Sicurezza Informatica, elaborato - traccia 2: Toxic Eye

Jacopo Mora [715149], Salvalai Matteo [715827]
Notebook distribuito con licenza **CC-BY**

In questo documento viene descritto il lavoro svolto dagli autori nell'ambito dello studio della robustezza di sistemi di riconoscimento automatico di malware. Ad alto livello, l'obiettivo principale consiste nell'addestrare un sistema di riconoscimento, basato su tecniche di Machine Learning, mediante un dataset noto, composto da dati sperimentali, e successivamente porlo come soggetto ad un attacco, in particolare a Fast Gradient Sign Method, in modo da generare una versione di malware che non venga riconosciuta come tale. Il malware scelto si chiama ToxicEye ed il sistema di riconoscimento consiste in una rete neurale profonda.

ToxicEye

ToxicEye è un *RAT*, scritto in C#, installabile su computer *Windows* e controllabile tramite l'app di messaggistica istantanea *Telegram*. Il malware è in grado di eseguire sulla macchina target una variegata lista di funzioni, che vanno dal semplice aprire/chiudere lo sportello del lettore CD al controllare webcam e microfono, o ancora fungere da keylogger e stealer di passwords. Il codice sorgente del malware è scaricabile da GitHub, dove è presente anche una guida dettagliata alla compilazione e all'uso.

Di ToxicEye se ne è parlato molto nel 2021: sono stati scritti articoli su autorevoli blog di sicurezza informatica come CheckPoint. In Italia è stato il soggetto di articoli da parte di HWUpgrade, Cyclonis e Mobile World.

Ambiente di lavoro

Essendo il malware progettato per funzionare su Windows, abbiamo deciso di utilizzare una macchina virtuale Windows 10, resa disponibile da Microsoft, dotata di Visual Studio, IDE pensato per lo sviluppo di applicazioni in linguaggi come C++ e C#. Da questa macchina virtuale, disconnessa dalla rete, è stato possibile effettuare la compilazione del malware e la modifica del suo codice sorgente in modo sicuro, consentendo di non mettere in pericolo sistemi esterni ad essa.

Al termine della compilazione del codice sorgente, l'antivirus in dotazione della macchina virtuale, Windows Defender, è stato in grado di identificare immediatamente l'eseguibile come malevolo.

Preparazione del codice sorgente, delle dipendenze necessarie e del dataset

Preparazione del codice sorgente

Prima di ottenere l'eseguibile vero e proprio è stato necessario modificare alcuni parametri del malware, tra cui la chiave API del bot di Telegram (il quale funge da tramite) e l'ID Telegram dell'utente proprietario, in modo da impedire che il bot sia sfruttato da utenti terzi. Inoltre, è stato necessario impostare la variabile *PreventStartOnVirtualMachine* a *false*, in modo da poter eseguire il malware sulla nostra macchina virtuale. La compilazione del codice sorgente è avvenuta mediante gli strumenti offerti da Visual Studio.

Preparazione delle dipendenze

Per l'analisi dell'header PE del malware utilizzeremo la libreria *pefile*, importata di seguito nel nostro ambiente di lavoro.

In [ ]:

```
!pip install pefile
import pefile
```

```
Collecting pefile
  Downloading
https://files.pythonhosted.org/packages/dc/8e/99fde2fe50afebbb1ef6f46508203e66e45c6e3966caacd219152
2e1/pefile-2021.5.24.tar.gz (66kB)
     |████████████████████████████████| 71kB 6.4MB/s
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (from pefile)
(0.16.0)
Building wheels for collected packages: pefile
  Building wheel for pefile (setup.py) ... done
  Created wheel for pefile: filename=pefile-2021.5.24-cp37-none-any.whl size=62592
sha256=f1886934b08ec5799136809736fe3f3f59491d03dc698bee22595b3da269ed42
  Stored in directory:
/root/.cache/pip/wheels/d6/7d/79/6d4efc404f6bd245244465f13a73bb7d303f83d70beb67b071
Successfully built pefile
Installing collected packages: pefile
Successfully installed pefile-2021.5.24
```

Per la gestione del dataset e la costruzione dei modelli di riconocimento utilizzeremo librerie come *Pandas*, *Numpy*, *SciKitLearn* e *Tensorflow*. Useremo *Tensorboard* per l'analisi dei modelli addestrati.

