Make phishing great again. VSTO office files are the new macro nightmare?

medium.com/@airlockdigital/make-phishing-great-again-vsto-office-files-are-the-new-macro-nightmare-e09fcadef010

April 15, 2022

Daniel Schell

Apr 14

.

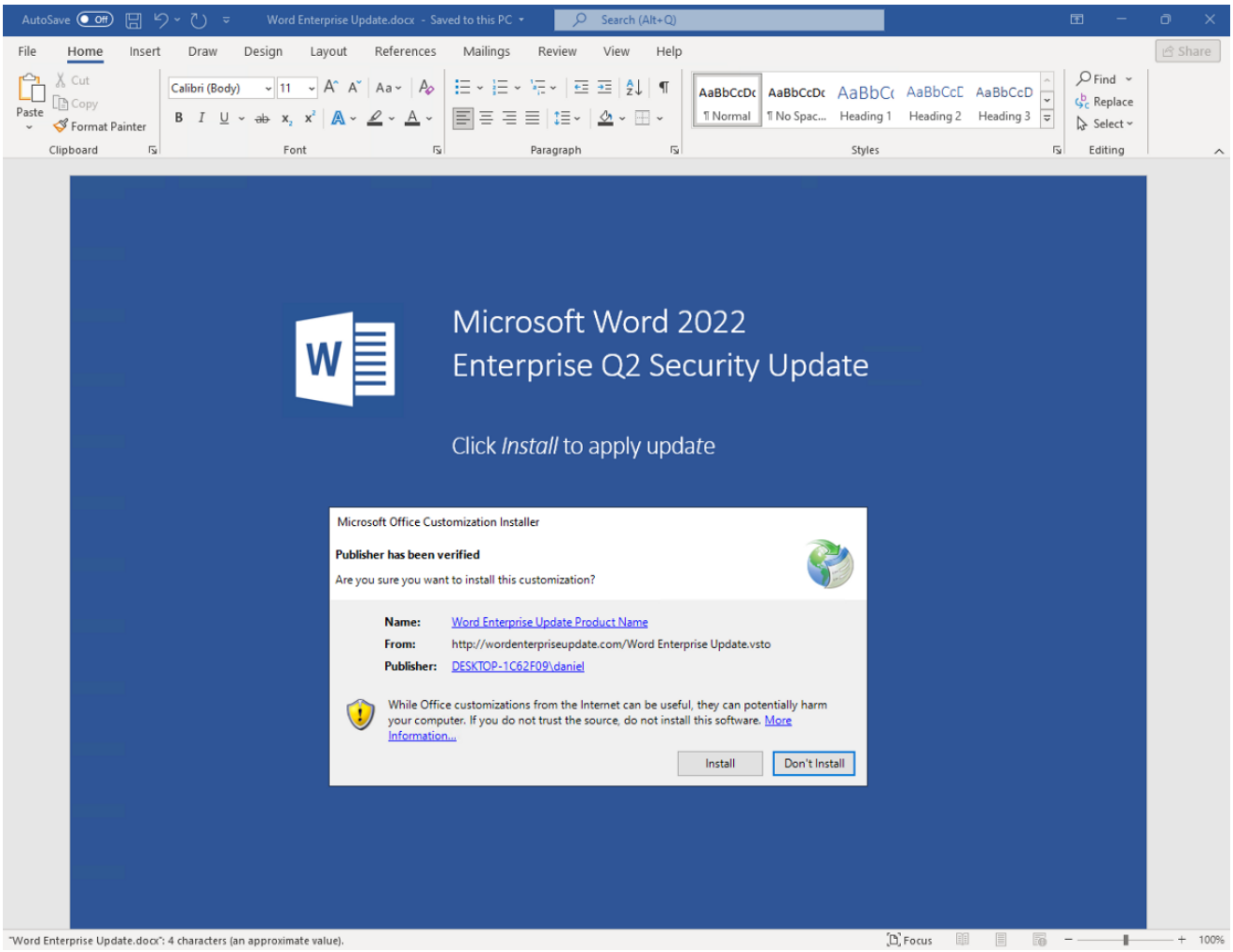
9 min read

.

Summary

Visual Studio Tools for Office (VSTO) has the capability to export an Add-In which is embedded inside an Office document file (such as a Word DOCX). If this document is delivered in the right way (to avoid some inbuilt mitigations) it provides rich capabilities for attackers to phish users and gain code execution on a remote machine through the installation of a word Add-In.

Office itself even provides an automatic update capability, which can be used by attackers to update payloads remotely.



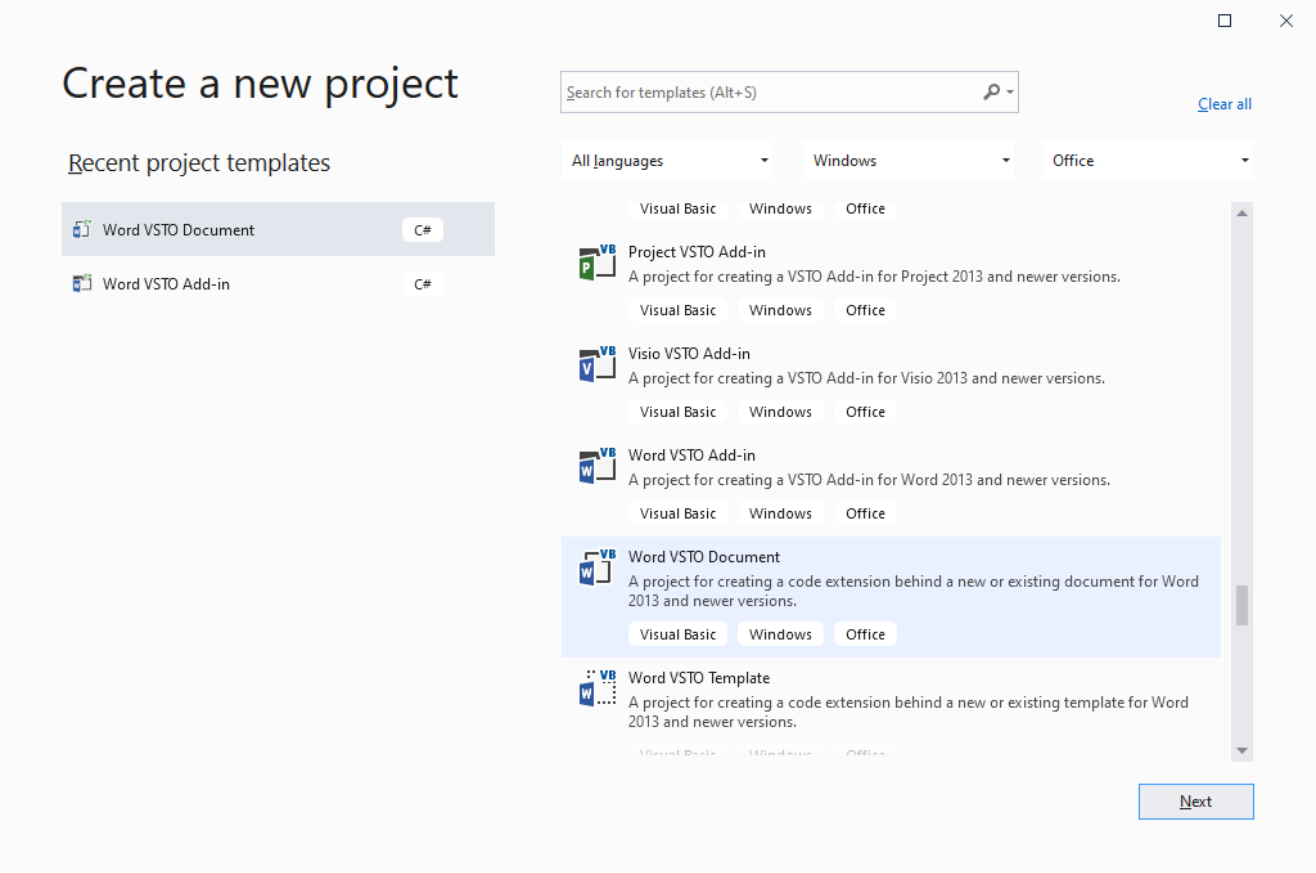
Not a macro nor VBA

Analysis

So I had a friend bring @bohops' excellent article from 2018, VSTO: The payload installer that probably defeats your application whitelisting rules, to my attention as a potential bypass for Airlock as it appeared to bypass AppLocker according to the article.

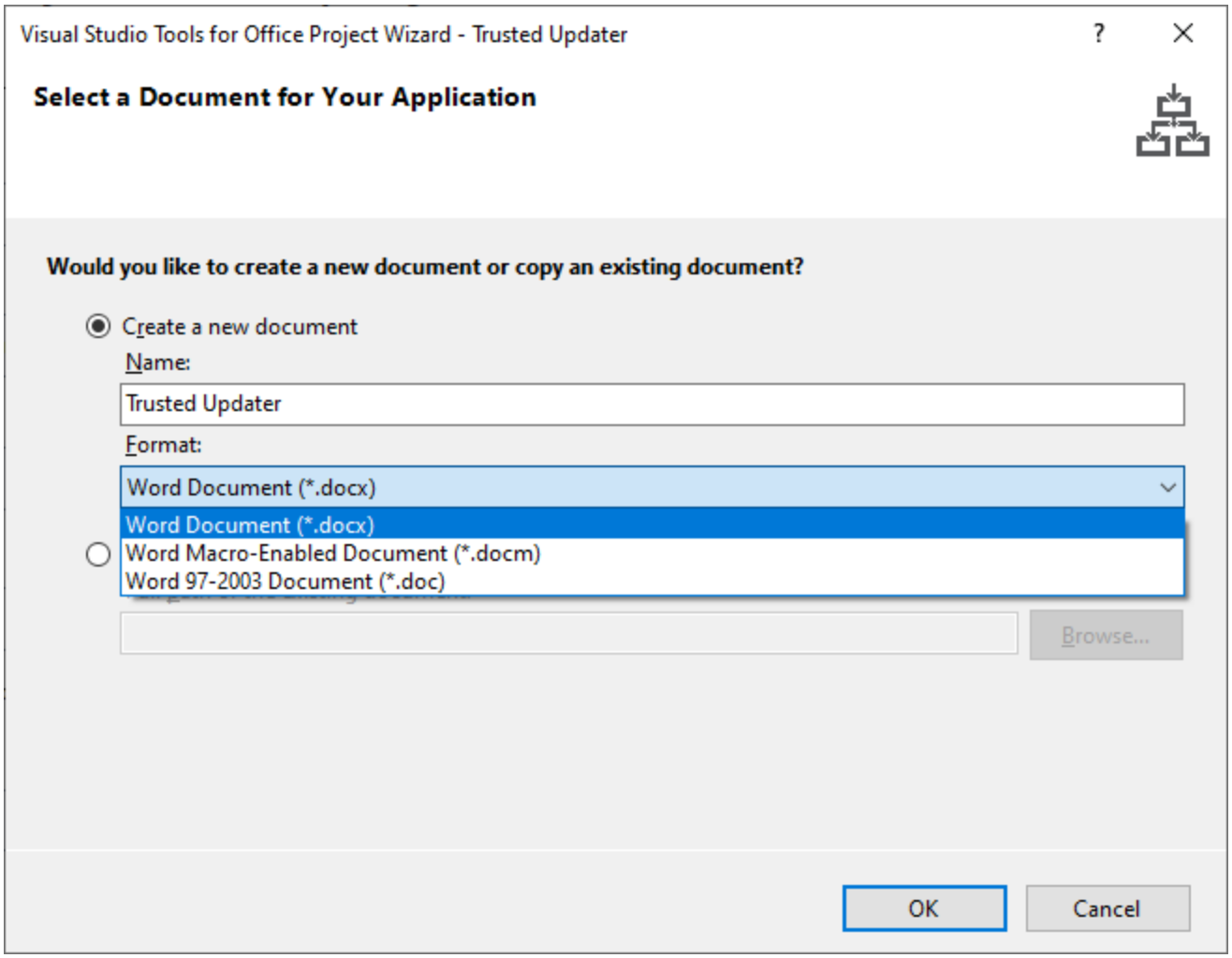
Following through the steps in the blog post, I managed to create a Word Add-in, which required manually launching a Visual Studio for Office (VSTO) file, and was pleased to see that Airlock did prevent the payload via the .NET reflection capability, however along the way something caught my eye in the Visual Studio project templates list — the **Word VSTO Document**.

