**IMPORTANT**

**The purpose of this guide is to show you how malware works and how it is run; not to encourage users to reproduce this in a live environment "in the wild." As security professionals, it is our job to understand how this dangerous stuff works, but only reproducing it in a safe and contained environment - please script responsibly.**

**Writing Ransomware in Python - Safely**

Python, a powerful high-level, general-purpose programming language, is one of the most sought after skills in the tech/cyber space! Almost every job description calls for at least some level of knowledge of this scripting language. Not only is it used for automating tasks or as a back end language used to build applications of all shapes and sizes, but a lot of complex malware is also written in Python as well. Now, it is incredibly important to understand how this malware is built and written, so that as security professionals you too know how to defend properly against it. So, today I will be teaching you just how powerful, easy, and dangerous python can be in the wrong hands. Let's get started!

**Requirements**

When working with incredibly dangerous content such as malware, it is important to do it in a safe and enclosed environment, so today we will be doing this lab in a Virtual Machine (VM). First, we are going to be downloading a Virtual Machine and Hypervisor - we will be using a [Kali-2024.04](https://www.kali.org/get-kali/#kali-virtual-machines) Virtual Machine and running it in a VMware hypervisor platform (you can use whatever hypervisor you want, but to make it easy we are using VMware). So download those two files, then upload that VM onto VirtualBox.



**Creating Test Directories and Files**

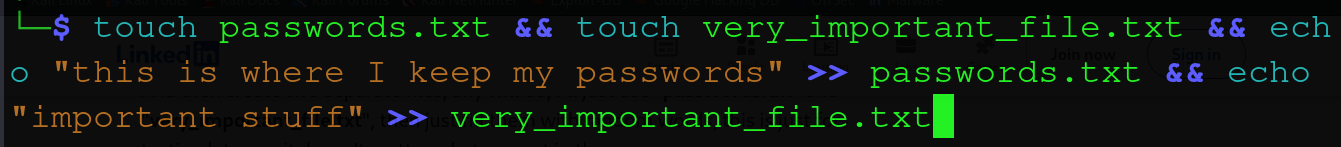
Once you have your VM up and running, go ahead and create a directory called "**Malware**" by running:



Then cd into that directory and create three files, our python script called "**fake\_malware.py**" by running:



And then create two separate files, any will do, I'll just use "**passwords.txt**" and "**very\_important\_file.txt**", then just fill them with random words, this is just for testing later, so it doesn't matter what you put in them:



**Creating the Malware**

Next, lets vim into our "**fake\_malware.py"** script, or whatever your favourite text editor is, and start our journey in creating malware!



The first thing that we will do is an **import os**, which is a library that will be used for interacting with the underlying operating system (OS).



Next, we are going to say **file = []**; this is just setting the variable "**file**" equal to an empty list:



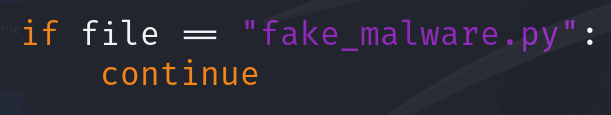
After this, we are going to be creating a “**for loop”**, which is just used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string), it will then look for a file in our listed directory.

We will do this by writing the following code inside our malware script:



All this loop is doing is that it is going through every file in our directory and doing something with it. But, we don't want to encrypt our malware script that we are creating, so let's do something to fix that.

Following the **for loop**, we are going to do a **TAB** then create the following lines of code:



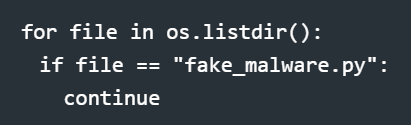
This allows us to be able to skip over our fake\_malware.py file.

Then underneath the **continue**, we will want to append our "**file**" list by using the "**append**" function:



Alright, so far, our script is going through every file in our current directory and then is adding it to our "**file**" list.

The whole block of code should look like the following:



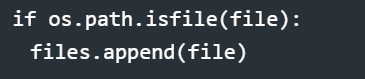
Let's try running this, so let's go ahead and print our variable "**file**":



Once running your script you should see it spit out every file in our current directory! How cool is that!? We are moving along on our malware scripting journey; feel like a criminal yet? Let's continue!