In [ ]:

```
import numpy as np
import pandas as pd
import sklearn

import tensorflow as tf

# Tensorboard
%load_ext tensorboard
import datetime
!rm -rf ./logs/
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

La libreria *adversarial-robustness-toolbox* fornisce le funzionalità necessarie per una semplice implementazione di **Fast Gradient Sign Method**.

In [ ]:

```
!pip install adversarial-robustness-toolbox
```

```
Collecting adversarial-robustness-toolbox
  Downloading
https://files.pythonhosted.org/packages/cc/eb/6c3bd5d5237d2c1ede42bdc8a0fa31eb3066440dc0623789e05b4
fe3/adversarial_robustness_toolbox-1.7.0-py3-none-any.whl (1.1MB)
     |████████████████████████████████| 1.1MB 13.5MB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from adversarial-
robustness-toolbox) (4.41.1)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-packages (from
adversarial-robustness-toolbox) (1.4.1)
Collecting numba~=0.53.1
  Downloading
https://files.pythonhosted.org/packages/bb/73/d9c127eddbe3c105a33379d425b88f9dca249a6eddf39ce886494
3f9/numba-0.53.1-cp37-cp37m-manylinux2014_x86_64.whl (3.4MB)
     |████████████████████████████████| 3.4MB 42.1MB/s
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.7/dist-packages (from
adversarial-robustness-toolbox) (1.19.5)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from
adversarial-robustness-toolbox) (57.0.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from adversarial-
robustness-toolbox) (1.15.0)
Requirement already satisfied: scikit-learn<0.24.3,>=0.22.2 in /usr/local/lib/python3.7/dist-
packages (from adversarial-robustness-toolbox) (0.22.2.post1)
Collecting llvmlite<0.37,>=0.36.0rc1
```

```
  Downloading
https://files.pythonhosted.org/packages/54/25/2b4015e2b0c3be2efa6870cf2cf2bd969dd0e5f937476fc13c102
f32/llvmlite-0.36.0-cp37-cp37m-manylinux2010_x86_64.whl (25.3MB)
     |████████████████████████████████| 25.3MB 120kB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from
scikit-learn<0.24.3,>=0.22.2->adversarial-robustness-toolbox) (1.0.1)
Installing collected packages: llvmlite, numba, adversarial-robustness-toolbox
  Found existing installation: llvmlite 0.34.0
    Uninstalling llvmlite-0.34.0:
      Successfully uninstalled llvmlite-0.34.0
  Found existing installation: numba 0.51.2
    Uninstalling numba-0.51.2:
      Successfully uninstalled numba-0.51.2
Successfully installed adversarial-robustness-toolbox-1.7.0 llvmlite-0.36.0 numba-0.53.1
```

Importeremo infine la cartella git in cui risiedono risorse essenziale come l'istanza addestrata del riconoscitore, l'eseguibile originale di ToxicEye e l'eseguibile modificato in modo da evadere il riconoscimento.

In [ ]:

```
!git clone https://github.com/jacknot/CyberSecStaticMalwareAnalysis.git
```

```
Cloning into 'CyberSecStaticMalwareAnalysis'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 26 (delta 8), reused 1 (delta 0), pack-reused 0
Unpacking objects: 100% (26/26), done.
```

Preparazione del dataset

Il dataset utilizzato in questo elaborato, è disponibile su Kaggle. L'autore del dataset ha estratto tramite *pefile* informazioni di una moltitudine di eseguibili, classificandoli come malware o meno.

Di seguito viene descritta la procedura usata dal creatore del dataset considerato per l'estrazione delle features a partire dai file PE oggetti di studio, recuperabile dalla sua repository Github. Le informazioni non sono estraibili unicamente partendo dagli eseguibili: per alcune fearures, come *SuspiciousNameSection* e *SuspiciousImportFunctions*, sono necessari ulteriori file, in formato .txt, sfortunatamente non resi disponibili. Problematiche relative a valori mancanti non sussistono nel caso del dataset originale, devono però essere gestite nel momento nel quale si voglio introdurre nuovi record. Per quanto riguarda la feature *SuspiciousNameSection* abbiamo deciso di usare il valore più frequente, ovvero 0. Per quanto riguarda la feature *SuspiciousImportFunctions*, invece, siamo riusciti a recuperare una lista di funzioni sospette, non però coincidente con la lista utilizzata dall'autore del dataset. Questa lista viene fornita da KoolimRezah in un file .txt.

In [ ]:

```
!wget https://raw.githubusercontent.com/KoolimRezah/antiMalware/master/suspiciousFunctions.txt
#il file non è l'originale usato dall'autore
sus_funcs = []
with open('suspiciousFunctions.txt') as f:
    sus_funcs = f.readlines()
sus_funcs = list(map(lambda x : x.strip(), sus_funcs))
```

```
--2021-07-05 18:23:51--
https://raw.githubusercontent.com/KoolimRezah/antiMalware/master/suspiciousFunctions.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133,
185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 1951 (1.9K) [text/plain]
Saving to: 'suspiciousFunctions.txt'

suspiciousFunctions 100%[===================>]   1.91K  --.-KB/s    in 0s

2021-07-05 18:23:51 (20.2 MB/s) - 'suspiciousFunctions.txt' saved [1951/1951]
```

In [ ]:

```
# https://github.com/amauricio/sklearn-antimalware/blob/master/test.py
def extract_features_from_executable(name, path):
    pe = pefile.PE(path)
    number_packers = 0

    entropy = map(lambda x:x.get_entropy(), pe.sections)
    raw_sizes = map(lambda x:x.SizeOfRawData, pe.sections)
    virtual_sizes = map(lambda x:x.Misc_VirtualSize, pe.sections)
    physical_address = map(lambda x:x.Misc_PhysicalAddress, pe.sections)
    virtual_address = map(lambda x:x.VirtualAddress, pe.sections)
    pointer_raw_data = map(lambda x:x.PointerToRawData, pe.sections)
    characteristics = map(lambda x:x.Characteristics, pe.sections)

    data = {'Name':name,
            'e_magic':pe.DOS_HEADER.e_magic,
            'e_cblp':pe.DOS_HEADER.e_cblp,
            'e_cp':pe.DOS_HEADER.e_cp,
            'e_crlc':pe.DOS_HEADER.e_crlc,
            'e_cparhdr':pe.DOS_HEADER.e_cparhdr,
            'e_minalloc':pe.DOS_HEADER.e_minalloc,
            'e_maxalloc':pe.DOS_HEADER.e_maxalloc,
            'e_ss':pe.DOS_HEADER.e_ss,
            'e_sp':pe.DOS_HEADER.e_sp,
            'e_csum':pe.DOS_HEADER.e_csum,
            'e_ip':pe.DOS_HEADER.e_ip,
            'e_cs':pe.DOS_HEADER.e_cs,
            'e_lfarlc':pe.DOS_HEADER.e_lfarlc,
            'e_ovno':pe.DOS_HEADER.e_ovno,
            'e_oemid':pe.DOS_HEADER.e_oemid,
            'e_oeminfo':pe.DOS_HEADER.e_oeminfo,
            'e_lfanew':pe.DOS_HEADER.e_lfanew,
            'Machine':pe.FILE_HEADER.Machine,
            'NumberOfSections':pe.FILE_HEADER.NumberOfSections,
            'TimeDateStamp':pe.FILE_HEADER.TimeDateStamp,
            'PointerToSymbolTable':pe.FILE_HEADER.PointerToSymbolTable,
            'NumberOfSymbols':pe.FILE_HEADER.NumberOfSymbols,
            'SizeOfOptionalHeader':pe.FILE_HEADER.SizeOfOptionalHeader,
            'Characteristics':pe.FILE_HEADER.Characteristics,
            'Magic':pe.OPTIONAL_HEADER.Magic,
            'MajorLinkerVersion':pe.OPTIONAL_HEADER.MajorLinkerVersion,
            'MinorLinkerVersion':pe.OPTIONAL_HEADER.MinorLinkerVersion,
            'SizeOfCode':pe.OPTIONAL_HEADER.SizeOfCode,
            'SizeOfInitializedData':pe.OPTIONAL_HEADER.SizeOfInitializedData,
            'SizeOfUninitializedData':pe.OPTIONAL_HEADER.SizeOfUninitializedData,
            'AddressOfEntryPoint':pe.OPTIONAL_HEADER.AddressOfEntryPoint,
            'BaseOfCode':pe.OPTIONAL_HEADER.BaseOfCode,
            'ImageBase':pe.OPTIONAL_HEADER.ImageBase,
            'SectionAlignment':pe.OPTIONAL_HEADER.SectionAlignment,
            'FileAlignment':pe.OPTIONAL_HEADER.FileAlignment,
            'MajorOperatingSystemVersion':pe.OPTIONAL_HEADER.MajorOperatingSystemVersion,
            'MinorOperatingSystemVersion':pe.OPTIONAL_HEADER.MinorOperatingSystemVersion,
            'MajorImageVersion':pe.OPTIONAL_HEADER.MajorImageVersion,
            'MinorImageVersion':pe.OPTIONAL_HEADER.MinorImageVersion,
            'MajorSubsystemVersion':pe.OPTIONAL_HEADER.MajorSubsystemVersion,
            'MinorSubsystemVersion':pe.OPTIONAL_HEADER.MinorSubsystemVersion,
            'SizeOfHeaders':pe.OPTIONAL_HEADER.SizeOfHeaders,
            'CheckSum':pe.OPTIONAL_HEADER.CheckSum,
            'SizeOfImage':pe.OPTIONAL_HEADER.SizeOfImage,
            'Subsystem':pe.OPTIONAL_HEADER.Subsystem,
            'DllCharacteristics':pe.OPTIONAL_HEADER.DllCharacteristics,
            'SizeOfStackReserve':pe.OPTIONAL_HEADER.SizeOfStackReserve,
            'SizeOfStackCommit':pe.OPTIONAL_HEADER.SizeOfStackCommit,
            'SizeOfHeapReserve':pe.OPTIONAL_HEADER.SizeOfHeapReserve,
            'SizeOfHeapCommit':pe.OPTIONAL_HEADER.SizeOfHeapCommit,
            'LoaderFlags':pe.OPTIONAL_HEADER.LoaderFlags,
            'NumberOfRvaAndSizes':pe.OPTIONAL_HEADER.NumberOfRvaAndSizes
        }

