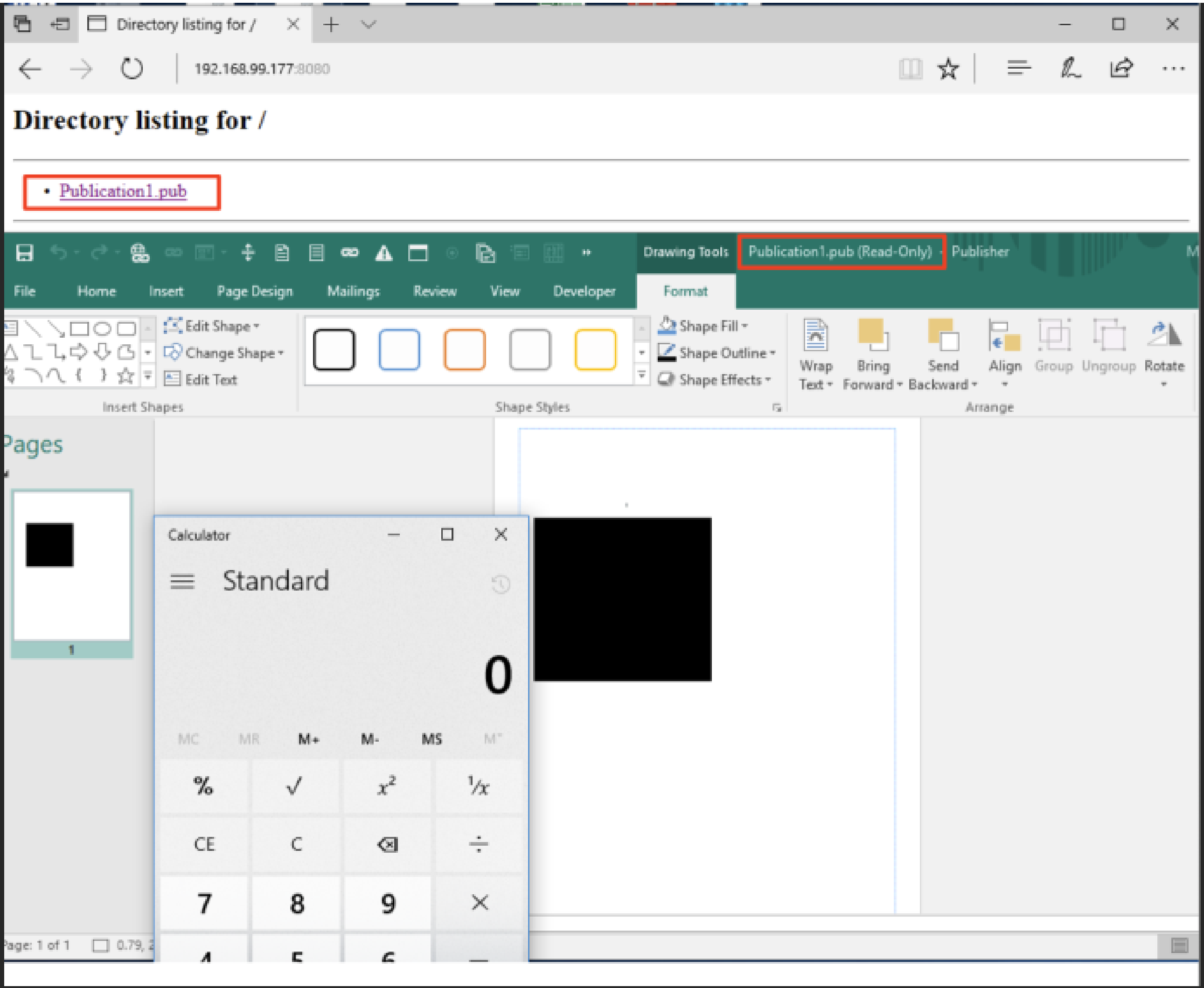
Name: payload.lnk

Publisher: **Unknown Publisher**

Type: Shortcut

While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not open this software. What's the risk?

