ICT2202 Digital Forensics Assignment 1

User Manual for Blitz-X

Deadline for Submission: 7th November 2021

Lab Group: P4

Team Name: Panzerwerfer

<u>Student Details</u>

Student Name	**Student ID**
Jonathan Tan Yu Shen	2001428
Koh Jun Jie	2000819
Ryan Goh Shao Chong	1802980

Contents

I. Overview

Blitz-eXtractor (Blitz-X) is a modular triage tool written in Python designed to access various forensic artefacts on Windows relating to user data exfiltration. The tool will parse the artefacts and present them in a format viable for analysis. The output may provide valuable insights during an incident response in a Windows environment while waiting for a full disk image to be acquired. The tool is meant to run on live systems on the offending User Account with administrative rights. Blitz-X's key features include:

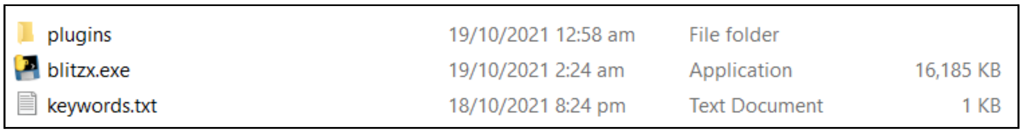
- **Minimised footprint on the target's system**
  Only "read" operations are performed on specific files specified by Blitz-X's modules on the target system.

- **User friendly**
  Blitz-X's command-line interface (CLI) only comprises two optional parameters, "--keydec" and "--keywords" and is easily understandable by a user of a non-technical nature. The "--keydec" parameter would run Blitz-X in "keydec" mode, described in Section VIII of the report. The "--keywords" parameter allows the user to specify a wordlist for Blitz-X to perform a keyword search on the extracted evidence files. Further information on the CLI's parameters can be found in Section II.

- **Highly configurable**
  Blitz-X is expandable with custom modules. Further information on adding and customising Blitz-X can be found in Sections V, VI and VII.

- **Provides chain of custody**
  The data files and report files generated by Blitz-X's modules are hashed, placed into a master hash file, and hashed with SHA-256. The master hash file and its SHA-256 hash are encrypted with a randomly generated AES-256 bit key. The AES-256 bit key is then exported and encrypted with the user's public key. The integrity of the data and report files are verified by reversing the above process. Further information on the chain of custody process can be found in Section II.

The compiled version of Blitz-X comprises the following files:

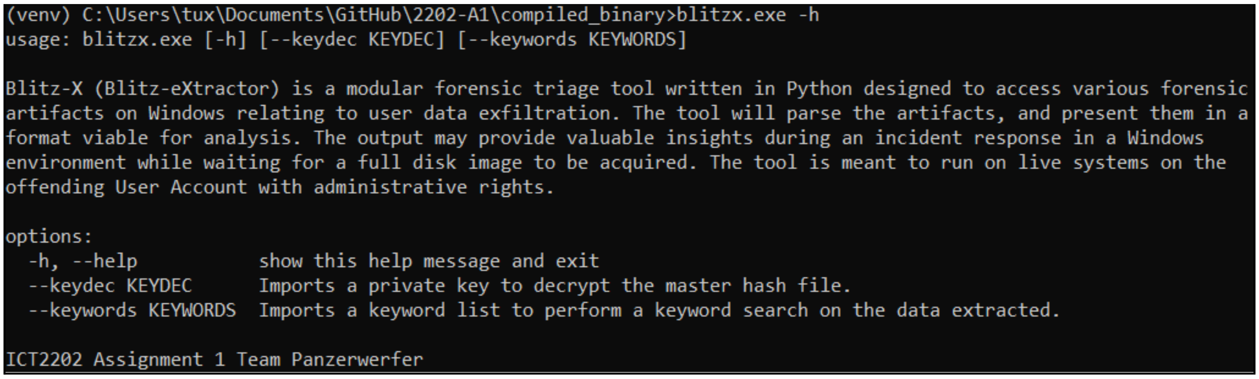


plugins	19/10/2021 12:58 am	File folder	
blitzx.exe	19/10/2021 2:24 am	Application	16,185 KB
keywords.txt	18/10/2021 8:24 pm	Text Document	1 KB

Name	**Description**
plugins	A folder containing the core and default supporting modules which are included with Blitz-X. A comprehensive description of these modules can be found in section IV.
blitzx.exe	The compiled binary of Blitz-X.
keywords.txt	A txt file containing a list of keywords for a module, “keyword_search”, one of Blitz-X’s core modules.

II. Arguments

Most of the functions in the tool are automatically run unless specified. The program supports two arguments: decrypt the master hash file and keywords to specify the keywords file for the “Keyword Search” module.



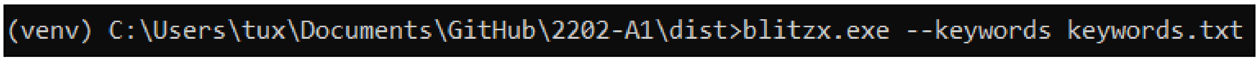
```
(venv) C:\Users\tux\Documents\GitHub\2202-A1\compiled_binary>blitzx.exe -h
usage: blitzx.exe [-h] [--keydec KEYDEC] [--keywords KEYWORDS]

Blitz-X (Blitz-eXtractor) is a modular forensic triage tool written in Python designed to access various forensic
artifacts on Windows relating to user data exfiltration. The tool will parse the artifacts, and present them in a
format viable for analysis. The output may provide valuable insights during an incident response in a Windows
environment while waiting for a full disk image to be acquired. The tool is meant to run on live systems on the
offending User Account with administrative rights.

options:
  -h, --help           show this help message and exit
  --keydec KEYDEC      Imports a private key to decrypt the master hash file.
  --keywords KEYWORDS  Imports a keyword list to perform a keyword search on the data extracted.

ICT2202 Assignment 1 Team Panzerwerfer
```

Argument Name	**Description**
--keydec	Users can decrypt the master hash file encrypted with a randomly generated AES-256 at the end of the tool runtime and its encrypted SHA-256 hash. The master hash files consist of all the MD5, SHA-1, SHA-256, and SHA-512 hashes of the extracted JSON files and HTML report files generated by Blitz-X’s modules. After decryption, the decrypted master hash file will be hashed again with SHA-256, and the hash value will be compared to the original decrypted SHA-256 hash. If both hashes match, it would mean that the master hash file was not tampered with during the chain of custody. The extracted JSON files and HTML report files will also be re-hashed, and their hash values will be stored in a text file. This text file is hashed with SHA-256 and compared with the decrypted master hash file’s SHA-256 hash. If both hashes match, the JSON files and HTML report are proven to be forensically sound and not tampered with.
--keyword	Users can specify the keyword text file to perform a recursive string search on the data files that Blitz-X's modules extracted. It is optional because the keyword search module would not run if the user did not specify it.