This led me down a multiday deep dive...



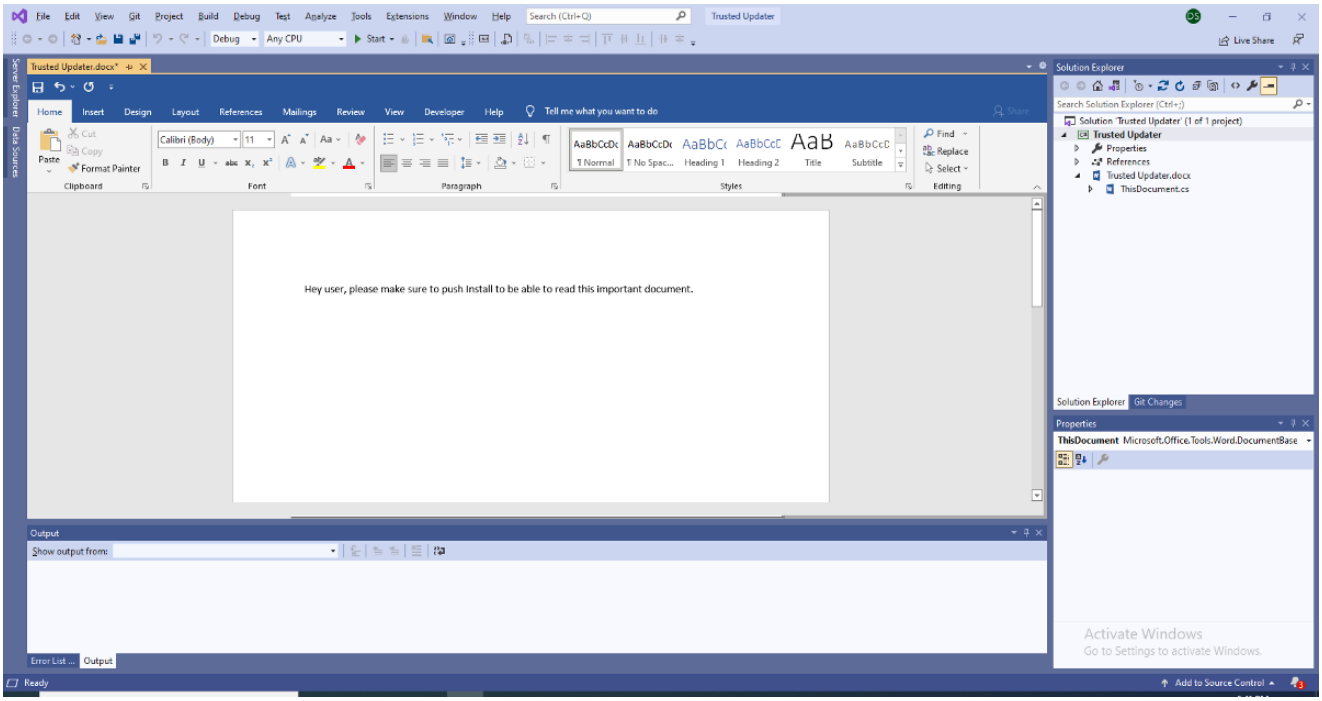
What are you?

Creating a Visual Studio project of this type allows you to select an existing document, or create a new one as a starting point for the project.



Let's go for Docx

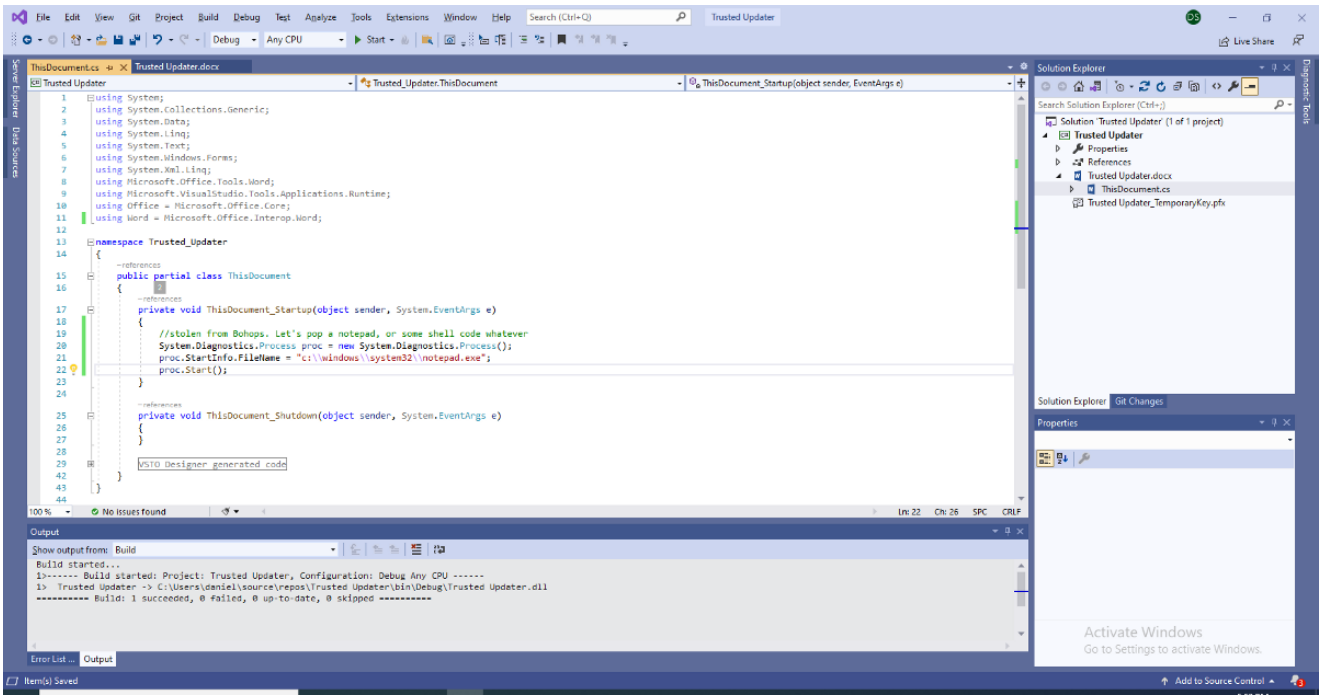
When project has opened up, you can see that you have embedded Word open with-in Visual Studio, allowing you to customize the document.



Oh cool, I can customize the document from inside Visual Studio. How helpful!

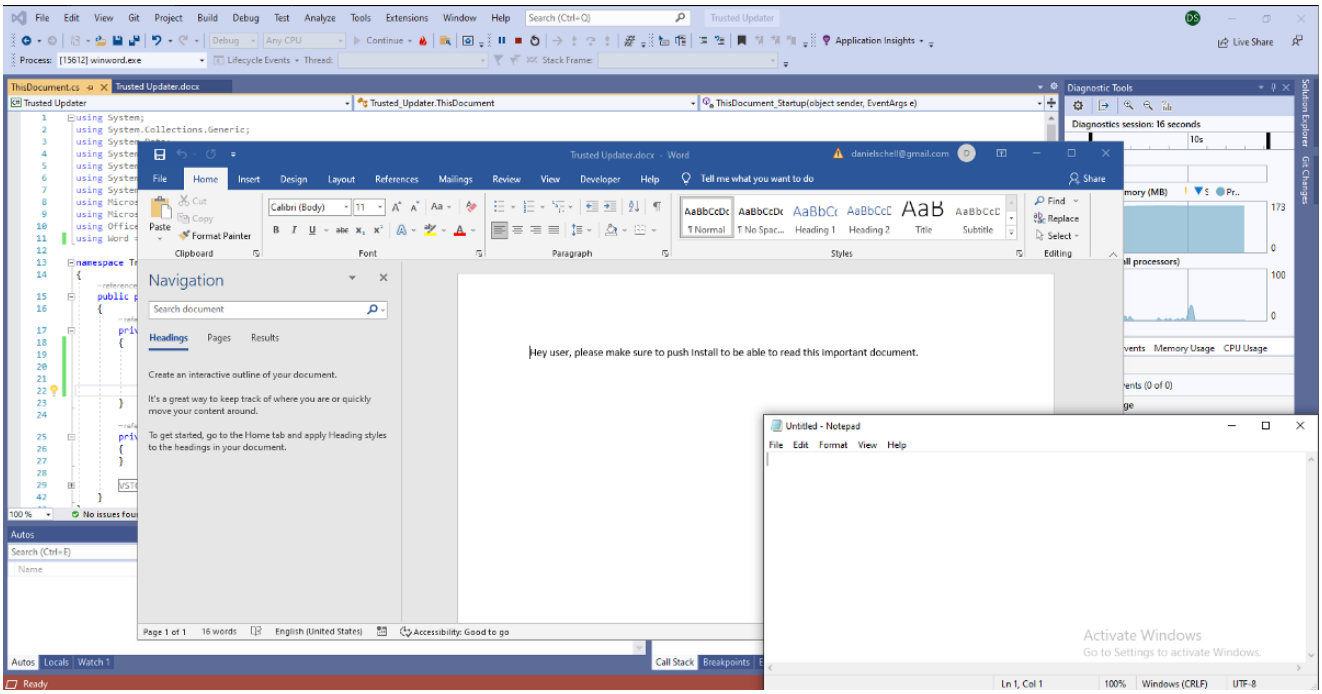
You can also view the C# code inside the document (ThisDocument.cs) which includes pre-built functions for the document loading or being shutdown.

As is customary, let's launch notepad.exe when the Document is loaded by Word.



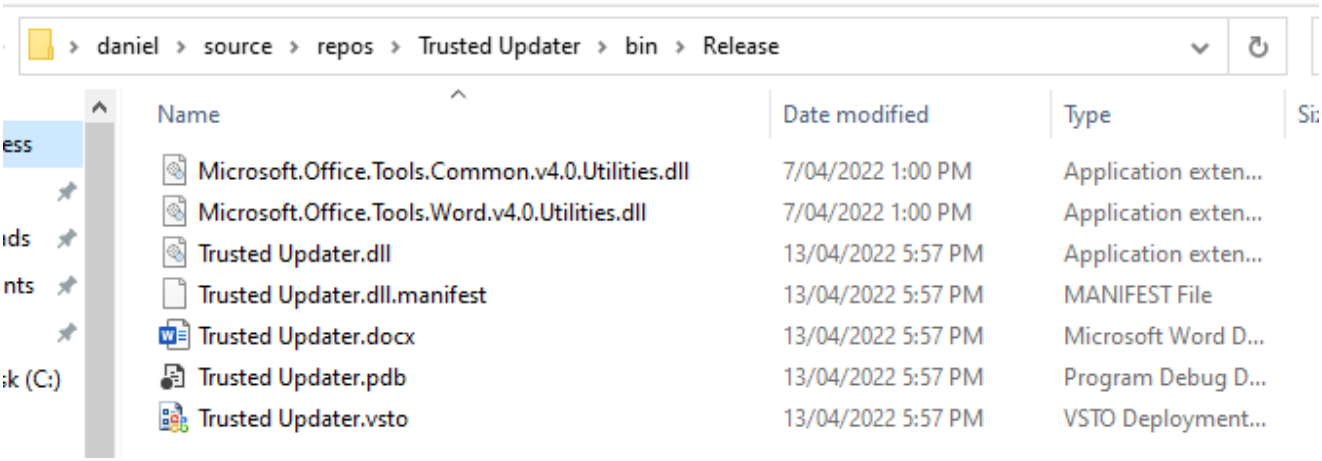
Let's pop a notepad.exe

Press play, and oh cool, Word opens and notepad.exe pops. At least in my development environment.