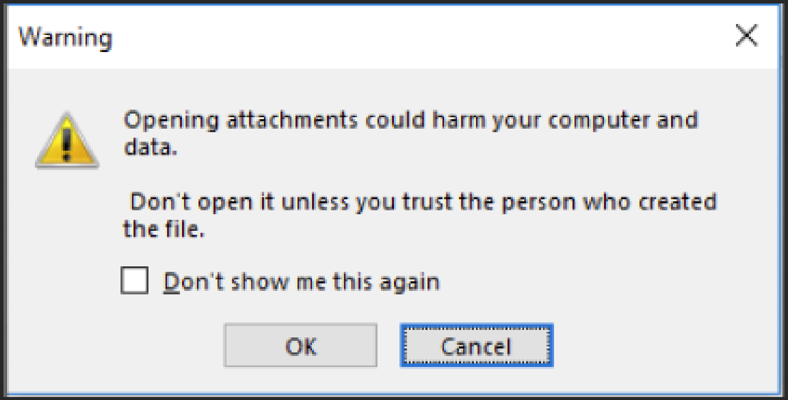
Clicking “Open” will cause the LNK to execute:



As you can see, double clicking the OLE object resulted in the LNK being executed (after a “Open File” prompt). Normally, Protected View would have prevented the OLE object from being activated until the user explicitly exited it.

Next, we will go into OneNote. OneNote allows for attaching files to note files. LNK files look a bit weird when attached to OneNote, so we will use a VBScript instead. For this example, this VBScript file will simply execute calc.exe via the Run method of the WScript.Shell COM object. For simplicity, I won't go into dressing the document up to entice the user.

If we host the OneNote file (.ONE) with the attached VBScript file, you will notice that Protected View does not activate. The user is presented with 1 dialogue:



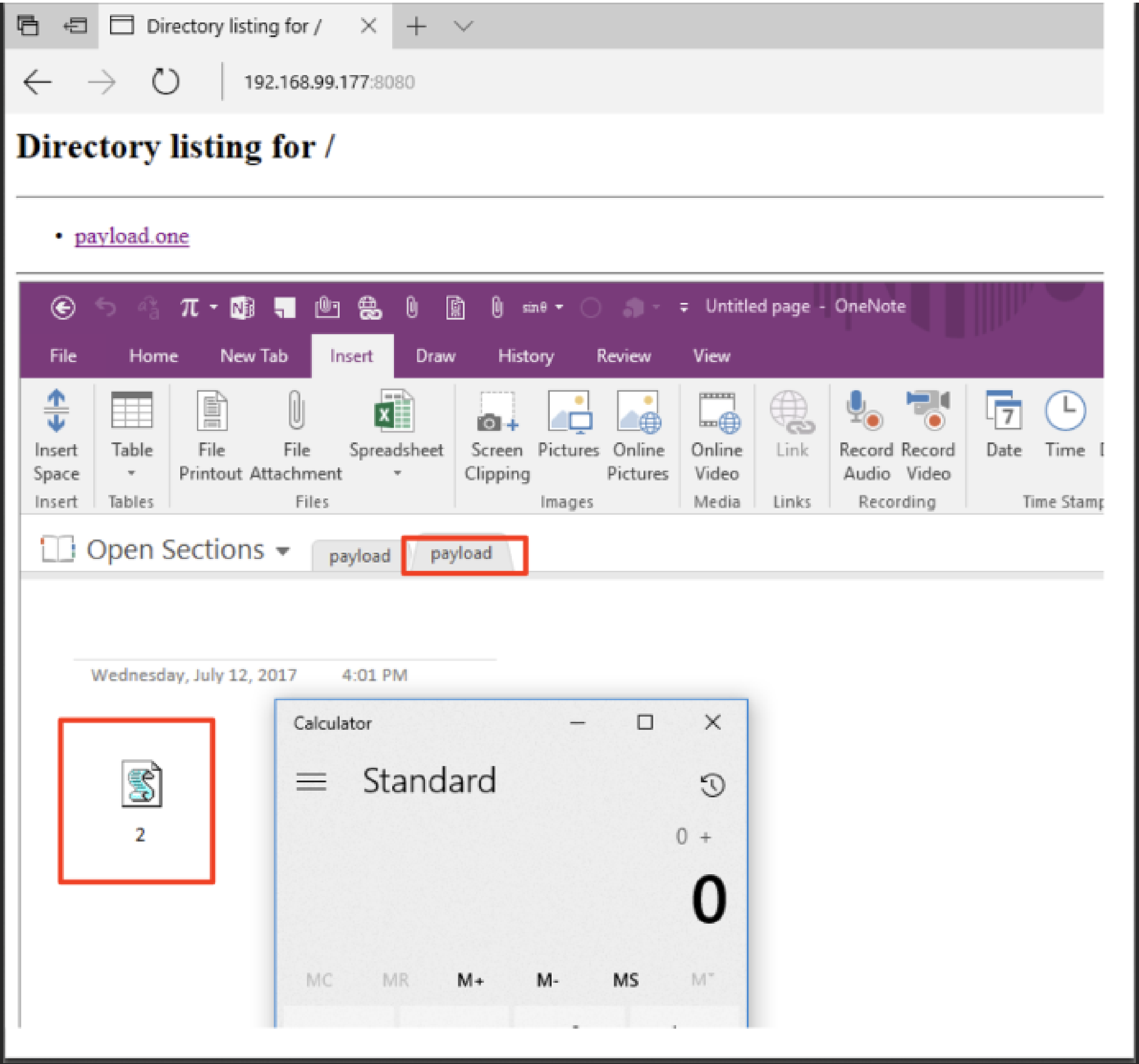
Warning

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

Clicking "OK" will result in the VBScript being executed:



Directory listing for /

192.168.99.177:8080

Directory listing for /

- payload.one

So far, we have Publisher files and OneNote files that don't trigger Protected View, but allow for OLE embedding, or something similar. Finally, there are Excel Symbolic Link files. This file format somewhat restricts the content it can host. In my testing, SLK files will strip OLE objects and any existing macro when saved. Fortunately, there are still attacks like Excel Formula Injection via DDE. If you don't know about this technique, you can read more about it here.

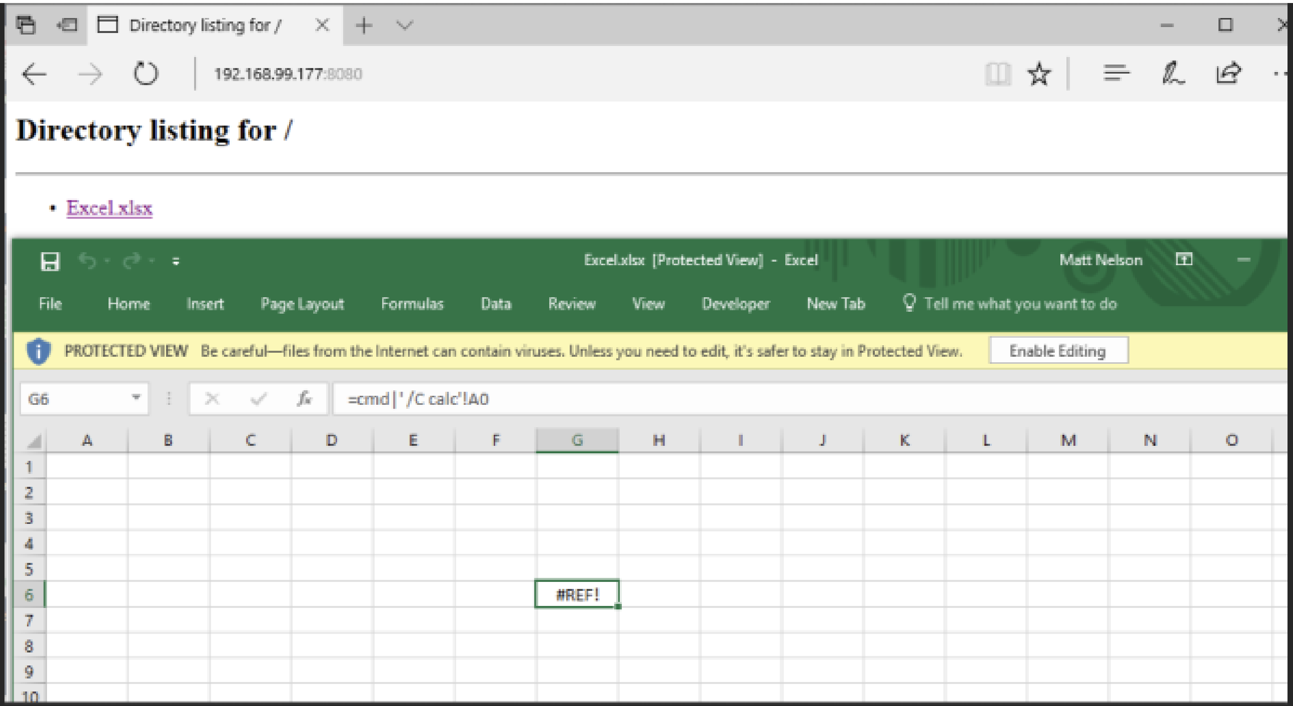
Normally, Protected View will prevent automatic cell updating, which renders this attack useless while in Protected View. If we add a malicious formula and save it as a Symbolic Link (.SLK) file, we can get around the Protected View portion of the attack.