III. Executing Blitz-X

Step 1: Launching Blitz-X

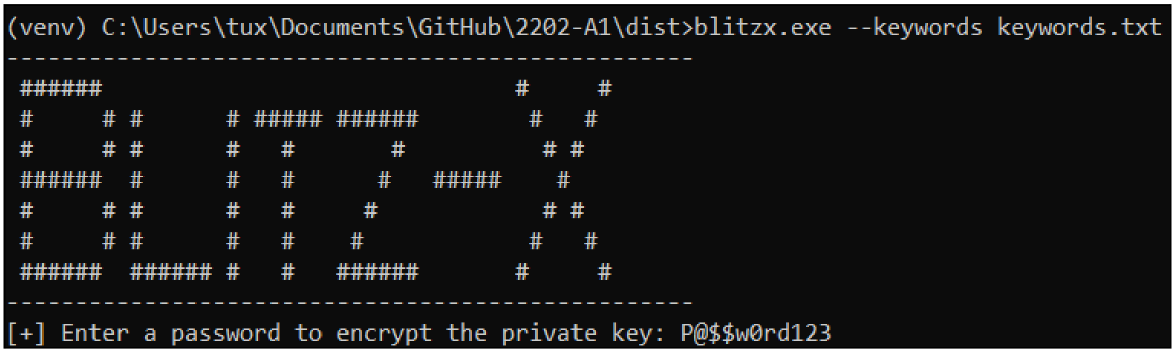
Load Blitz-X onto a thumb drive and plug it into the offender’s machine. Blitz-X requires administrative rights to be able to run. This is a command-line tool; therefore, it needs to be run from the command prompt. After supplying the tool with appropriate arguments, the tool will run and collect data based on the compiled plugins.



```
(venv) C:\Users\tux\Documents\GitHub\2202-A1\dist>blitzx.exe --keywords keywords.txt
```

Step 2: Generating the Private Key

Upon running Blitz-X, if a public key is not present in the same directory, it would prompt the user to enter a password to generate a new private key. That private key will be used to decrypt files later on that will be encrypted by the tool.

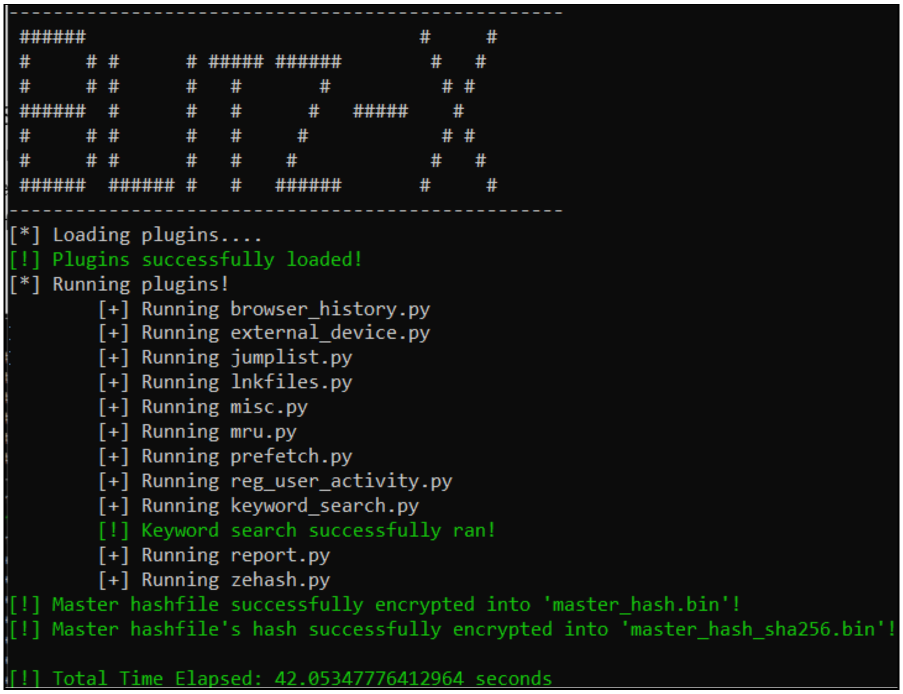


```
(venv) C:\Users\tux\Documents\GitHub\2202-A1\dist>blitzx.exe --keywords keywords.txt
--------------------------------------------------
 ######                              #     #
 #     # #      # ##### ######        #   #
 #     # #      #   #       #          # #
 ######  #      #   #      #   #####    #
 #     # #      #   #     #            # #
 #     # #      #   #    #            #   #
 ######  ###### #   #   ######       #     #
--------------------------------------------------
[+] Enter a password to encrypt the private key: P@$$w0rd123
```

*The naming convention of the private key file has to specifically follow “public_key.pem”.

Step 3: Wait for Blitz-X to Successfully Execute

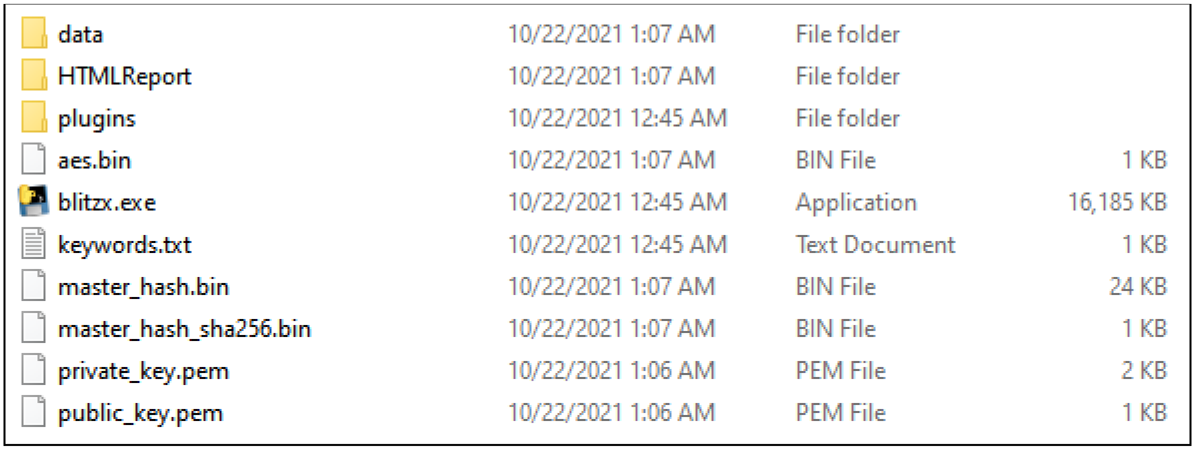
If no errors occurred during runtime, success messages highlighted in green would be displayed. Upon the tool's successful execution, remove the thumb drive containing Blitz-X and its extracted artefacts from the offender's machine and plug it into a forensically sound machine for analysis.



```
--------------------------------------------------
 ######                              #     #
 #     # #      # ##### ######        #   #
 #     # #      #   #       #          # #
 ######  #      #   #      #   #####    #
 #     # #      #   #     #            # #
 #     # #      #   #    #            #   #
 ######  ###### #   #   ######       #     #
--------------------------------------------------
[*] Loading plugins....
[!] Plugins successfully loaded!
[*] Running plugins!
        [+] Running browser_history.py
        [+] Running external_device.py
        [+] Running jumplist.py
        [+] Running lnkfiles.py
        [+] Running misc.py
        [+] Running mru.py
        [+] Running prefetch.py
        [+] Running reg_user_activity.py
        [+] Running keyword_search.py
        [!] Keyword search successfully ran!
        [+] Running report.py
        [+] Running zehash.py
[!] Master hashfile successfully encrypted into 'master_hash.bin'!
[!] Master hashfile's hash successfully encrypted into 'master_hash_sha256.bin'!

[!] Total Time Elapsed: 42.05347776412964 seconds
```