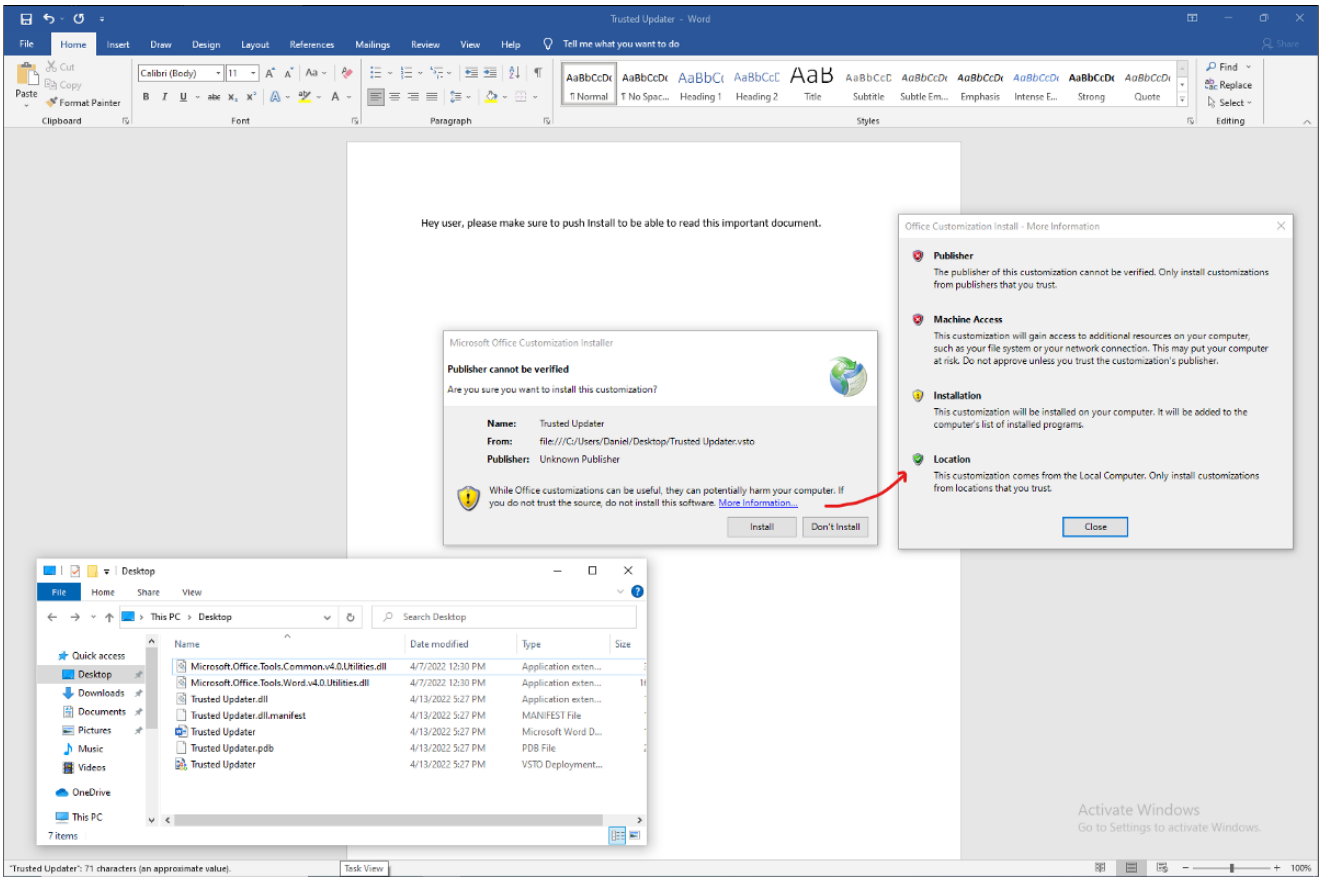
Okay, so a Word document can run it's own C# assembly. What a time to be alive!

And here is where things get interesting. When we build the solution as a release we can see a bunch of files, in addition to the Word document itself.



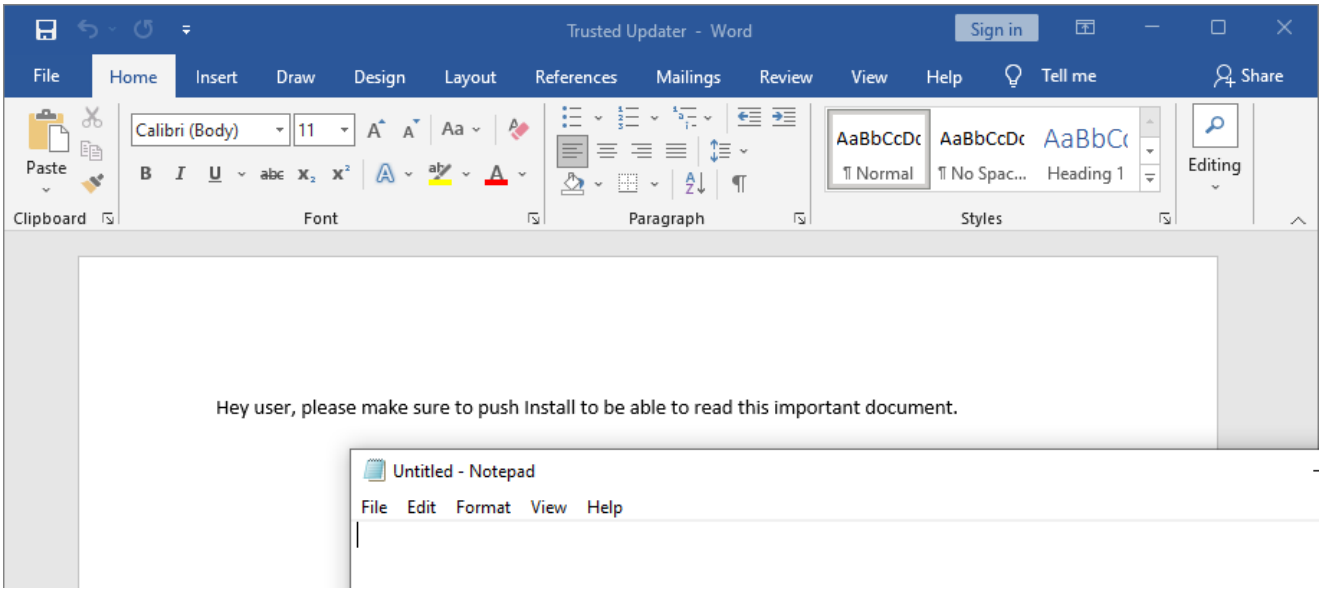
So we have the VSTO, the .NET Assembly containing our code, and dependencies assemblies

Grabbing these files and copying them to a standalone VM with Office and executing the Word document brings up a prompt that Publisher cannot be verified, as it is not signed, however you are able to Install the customization as all of the files exist locally on the machine.



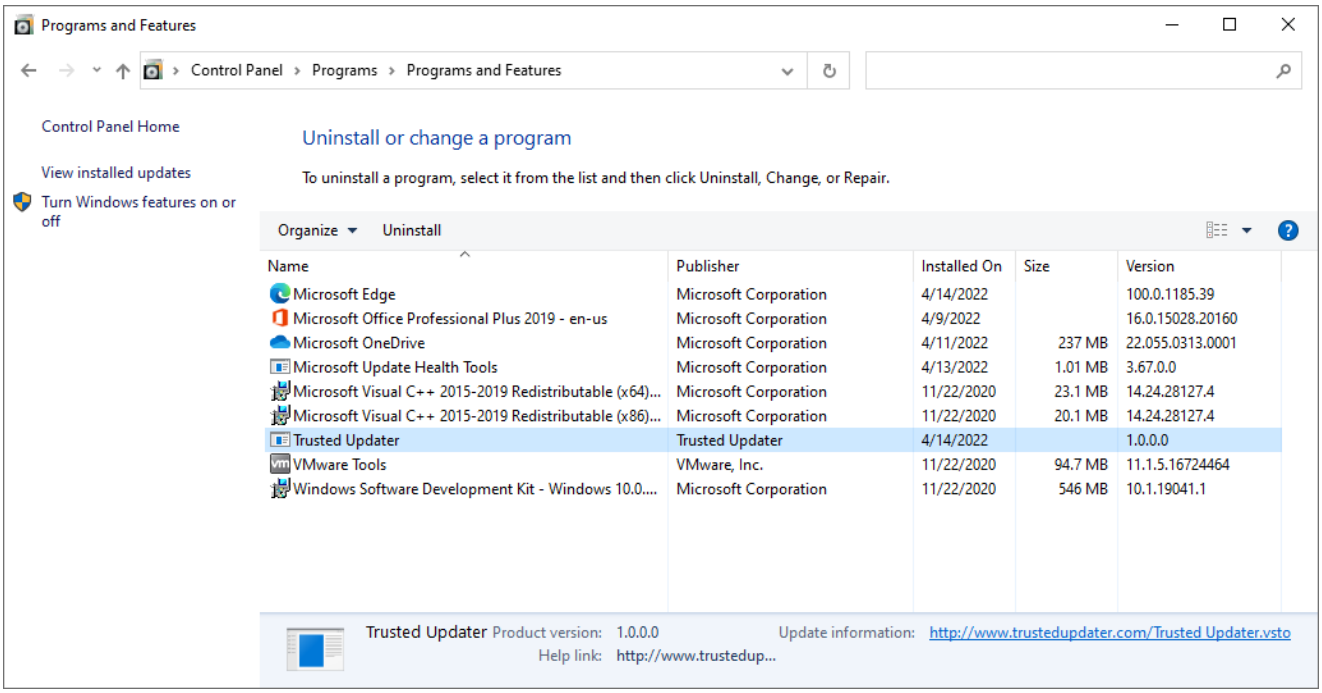
The link was really useful in stepping through this

Once you press install, the Document .NET assembly is loaded and notepad pops.



Neat!

On subsequent loads of the document the user is no longer prompted to install an Add-In, this occurs because the Add-In is now installed on the system. This behavior will continue until the add-in has been removed via Add/Remove programs.



ClickOnce installed applications always create an entry in Add/Remove programs

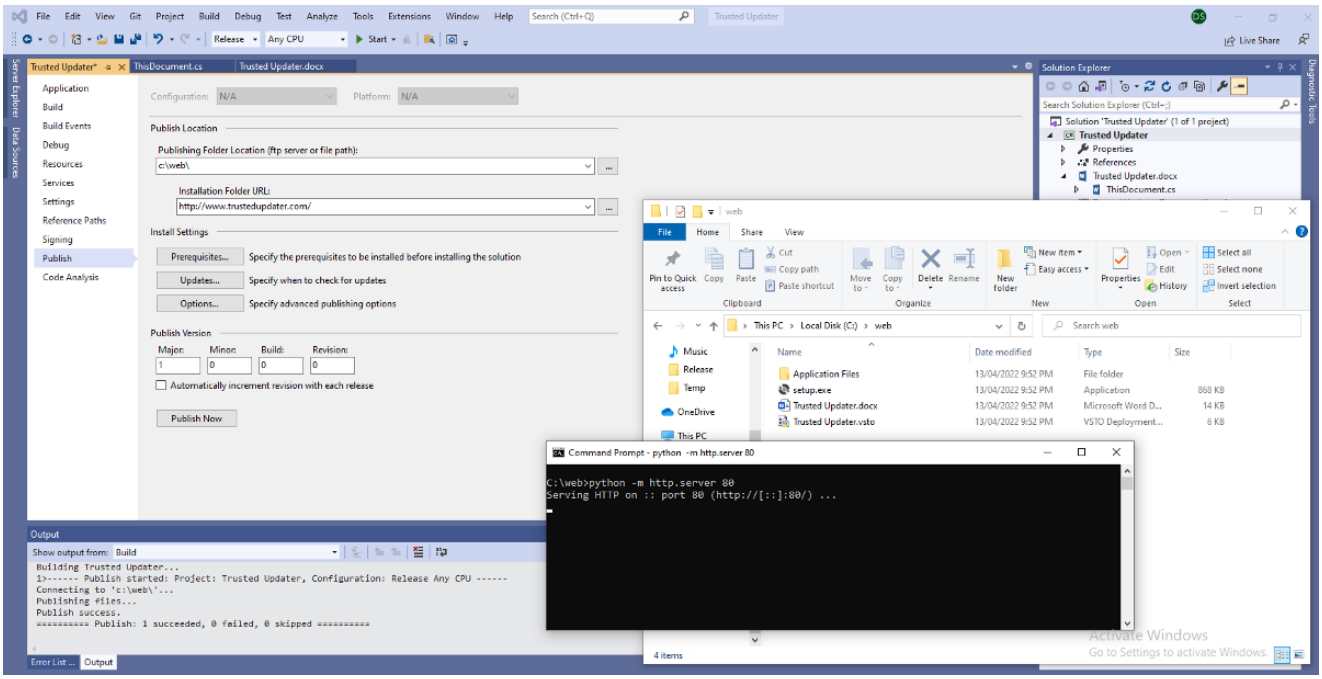
Distribution of this payload, in this form, requires all of the project files to be bundled in a single location or bundled in a container such as an ISO file, as the document in this configuration requires files to be on the local machine.