    sus_functions_count = 0
    for entry in getattr(pe, 'DIRECTORY_ENTRY_IMPORT', []):
        for func in entry.imports:
            if func.name.decode('utf-8') in sus_funcs:
                sus_functions_count += 1
    data['SuspiciousImportFunctions'] = sus_functions_count
```

```
    data['SuspiciousNameSection'] = 0

    try:
        data['SectionsLength'] = len(pe.sections)
    except (ValueError, TypeError):
        data['SectionsLength'] = 0
    try:
        data['SectionMinEntropy'] = min(entropy)
    except (ValueError, TypeError):
        data['SectionMinEntropy'] = 0
    try:
        data['SectionMaxEntropy'] = max(entropy)
    except (ValueError, TypeError):
        data['SectionMaxEntropy'] = 0
    try:
        data['SectionMinRawsize'] = min(raw_sizes)
    except (ValueError, TypeError):
        data['SectionMinRawsize'] = 0
    try:
        data['SectionMaxRawsize'] = max(raw_sizes)
    except (ValueError, TypeError):
        data['SectionMaxRawsize'] = 0
    try:
        data['SectionMinVirtualsize'] = min(virtual_sizes)
    except (ValueError, TypeError):
        data['SectionMinVirtualsize'] = 0
    try:
        data['SectionMaxVirtualsize'] = max(virtual_sizes)
    except (ValueError, TypeError):
        data['SectionMaxVirtualsize'] = 0
    try:
        data['SectionMaxVirtualsize'] = max(virtual_sizes)
    except (ValueError, TypeError):
        data['SectionMaxVirtualsize'] = 0

    try:
        data['SectionMaxPhysical'] = max(physical_address)
    except (ValueError, TypeError):
        data['SectionMaxPhysical'] = 0
    try:
        data['SectionMinPhysical'] = min(physical_address)
    except (ValueError, TypeError):
        data['SectionMinPhysical'] = 0

    try:
        data['SectionMaxVirtual'] = max(virtual_address)
    except (ValueError, TypeError):
        data['SectionMaxVirtual'] = 0
    try:
        data['SectionMinVirtual'] = min(virtual_address)
    except (ValueError, TypeError):
        data['SectionMinVirtual'] = 0

    try:
        data['SectionMaxPointerData'] = max(pointer_raw_data)
    except (ValueError, TypeError):
        data['SectionMaxPointerData'] = 0

    try:
        data['SectionMinPointerData'] = min(pointer_raw_data)
    except (ValueError, TypeError):
        data['SectionMinPointerData'] = 0

    try:
        data['SectionMaxChar'] = max(characteristics)
    except (ValueError, TypeError):
        data['SectionMaxChar'] = 0

    try:
        data['SectionMinChar'] = min(characteristics)
    except (ValueError, TypeError):
        data['SectionMainChar'] = 0

    try:
        data['DirectoryEntryImport'] = (len(pe.DIRECTORY_ENTRY_IMPORT))
        imports = sum([x.imports for x in pe.DIRECTORY_ENTRY_IMPORT], [])
        data['DirectoryEntryImportSize'] = (len(imports))
```