Step 4: Decrypting Master Hash File

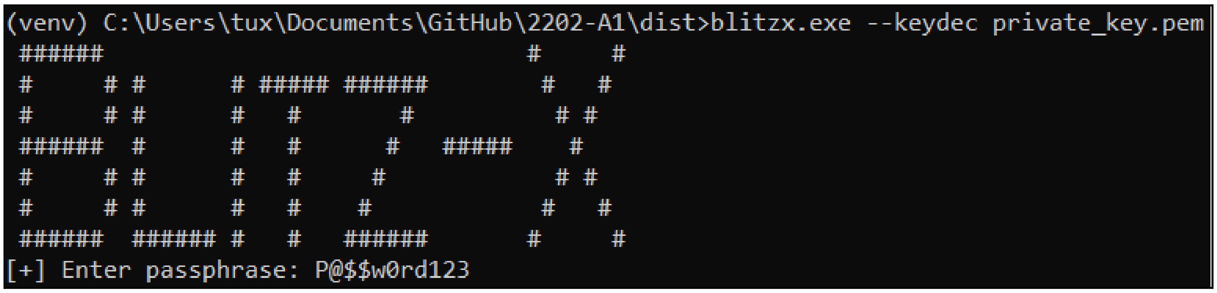
The notable files generated by Blitz-X comprises of the following:



data	10/22/2021 1:07 AM	File folder	
HTMLReport	10/22/2021 1:07 AM	File folder	
plugins	10/22/2021 12:45 AM	File folder	
aes.bin	10/22/2021 1:07 AM	BIN File	1 KB
blitzx.exe	10/22/2021 12:45 AM	Application	16,185 KB
keywords.txt	10/22/2021 12:45 AM	Text Document	1 KB
master_hash.bin	10/22/2021 1:07 AM	BIN File	24 KB
master_hash_sha256.bin	10/22/2021 1:07 AM	BIN File	1 KB
private_key.pem	10/22/2021 1:06 AM	PEM File	2 KB
public_key.pem	10/22/2021 1:06 AM	PEM File	1 KB

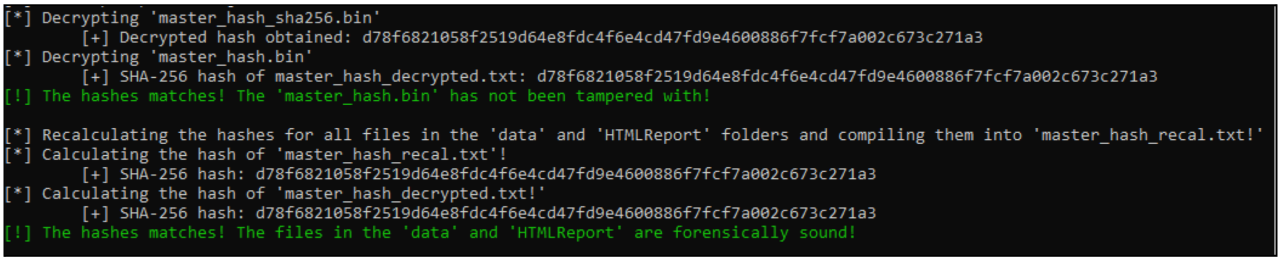
File / Folder Name	Description
data	A folder that contains the data artefacts extracted by Blitz-X in json format.
HTMLReport	A folder that contains a HTML-representation of the data in the “data” folder.
aes.bin	A random 256-bit symmetric key that is used to encrypt the master hash file using AES.
master_hash.bin	The encrypted “master_hash.txt”, a .txt file which contains all of the hashes of all the files in the “data” folder as well as all of the files in the “HTMLReport” folder.
master_hash_sha256.bin	An encrypted .txt file that contains the SHA-256 hash of “master_hash.txt” before it gets encrypted.

Blitz-X has an inbuilt function that decrypts the encrypted master hash file. The user will first input their private key file and provide the credentials to decrypt it.



```
(venv) C:\Users\tux\Documents\GitHub\2202-A1\dist>blitzx.exe --keydec private_key.pem
 ######                              #     #
 #     # #      # ##### ######        #   #
 #     # #      #   #       #          # #
 ######  #      #   #      #   #####    #
 #     # #      #   #     #            # #
 #     # #      #   #    #            #   #
 ######  ###### #   #   ######       #     #
[+] Enter passphrase: P@$$w0rd123
```

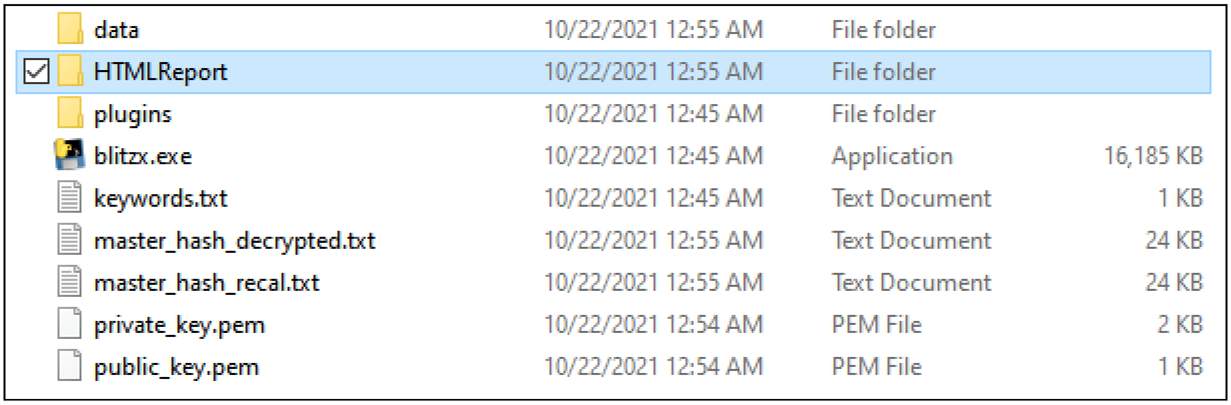
The user will be notified if the master hash file has been tampered with while it was in the chain of custody. If the file has not been tampered with, the user can then verify the checksums of the data files to determine if they are forensically sound.



```
[*] Decrypting 'master_hash_sha256.bin'
        [+] Decrypted hash obtained: d78f6821058f2519d64e8fdc4f6e4cd47fd9e4600886f7fcf7a002c673c271a3
[*] Decrypting 'master_hash.bin'
        [+] SHA-256 hash of master_hash_decrypted.txt: d78f6821058f2519d64e8fdc4f6e4cd47fd9e4600886f7fcf7a002c673c271a3
[!] The hashes matches! The 'master_hash.bin' has not been tampered with!

[*] Recalculating the hashes for all files in the 'data' and 'HTMLReport' folders and compiling them into 'master_hash_recal.txt!'
[*] Calculating the hash of 'master_hash_recal.txt'!
        [+] SHA-256 hash: d78f6821058f2519d64e8fdc4f6e4cd47fd9e4600886f7fcf7a002c673c271a3
[*] Calculating the hash of 'master_hash_decrypted.txt!'
        [+] SHA-256 hash: d78f6821058f2519d64e8fdc4f6e4cd47fd9e4600886f7fcf7a002c673c271a3
[!] The hashes matches! The files in the 'data' and 'HTMLReport' are forensically sound!
```