In this example, the Excel formula will be something like this:

=cmd|' /C calc'!A0

It is important to note that the DDE injection attack does present the user with 2 security warnings. There may be additional functionality in Excel SLK outside of DDE that won't prompt 2 security dialogues…I encourage you to use your imagination 🙂

If we save the file as a normal Excel file, you will notice that Protected View blocks the automatic "Enable" prompt, and requires the user to exit Protected View first:



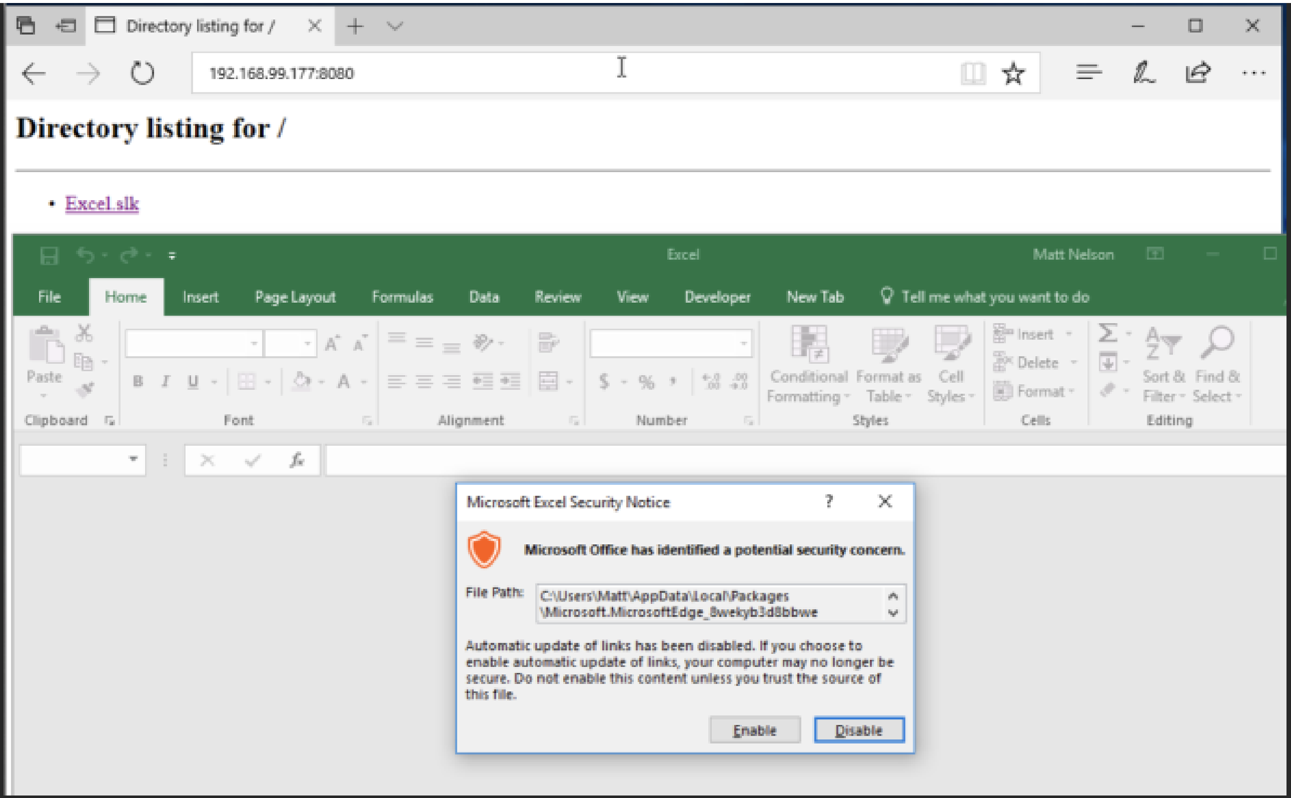
Directory listing for /

- Excel.xlsx

PROTECTED VIEW Be careful—files from the Internet can contain viruses. Unless you need to edit, it's safer to stay in Protected View.

```
=cmd|'/C calc'!A0
```