It's worth mentioning here that I have all Macros and VBA capabilities disabled in group policy and that the Documents shown below are not prevented from executing with these restrictions in place.

But wait, how about just the docx file?

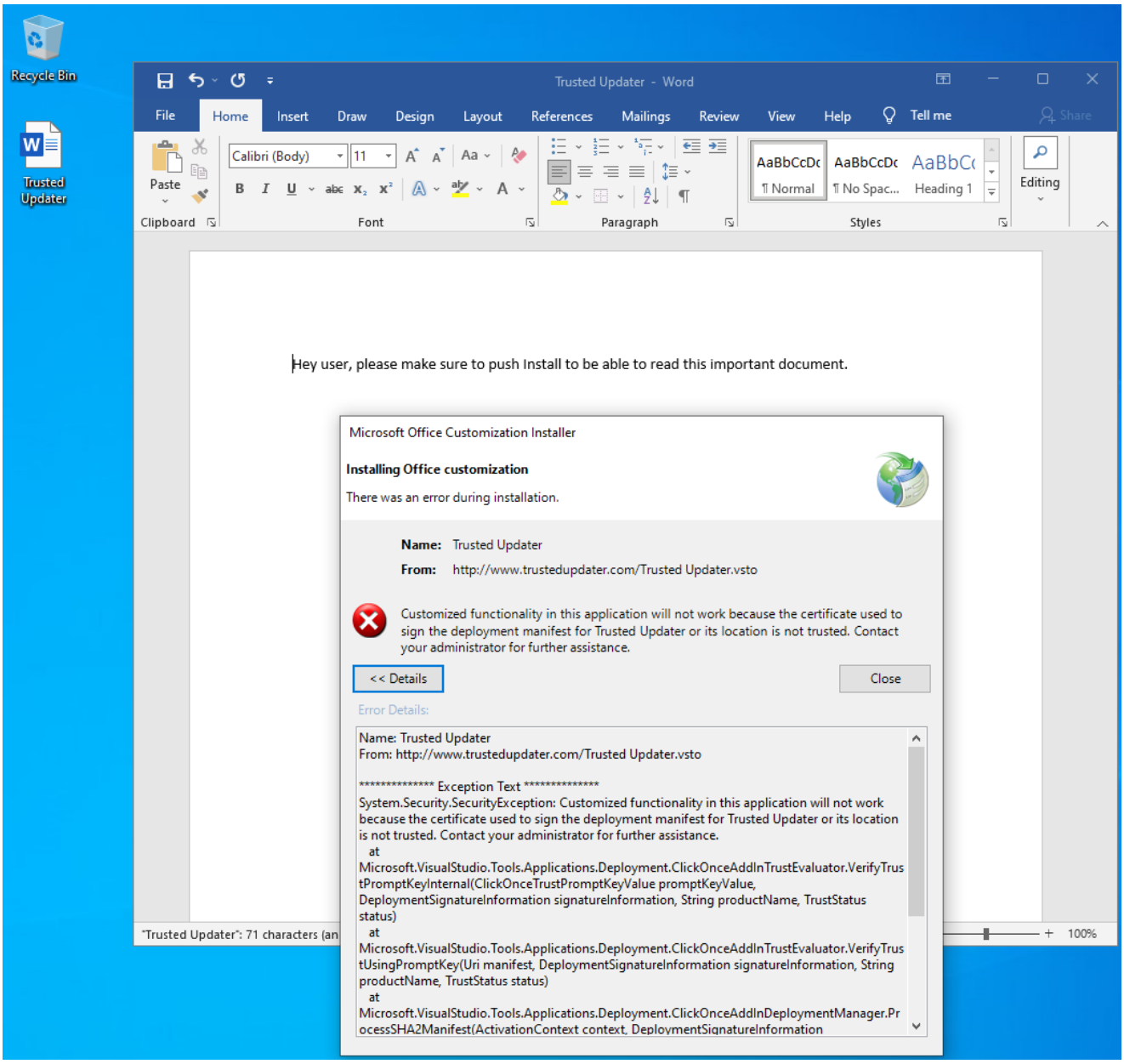
aka Let's just host the dependencies on the internet

So this is where it becomes more interesting as the Visual Studio has a Publish feature (aka ClickOnce) that allows you to Publish the project to a website.

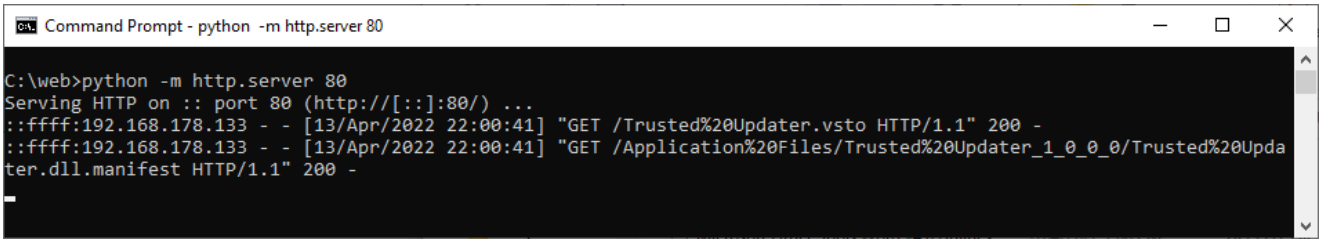


You can publish a VSTO project to a website. In the screenshot python is shown hosting the website.

When you now attempt to run only the document, you will be able to see it retrieve the files from the web server, but then fail to provide the Office Customization prompt as the assembly is not signed or not from a trusted location.



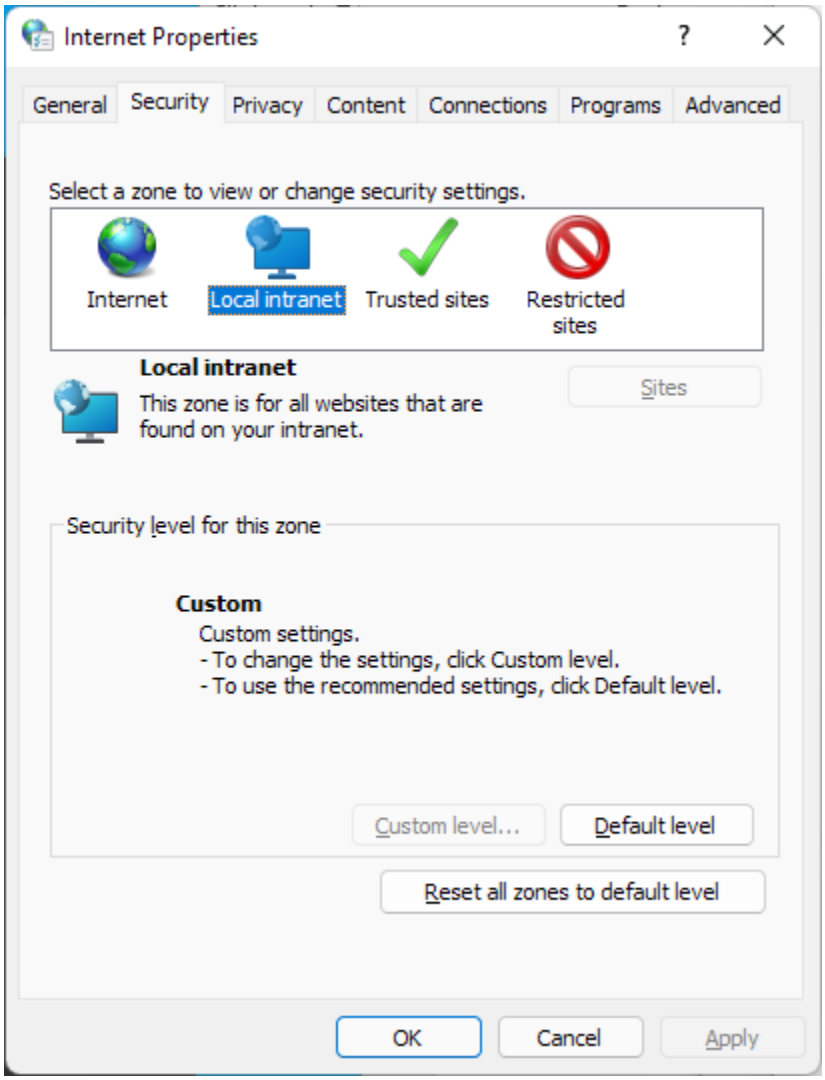
No go



The VSTO and assembly manifest files retrieved by Word

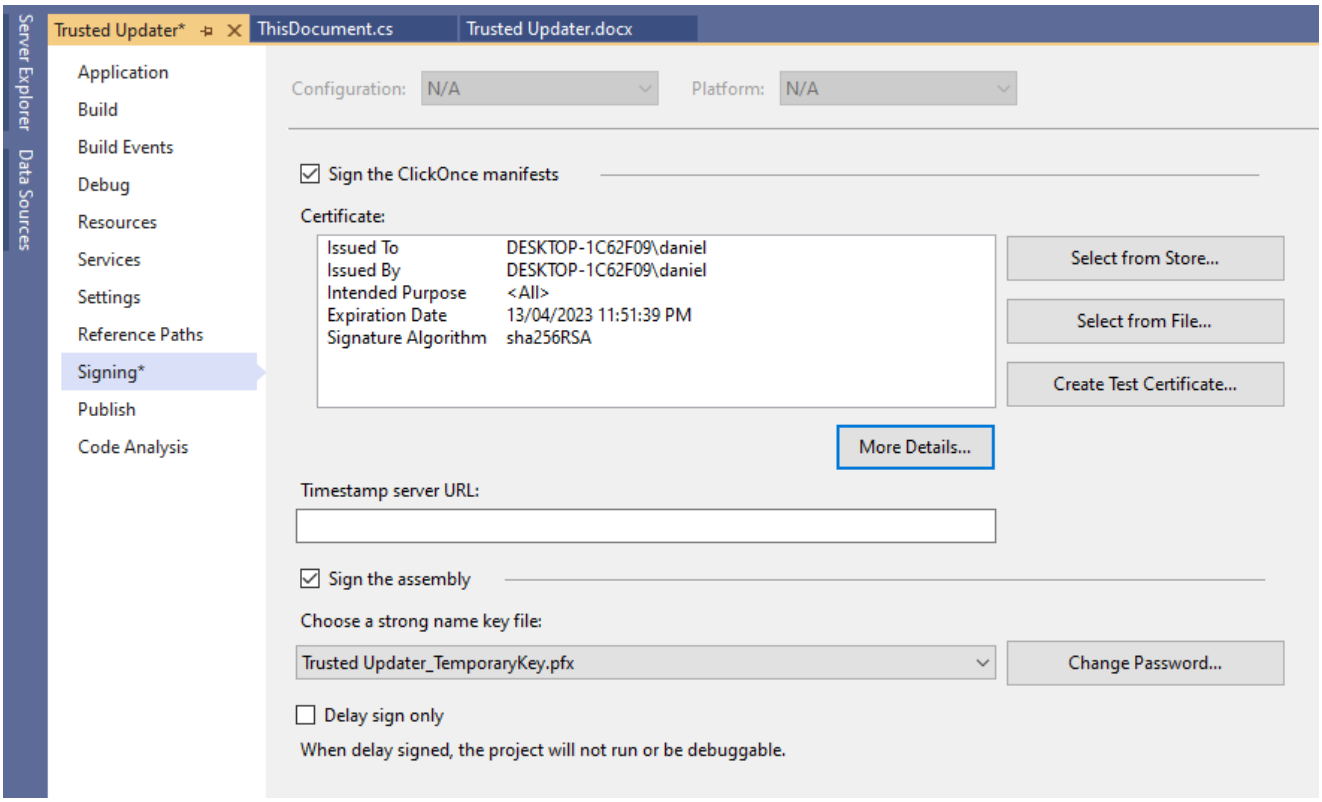
Location, Location, Location

For the Office Customization Installer prompt to appear, the URL hosting the project needs to be in a , or in the zone **OR signed by a trusted publisher** i.e. a code signing certificate that will be trusted by the computer.



