Encoding, Hashing and Encryption

Written by Tyler Weiss

Exercise 1 - Encoding

Task 1-2

---

1. Encode the following message using base64 encoder.
2. Take the the encoded string and decode it using the base64 encoder.
   The output of echo is piped to the base64 program as input allowing us to encode the echoed string. Since the original message was produced by decoding the encoded string this proves that the encoder worked properly.

```
echo "You guys are AWESOME!" | base64
echo "WW91IGd1eXMgYXJlIEFXRVNPTUUhCg==" | base64 --decode
```

```
rec0nrat@demoserver1:~$ echo "You guys are AWESOME!" | base64
WW91IGd1eXMgYXJlIEFXRVNPTUUhCg==
rec0nrat@demoserver1:~$ echo "WW91IGd1eXMgYXJlIEFXRVNPTUUhCg==" | base64 --decode
You guys are AWESOME!
rec0nrat@demoserver1:~$
```

Task 3

---

1. Append the string 'This is evil naughty naughty malware' to a file called 'malware.txt'.
2. Cat the contents of 'malware.txt' to show that it contains the string.
3. Cat the contents of 'malware.txt' and redirect the output to the base64 encoder appending the encoded string to 'notmalwarenoreally.txt'.
4. Cat 'notmalwarenoreally.txt' to prove the contents of the file and show the encoded string.

5. Cat 'notmalwarenoreally.txt' and redirect the output to the base64 decoder as input. This series of actions proves that the encoded value in 'notmalwarenoreally.txt' is the base64 equivalent of the string 'This is evil naughty naughty malware' that is stored in 'malware.txt'.

```
rec0nrat@demoserver1:~$ echo This is evil naughty naughty malware > malware.txt
rec0nrat@demoserver1:~$ cat malware.txt
This is evil naughty naughty malware
rec0nrat@demoserver1:~$ cat malware.txt | base64 > notmalwarenoreally.txt
rec0nrat@demoserver1:~$ cat notmalwarenoreally.txt
VGhpcyBpcyBldmlsIG5hdWdodHkgbmF1Z2h0eSBtYWx3YXJlCg==
rec0nrat@demoserver1:~$ cat notmalwarenoreally.txt | base64 -d
This is evil naughty naughty malware
rec0nrat@demoserver1:~$
```

Task 4

1. Produce an md5 hash of 'malware.txt' using 'md5sum'
2. Produce an md5 hash of 'notmalwarenoreally.txt' using 'md5sum'
   Even though the contents of 'notmalwarenoreally.txt' is the base64 encoded version of the contents of 'malware.txt' the hashes are different because the contents of the files are different. This illustrates a method of evading anti-virus software by obfuscating the the file contents thus changing the digital signature of the original file.

```
rec0nrat@demoserver1:~$ md5sum malware.txt
de54f727fe1eb6d1d0ca9411db64024a  malware.txt
rec0nrat@demoserver1:~$ md5sum notmalwarenoreally.txt
6606d6223afdfe6c969d31197e9400e1  notmalwarenoreally.txt
rec0nrat@demoserver1:~$
```

Exercise 2 - Hashing

Task 1

1. Echo 'hello world' and redirect the output to 'file1.txt'
2. Echo 'hello world' and redirect the output to 'file2.txt'
3. Cat the contents of both files.

4. Compare the md5 hash of the files using 'md5sum'.

```
rec0nrat@demoserver1:~/HashEncrypt$ echo hello world > file1.tx
rec0nrat@demoserver1:~/HashEncrypt$ echo hello world > file2.tx
rec0nrat@demoserver1:~/HashEncrypt$ rm file*
rec0nrat@demoserver1:~/HashEncrypt$ echo hello world > file1.txt
rec0nrat@demoserver1:~/HashEncrypt$ echo hello world > file2.txt
rec0nrat@demoserver1:~/HashEncrypt$ cat file*.txt
hello world
hello world
rec0nrat@demoserver1:~/HashEncrypt$ md5sum file*.txt
6f5902ac237024bdd0c176cb93063dc4  file1.txt
6f5902ac237024bdd0c176cb93063dc4  file2.txt
rec0nrat@demoserver1:~/HashEncrypt$
```

Please note that the MD5 hash of the outputs are identical to each other. That means when you created the files, they were identical, so were their hashes. This proves the deterministic nature of using the same hash function, which in this case was md5.