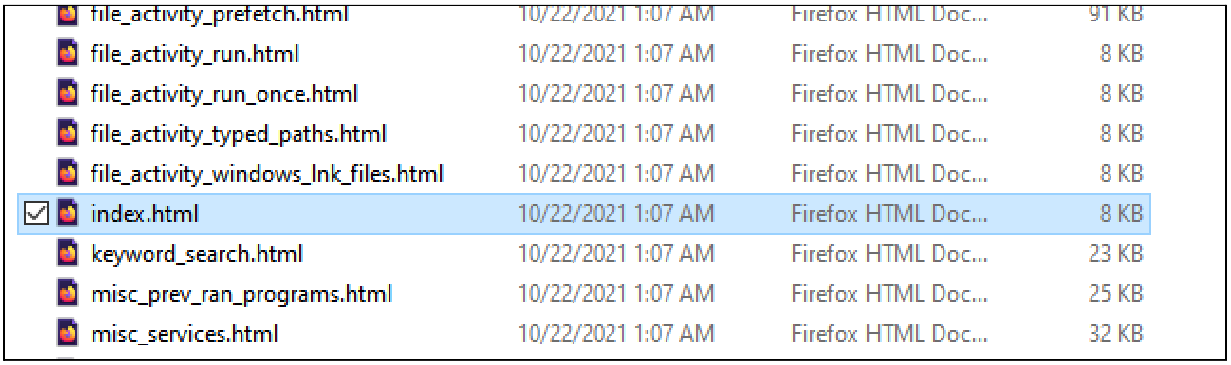
Step 5: Navigating the HTML Report

Once it is determined that the data extracted are still forensically sound, the user can then view all the extracted data. The data extracted are converted into a presentable form and are stored in the “HTMLReport” folder.

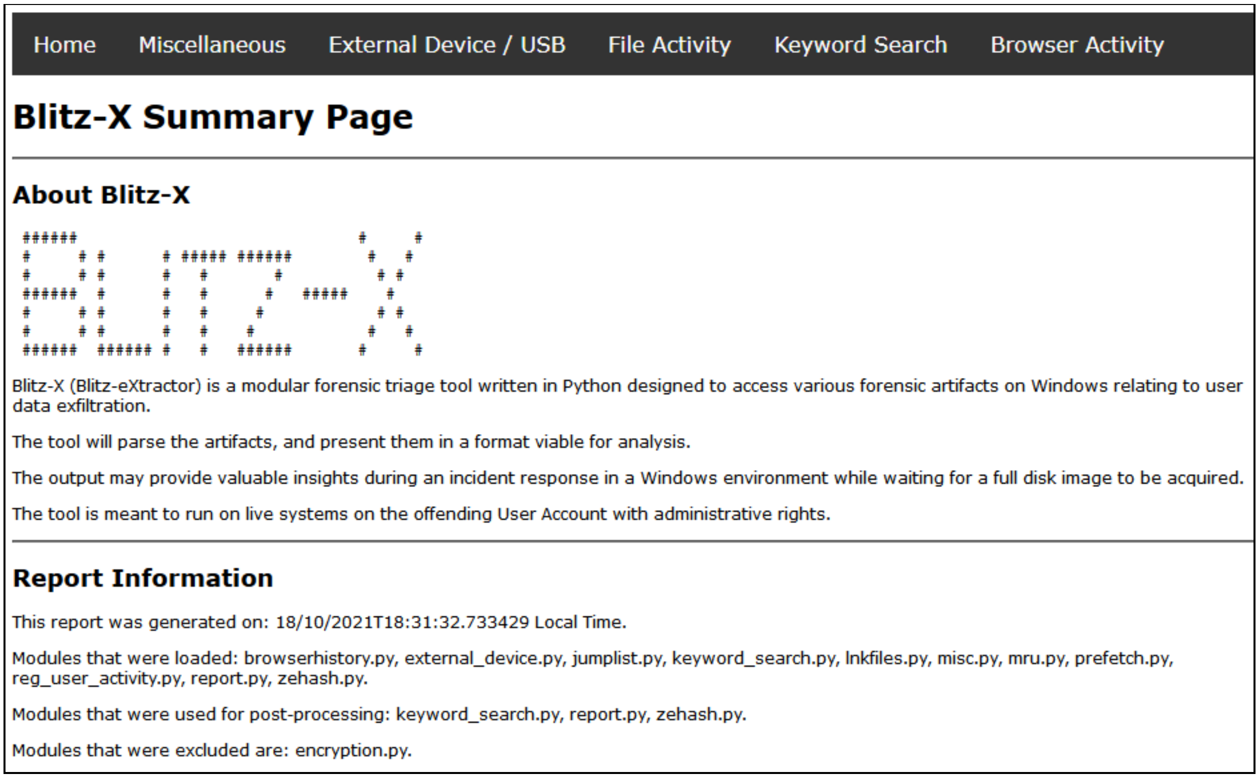


data	10/22/2021 12:55 AM	File folder	
☑ HTMLReport	10/22/2021 12:55 AM	File folder	
plugins	10/22/2021 12:45 AM	File folder	
blitzx.exe	10/22/2021 12:45 AM	Application	16,185 KB
keywords.txt	10/22/2021 12:45 AM	Text Document	1 KB
master_hash_decrypted.txt	10/22/2021 12:55 AM	Text Document	24 KB
master_hash_recal.txt	10/22/2021 12:55 AM	Text Document	24 KB
private_key.pem	10/22/2021 12:54 AM	PEM File	2 KB
public_key.pem	10/22/2021 12:54 AM	PEM File	1 KB

A summary page will be generated for the HTML report named “index.html”. The summary page would contain information such as the date and time of when the report was generated, along with modules that were loaded, excluded and used for post-processing.