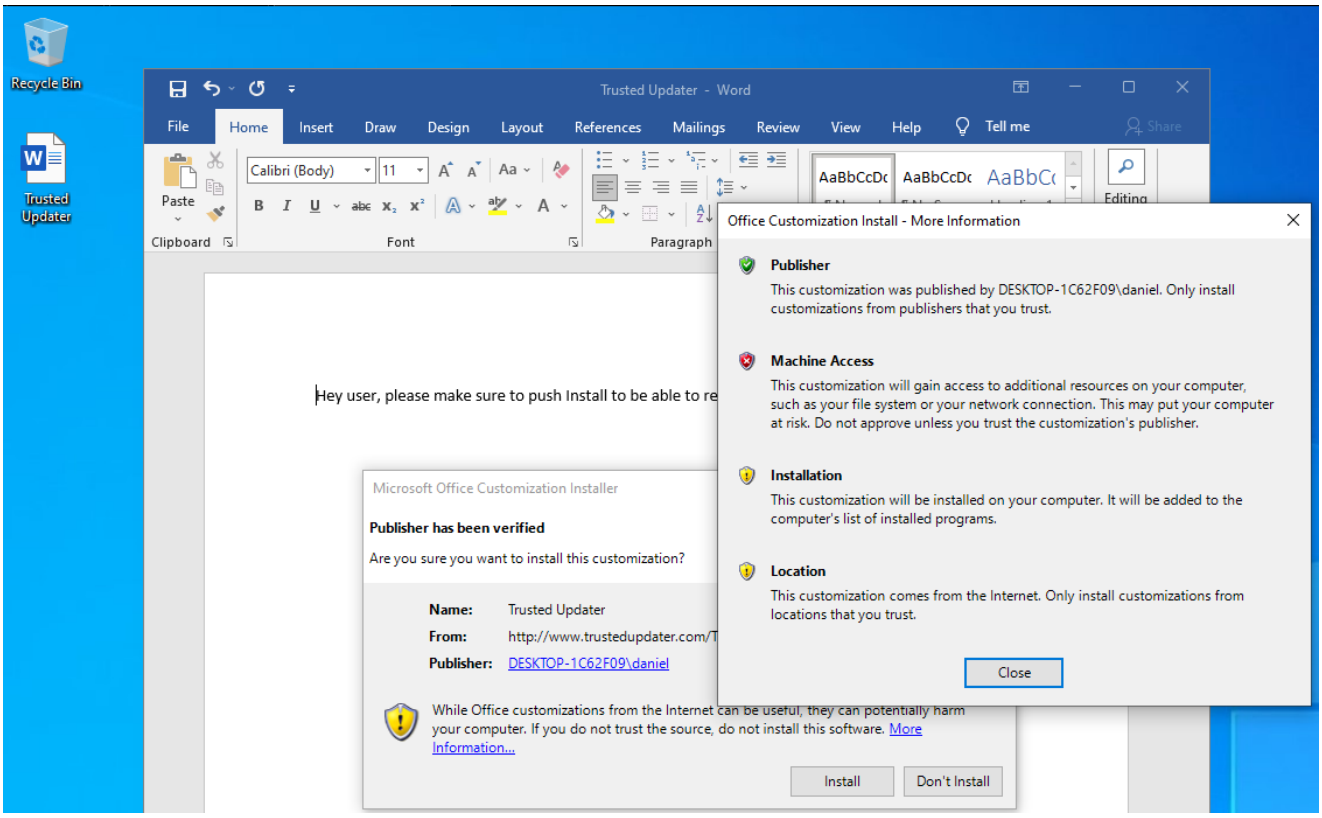
Remember these? The scope of the local intranet zone can be interesting.

So for the price of a standard code signing or internal code signing certificate that would be trusted by the target system (including the user certificate store) you are able to sign the project. Visual studio nicely takes care of this for you without having to rely on the Windows SDK tools such as mage.exe and sn.exe.



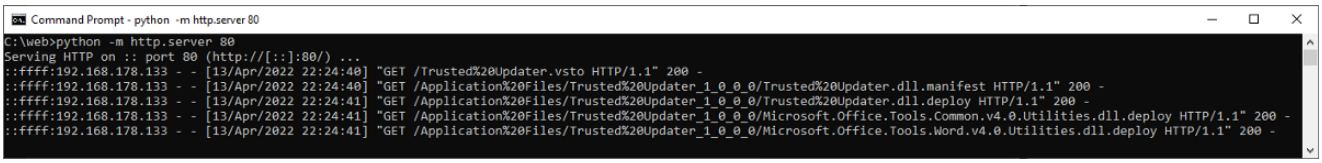
Sign me up

Now that the project files are signed by a certificate that is trusted by the target machine they will again receive Microsoft Office Customization Installer Window, this time sporting the nice **Publisher has been verified** text.

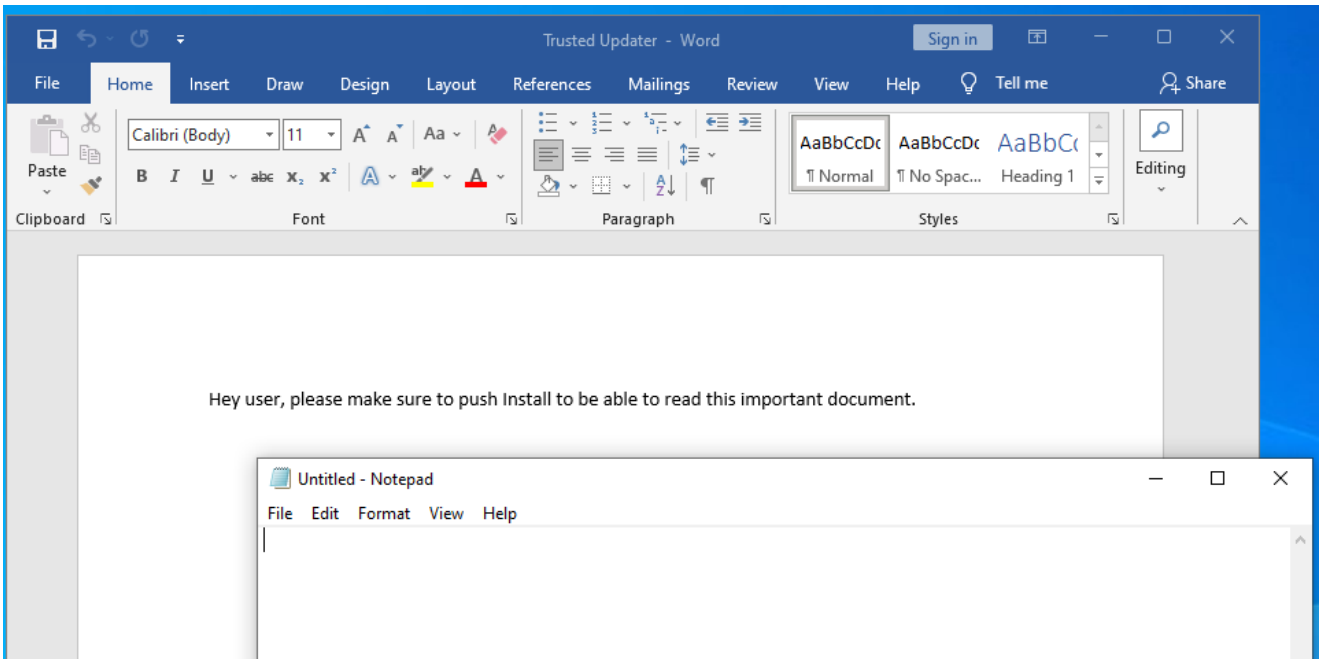


The location is no longer relevant as either a trusted publisher or location makes the determination

Pressing Install now successfully downloads the additional dependencies of the Document and payload is executed on startup and notepad executes.



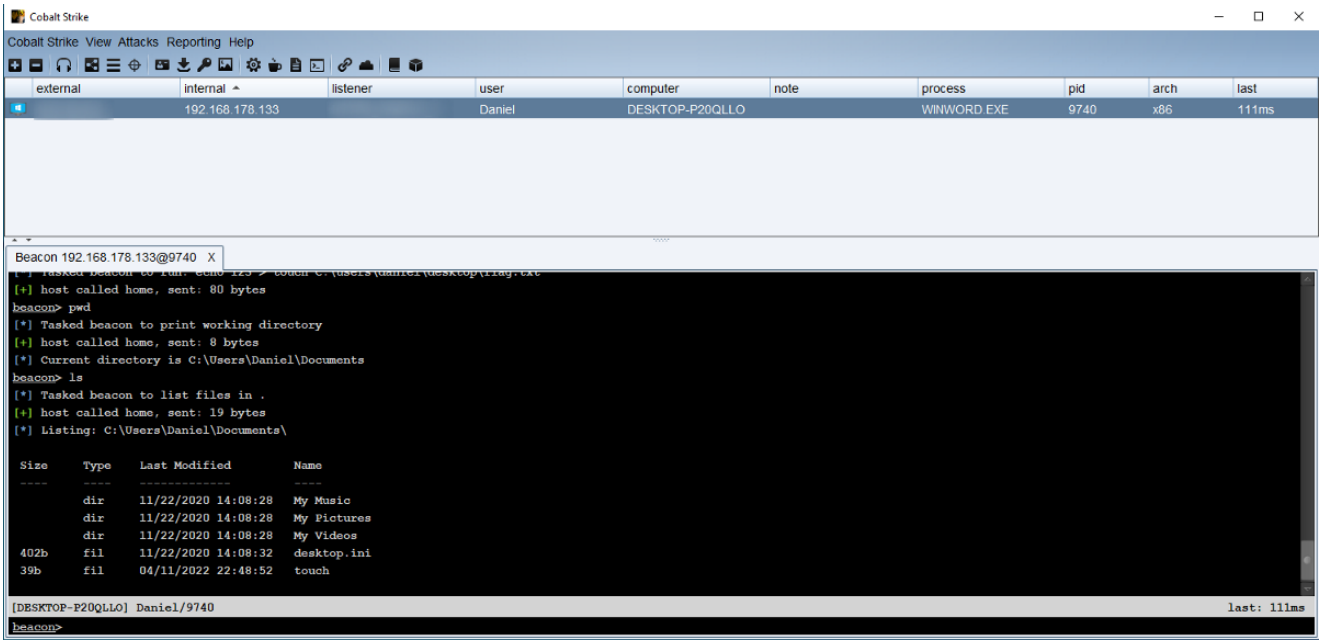
Word now downloads additional dependencies and the assembly itself



Wew

What could an attacker do?

Pretty much anything within the context of the user who launched Word. As part of the research, we verified shellcode that connected back to a Cobalt Strike C2 without any problems.

