John-Peter Krause
u23533529
Cos 330 Prac 3
github repo:
https://github.com/johnpeterprogramming/COS330/tree/master/Prac3

Task 1

For my benign apps I used the following well-known benign apps: vlc, putty, notepad++, firefox and gimp.

Firefox:

```
johna  master  …  Prac3  benign  firefox  1  sha256sum Firefox\ Setup\ 144.0b4.exe
25c03c8b477e91d0d47d19a47278e7ae20917fa9c5d5d0380857e8b4bf77f536  Firefox Setup 144.0b4.exe
johna  master  …  Prac3  benign  firefox  cat SHA256SUMS
25c03c8b477e91d0d47d19a47278e7ae20917fa9c5d5d0380857e8b4bf77f536  win64/en-US/Firefox Setup 144.0b4.exe
```

VLC:

```
johna  master  …  Prac3  benign  vlc  sha256sum vlc-3.0.21-win64.exe
9742689a50e96ddc04d80ceff046b28da2beefd617be18166f8c5e715ec60c59  vlc-3.0.21-win64.exe
johna  master  …  Prac3  benign  vlc  cat checksum.sha256
9742689a50e96ddc04d80ceff046b28da2beefd617be18166f8c5e715ec60c59
johna  master  …  Prac3  benign  vlc
```

Putty:

```
johna  master  …  Prac3  benign  putty  gpg --import release-2023.asc
gpg: key 1993D21BCAD1AA77: 1 signature not checked due to a missing key
gpg: key 1993D21BCAD1AA77: public key "PuTTY Releases <putty@projects.tartarus.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2027-03-13
johna  master  …  Prac3  benign  putty  gpg --verify putty-64bit-0.83-installer.msi.gpg putty-64bit-0.83-installer.msi
gpg: Signature made Sat 08 Feb 2025 12:29:06 PM SAST
gpg:                using RSA key F412BA3AA30FDC0E77B4E3871993D21BCAD1AA77
gpg: Good signature from "PuTTY Releases <putty@projects.tartarus.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: F412 BA3A A30F DC0E 77B4  E387 1993 D21B CAD1 AA77
```

Notepad++:

```
johna  master  …  Prac3  benign  notepad-plus-plus  gpg --fingerprint 6C429F1D8D84F46E
pub   rsa4096 2019-03-11 [SC] [expires: 2027-03-13]
      14BC E436 2749 B2B5 1F8C  7122 6C42 9F1D 8D84 F46E
uid           [ultimate] Notepad++ <don.h@free.fr>

johna  master  …  Prac3  benign  notepad-plus-plus  gpg --verify npp.8.8.5.Installer.x64.exe.sig npp.8.8.5.Installer.x64.exe
gpg: Signature made Thu 14 Aug 2025 01:27:24 AM SAST
gpg:                using RSA key 14BCE4362749B2B51F8C71226C429F1D8D84F46E
gpg: Good signature from "Notepad++ <don.h@free.fr>" [ultimate]
johna  master  …  Prac3  benign  notepad-plus-plus
```

Gimp:

```
→  gimp git:(master) ✗ sha256sum gimp-3.0.4-setup.exe                    [INSERT]
385e36fe577cbdbfc71ba79d6c046d6f4eaabc01effd7f067bf15fd98410b2a1  gimp-3.0.4-setup.exe
```

Security

The SHA256 hash sum for `gimp-3.0.4-setup.exe` is:

`385e36fe577cbdbfc71ba79d6c046d6f4eaabc01effd7f067bf15fd98410b2a1`

Check it on VirusTotal: gimp-3.0.4-setup.exe

For malware I used malwareBazaar, it ended up being a lot easier than using virusshare.com and theZoo, because I could filter by exe and didn't have to get special access or use an api to grab files. The malware I got didn’t have proper names, so I will refer to them by their numbers.
What makes the integrity even more obvious is that the executables are named with their hashes(something malwareBazaar does) so it’s easy to see that the hashes match as when comparing. Additionally I included a screenshot from what the site says the hash is from the download page.