```
    except AttributeError:
        data['DirectoryEntryImport'] = 0
        data['DirectoryEntryImportSize'] =0
    #Exports
    try:
        data['DirectoryEntryExport']  = (len(pe.DIRECTORY_ENTRY_EXPORT.symbols))
    except AttributeError:
        # No export
        data['DirectoryEntryExport']  = 0


    data[ 'ImageDirectoryEntryExport' ] = pe.OPTIONAL_HEADER.DATA_DIRECTORY[pefile.DIRECTORY_ENTRY[
'IMAGE_DIRECTORY_ENTRY_EXPORT']].VirtualAddress
    data[ 'ImageDirectoryEntryImport' ] = pe.OPTIONAL_HEADER.DATA_DIRECTORY[pefile.DIRECTORY_ENTRY[
'IMAGE_DIRECTORY_ENTRY_IMPORT']].VirtualAddress
    data[ 'ImageDirectoryEntryResource' ] = pe.OPTIONAL_HEADER.DATA_DIRECTORY[pefile.DIRECTORY_ENTR
Y['IMAGE_DIRECTORY_ENTRY_RESOURCE']].VirtualAddress
    data[ 'ImageDirectoryEntryException' ] = pe.OPTIONAL_HEADER.DATA_DIRECTORY[pefile.DIRECTORY_ENT
RY['IMAGE_DIRECTORY_ENTRY_EXCEPTION']].VirtualAddress
    data[ 'ImageDirectoryEntrySecurity' ] = pe.OPTIONAL_HEADER.DATA_DIRECTORY[pefile.DIRECTORY_ENTR
Y['IMAGE_DIRECTORY_ENTRY_SECURITY']].VirtualAddress

    return pd.DataFrame(data, index=[0])
```

Di seguito vengono mostrate le informazioni sull'header di Toxic Eye.

In [ ]:

```
pe =  pefile.PE("CyberSecStaticMalwareAnalysis/TelegramRAT.exe")
print(pe.dump_info())
```

```
----------DOS_HEADER----------

[IMAGE_DOS_HEADER]
0x0        0x0   e_magic:                       0x5A4D
0x2        0x2   e_cblp:                        0x90
0x4        0x4   e_cp:                          0x3
0x6        0x6   e_crlc:                        0x0
0x8        0x8   e_cparhdr:                     0x4
0xA        0xA   e_minalloc:                    0x0
0xC        0xC   e_maxalloc:                    0xFFFF
0xE        0xE   e_ss:                          0x0
0x10       0x10  e_sp:                          0xB8
0x12       0x12  e_csum:                        0x0
0x14       0x14  e_ip:                          0x0
0x16       0x16  e_cs:                          0x0
0x18       0x18  e_lfarlc:                      0x40
0x1A       0x1A  e_ovno:                        0x0
0x1C       0x1C  e_res:
0x24       0x24  e_oemid:                       0x0
0x26       0x26  e_oeminfo:                     0x0
0x28       0x28  e_res2:
0x3C       0x3C  e_lfanew:                      0x80

----------NT_HEADERS----------

[IMAGE_NT_HEADERS]
0x80       0x0   Signature:                     0x4550

----------FILE_HEADER----------

[IMAGE_FILE_HEADER]
0x84       0x0   Machine:                       0x14C
0x86       0x2   NumberOfSections:              0x3
0x88       0x4   TimeDateStamp:                 0xBF850362 [Tue Oct 27 09:56:50 2071 UTC]
0x8C       0x8   PointerToSymbolTable:          0x0
0x90       0xC   NumberOfSymbols:               0x0
0x94       0x10  SizeOfOptionalHeader:          0xE0
0x96       0x12  Characteristics:               0x22
Flags: IMAGE_FILE_EXECUTABLE_IMAGE, IMAGE_FILE_LARGE_ADDRESS_AWARE

----------OPTIONAL_HEADER----------

[IMAGE_OPTIONAL_HEADER]
0x98       0x0   Magic:                         0x10B
```