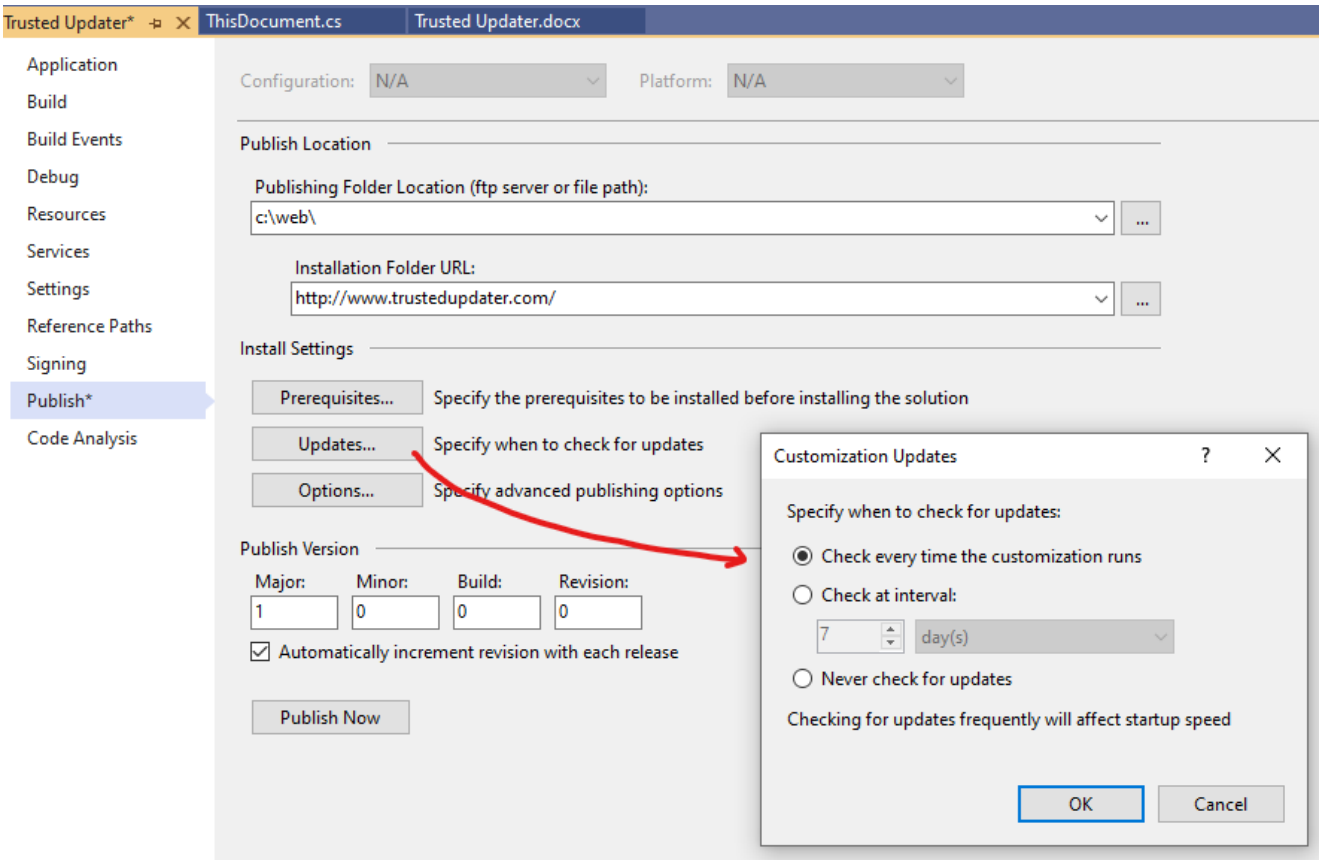


Is it even a infosec blog without this screenshot?

But wait, there's more!

Self updating .NET assemblies

While exploring the **Publish** menu in Visual Studio I discovered the **Updates** option. This feature allows you to configure the Document add-in to automatically check for published updates at startup.

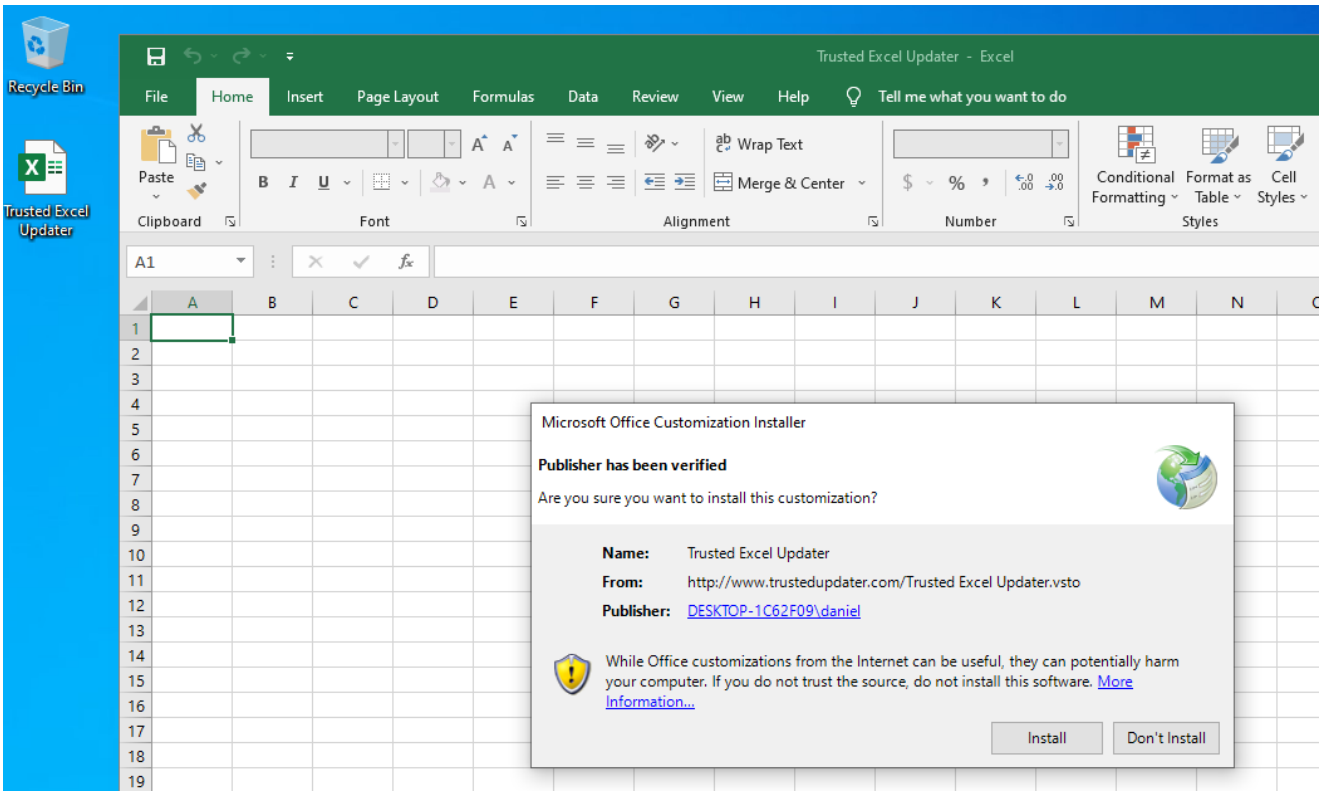


<Chef kiss emoji here>

This means that you can actually change the payload, and publish an updated version of the project to the web, and the next time the document is run it will automatically download and run the latest version of the assembly. This allows an attacker to update their tradecraft without having the replace the document which may have already been distributed.

It's not just word files

You can create these Document add-ins for pretty much all of the different office document types such as PowerPoint and Visio documents.



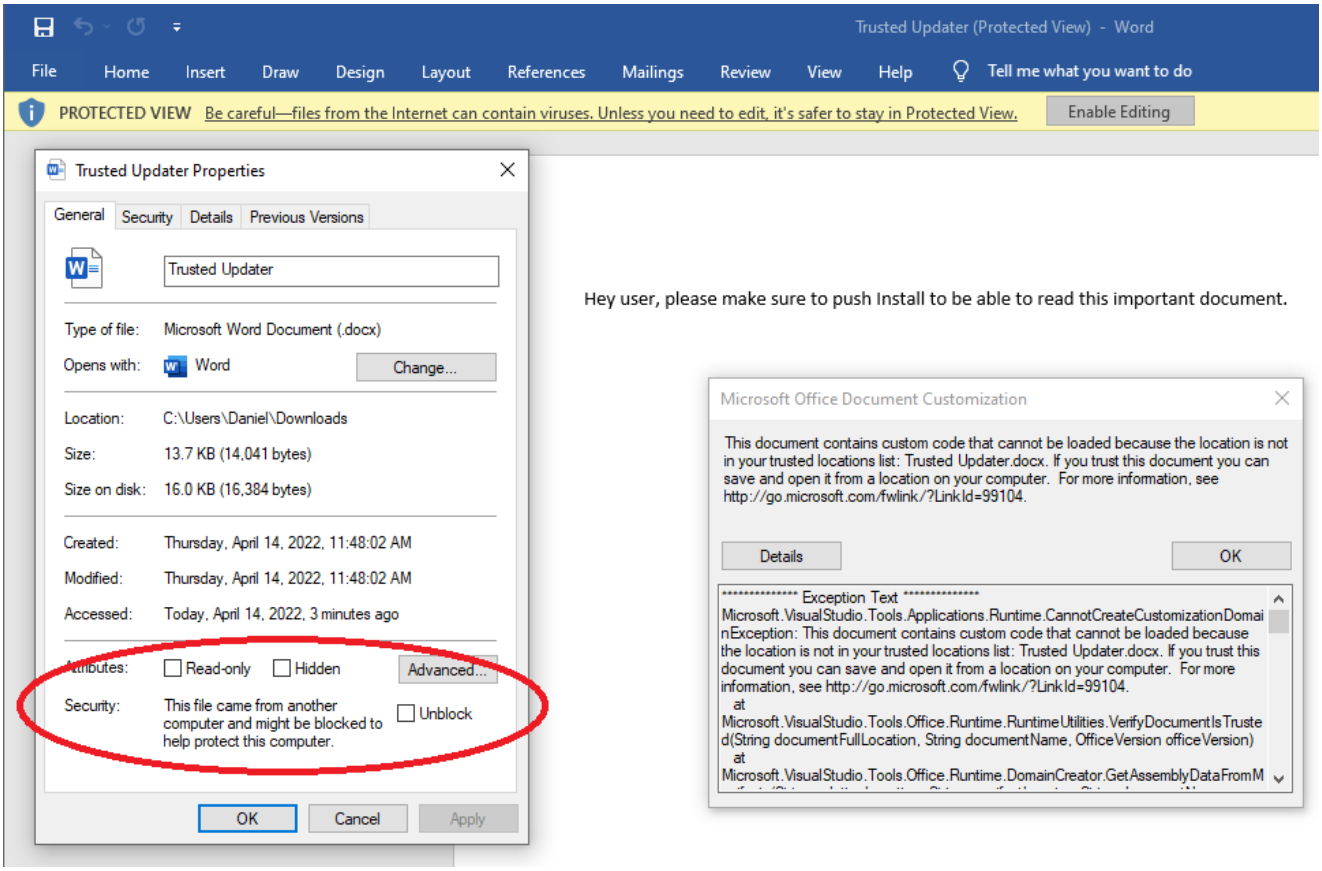
Word is the old me. This is me now.

Default Mitigations

The default Office / Windows mitigations in place to help minimize the risk of a successful attack are the **Mark of the web** and the fact that the publisher needs to be signed by a certificate that is trusted by the target user or computer certificate store.

Mark of the web explained

Thankfully for defenders, documents with the “mark of the web” (i.e. downloaded from certain browsers and email clients) will not run unless they are run from a trusted location.