Now, when you ran the script, and if you have another directory within your current directory, you will notice that it actually listed that **specific directory**. We don't want that, so let's jump back into the script to fix that!

We are going to use a built in function within our os library called "**os.path.isfile()**", which is essentially determining if something within our directory is a file or not. Now let's use this in a form of another "**if statement**". But first, let's delete our append function, because we will insert this after the **if statement**:



So, again, what this is saying is that **if** we **ONLY** find files within our directory, we will append our list and add those files **ONLY**.

Now, if we go ahead and run our script, using that same print statement from above "**print(file)**", we should see an output that only reflects our files, and not our directories.

Now, it is time to encrypt these files! Let's go ahead and jump back into our script.

**Encryption**

In order to run encryption syntax inside our script, we must first import an encryption library at the top of our script; so right underneath our **import os**, we will enter the following:



So, Fernet is a super cool tool to use for encryption. Fernet is actually a cryptography passage that provides symmetric encryption and authentication to data; it makes sure that the message encrypted cannot be manipulated/read without the key - it also uses 128 bit AES. It is a part of the cryptography library for Python, which is developed by the Python Cryptographic Authority (PYCA).

Then, back at the bottom of the script, we will insert:



Here we are using the variable "**key**" to use to create our key.

Then, we need to save our key to a file - we want to be able to use that key to unlock our files later; if you aren't familiar with python syntax, I will break down this next part for you.

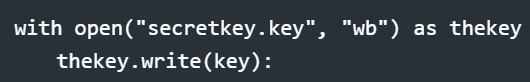


So, we are using the open function to create or open a file called "**secretkey.key,**" then we are going to open it in a specific mode, which is called "**write mode,**" then we are going to refer to it in our function by calling it "**thekey.**" I know that was a lot, but all of this is about learning the "**why**" behind what you are doing; so if you need, take a second and look back at the code and study it; it is okay to take your time with this complex stuff.

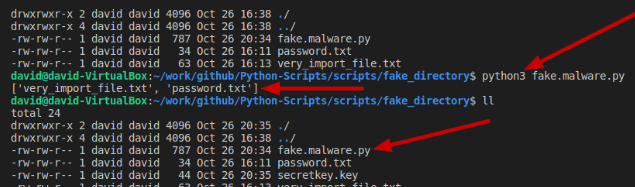
Next, we are going to hit **Enter** and **TAB** and underneath our statement above we are going to write:



So here, we are referencing our function "**thekey**" and telling it to write to our variable "key." That entire block of code altogether will look like the following:



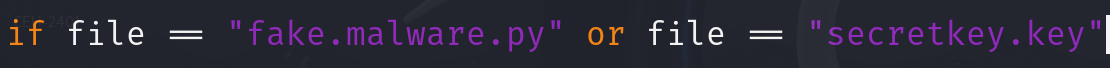
So, let's go ahead and save and run our script again to see if it works!



And would you look at that! After running the script you should see the secret key we created right there in your directory where you are running the script! Super scary - we are on our way to being a hacker yet!

Next, you might be asking yourself, "won't I encrypt my key if I run the script?" Good question! We need to hop back into our script and make sure this doesn't happen; don’t worry, this will be really easy to fix.

Right next to our first "**if statement,**" we are going to add an "**or**" operator and say:



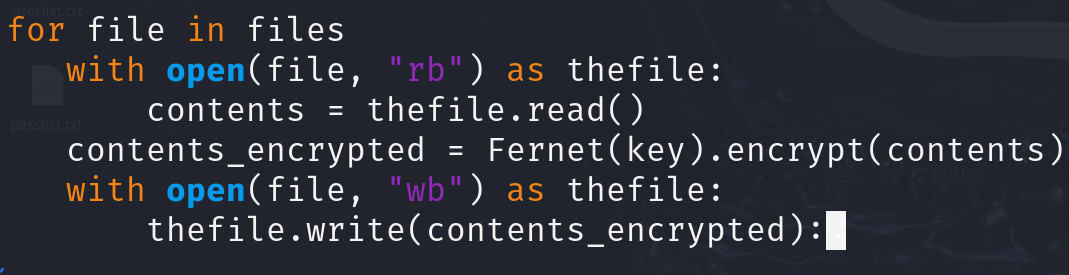
And just like that, we no longer need to worry about encrypting our secret key!