Task 2

1. Echo 'hello world!' and redirect the output to 'file3.txt'
2. Cat 'file3.txt'.
3. Compare the md5 hash of the previous files with that of 'file3.txt' using 'md5sum'

```
rec0nrat@demoserver1:~/HashEncrypt$ echo hello world! > file3.txt
rec0nrat@demoserver1:~/HashEncrypt$ cat file3.txt
hello world!
rec0nrat@demoserver1:~/HashEncrypt$ md5sum file*.txt
6f5902ac237024bdd0c176cb93063dc4  file1.txt
6f5902ac237024bdd0c176cb93063dc4  file2.txt
c897d1410af8f2c74fba11b1db511e9e  file3.txt
rec0nrat@demoserver1:~/HashEncrypt$
```

Please note that by inputting different text into file3, that the hash is wildly different that the files in Task 1. This is due to the Avalanche effect in Hashing.

Task 3

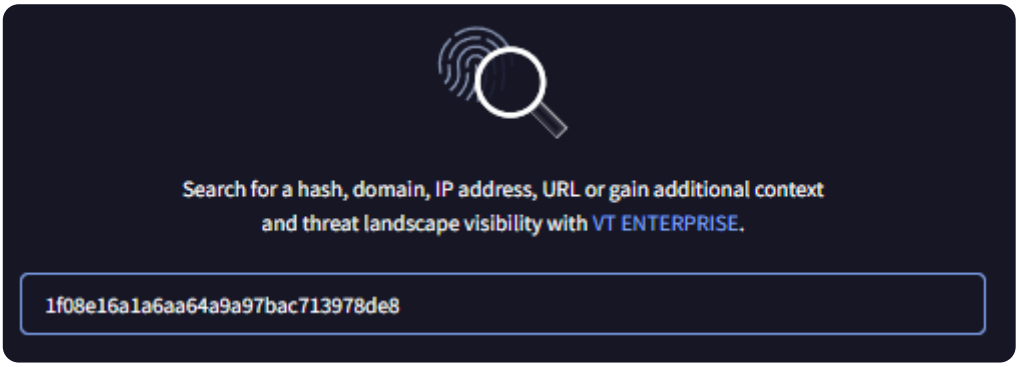
1. Run the following command to download 'catpicturess.jpg' using 'wget'.

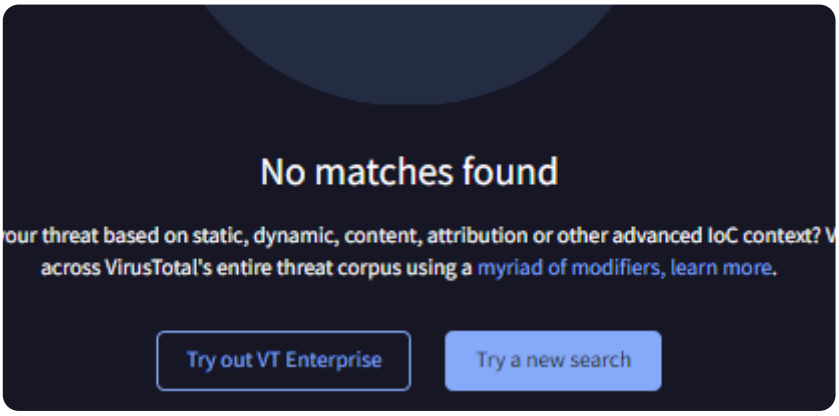
```
wget
https://github.com/ajay63/BlackTowerAcademy/blob/main/catpicturess.jpg
```