Malware 1:

```
→  malware1 git:(master) ✗ sha256sum 7109c74b24a883dbd37cf5d23a11642ed056d876e51
20102ab860da498550e33.exe
7109c74b24a883dbd37cf5d23a11642ed056d876e5120102ab860da498550e33  7109c74b24a883
dbd37cf5d23a11642ed056d876e5120102ab860da498550e33.exe
```

This page let you download the following malware sample: **SHA256 7109c74b24a883dbd37cf5d23a11642ed056d876e5120102ab860da498550e33**

Malware 2:

```
→  malware2 git:(master) ✗ sha256sum aae142810c653716d5acd0c128bd05ed96c30861188
a09541ed16099e17de005.exe
aae142810c653716d5acd0c128bd05ed96c30861188a09541ed16099e17de005  aae142810c6537
16d5acd0c128bd05ed96c30861188a09541ed16099e17de005.exe
```

This page let you download the following malware sample: **SHA256 aae142810c653716d5acd0c128bd05ed96c30861188a09541ed16099e17de005**

Malware 3:

```
→  malware3 git:(master) ✗ sha256sum c3c451fa65b2a9d0863a02708bb3187630eff2f42d4
7b2455f2b53a621ea8bc7.exe
c3c451fa65b2a9d0863a02708bb3187630eff2f42d47b2455f2b53a621ea8bc7  c3c451fa65b2a9
d0863a02708bb3187630eff2f42d47b2455f2b53a621ea8bc7.exe
```

This page let you download the following malware sample: **SHA256 468b1a3d163c9123ff825af496e42cc29a7c8d2fd63bd5593f411f22150c76c8**

Malware 4:

```
→  malware4 git:(master) ✗ sha256sum 0e950d396f054459d624c7734c02e9357f2a0fa21bad98edc52d46169b
3487eb.exe
0e950d396f054459d624c7734c02e9357f2a0fa21bad98edc52d46169b3487eb  0e950d396f054459d624c7734c02e
9357f2a0fa21bad98edc52d46169b3487eb.exe
```

This page let you download the following malware sample: **SHA256 0e950d396f054459d624c7734c02e9357f2a0fa21bad98edc52d46169b3487eb**

Malware 5:

```
→  malware5 git:(master) ✗ sha256sum e2d9f04171f5c46f9a3844d738ef819fdfcfb43cb7c
ca86b43a348fb89930423.exe
e2d9f04171f5c46f9a3844d738ef819fdfcfb43cb7cca86b43a348fb89930423  e2d9f04171f5c4
6f9a3844d738ef819fdfcfb43cb7cca86b43a348fb89930423.exe
```

This page let you download the following malware sample: **SHA256 e2d9f04171f5c46f9a3844d738ef819fdfcfb43cb7cca86b43a348fb89930423**

Task 2

I moved the malware files into my virtual machine in order to prevent accidental damage to my machine, but I also removed execute permissions as an extra precaution. Other precautions I added was not including any executables into version control and using a chroot so that my vm was isolated further.

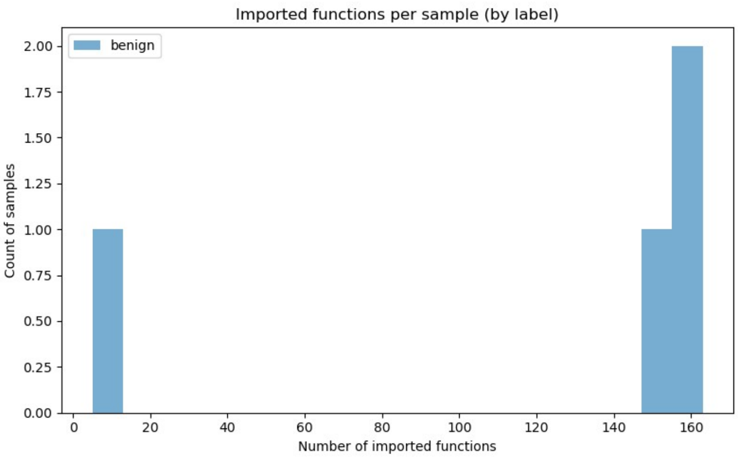
I created a bash script that retrieves data from the executables. The script also made use of upx and a python dependency pefile.
I successfully ran the script that extracts file types, import tables, reading metadata like timestamps, compiler information and urls and strings.

