Python Simple HTTP Server With SSL Certificate (Encrypted Traffic)

The easy way to make a temporary HTTP server with TLS encryption.

Febi Mudiyanto · Follow

Published in Python in Plain English

3 min read · Aug 19, 2022

Photo by Mathew Schwartz on Unsplash

The simple HTTP server is a feature from python that allows us to create an HTTP server in a simple way. In another way, usually, hackers or penetration testers use this method to transfer files between the attacker machine (Kali Linux) to the victim machine. Because of the assumption that firewalls usually allow access from inbound port 80 or HTTP.

The cons of this method are, that all the communication is unencrypted because just with HTTP. While the attacker transferred the malware into the victim machine, so, the Blue Team / Threat Hunter found our methodology to attack easily.

There is one answer to making the Blue Team hard to do forensics, it encrypts the communication while transferring malware.

But, how?

Let's get started

Actually, I got this script from Red Team Field Manual (RTFM), I attach the link if you are interested in this book.

Before going to the script, firstly prepare for the SSL certificate (private key and public certificate). If you are new in this field, just looking my previous article about SSL certificates.

Automate the Local Certificate Authority Registration with Python

How to make a self-signed SSL certificate with your own CA

python.plainenglish.io

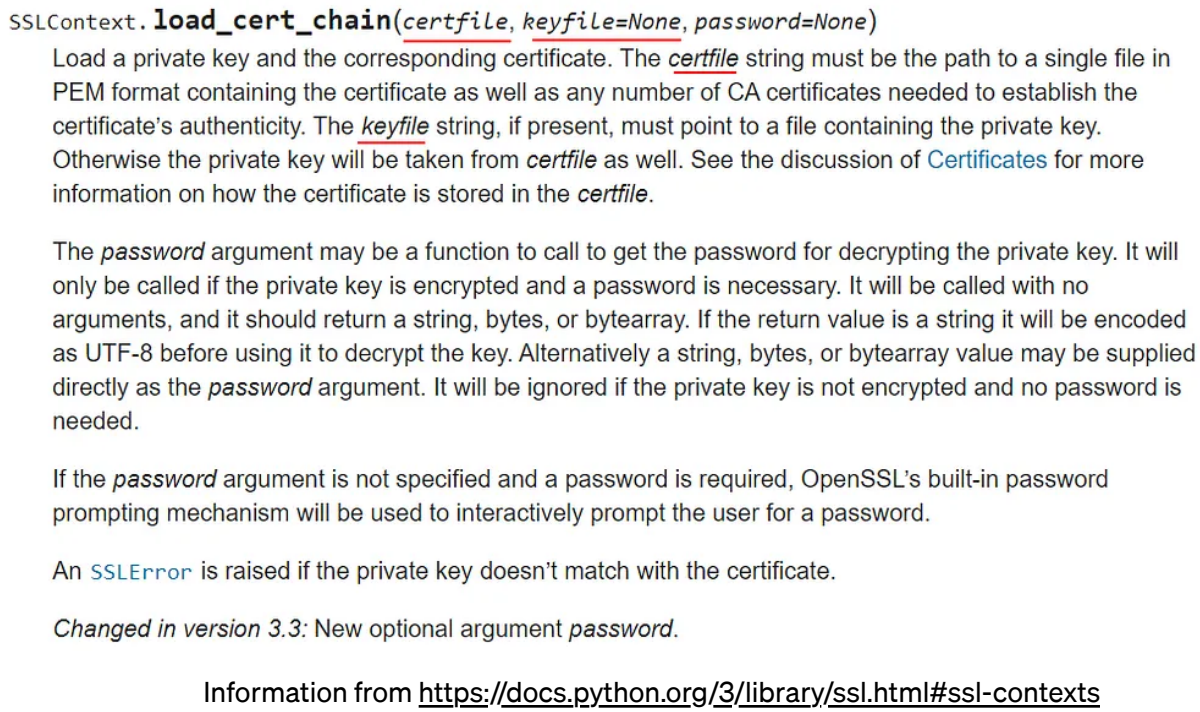
or you can do this command in your Linux terminal

```
openssl req -new -x509 -keyout cert.pem -out cert.pem -days 365
-nodes
```

Let's say you have a cert file and a private key or you run the command above and have a single cert.pem file.