file_activity_prefetch.html	10/22/2021 1:07 AM	Firefox HTML Doc...	91 KB
file_activity_run.html	10/22/2021 1:07 AM	Firefox HTML Doc...	8 KB
file_activity_run_once.html	10/22/2021 1:07 AM	Firefox HTML Doc...	8 KB
file_activity_typed_paths.html	10/22/2021 1:07 AM	Firefox HTML Doc...	8 KB
file_activity_windows_lnk_files.html	10/22/2021 1:07 AM	Firefox HTML Doc...	8 KB
☑ index.html	10/22/2021 1:07 AM	Firefox HTML Doc...	8 KB
keyword_search.html	10/22/2021 1:07 AM	Firefox HTML Doc...	23 KB
misc_prev_ran_programs.html	10/22/2021 1:07 AM	Firefox HTML Doc...	25 KB
misc_services.html	10/22/2021 1:07 AM	Firefox HTML Doc...	32 KB



Blitz-X Summary Page

About Blitz-X

```
######                              #     #
#     # #      # ##### ######        #   #
#     # #      #   #       #          # #
######  #      #   #      #   #####    #
#     # #      #   #     #            # #
#     # #      #   #    #            #   #
######  ###### #   #   ######       #     #
```

Blitz-X (Blitz-eXtractor) is a modular forensic triage tool written in Python designed to access various forensic artifacts on Windows relating to user data exfiltration.

The tool will parse the artifacts, and present them in a format viable for analysis.

The output may provide valuable insights during an incident response in a Windows environment while waiting for a full disk image to be acquired.

The tool is meant to run on live systems on the offending User Account with administrative rights.

Report Information

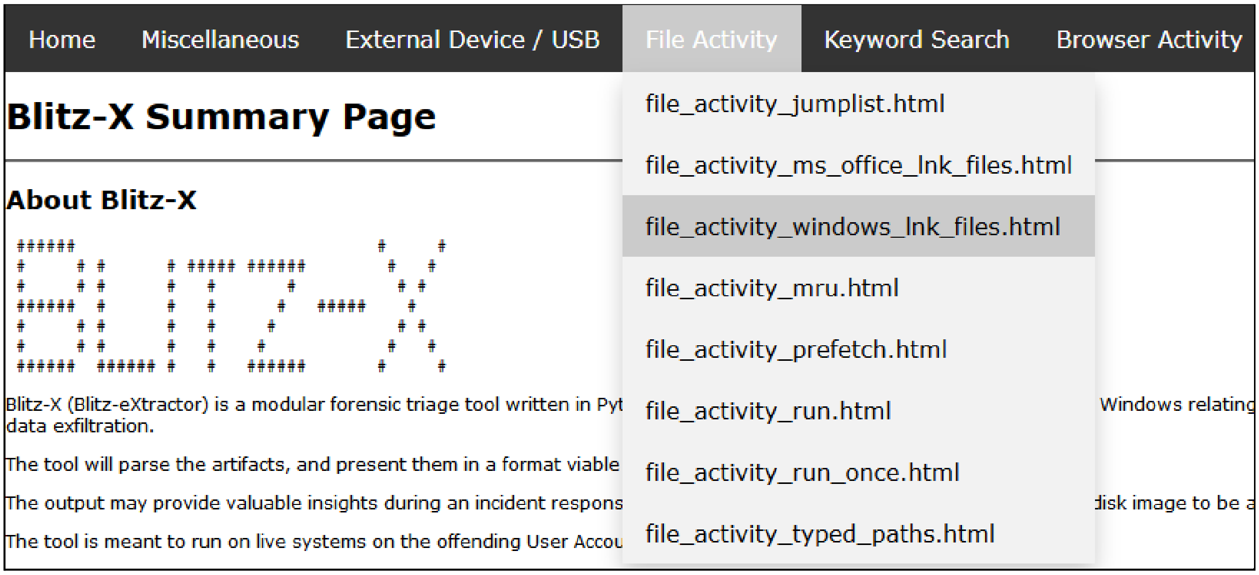
This report was generated on: 18/10/2021T18:31:32.733429 Local Time.

Modules that were loaded: browserhistory.py, external_device.py, jumplist.py, keyword_search.py, lnkfiles.py, misc.py, mru.py, prefetch.py, reg_user_activity.py, report.py, zehash.py.

Modules that were used for post-processing: keyword_search.py, report.py, zehash.py.

Modules that were excluded are: encryption.py.

A navigation bar will also be generated in all the HTML pages for ease of navigation. The top-level navigation bar reflects the core categories of the tool. A dropbox will appear when the categories are hovered over, allowing the user to access and read the different data extracted by the different modules.


Home
Miscellaneous
External Device / USB
File Activity
Keyword Search
Browser Activity
file_activity_jumplist.html
file_activity_ms_office_lnk_files.html
file_activity_windows_lnk_files.html
file_activity_mru.html
file_activity_prefetch.html
file_activity_run.html
file_activity_run_once.html
file_activity_typed_paths.html

Blitz-X Summary Page

About Blitz-X

```
######                              #     #
#     # #      # ##### ######        #   #
#     # #      #   #       #          # #
######  #      #   #      #   #####    #
#     # #      #   #     #            # #
#     # #      #   #    #            #   #
######  ###### #   #   ######       #     #
```

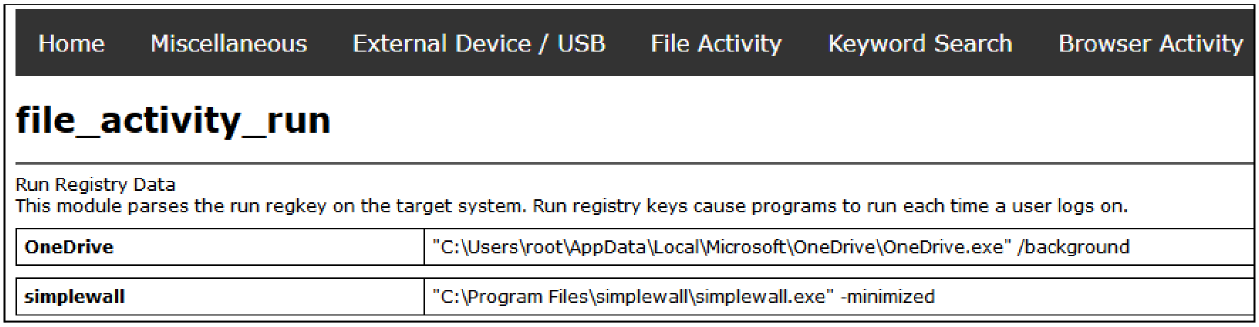
Blitz-X (Blitz-eXtractor) is a modular forensic triage tool written in Pyt Windows relating data exfiltration.

The tool will parse the artifacts, and present them in a format viable

The output may provide valuable insights during an incident respons disk image to be a

The tool is meant to run on live systems on the offending User Accou

To read a HTML file, the file name will be the header of the web page. There will also be a title of the module followed by the description. The data will be presented in HTML tables. In the example below, the data extracted from the registry shows the programs that will run when the user logs onto the computer.



file_activity_run

Run Registry Data
This module parses the run regkey on the target system. Run registry keys cause programs to run each time a user logs on.