```
0x98        0x0   Magic:                          0x10B
0x9A        0x2   MajorLinkerVersion:             0x30
0x9B        0x3   MinorLinkerVersion:             0x0
0x9C        0x4   SizeOfCode:                      0x1B200
0xA0        0x8   SizeOfInitializedData:          0x800
0xA4        0xC   SizeOfUninitializedData:        0x0
0xA8        0x10  AddressOfEntryPoint:            0x1D1DA
0xAC        0x14  BaseOfCode:                     0x2000
0xB0        0x18  BaseOfData:                     0x1E000
0xB4        0x1C  ImageBase:                      0x400000
0xB8        0x20  SectionAlignment:               0x2000
0xBC        0x24  FileAlignment:                  0x200
0xC0        0x28  MajorOperatingSystemVersion:    0x4
0xC2        0x2A  MinorOperatingSystemVersion:    0x0
0xC4        0x2C  MajorImageVersion:              0x0
0xC6        0x2E  MinorImageVersion:              0x0
0xC8        0x30  MajorSubsystemVersion:          0x6
0xCA        0x32  MinorSubsystemVersion:          0x0
0xCC        0x34  Reserved1:                      0x0
0xD0        0x38  SizeOfImage:                    0x22000
0xD4        0x3C  SizeOfHeaders:                  0x200
0xD8        0x40  CheckSum:                       0x0
0xDC        0x44  Subsystem:                      0x3
0xDE        0x46  DllCharacteristics:             0x8560
0xE0        0x48  SizeOfStackReserve:             0x100000
0xE4        0x4C  SizeOfStackCommit:              0x1000
0xE8        0x50  SizeOfHeapReserve:              0x100000
0xEC        0x54  SizeOfHeapCommit:               0x1000
0xF0        0x58  LoaderFlags:                    0x0
0xF4        0x5C  NumberOfRvaAndSizes:            0x10
DllCharacteristics: IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE,
IMAGE_DLLCHARACTERISTICS_HIGH_ENTROPY_VA, IMAGE_DLLCHARACTERISTICS_NO_SEH,
IMAGE_DLLCHARACTERISTICS_NX_COMPAT, IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE

----------PE Sections----------

[IMAGE_SECTION_HEADER]
0x178       0x0   Name:                           .text
0x180       0x8   Misc:                           0x1B200
0x180       0x8   Misc_PhysicalAddress:           0x1B200
0x180       0x8   Misc_VirtualSize:               0x1B200
0x184       0xC   VirtualAddress:                 0x2000
0x188       0x10  SizeOfRawData:                  0x1B200
0x18C       0x14  PointerToRawData:               0x200
0x190       0x18  PointerToRelocations:           0x0
0x194       0x1C  PointerToLinenumbers:           0x0
0x198       0x20  NumberOfRelocations:            0x0
0x19A       0x22  NumberOfLinenumbers:            0x0
0x19C       0x24  Characteristics:                0x60000020
Flags: IMAGE_SCN_CNT_CODE, IMAGE_SCN_MEM_EXECUTE, IMAGE_SCN_MEM_READ
Entropy: 5.709185 (Min=0.0, Max=8.0)
MD5     hash: 5bf544c5223fa3ae3248d63a9bdf398c
SHA-1   hash: c573697b335bf23ce5fc88a6adc284de076e2f3f
SHA-256 hash: d6ac3e077e1eef377f4e10f942f10c68b9f72b641f23e202403b842ebee24f08
SHA-512 hash:
47ca6d45ef3e198f1dcc0af8b69edd0639db9ac66de82223e6d90d4f604adb14f7a99410a949755ad2f79342373e2bbeeab
a40526fe9be6dba6354f027d0