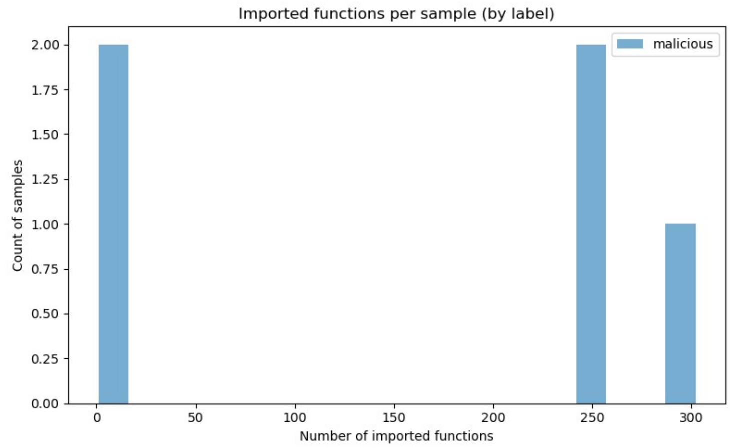
Malware analysis
I opted to use Rizin for binary analysis, because I have a Linux machine. Some samples could not be parsed by Rizin due to malformed PE headers, which itself could be indicative of packing or anti-analysis techniques used in malware.

I used upx analysis to see if any of the executables were packed by upx, but none were.
Binwalk also did not identify any embedded file signatures, which probably means they used non-standard packing or encryption methods to conceal the payload.
The absence of these results don't mean my analysis failed, instead they are meaningful in themselves. Benign executables are typically uncompressed while malware often employs custom packing techniques undetectable by standard tools.

Benign analysis
Rizin also failed for all the benign exe's, so I cannot use that meaningfully when creating my YARA scripts to classify malware.
Binwalk also didn't identify any embedded file signatures.
None of the benign apps were packed by upx, which makes sense, but also I cannot use this meaningfully when creating my yara files.
The benign apps mostly have trusted imbedded urls for example microsoft.com and digicert.com, which will make it easier to identify malware when compared to their urls - which are urls like chiark.greenend.org.uk.

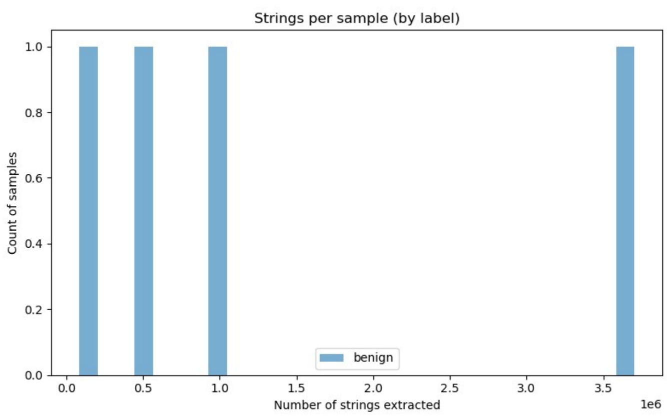
Comparison of graphs
I used a python script to retrieve data from bash script output, organize into csv and plot them using matplotlib.
Import function count:

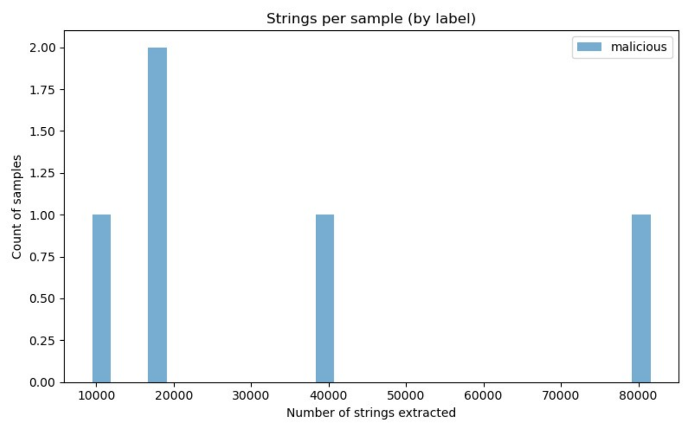




benign exe's typically import 10-160 functions, with most clustering around 150-160 and around 10. Malicious exe's show a much wider distribution importing 0-300+ functions. I will create yara rules that flag exe's with unusually high import counts >200 or suspiciously low < 5 imports.

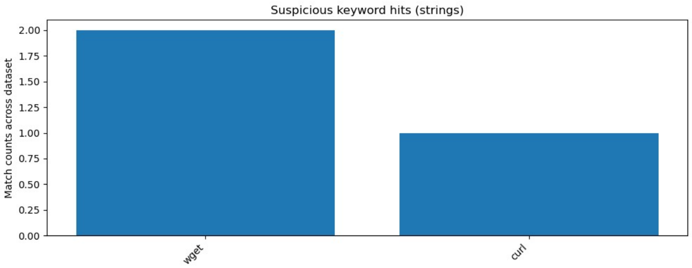
Strings:

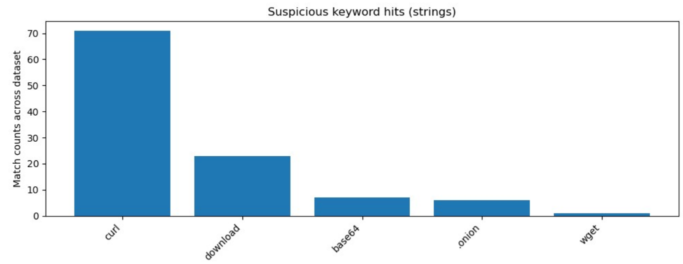




The benign apps have significantly higher strings ranging from 0-4 millions whereas the malicious have around 10-80k strings. Thus files with unusually low strings counts in the 10k-80k range will be flagged.

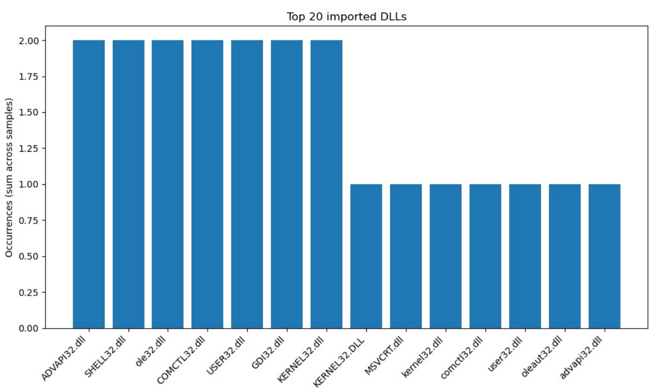
Suspicious Keywords:

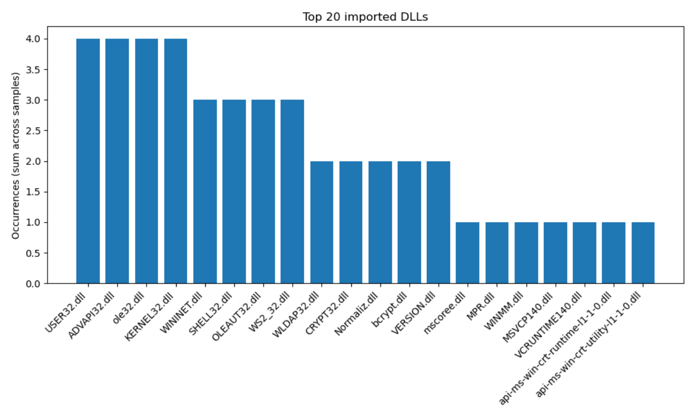




Malicious files have high amounts of curl, download and moderate amount of base64, onion(very suspicious) and wget. Benign apps have minimum presence of these keywords. I will high occurences of curl, any occurence of onion and download + base64 and wget to flag malware in my yara files.