Now, if we save the file as .SLK and host it, you will notice that Protected View is not activated, and the user is automatically presented with the “Enable, Disable” prompt.



Microsoft Excel Security Notice

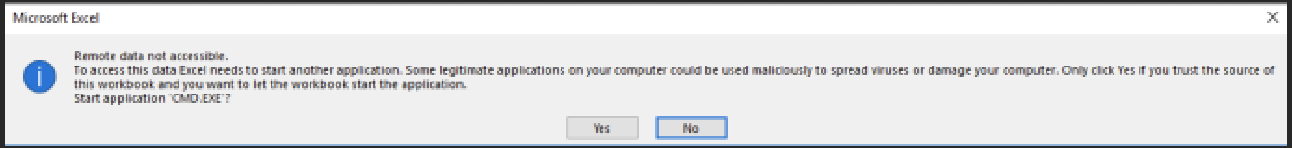
Microsoft Office has identified a potential security concern.

File Path: C:\Users\Matt\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe

Automatic update of links has been disabled. If you choose to enable automatic update of links, your computer may no longer be secure. Do not enable this content unless you trust the source of this file.

Enable | Disable

Clicking “Enable” will present the user with the following dialogue. I’ll be the first to say that users love to click “Yes” on this prompt 😉

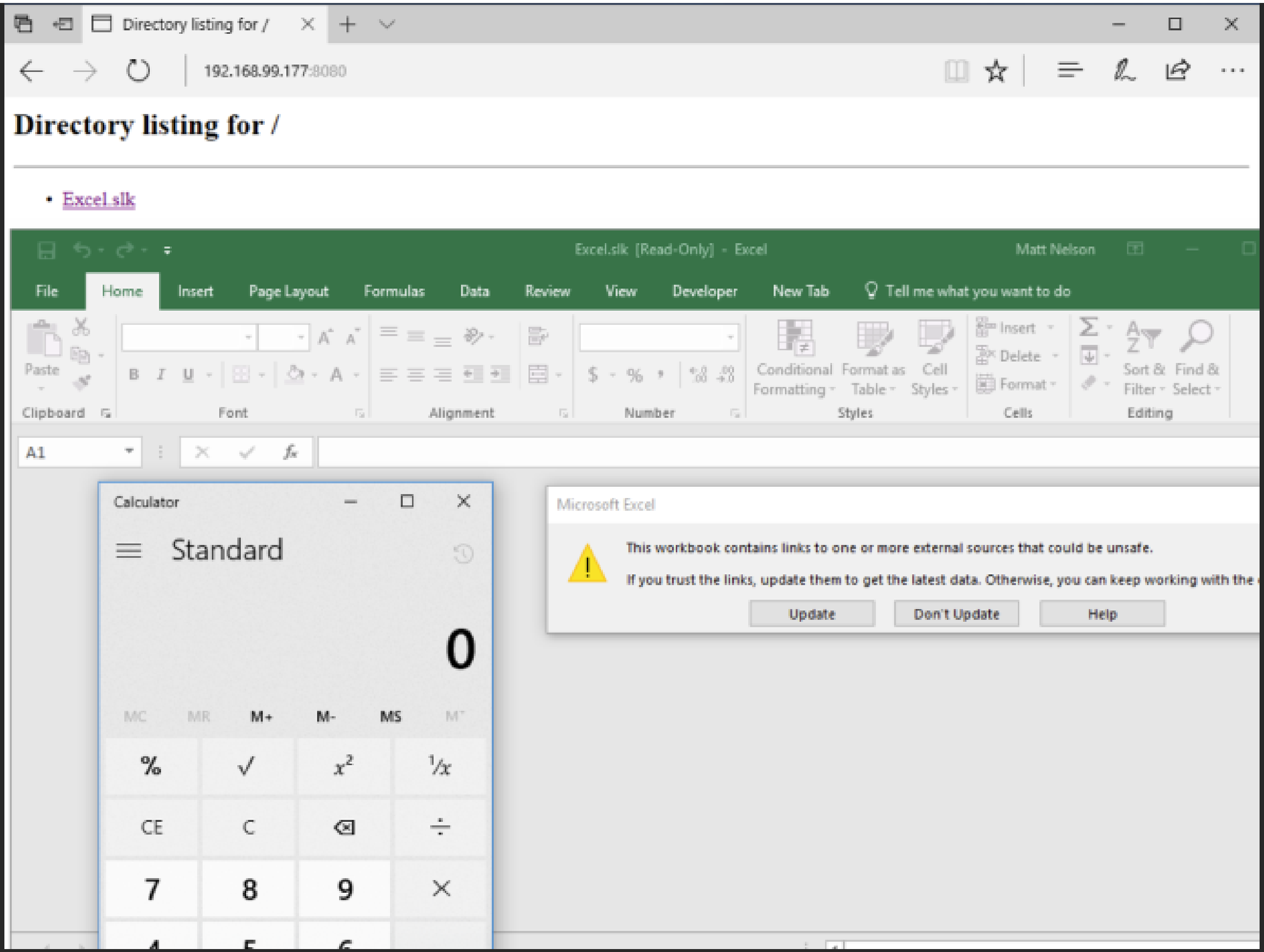


Microsoft Excel

Remote data not accessible.
To access this data Excel needs to start another application. Some legitimate applications on your computer could be used maliciously to spread viruses or damage your computer. Only click Yes if you trust the source of this workbook and you want to let the workbook start the application.
Start application 'CMD.EXE'?

Yes | No

Clicking “Yes” will result in the command being executed:



Directory listing for /

- Excel.slk

Microsoft Excel

This workbook contains links to one or more external sources that could be unsafe.

If you trust the links, update them to get the latest data. Otherwise, you can keep working with the

While the user is presented with 2 prompts for the .SLK attack, users are often less likely to exit Protected View then click on the displayed prompts. From a Red Team perspective, any way to get around Protected View is worth the investment in terms of payload delivery.