OneDrive	"C:\Users\root\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
simplewall	"C:\Program Files\simplewall\simplewall.exe" -minimized

IV. Included Modules

Blitz-X ships with a plethora of core and supporting modules by default. They comprises of:

Core Modules

Module Name	Description
encryption.py	This module ensures the integrity of the data produced by the “zehash.py” module, especially during transit. The master hash file itself would be hashed with SHA-256. The hash would be output into a text file, and it is then encrypted with the user’s public key file. The master hash file is then encrypted with a randomly generated AES-256 bit key, and this key is then exported and encrypted with the user’s public key. This ensures that the data will not be able to be easily tampered with. a private key must be provided to the tool using the “--keydec” argument to decrypt the master hash file. If the correct key is provided, the files will be decrypted, and then the tool will compare the SHA-256 hashes and output the result accordingly. If the hashes match, it shows that the master hash file has not been tampered with, and all the hashes contained in the master hash file are sound. The files that are associated with the hashes in the master hash file are re-hashed, and the recalculated master hash file is created. The recalculated master hash file is then hashed with SHA-256, and its hash is compared with the SHA-256 of the earlier decrypted master hash file.
keyword_search.py	This module parses the “data” folder and does a recursive keyword search for all the keywords specified in the keyword list provided as an argument on all the .json files.
report.py	This module will generate a readable report in HTML format from the JSON files containing the data extracted from the target machine. The HTML report will have a summary page containing details such as the report information. It would also generate a proper navigation bar for the user to navigate around the report easily. There are categories defined by the module; these are the core categories that the tool uses:

	- Miscellaneous - External Device / USB - File Activity - Keyword Search - Browser Activity - Other Plugins If a category does not exist, the module will not generate that category in the navigation bar. The module utilises naming conventions to determine which category that a generated report should go to. Files that do not use the naming conventions would go into the “Other Plugins” category.
zehash.py	This module enables a chain of custody to be practised by investigators whose objective is to procure forensically sound evidence. This module will hash all files produced by the tool, such as the data files as well as the report. Multiple checksums are supported for backward compatibility purposes: - MD5 - SHA-1 - SHA-256 - SHA-512 After hashing the appropriate files, the hashes will be dumped into a text file named “master_hash.txt”.

Supporting Modules

Module Name	Description
browser_history.py	This module attempts to obtain data from the three mainstream browsers: - Google Chrome - Mozilla Firefox - Microsoft Edge The data it attempts to obtain are history, cookies, and bookmarks. If the target browser is not installed, it will be skipped. Data obtained will be dumped into a JSON file.

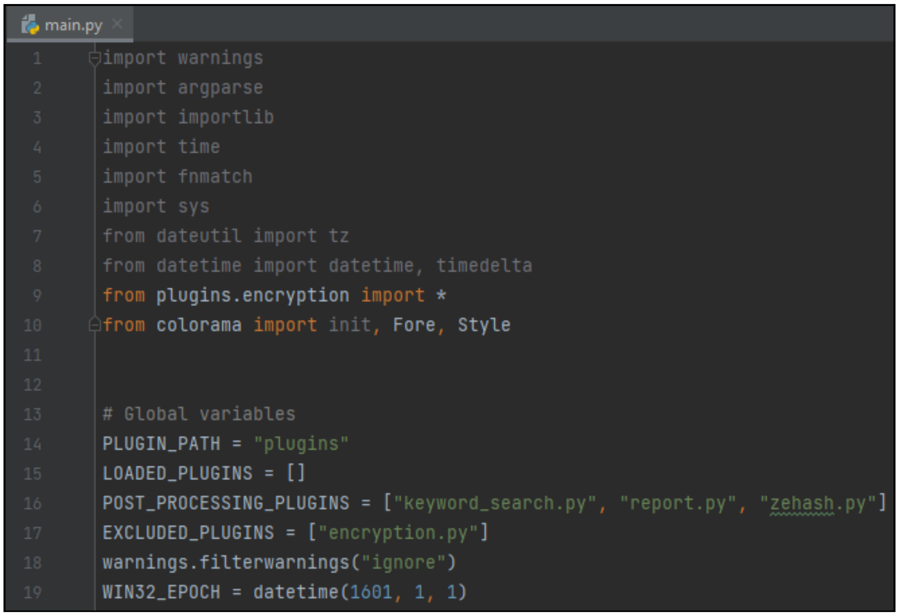
external_device.py	This module attempts to obtain data related to external devices such as USB usage. From the registry, it attempts to get the USB devices plugged into the machine, the last drive letter of the USB device when it was plugged into the machine, the volume serial number and it also enables finding the user that last used the unique USB device. From the system log file, it attempts to list all plug and play driver install logs. From the plug and play log files, it can get the date and time that the device is first plugged in. Data obtained will be dumped into a JSON file.
jumplist.py	This module attempts to obtain and parse jumplist files on the target system. Jump lists are artefacts created by Windows or applications to quickly access recently opened files or folders. They contain information about recently accessed applications and files. Data obtained will be dumped into a JSON file.
lnkfiles.py	This module attempts to obtain and parse lnk files on the target system. Lnk files are shortcut files that link to an application or file, which are commonly found on a user's Desktop. It can also be found throughout Windows with an extension of ".lnk". Lnk files can be used to prove the execution of a program, the opening of a document or malicious code startup. Data obtained will be dumped into a JSON file.
misc.py	This module attempts to obtain miscellaneous data from the target machine registry. It aims to give extra information to aid in investigations potentially. These data are: ● Previously ran programs ● Services ● Startup Applications ● System Environment ● Windows Version Data obtained will be dumped into a JSON file.
mru.py	This module attempts to obtain and parse the most recently used (MRU) entries on the target system. The MRU list is a Windows-based application that includes a registry of recently opened web pages, documents, files, images, and other applications. Data obtained will be dumped into a JSON file.
prefetch.py	This module attempts to obtain and parse prefetch files on the target system. Prefetch files are created when an executable program is run from a particular location for the very first time. They can be used to

	determine what was running and when. Data obtained will be dumped into a JSON file.
reg_user_activity.py	This module attempts to obtain and parse user related activity that is stored in the registry. The appropriate keys are: ● SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce ● SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run ● SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\TypedPaths The data stored within the Run and RunOnce keys can tell which programs run at startup. The TypePaths keys can record the last 25 directories the user manually typed into the file explorer path bar, proving what file or application the user was trying to search or open.

V. Plugin Architecture

The tool is designed with a modular architecture in mind, where its functionalities come in the form of plugins. Each plugin follows a specific boilerplate, which will be further elaborated on later in section VI. The core principle of designing such an architecture is that non-essential modules can work without depending on each other's codebase. For example, removing a module that extracts Windows jumplist files will not cause the tool to throw an error and exit. Removing that module will not cause the report generation or any other modules to stop working.