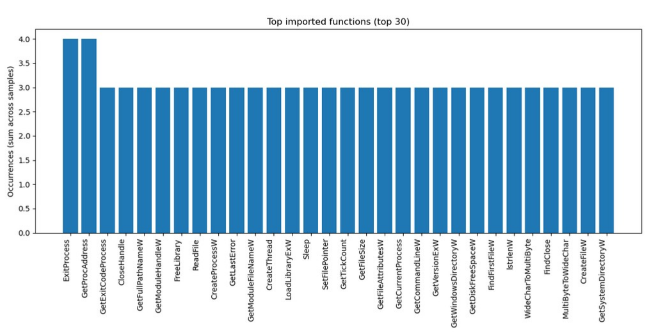
Dll imports:

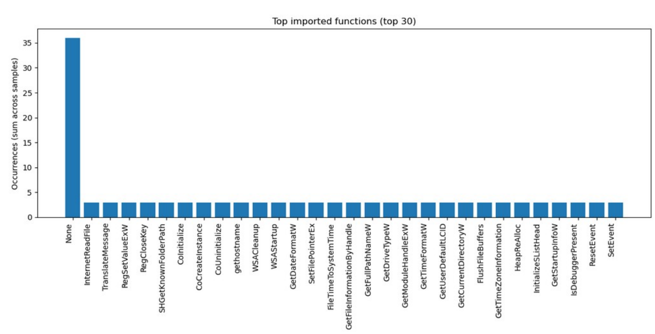




Malware has a heavy reliance on USER32.dll, ADVAPI32.dll, OLE32.dll and KERNEL32.dll and moderate use of WININET.dll, SHELL32.dll, OLEAUT32.dll, WS2_32.dll. Benign apps have way more diverse dll usage with lower individual concentrations. In my yara I will flag certain dll combinations.

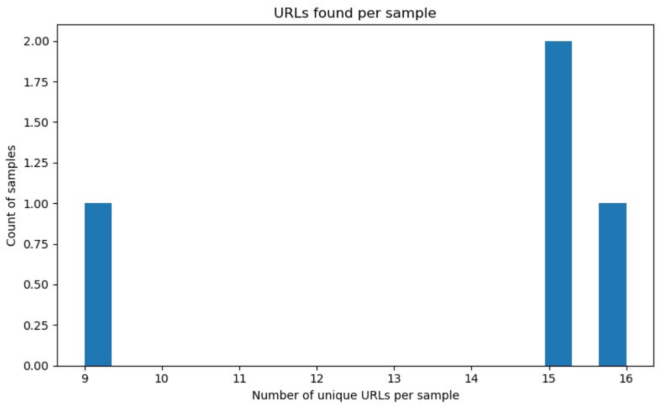
Import function analysis:

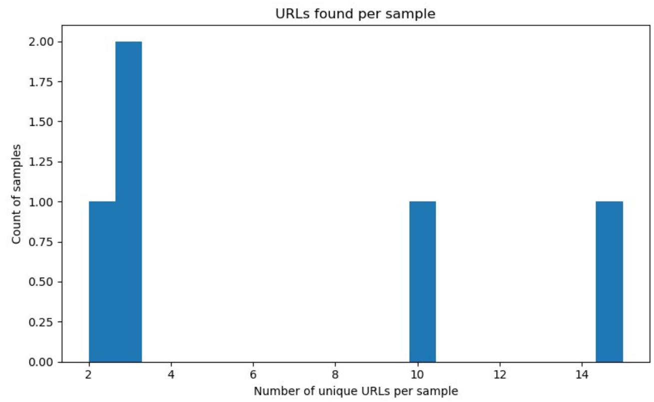




Malicious files show an extreme spike in one unnamed function 36+ occurences, while benign files show a more balanced function usage. For yara I will look at unusual concentrations of specific imported functions.