2. Get the md5 hash of 'catpicturess.jpg' using 'md5sum'.

```
rec0nrat@demoserver1:~/HashEncrypt$ wget https://github.com/ajay63/BlackTowerAcademy/blob/main/catpicturess.jpg
--2024-04-14 06:56:37--  https://github.com/ajay63/BlackTowerAcademy/blob/main/catpicturess.jpg
Resolving github.com (github.com)... 140.82.112.3
Connecting to github.com (github.com)|140.82.112.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: ‘catpicturess.jpg’

catpicturess.jpg                [ <=>                                   ] 144.48K  --.-KB/s    in 0.1s

2024-04-14 06:56:38 (979 KB/s) - ‘catpicturess.jpg’ saved [147943]

rec0nrat@demoserver1:~/HashEncrypt$ md5sum catpicturess.jpg
1f08e16a1a6aa64a9a97bac713978de8  catpicturess.jpg
rec0nrat@demoserver1:~/HashEncrypt$
```

3. Browse to virus total: https://www.virustotal.com/gui/home/search
4. Copy the hash above for catpicturess.jpg and paste it into the search field in virustotal.
5. No matches were found meaning that 'catpicturess.jpg' was not a virus.

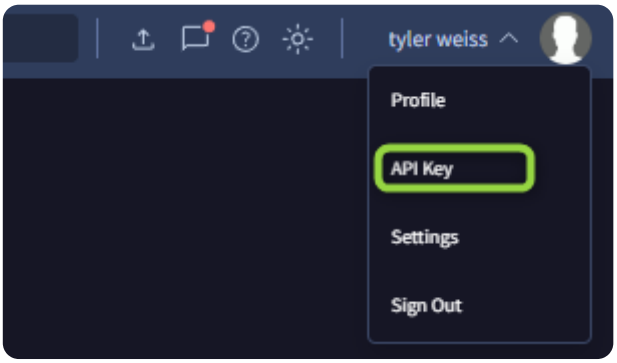


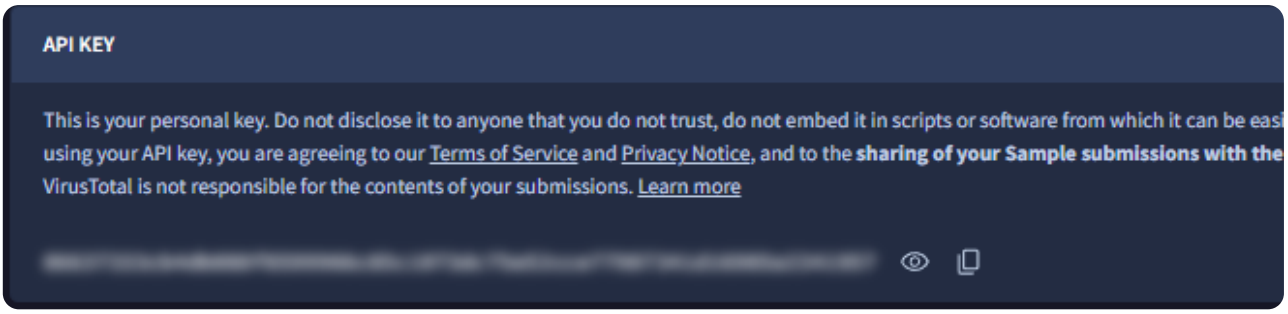


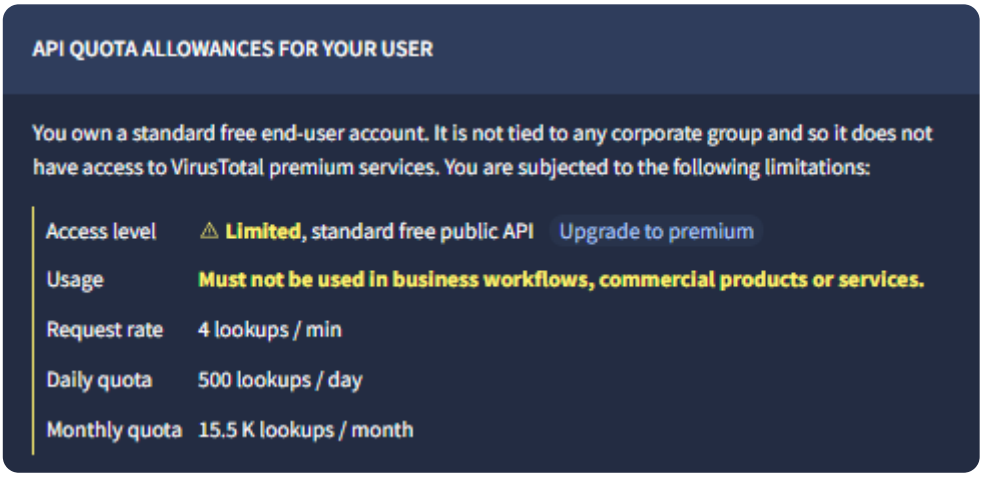
Task 4

1. Open a browser to https://www.virustotal.com.
2. Create an account.

3. Navigate to the API key.







4. We have to download the pre-compiled VirusTotal Application to the Linux server. Use the following command to download the zipped file.
5. List the directory to check that the file exists.

```
wget https://github.com/VirusTotal/vt-cli/releases/download/1.0.0/Linux64.zip
```

```
rec0nrat@demoserver1:~/HashEncrypt$ wget https://github.com/VirusTotal/vt-cli/releases/download/1.0.0/Linux64.zip
--2024-04-14 07:06:35--  https://github.com/VirusTotal/vt-cli/releases/download/1.0.0/Linux64.zip