In the screenshot below, the tool has global variables that essentially categorises modules into three main states.

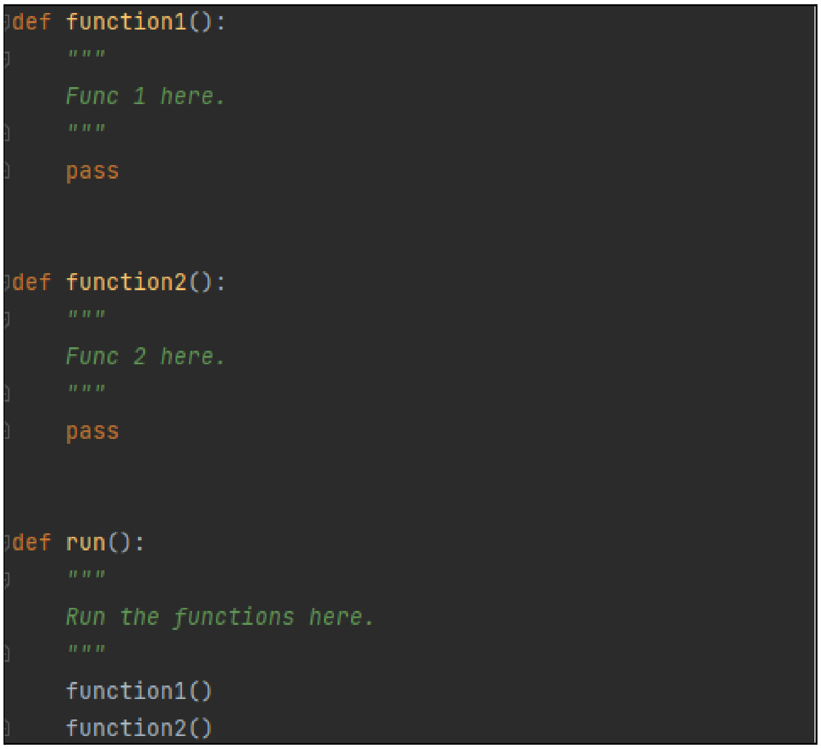


```python
import warnings
import argparse
import importlib
import time
import fnmatch
import sys
from dateutil import tz
from datetime import datetime, timedelta
from plugins.encryption import *
from colorama import init, Fore, Style


# Global variables
PLUGIN_PATH = "plugins"
LOADED_PLUGINS = []
POST_PROCESSING_PLUGINS = ["keyword_search.py", "report.py", "zehash.py"]
EXCLUDED_PLUGINS = ["encryption.py"]
warnings.filterwarnings("ignore")
WIN32_EPOCH = datetime(1601, 1, 1)
```

Global Variables	Description
LOADED_PLUGINS	Plugins that extract forensic artefacts.
POST_PROCESSING_PLUGINS	Plugins that do not perform any forensic artefacts extraction functions but do processing on the extracted data itself.
EXCLUDED_PLUGINS	The user is able to exclude plugins at their own risk, which essentially prevents the plugin from running.

VI. Plugin Boilerplate

An example boilerplate of writing a module for the tool is shown in the screenshot below.

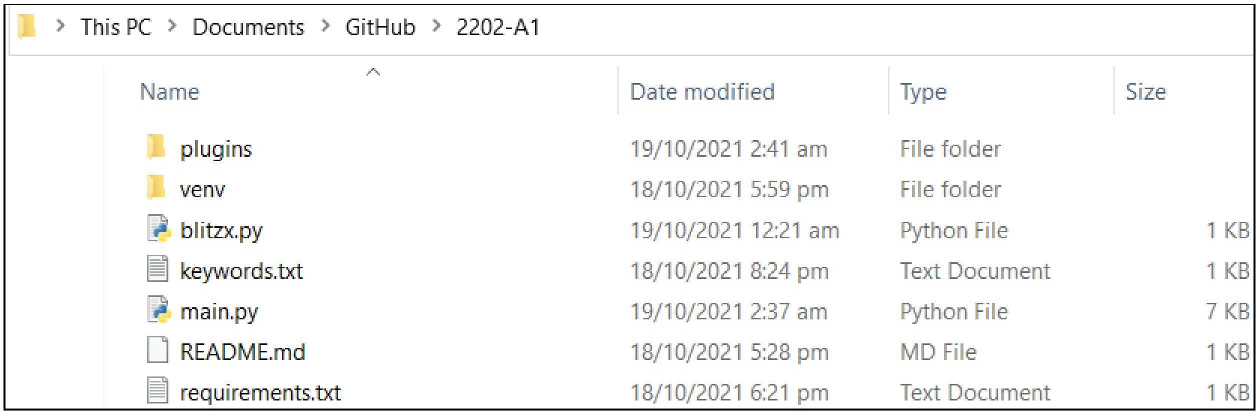


```
def function1():
    """
    Func 1 here.
    """
    pass


def function2():
    """
    Func 2 here.
    """
    pass


def run():
    """
    Run the functions here.
    """
    function1()
    function2()
```

The “function1” and “function2” placeholders are examples of functions that the user would want the module to perform internally. The only prerequisite for the module to be executed by Blitz-X is to have a function named “run”. The “run” function will contain whatever code the user wants the module to execute.

VII. Addition of Custom Modules

The source code of the tool can be compiled using PyInstaller. But before that can be done, if new custom plugins are introduced into the tool, some prerequisite steps need to be achieved before it can be compiled. The uncompiled source is as shown below.