Now, you are probably wondering; "when am I going to be able to lock up some of my files?!" Patience young padawan! That is coming next.

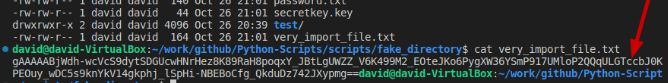
So far, we have created our secret key and set our variable "**key**" that is awaiting some sweet-sweet files to encrypt, which we will do that here! We are going to be using another **"for loop**" to encrypt our files, so get ready once again.

The first thing we are going to do is create another **for loop**; this **for loop** will check every file in the variable "**files**". The next thing that we are going to be doing is opening the files in read binary with our function "**thefile**." Next, we are going to opening the files in read binary mode, as "**thefile**," and then using the variable "contents" to be equal to **thefile.read**, which is just reading all of the files. Then, we are opening the files and encrypting them with the **Fernet** cryptography packet that we imported earlier. And lastly we are writing back to the files with the encryption done above.

Altogether this will look like the following:



Now, let's go ahead and save this and run our script again.



WOW, look at that! All our files in our directory are encrypted! But, don't worry, you don't have to pay any crazy sum of money to get them back. But, you are probably wondering… "okay, now how do I decrypt them." Well, again, I am glad you asked! We are going to actually write a decrypt script that will do just that!

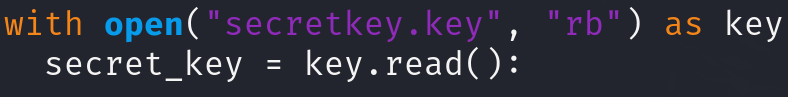
**Decrypting our Files**

So let's actually copy the contents of our **fake\_malware.py** script to another script, because as tech professionals, we are always looking for shortcuts.We can do that easily by doing the following:



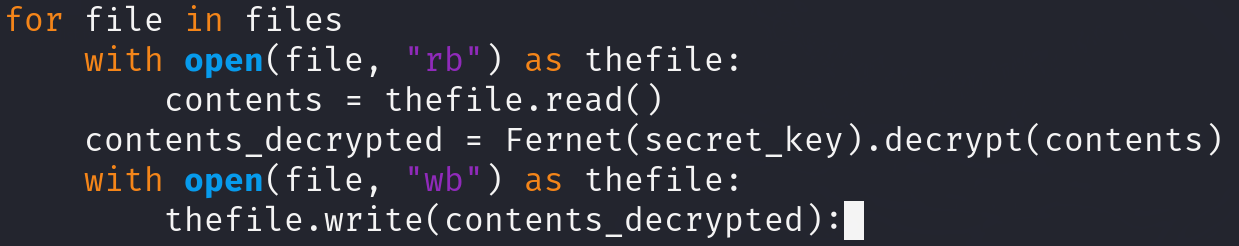
Great, now we have the start to our decrypt script. Let's go ahead and jump into that decrypt script and tweak a few things!

The first thing we are going to do is get rid of our "**key**" variable and delete the area where we are writing the encryption key. Instead, the following code will be written:



Here we are opening the "**secret\_key.key**" file in read binary mode as "**key**," and then we are creating a new variable called "**secret\_key**" and making that equal to the contents of that file.

Lastly we are going to tweak our encrypt "**for loop**" to decrypt instead! This will look something like this:



So, it might look the same at first, but when you look a little closer, you will notice some differences. This "for loop" will check every file in the variable **files**, like last time, then we are opening the files in read binary with our function "**thefile**," then we are using the variable "**contents\_decrypted**" to be equal to "**thefile.read**", so it is reading the contents within that file, then we are opening the files and decrypting them with Fernet and our **"secret\_key"** variable above, and then lastly we are writing the decrypted contents back to the file.

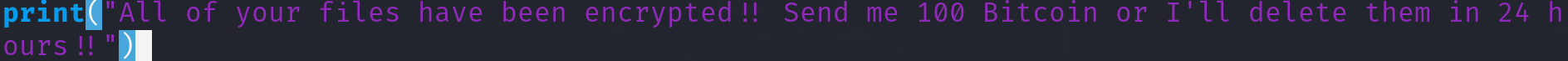
Let's go ahead and run that script over our encrypted files and see if that works!

BOOM! There we go! Our files have been returned to their original form! You can breathe a sigh of relief now; but remember, this is only a simulation, no actual crypto was used in this process.