In this example, I use step 2 for generating an SSL certificate. So, now I just have **cert.pem**.

If you have a private key and cert file. Just read this information below for a little enhancement in your code.



`SSLContext.`**`load_cert_chain`**`(certfile, keyfile=None, password=None)`

Load a private key and the corresponding certificate. The *certfile* string must be the path to a single file in PEM format containing the certificate as well as any number of CA certificates needed to establish the certificate's authenticity. The *keyfile* string, if present, must point to a file containing the private key. Otherwise the private key will be taken from *certfile* as well. See the discussion of Certificates for more information on how the certificate is stored in the *certfile*.

The *password* argument may be a function to call to get the password for decrypting the private key. It will only be called if the private key is encrypted and a password is necessary. It will be called with no arguments, and it should return a string, bytes, or bytearray. If the return value is a string it will be encoded as UTF-8 before using it to decrypt the key. Alternatively a string, bytes, or bytearray value may be supplied directly as the *password* argument. It will be ignored if the private key is not encrypted and no password is needed.

If the *password* argument is not specified and a password is required, OpenSSL's built-in password prompting mechanism will be used to interactively prompt the user for a password.

An `SSLError` is raised if the private key doesn't match with the certificate.

Changed in version 3.3: New optional argument *password*.

Information from https://docs.python.org/3/library/ssl.html#ssl-contexts

The next step is to create a file called **https-simple-server.py** or anything you want to be the name of the file.

Put the code below into that file:

```
# Date : 17 - 08 - 2022
# Ref: RTFM v2
'''
Python3 simple-https-server.py
'''

import http.server, ssl, socketserver

context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain("cert.pem") # PUT YOUR cert.pem HERE
server_address = ("192.168.43.210", 4443)  # CHANGE THIS IP & PORT

handler = http.server.SimpleHTTPRequestHandler
with socketserver.TCPServer(server_address, handler) as httpd:
    httpd.socket = context.wrap_socket(httpd.socket, server_side=True)
    httpd.serve_forever()
```

simple-https-server.py hosted with ❤️ by **GitHub** **view raw**

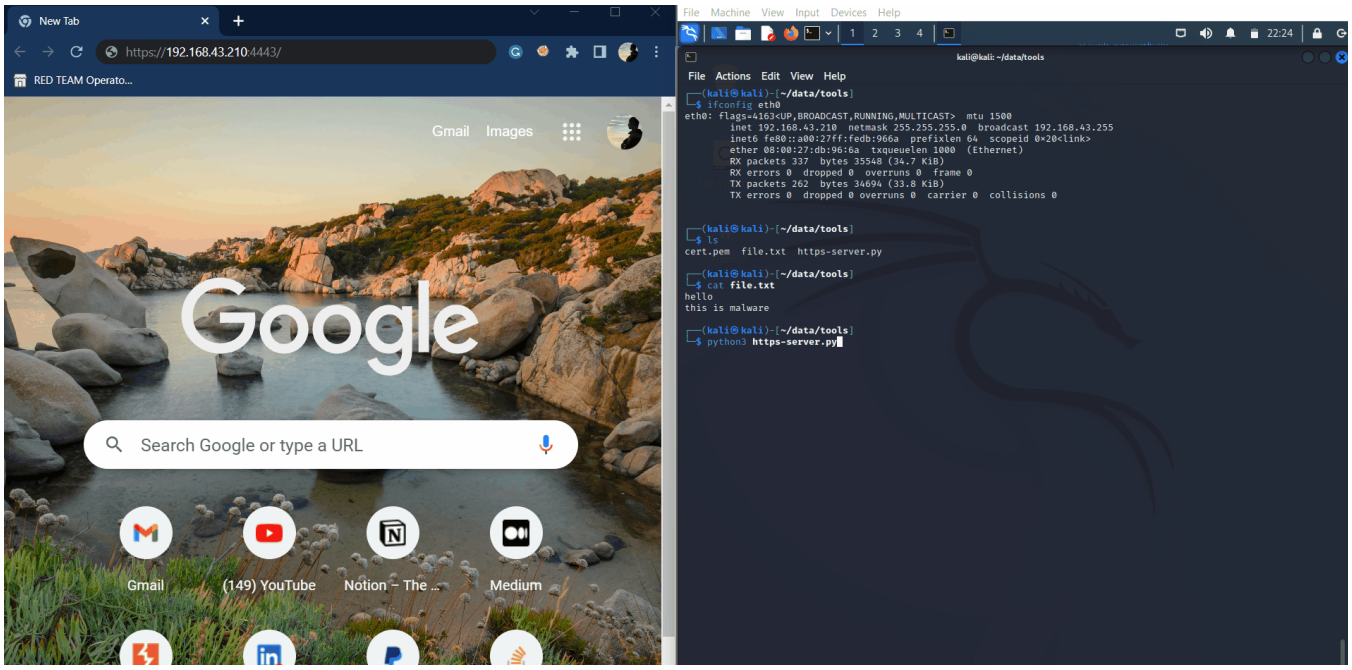
Run this program with this command:

```
Python3 simple-https-server.py
```