A document with Mark of the web not allowing the code to run

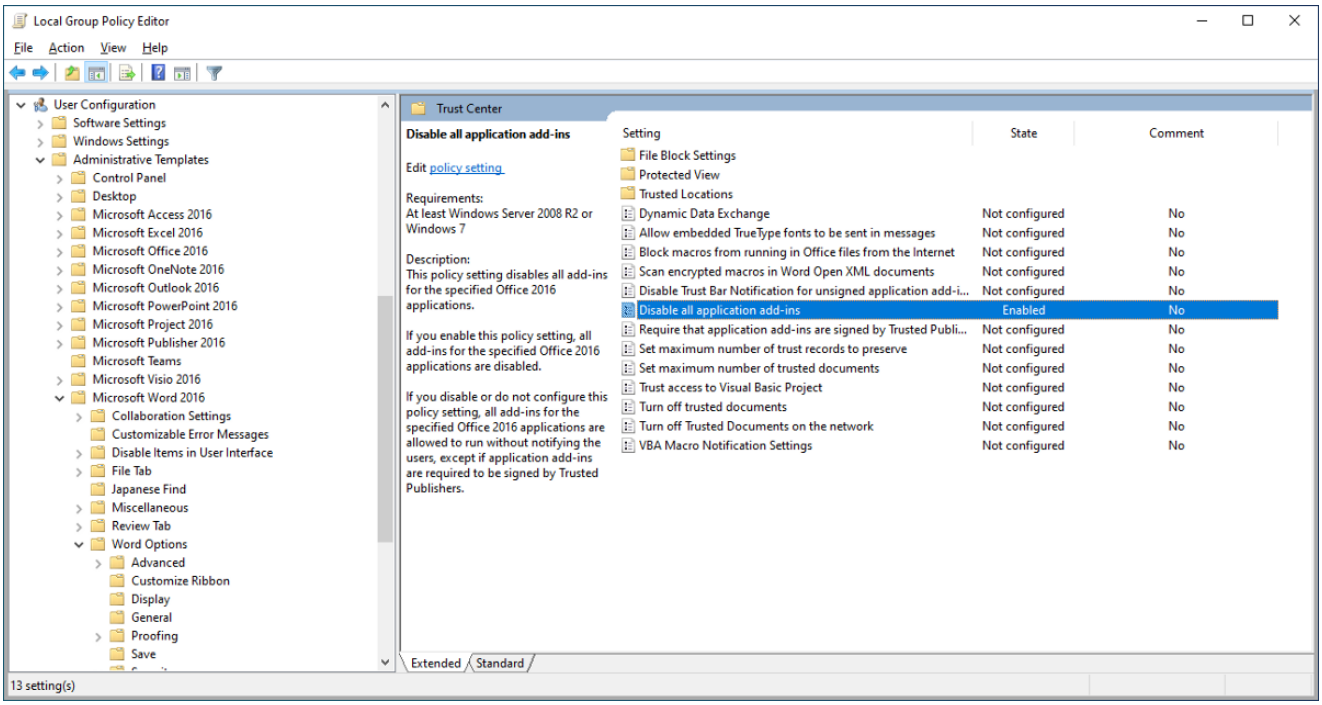
There are ways to Subvert Trust Controls: Mark-of-the-Web Bypass, Sub-technique T1553.005, such as packaging the docx into an ISO file, however it does force the user through additional steps.

Additional Mitigations

Group policy

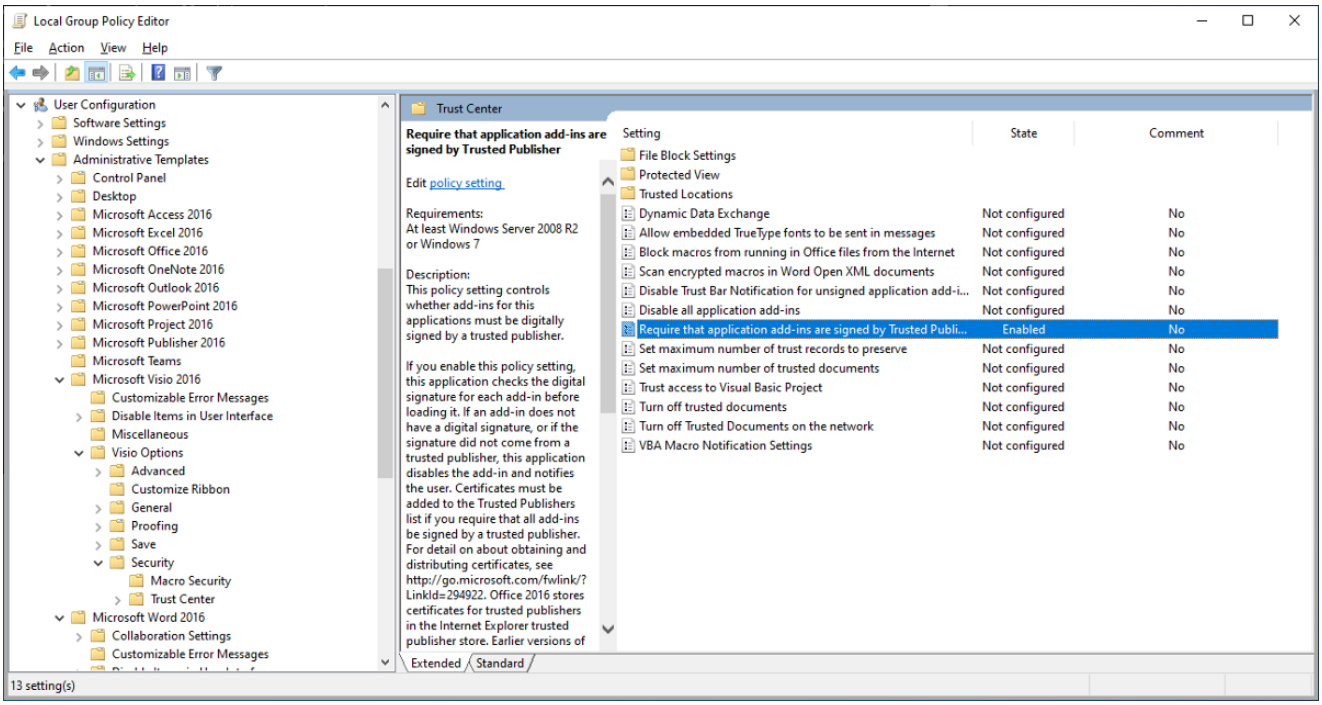
There are a couple of Group Policy settings that can be used to prevent/control Add-ins from running. You need to set this group policy setting for each office application you want to control Add-ins from, i.e. Word, Excel, Visio, etc.

The first option, **Disable all application add-ins**, is the nuke from orbit option which will flat out prevent add-ins from being loaded. On loading the document the user is not presented with any prompt, and active content will not run. This setting will not just disable VSTO content in documents, but also other office add-ins so although it may appear to be a simple fix, making this change would be a challenge for many organizations to implement as there are probably more Office add-ins in use than you would expect, such as Grammarly, DocuSign, label makers, or documentation classification add-ins such as used by Government organizations.



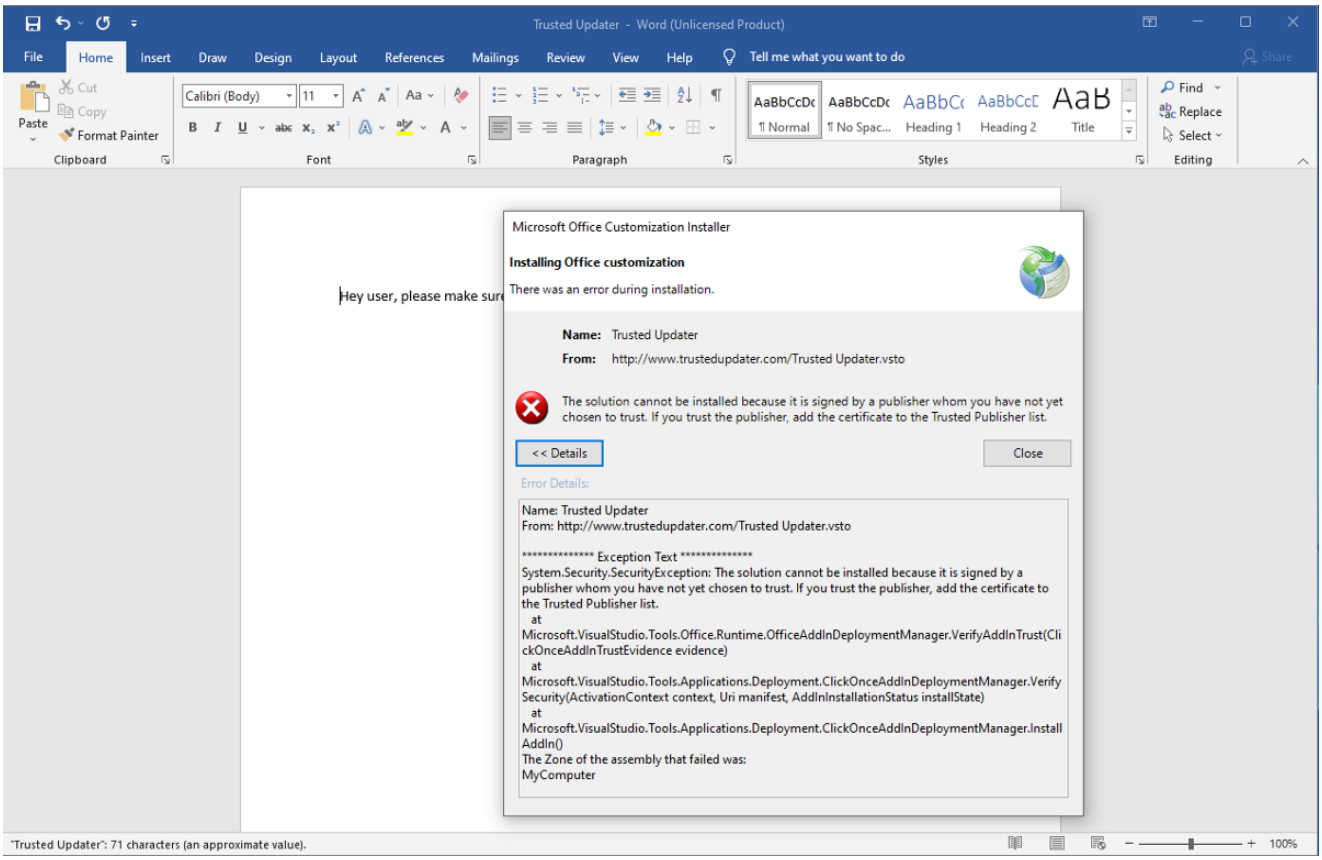
Leave no survivors

The second Group Policy setting that can be used to control add-ins, is the **Require that application add-ins are signed by Trusted Publisher**.

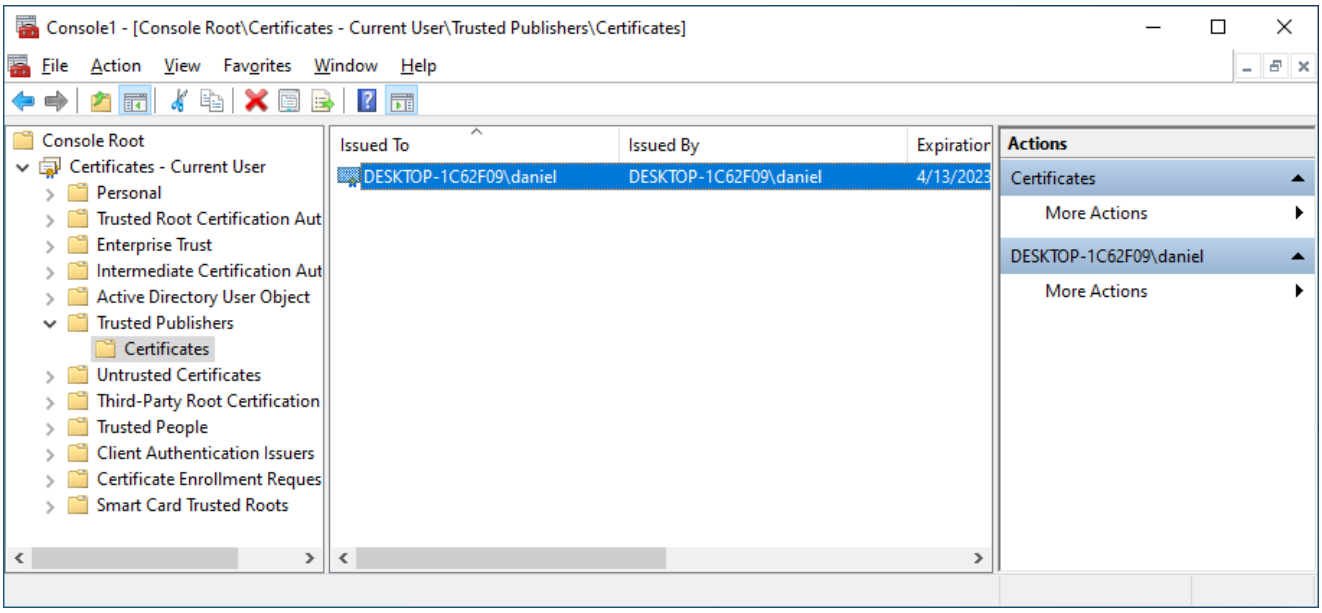


Better

When this option is enabled the document Add-in cannot be installed unless the publishers' certificate is added to the **Trusted Publishers** certificate store.