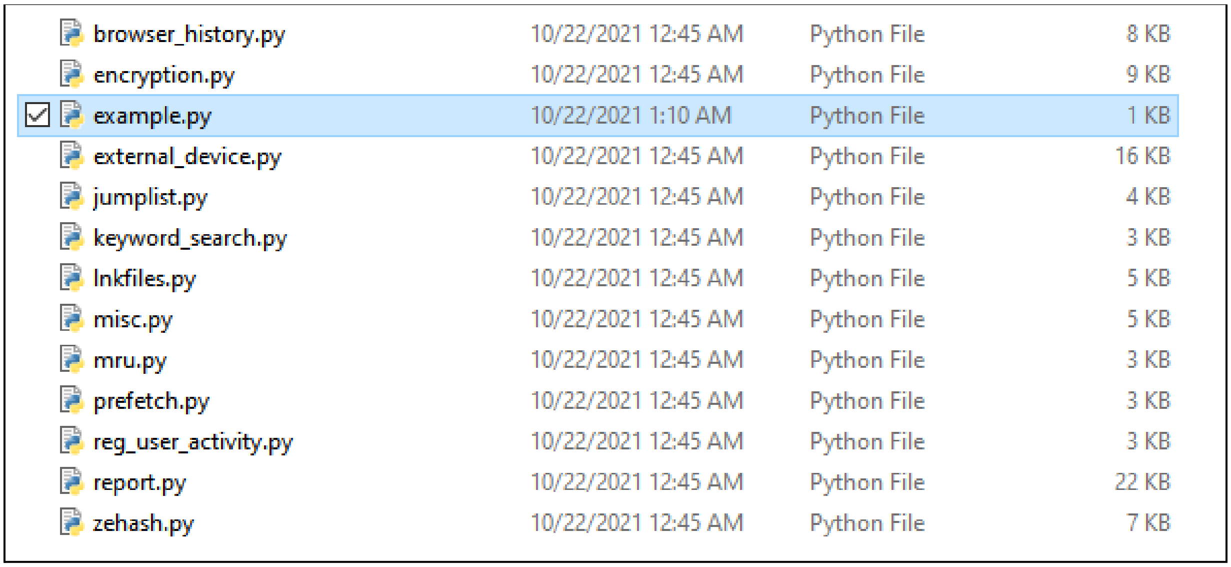


This PC > Documents > GitHub > 2202-A1

Name	Date modified	Type	Size
plugins	19/10/2021 2:41 am	File folder	
venv	18/10/2021 5:59 pm	File folder	
blitzx.py	19/10/2021 12:21 am	Python File	1 KB
keywords.txt	18/10/2021 8:24 pm	Text Document	1 KB
main.py	19/10/2021 2:37 am	Python File	7 KB
README.md	18/10/2021 5:28 pm	MD File	1 KB
requirements.txt	18/10/2021 6:21 pm	Text Document	1 KB

Step 1: Copy prospective module into Blitz-X's "plugins" folder

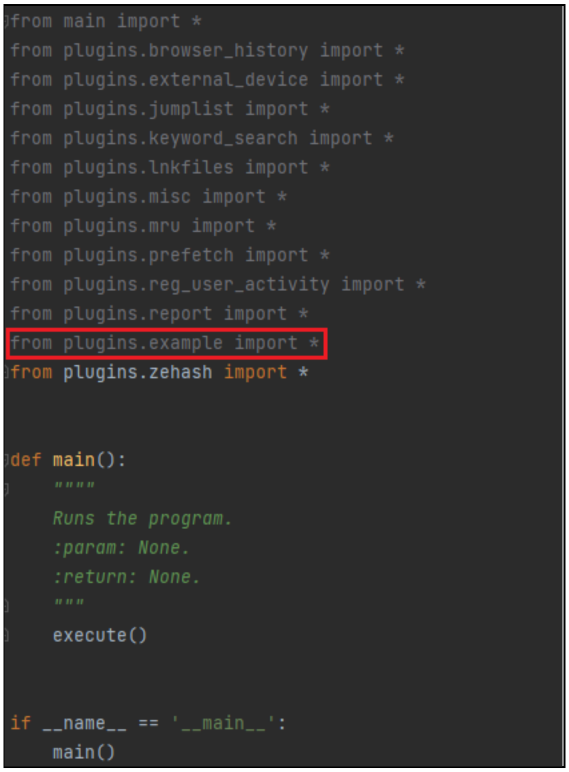
The prospective module shall be placed into the plugins folder of the uncompiled source code. A module named "example.py" is placed into the plugins folder where all other default modules reside in the screenshot below.



	browser_history.py	10/22/2021 12:45 AM	Python File	8 KB
	encryption.py	10/22/2021 12:45 AM	Python File	9 KB
☑	example.py	10/22/2021 1:10 AM	Python File	1 KB
	external_device.py	10/22/2021 12:45 AM	Python File	16 KB
	jumplist.py	10/22/2021 12:45 AM	Python File	4 KB
	keyword_search.py	10/22/2021 12:45 AM	Python File	3 KB
	lnkfiles.py	10/22/2021 12:45 AM	Python File	5 KB
	misc.py	10/22/2021 12:45 AM	Python File	5 KB
	mru.py	10/22/2021 12:45 AM	Python File	3 KB
	prefetch.py	10/22/2021 12:45 AM	Python File	3 KB
	reg_user_activity.py	10/22/2021 12:45 AM	Python File	3 KB
	report.py	10/22/2021 12:45 AM	Python File	22 KB
	zehash.py	10/22/2021 12:45 AM	Python File	7 KB

Step 2: Modify “blitzx.py” and Import Prospective Module

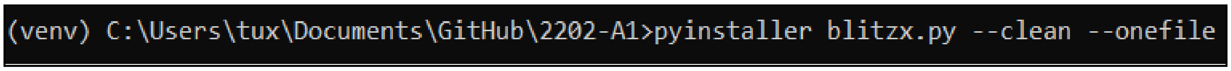
After doing so, an entry in “blitzx.py”, which resides in the project root directory, shall be inserted. In the screenshot below, an entry has been made named “from plugins.example import *”. This is done so that PyInstaller is able to “see” the module and its dependencies so that it can be included during the compilation.



```
from main import *
from plugins.browser_history import *
from plugins.external_device import *
from plugins.jumplist import *
from plugins.keyword_search import *
from plugins.lnkfiles import *
from plugins.misc import *
from plugins.mru import *
from plugins.prefetch import *
from plugins.reg_user_activity import *
from plugins.report import *
from plugins.example import *
from plugins.zehash import *


def main():
    """
    Runs the program.
    :param: None.
    :return: None.
    """
    execute()


if __name__ == '__main__':
    main()
```

Step 3: Compile Blitz-X’s Uncompiled Source Code using PyInstaller

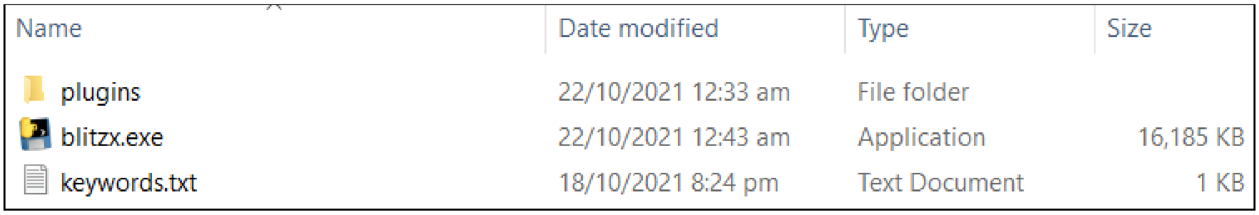
After all the preparation is done, the source code is ready to be compiled using PyInstaller. An example command is shown below:



```
(venv) C:\Users\tux\Documents\GitHub\2202-A1>pyinstaller blitzx.py --clean --onefile
```

Step 4: Copy the “plugins” Folder into the Compiled Folder

After compiling, copy the “plugins” folder into the tool’s compiled directory. Export the entire compiled directory onto a thumb drive, and Blitz-X is ready to be used.



Name	Date modified	Type	Size
plugins	22/10/2021 12:33 am	File folder	
blitzx.exe	22/10/2021 12:43 am	Application	16,185 KB
keywords.txt	18/10/2021 8:24 pm	Text Document	1 KB