Then access the server with the browser of the other machine:

https://<IP>:<port>

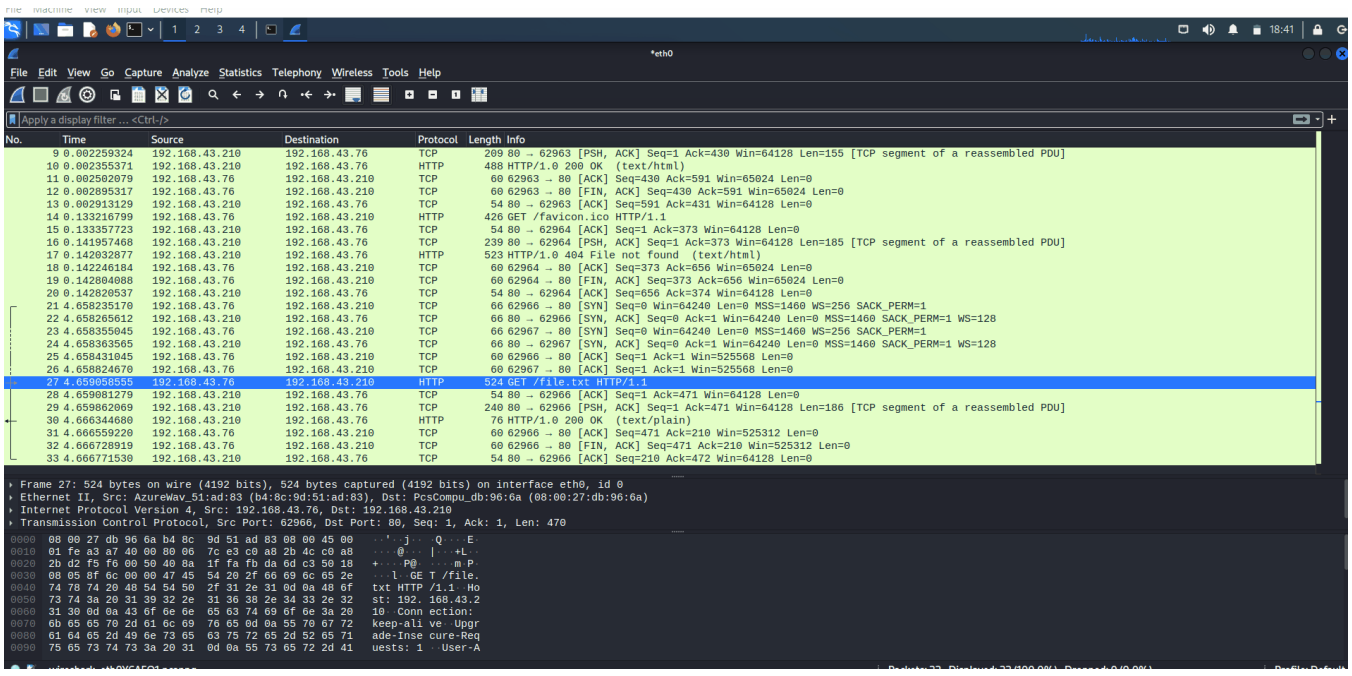
Finally, enjoy your encrypted transfer file with this simple HTTP server + SSL certificate.