Resolving github.com (github.com)... 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/133561480/bb8bbce9-0ce3-43
39d-124ec1ecb76f?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAVCODYLSA53PQK4ZA%2F20240414%2Fus-east-1%2F
Faws4_request&X-Amz-Date=20240414T070635Z&X-Amz-Expires=300&X-Amz-Signature=993b8fd2369bb4777e4675283f202bbb92f76
feecdd76e70d5d4a69ca29c&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=133561480&response-content-dispositi
ttachment%3B%20filename%3DLinux64.zip&response-content-type=application%2Foctet-stream [following]
--2024-04-14 07:06:35--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/133561480/b
ce9-0ce3-431d-839d-124ec1ecb76f?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAVCODYLSA53PQK4ZA%2F20240414
s-east-1%2Fs3%2Faws4_request&X-Amz-Date=20240414T070635Z&X-Amz-Expires=300&X-Amz-Signature=993b8fd2369bb4777e4675
202bbb92f763479feecdd76e70d5d4a69ca29c&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=133561480&response-co
t-disposition=attachment%3B%20filename%3DLinux64.zip&response-content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.109.133, 185.199.111.133, 185.
108.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7120233 (6.8M) [application/octet-stream]
Saving to: ‘Linux64.zip’

Linux64.zip                   100%[=============================================>]   6.79M  --.-KB/s    in 0.1s

2024-04-14 07:06:35 (56.2 MB/s) - ‘Linux64.zip’ saved [7120233/7120233]

rec0nrat@demoserver1:~/HashEncrypt$
```

```
rec0nrat@demoserver1:~/HashEncrypt$ ls
catpicturess.jpg  file1.txt  file2.txt  file3.txt  Linux64.zip  malware.txt  notmalwarenoreally.txt
rec0nrat@demoserver1:~/HashEncrypt$
```