Prevention: I am not currently aware of a way to manually enroll Publisher, OneNote and .SLK files into Protected View. User awareness training is recommended. If your end users do not use OneNote and Publisher, one solution would be to uninstall these.

- Matt N.

SHARE THIS:

Twitter Facebook 9 G+ Google

★ Like

Be the first to like this.

RELATED

Reviving DDE: Using OneNote and Excel for Code Execution

"Fileless" UAC Bypass Using eventvwr.exe and Registry Hijacking
With 27 comments

Bypassing UAC using App Paths
With 3 comments

Bookmark the permalink.

LEAVE A REPLY

Enter your comment here...

Search … Search

ARCHIVES

- January 2018
- November 2017
- October 2017
- September 2017
- August 2017
- July 2017
- April 2017
- March 2017
- January 2017
- November 2016
- August 2016
- July 2016
- May 2016
- March 2016
- February 2016
- January 2016
- October 2015
- August 2015
- April 2015
- March 2015
- January 2015
- October 2014
- July 2014
- June 2014
- March 2014
- January 2014

RECENT POSTS

- Reviving DDE: Using OneNote and Excel for Code Execution
- Lateral Movement Using Outlook's CreateObject Method and DotNetToJScript
- A Look at CVE-2017-8715: Bypassing CVE-2017-0218 using PowerShell Module Manifests
- UMCI Bypass Using PSWorkFlowUtility: CVE-2017-0215
- Lateral Movement using Excel.Application and DCOM

CATEGORIES

- Uncategorized

RECENT COMMENTS

Soc on Defeating Device Guard: A

"Fileless… on "Fileless" UAC Byp…

"Fileless… on Bypassing UAC usin App P…

Windows 10 UAC Looph… on Bypa UAC using App P…

NexusLogger: A New C… on "Filele UAC Byp…

META

- Register
- Log in
- Entries RSS
- Comments RSS
- WordPress.com

Blog at WordPress.com.