Prove of Concept with Wireshark

Unencrypted

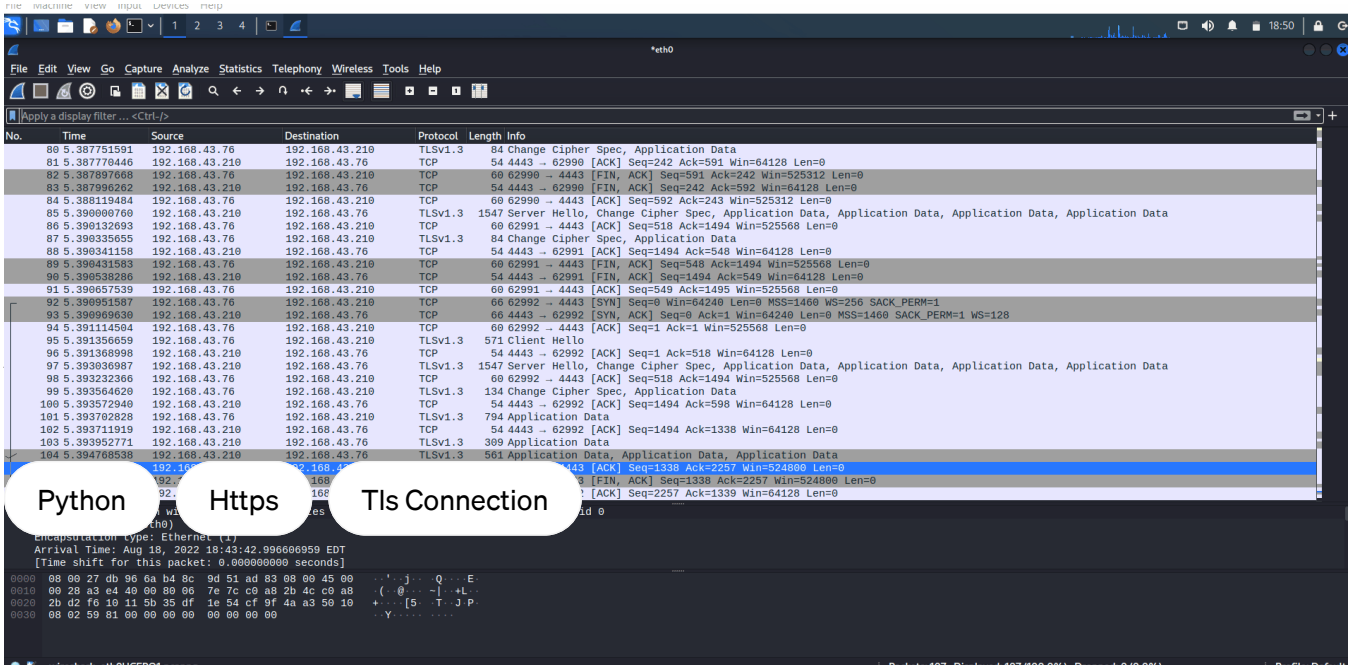
If you use HTTP in transferring files, it's will seem like this:



Wireshark captures HTTP traffic

Encrypted

On the other side, when you use HTTPS to transfer files, all of the information will be encrypted.



Wireshark captures HTTPS traffic

Conclusion

Keep your way as simple as you know, so you are not afraid of tools and processes. This python feature makes me feel easy to set up the transferring file in the daily job or night job as a CTF player.

RTFM: Red Team Field Manual v2

Follow

Written by Febi Mudiyanto

383 Followers · Writer for Python in Plain English

Just a Learner and CTFs Player on a quite night.

More from Febi Mudiyanto and Python in Plain English