6. Install 'unzip'.
7. Unzip the file 'Linux64.zip'
8. List the directory to prove the virustotal 'vt' executable exists.
9. Initialize the program by running './vt init'.
10. Input the API key from the virustotal website.

```
rec0nrat@demoserver1:~/HashEncrypt$ sudo apt install unzip -y
[sudo] password for rec0nrat:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
unzip is already the newest version (6.0-26ubuntu3.2).
0 upgraded, 0 newly installed, 0 to remove and 19 not upgraded.
rec0nrat@demoserver1:~/HashEncrypt$ unz
unzip     unzipsfx  unzstd
rec0nrat@demoserver1:~/HashEncrypt$ unzip Linux64.zip
Archive:  Linux64.zip
  inflating: vt
rec0nrat@demoserver1:~/HashEncrypt$ ls
catpicturess.jpg  file1.txt  file2.txt  file3.txt  Linux64.zip  malware.txt  notmalwarenoreally.txt  vt
```

```
rec0nrat@demoserver1:~/HashEncrypt$ ./vt init
VIRUSTOTAL

VirusTotal Command-Line Interface: Threat Intelligence at your fingertips.

Enter your API key: [redacted]
Your API key has been written to config file /home/rec0nrat/.vt.toml
rec0nrat@demoserver1:~/HashEncrypt$
```

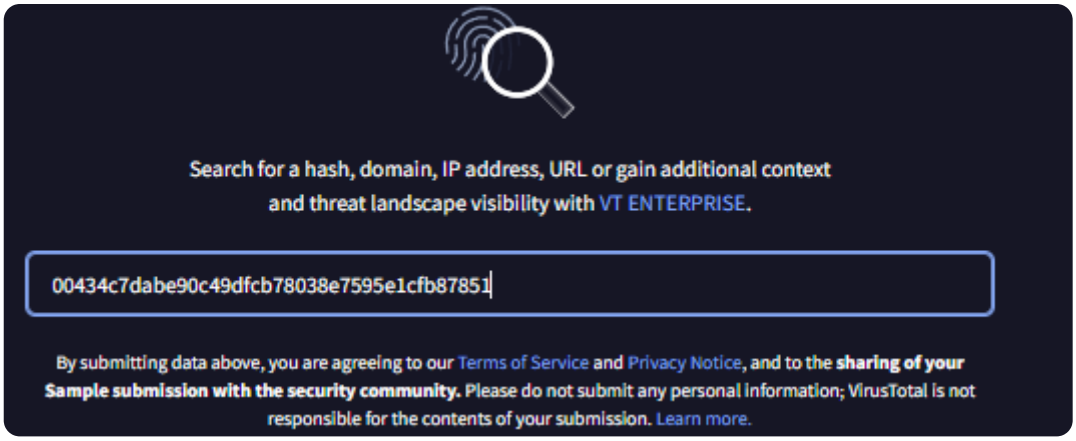
11. Get the md5 hash of 'catpicture.jpg'.
12. Check the hash to see if it is malware using the 'vt' program and the 'file' option.

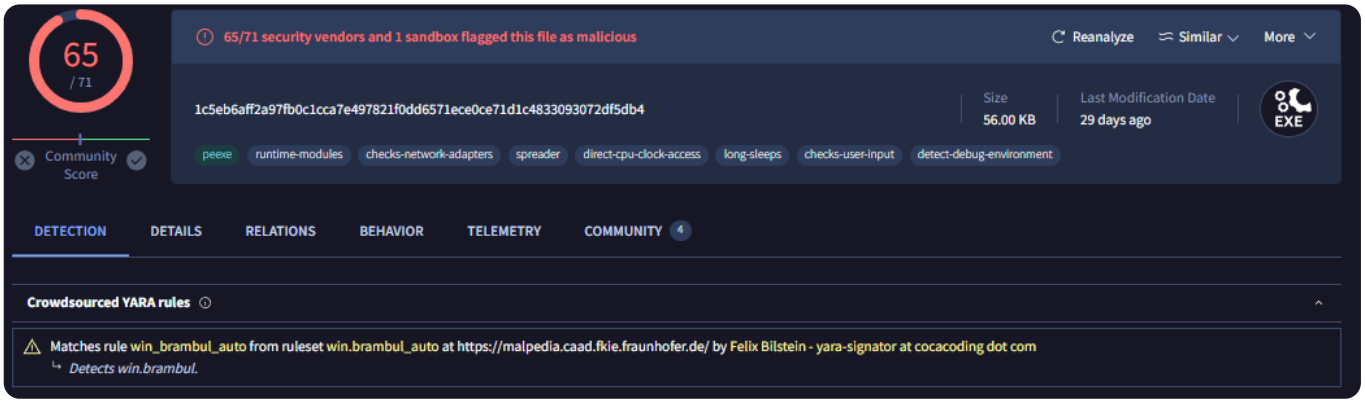
Since the searched returned 'File <hash> not found' we can assume it is not a virus and has no listing on virustotal.