Now, the last thing that we will do is add the "stereotypical messages" that most ransom ware victims get; the - "I HAVE STOLEN YOUR FILES! GIVE ME CRYPTO…" or something along those lines.

**Adding Ransom-ware Print Statements and Secret Phrase:**

Here, you can get creative! It is your encryption script! So let's jump back into our "**fake\_malware.py**" script, and jump to the bottom! Here we will add a simple print statement, of your choice, that expresses that the victim has been hacked! I decided to keep it classic and do something like this:

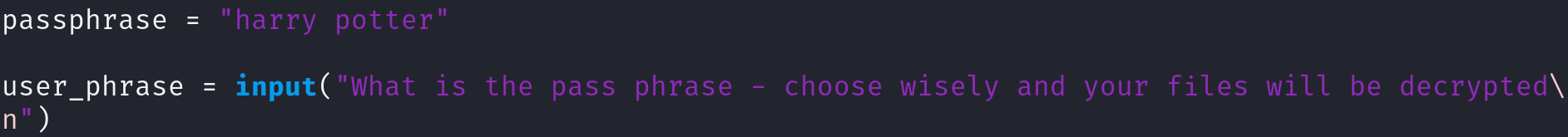


Now, if we jump out of it, and run it again, we will see that it lists the encrypted files AND our fun print statement as well.

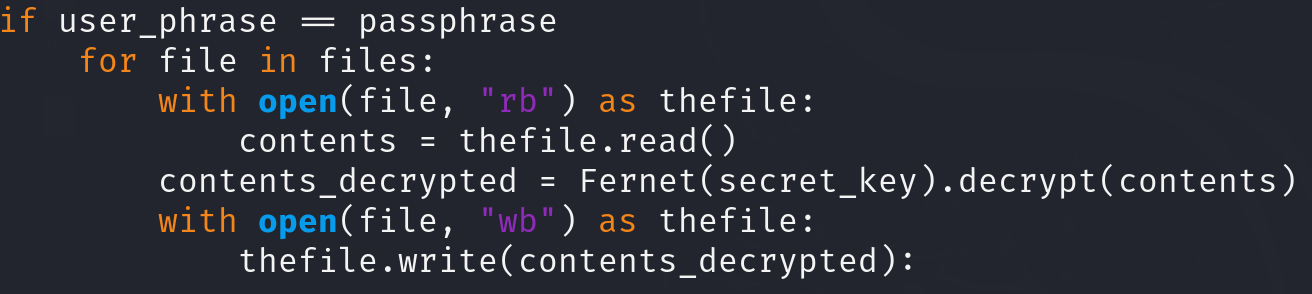


Now, let's make our decrypt script a bit more on the fun side! In order for their files to be decrypted, after they have given us the **bitcoin**, we will give them a secret pass phrase, and if and only if they have that will their files be decrypted. So let's jump back into our decrypt script!

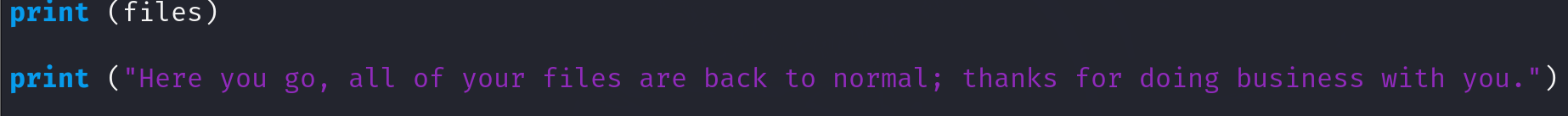
The first thing we will add is a variable called "passphrase" and set it equal to our "secret pass phrase of choice," which will be something like "harry potter", followed by another variable, "user\_passphrase", which then we will set it equal to an input statement, followed by a "\n" which is just a line break; this will prompt the user to enter in the passphrase that we set above, which in this case is "harry potter" - the code will look like the following:



Then, we are going to create an if statement where we set our variable "**user\_phrase**" equal to our "**passphrase**" variable, and if those are equal, we will run our decrypt for loop, "**else**," we will run some print statement and the for loop doesn't run. In order to do that, we have to nest the for loop under the "**if statement**," and if you are new to Python, that means we will have to tab over our for loop once per line. It should look something like this:

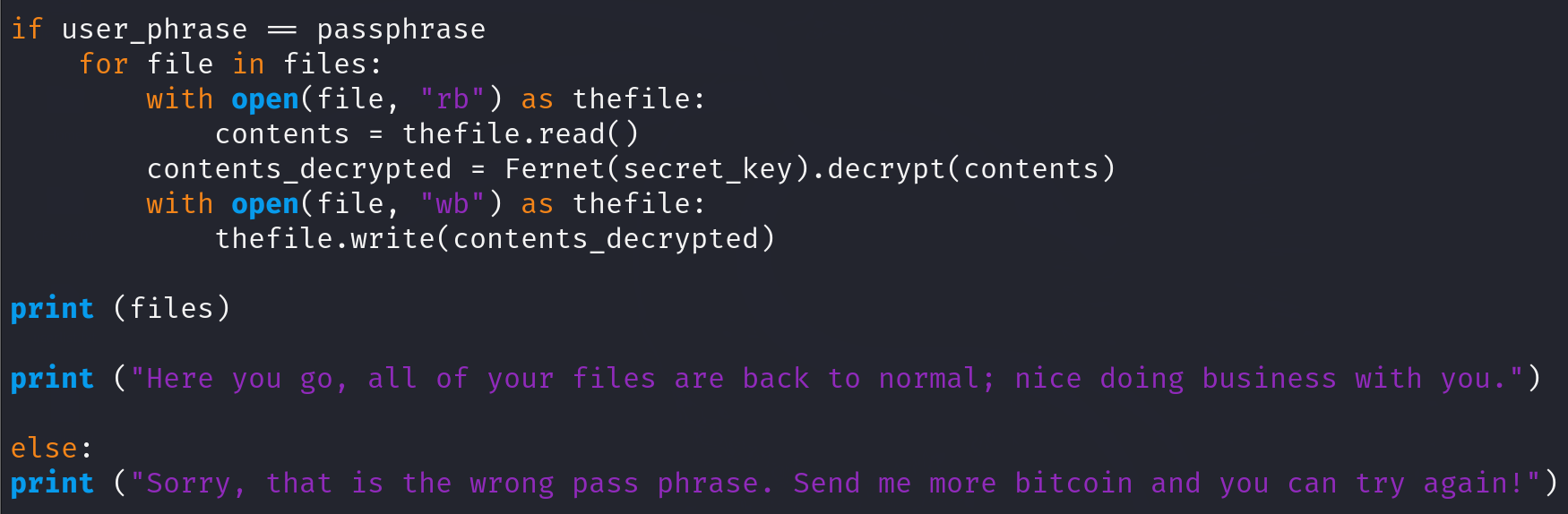


Then, if the passphrase was entered successfully, we'll print our list "**files**" and follow that with another **print** statement that confirms to the user that their files are encrypted; this should be entered right underneath our **for loop**:

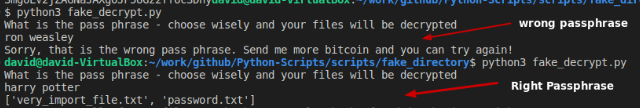


Lastly, in order to complete our **if statement**, we have to add an "**else**;" this signifies that **IF** the passphrase is wrong, **ELSE** run what follows, which will be a **print** statement stating that the pass phrase was incorrect.

So, our if/else statement, with a nested for loop, should look something like this:



So, if we run it again, we should see something like this.



There we go! 100% success! You just created malware, give yourself a hand! But only try this in a throwaway VM environment ONLY! I trust you though.

**Summary**

So, today we learned some powerful but fun python syntax! We learned how to import libraries, use if/else statements, print statements, and how to use some complex for loops to encrypt and decrypt files! I hope you are able to see the sheer power and simplicity that malware can be - obviously there is a LOT more that goes into stuff like this, but the reasoning for me doing this was to help educate security professionals on python, and malware itself. We need to be able to understand malware before we are able to protect against this in our day to day roles. Hope you found some value in what I wrote today, and remember, always practice safe scripting.

***S. H. RAFAT. Security Specialist. Creative Business Group | Creative IT Institute | Creative Skills.***

***Do not engage in any illegal activities under any circumstances. Never attempt actions involving Government, Private Organizations, or Military Sectors.***