Urls patterns:





Malware contains fewer unique urls whereas benign show a more diverse set of urls, however in my samples I didn't get a large enough diference in my opinion to apply a hard rule for my yara files.

Task 3

I created 3 yara rules for the following:

- Rule 1: Should trigger on malware samples with high import counts, minimal false positives on legitimate software
- Rule 2: Should have high detection rate on downloaders/droppers, very low false positive rate
- Rule 3: Should catch packed/obfuscated samples

1. HighImportCountMalware:

- Based on analysis showing malicious files can have 250-300+ imports
- Targets the specific DLL combination pattern (USER32 + ADVAPI32 + WININET)
- Requires both network and system manipulation capabilities
- Low false positive risk as benign software rarely combines high imports with these specific capabilities

2. NetworkDownloaderMalware:

- Targets the stark difference in "curl" usage (70+ in malware vs ~2 in benign)
- Focuses on download + encoding combinations (common in malware payloads)
- Onion references are strong indicators with minimal false positive risk
- Uses threshold-based detection for high-frequency terms

3. LowStringCountObfuscatedMalware:

- Addresses the corrected finding that malware has fewer strings (10K-80K vs millions)
- Detects files with functionality but suspiciously few readable strings
- Targets obfuscated/packed malware that tries to hide its true nature
- Uses dynamic loading functions as indicators of evasive techniques

Results of yara file on benign apps:

```
➜  Prac3 git:(master) ✗ ./batch_run_yara.sh                                    [INSERT]
Testing: firefox.exe
Testing: gimp-3.0.4-setup.exe
Testing: npp.8.8.5.Installer.x64.exe
LowStringCountObfuscatedMalware ./benign/binaries/npp.8.8.5.Installer.x64.e
xe
Testing: vlc-3.0.21-win64.exe
```

1 False positive – it flagged notepad plus plus for having a suspiciously low string count

Results of yara file on malicious files:

```
betatesters@betatestersvm:~/malware$ ./batch_run_yara.sh
Testing: 0e950d396f054459d624c7734c02e9357f2a0fa21bad98edc52d46169b3487eb.exe
Testing: 7109c74b24a883dbd37cf5d23a11642ed056d876e5120102ab860da498550e33.exe
NetworkDownloaderMalware ./binaries/7109c74b24a883dbd37cf5d23a11642ed056d876e5120102ab860da498550e33.
exe
LowStringCountObfuscatedMalware ./binaries/7109c74b24a883dbd37cf5d23a11642ed056d876e5120102ab860da498
550e33.exe
Testing: aae142810c653716d5acd0c128bd05ed96c30861188a09541ed16099e17de005.exe
NetworkDownloaderMalware ./binaries/aae142810c653716d5acd0c128bd05ed96c30861188a09541ed16099e17de005.
exe
LowStringCountObfuscatedMalware ./binaries/aae142810c653716d5acd0c128bd05ed96c30861188a09541ed16099e1
7de005.exe
Testing: c3c451fa65b2a9d0863a02708bb3187630eff2f42d47b2455f2b53a621ea8bc7.exe
NetworkDownloaderMalware ./binaries/c3c451fa65b2a9d0863a02708bb3187630eff2f42d47b2455f2b53a621ea8bc7.
exe
LowStringCountObfuscatedMalware ./binaries/c3c451fa65b2a9d0863a02708bb3187630eff2f42d47b2455f2b53a621
ea8bc7.exe
Testing: e2d9f04171f5c46f9a3844d738ef819fdfcfb43cb7cca86b43a348fb89930423.exe
```

2 malicious files weren't flagged as malware. This indicates that I need to tune my values a bit more, but for now I am satisfied with the results.