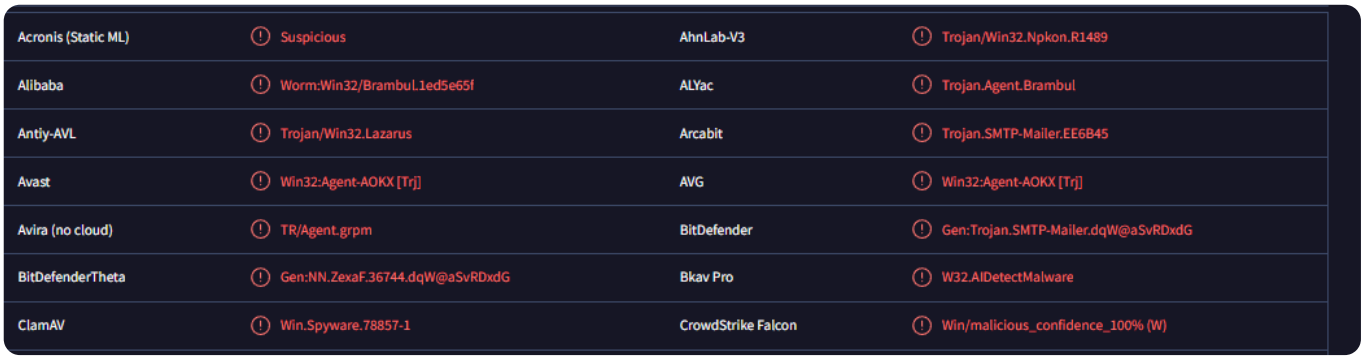
```
rec0nrat@demoserver1:~/HashEncrypt$ ./vt file $(md5sum catpicturess.jpg)
File "1f08e16a1a6aa64a9a97bac713978de8" not found
rec0nrat@demoserver1:~/HashEncrypt$
```

Let's use a known bad hash now.

13. Pull up our Virus Total Web page, and paste in the following known bad hash: 00434c7dabe90c49dfcb78038e7595e1cfb87851







14. Check the same file hash using the 'vt' program.

```
rec0nrat@demoserver1:~/HashEncrypt$ ./vt file 00434c7dabe90c49dfcb78038e7595e1cfb87851
- _id: "1c5eb6aff2a97fb0c1cca7e497821f0dd6571ece0ce71d1c4833093072df5db4"
  _type: "file"
  authentihash: "ff3530eb0fcd6f36777a9dd5c2fa211b8841ced09736c673645ee803db73eb7e"
  creation_date: 1255524354  # 2009-10-14 12:45:54 +0000 UTC
  crowdsourced_ids_results:
  - alert_context:
    - src_ip: "93.220.189.23"
    alert_severity: "medium"
    rule_category: "attempted-recon"
    rule_id: "116:441"
    rule_msg: "(icmp4) ICMP destination unreachable communication administratively prohibited"
    rule_raw: "alert ( gid:116; sid:441; rev:2; msg:\"(icmp4) ICMP destination unreachable communication administrati
vely prohibited\"; metadata: rule-type decode; classtype:attempted-recon;)"
    rule_source: "Snort registered user ruleset"
    rule_url: "https://www.snort.org/downloads/#rule-downloads"
  - alert_context:
    - src_ip: "173.44.201.217"
    alert_severity: "medium"
```