Womp womp

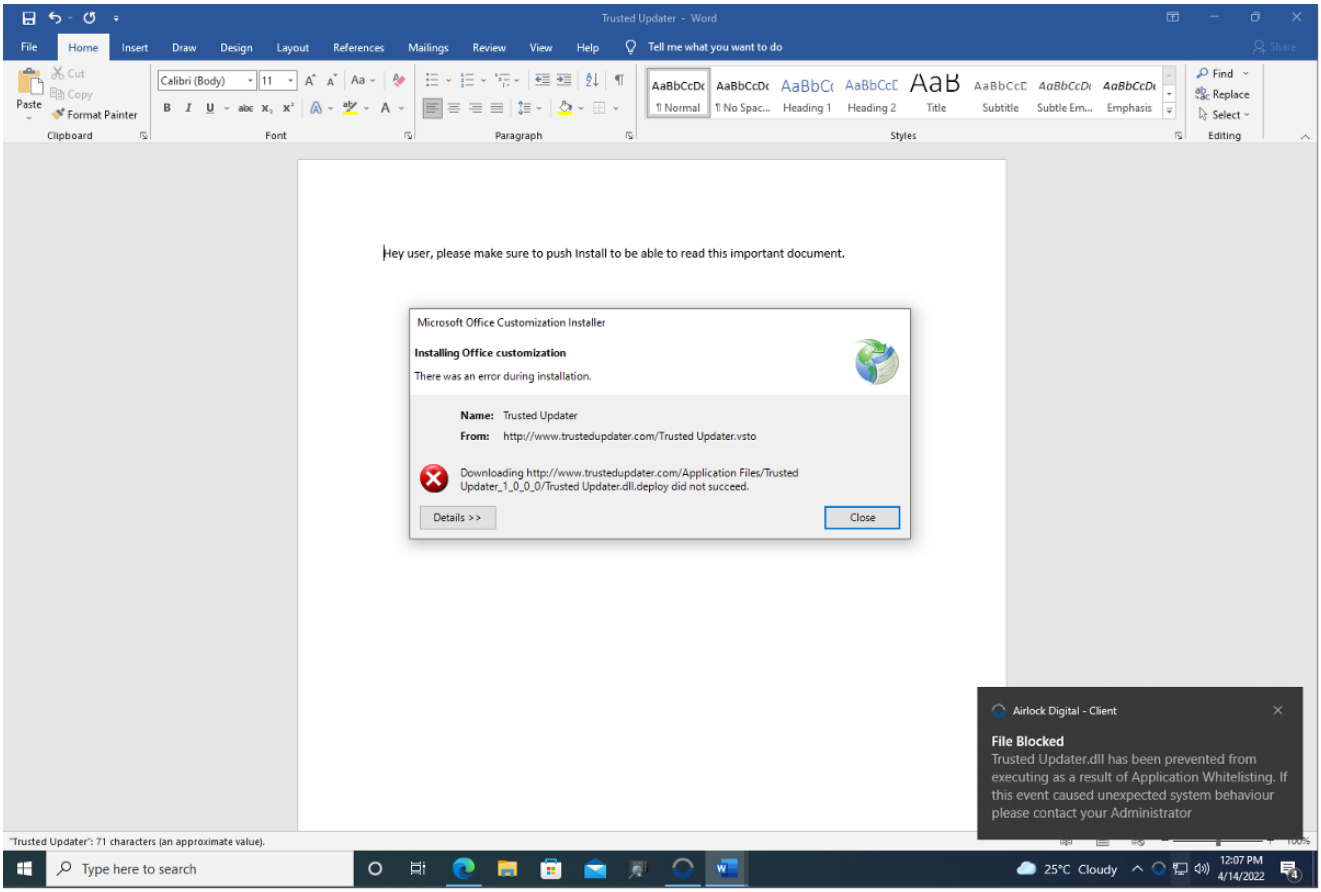


Certificates for trusted publishers must be added here when the GP restriction is in place

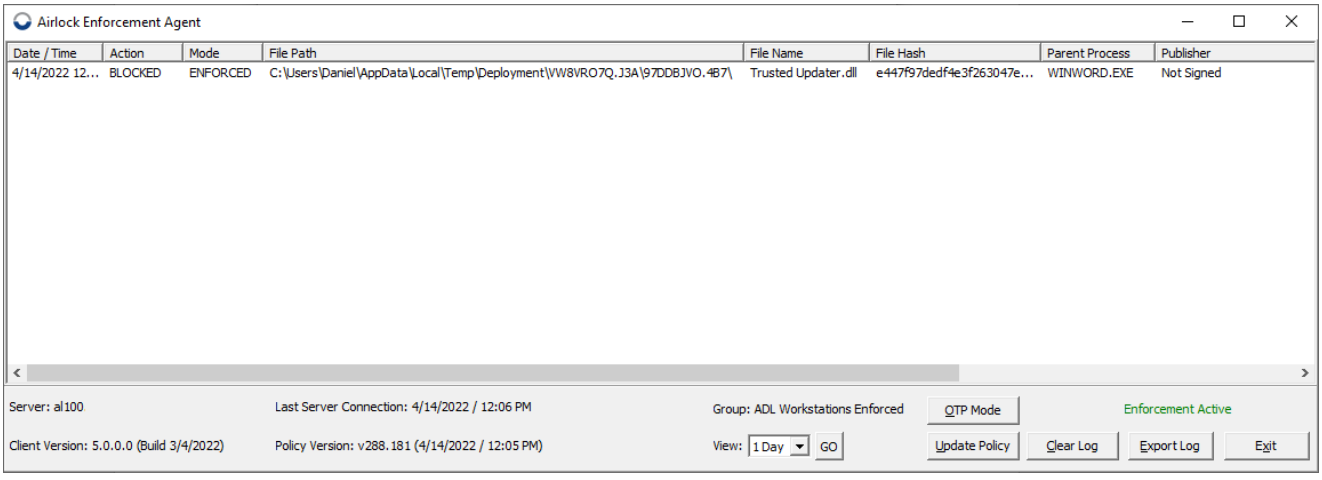
What is interesting here, regardless of the policy setting, is that if the signer is in the Trusted Publishers store, you are no longer prompted to install the add-in, it is just automatically installed.

Allowlisting

By preventing untrusted code from running in your organization (assuming the technology you have chosen is capable of blocking .NET assembly loads from trusted applications) you are able to gain visibility, and choose to block or allow individual add-ins to run based on their hash, publisher, etc.



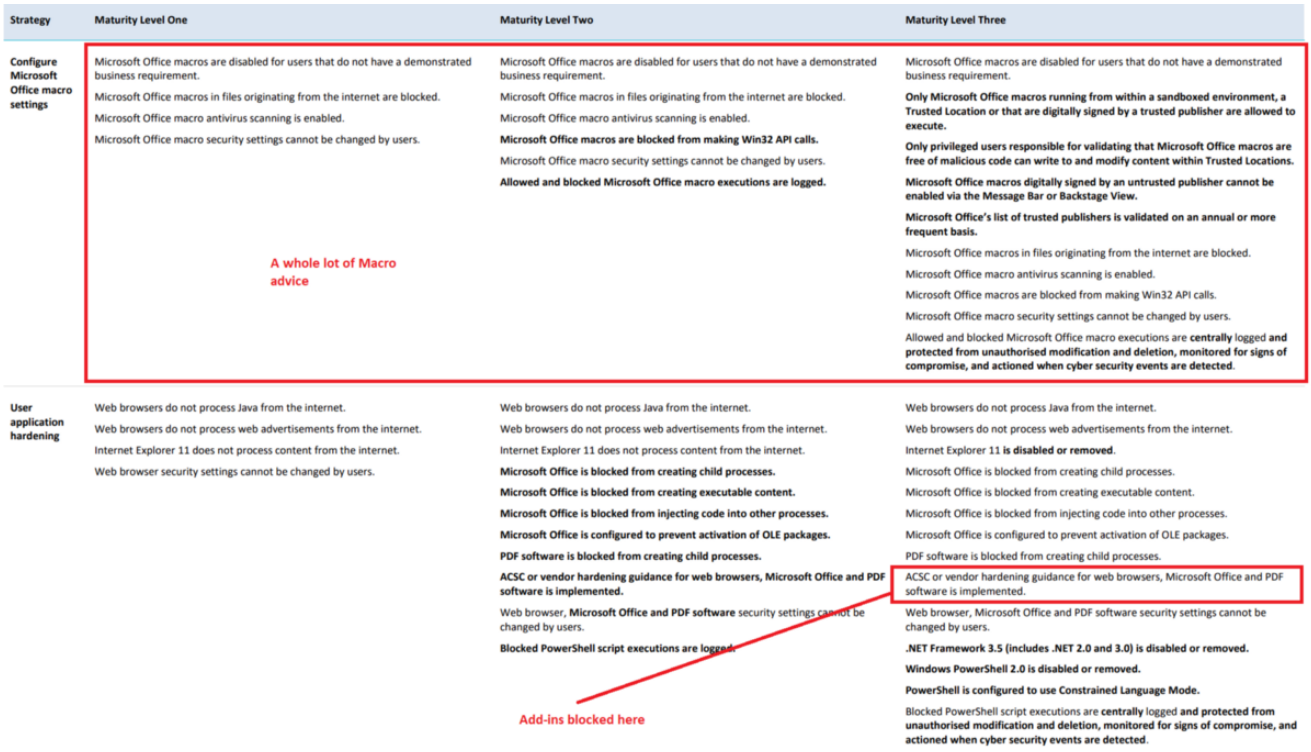
Untrusted document prevented from launching .NET assembly pulled from the internet



Conclusion

Testing has found VSTO Office files to be nearly invisible to AV/Endpoint security products today (somewhat dependent on their behavior) and while VSTO office files have all, if not more, of the capabilities of Macros, they have not seen to be weaponized to the same extent. Given that VSTO office files can be distributed in a single office document this may represent an emerging technique to be leveraged in the future.

It can be seen in the current ACSC Essential 8 Maturity Model there are requirements for hardening systems against Macros at all levels of the maturity model, where Office add-ins are only covered in a supporting document at the highest maturity level.



An extract of the ACSC Essential 8 Maturity Model

The research has also highlighted the many layers of defense in depth, created over many years, in the Microsoft / Office ecosystem today. The complexity in walking through the different protections, such as Mark of the web, the certificate stores, signing, trusted locations (URL & file systems), and trusted publishers makes it challenging for both attackers and defenders to successfully achieve their objectives.

About the author

Daniel Schell (@danonit) is the CTO & Co-Founder at Airlock Digital, creator of the endpoint solution Airlock which makes Allowlisting practical. Learn more at https://www.airlockdigital.com/

Prior Research / References

VSTO: The Payload Installer That Probably Defeats Your Application Whitelisting Rules — bohops

ClickOnce (Twice or Thrice): A Technique for Social Engineering and (Un)trusted Command Execution — bohops

ericlaw — Downloads and the Mark-of-the-Web — text/plain (textslashplain.com)

Hardening Microsoft 365, Office 2021, Office 2019 and Office 2016 | Cyber.gov.au

Thanks to

bohops (@bohops) / Twitter — prior art
bitstorm (@marc_cybersec) / Twitter — sanity check, shell code genius.
C Sto (@C__Sto) / Twitter — sanity check and probing questions.
People who prefer to remain anonymous