15. Redirect the output to a file named 'ebil.txt'.
16. Use one of the Linux text readers to read 'ebil.txt'

```
rec0nrat@demoserver1:~/HashEncrypt$ ./vt file 00434c7dabe90c49dfcb78038e7595e1cfb87851 > ebil.txt
rec0nrat@demoserver1:~/HashEncrypt$ ls
catpicturess.jpg  ebil.txt  file1.txt  file2.txt  file3.txt  Linux64.zip  malware.txt  notmalwarenoreally.txt  vt
rec0nrat@demoserver1:~/HashEncrypt$ less ebil.txt
```

```
1ece0ce71d1c4833093072df5db4"
  _type: "file"
  authentihash: "ff3530eb0fcd6f36777a9dd5c2fa211b8841ced09736c673645ee803db73eb7e"
  creation_date: 1255524354  # 2009-10-14 12:45:54 +0000 UTC
  crowdsourced_ids_results:
  - alert_context:
    - src_ip: "93.220.189.23"
    alert_severity: "medium"
    rule_category: "attempted-recon"
    rule_id: "116:441"
    rule_msg: "(icmp4) ICMP destination unreachable communication administratively prohibited"
    rule_raw: "alert ( gid:116; sid:441; rev:2; msg:\"(icmp4) ICMP destination unreachable communication administrati
vely prohibited\"; metadata: rule-type decode; classtype:attempted-recon;)"
    rule_source: "Snort registered user ruleset"
    rule_url: "https://www.snort.org/downloads/#rule-downloads"
  - alert_context:
    - src_ip: "173.44.201.217"
    alert_severity: "medium"
```

Now by using the command line interface text connection to virus total, I can bring in automatable log enrichment and information that can be used with other code to provide good threat intelligence and way to populate cybersecurity systems.

Exercise 3 - Encryption

Encrypting and Decrypting Files with GPG.
Objective: Learn how to encrypt and decrypt files using GnuPG (GPG) on a Linux system.
Prerequisites

- Access to a Linux system
- Install GnuPG installed.
- Basic knowledge of navigating the Linux command line interface.

Task 1

Installation: Ensure that GnuPG is installed on your system. You can install it using the package manager of your Linux distribution. For example, on Ubuntu or Debian, you would use:

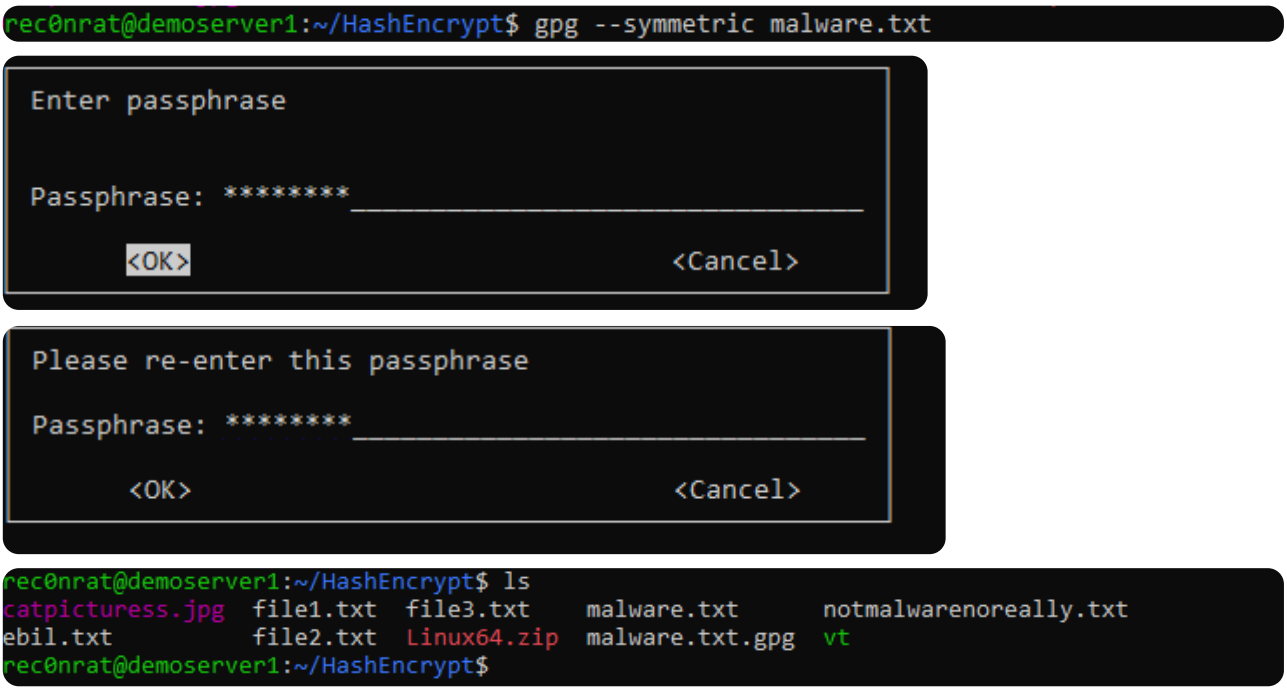
```
sudo apt install gnupg -y
```

Task 2

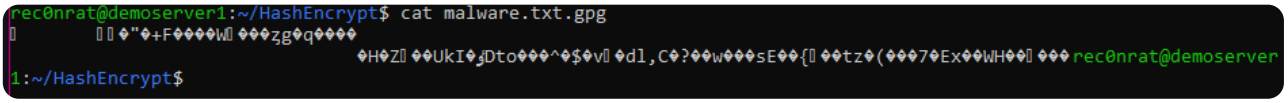
1. To encrypt the file 'malware.txt' symmetrically (using a passphrase), run:

```
gpg --symmetric malware.txt
```

2. GPG will prompt you to enter a passphrase. Choose a strong, memorable passphrase. Confirm the passphrase when prompted.
3. After encryption, a new file with the extension '.gpg' will be created (in this case, malware.txt.gpg).
4. You can see this new file by listing the contents of the directory with ls.



5. Attempt to view the encrypted file's contents using cat 'malware.txt.gpg'. You'll see that the output is garbled, indicating the content is encrypted.



Keep in mind if you are decrypting locally, during the encryption process earlier you should have seen:

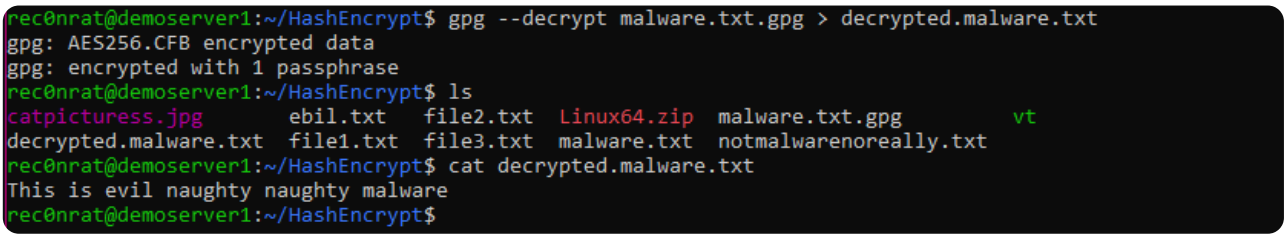
```
gpg: directory '/home/ajay/.gnupg' created
gpg: keybox '/home/ajay/.gnupg/pubring.kbx' created
```

This means there is a pubring, is a local file that is keeping your symmetric keys for you. When you go to decrypt locally, you won't be prompted to provide the password. If you provide it to another person, you will need to provide the symmetric private key.

6. To decrypt the 'malware.txt.gpg' file, run:

```
gpg --decrypt malware.txt.gpg > decrypted.malware.txt
```

7. View the contents of the decrypted file using 'cat decrypted.malware.txt'.
8. The output should match the original malware.txt file, in this case, displaying "This is evil naughty naughty malware"



Remember, the security of encrypted files depends on the strength of the passphrase and the security of the system used for encryption and decryption.