[IMAGE_SECTION_HEADER]
0x1A0       0x0   Name:                           .rsrc
0x1A8       0x8   Misc:                           0x5BC
0x1A8       0x8   Misc_PhysicalAddress:           0x5BC
0x1A8       0x8   Misc_VirtualSize:               0x5BC
0x1AC       0xC   VirtualAddress:                 0x1E000
0x1B0       0x10  SizeOfRawData:                  0x600
0x1B4       0x14  PointerToRawData:               0x1B400
0x1B8       0x18  PointerToRelocations:           0x0
0x1BC       0x1C  PointerToLinenumbers:           0x0
0x1C0       0x20  NumberOfRelocations:            0x0
0x1C2       0x22  NumberOfLinenumbers:            0x0
0x1C4       0x24  Characteristics:                0x40000040
Flags: IMAGE_SCN_CNT_INITIALIZED_DATA, IMAGE_SCN_MEM_READ
Entropy: 4.115205 (Min=0.0, Max=8.0)
MD5     hash: 7907b9697008599683b66a56d4c253ab
SHA-1   hash: 35d2f25bf86a64c8d2c29daed1f7a7d03e6be079
SHA-256 hash: 9b4dbab148442f10dfc2440351b59abe3eca30d89817a0046d39ceff5260e8fa
SHA-512 hash:
f7313a19a443983f81515416a3a6f9220da65bbef87071433431aa8d93403e7faa7388aadd1dfba3ad81df18adfb003a5c
```

17313a19c4439631615159416c3c619220da65bbe1670714333431cc8d93403e71ca7366eaddidiba3cd81df18edfb003a3c
dc48d645f53285a57d5feab8d

[IMAGE_SECTION_HEADER]
0x1C8 0x0 Name: .reloc
0x1D0 0x8 Misc: 0xC
0x1D0 0x8 Misc_PhysicalAddress: 0xC
0x1D0 0x8 Misc_VirtualSize: 0xC
0x1D4 0xC VirtualAddress: 0x20000
0x1D8 0x10 SizeOfRawData: 0x200
0x1DC 0x14 PointerToRawData: 0x1BA00
0x1E0 0x18 PointerToRelocations: 0x0
0x1E4 0x1C PointerToLinenumbers: 0x0
0x1E8 0x20 NumberOfRelocations: 0x0
0x1EA 0x22 NumberOfLinenumbers: 0x0
0x1EC 0x24 Characteristics: 0x42000040
Flags: IMAGE_SCN_CNT_INITIALIZED_DATA, IMAGE_SCN_MEM_DISCARDABLE, IMAGE_SCN_MEM_READ
Entropy: 0.101910 (Min=0.0, Max=8.0)
MD5 hash: ac253f4ede3d7beb9eb50fd783785da9
SHA-1 hash: 431caf4bc9cda67260b716655915fae080d2f9e8
SHA-256 hash: 2fb733825d7c48d9d1ddca8cf6c548d6569cfce5fdc422ce1bc3c1fffe8aa0cb
SHA-512 hash:
9f5da82ddf9fd63913026f7c3ec04a46a69b3b55891df9c8783663e6496a75984f0e8de88187eb36be5db841baab54ceef8
d0cabaca4d64f8ac49e32831e

----------Directories----------

[IMAGE_DIRECTORY_ENTRY_EXPORT]
0xF8 0x0 VirtualAddress: 0x0
0xFC 0x4 Size: 0x0
[IMAGE_DIRECTORY_ENTRY_IMPORT]
0x100 0x0 VirtualAddress: 0x1D185
0x104 0x4 Size: 0x4F
[IMAGE_DIRECTORY_ENTRY_RESOURCE]
0x108 0x0 VirtualAddress: 0x1E000
0x10C 0x4 Size: 0x5BC
[IMAGE_DIRECTORY_ENTRY_EXCEPTION]
0x110 0x0 VirtualAddress: 0x0
0x114 0x4 Size: 0x0
[IMAGE_DIRECTORY_ENTRY_SECURITY]
0x118 0x0 VirtualAddress: 0x0
0x11C 0x4 Size: 0x0
[IMAGE_DIRECTORY_ENTRY_BASERELOC]
0x120 0x0 VirtualAddress: 0x20000
0x124 0x4 Size: 0xC
[IMAGE_DIRECTORY_ENTRY_DEBUG]
0x128 0x0 VirtualAddress: 0x1D0EC
0x12C 0x4 Size: 0x38
[IMAGE_DIRECTORY_ENTRY_COPYRIGHT]
0x130 0x0 VirtualAddress: 0x0
0x134 0x4 Size: 0x0
[IMAGE_DIRECTORY_ENTRY_GLOBALPTR]
0x138 0x0 VirtualAddress: 0x0
0x13C 0x4 Size: 0x0
[IMAGE_DIRECTORY_ENTRY_TLS]
0x140 0x0 VirtualAddress: 0x0
0x144 0x4 Size: 0x0
[IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG]
0x148 0x0 VirtualAddress: 0x0
0x14C 0x4 Size: 0x0
[IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT]
0x150 0x0 VirtualAddress: 0x0
0x154 0x4 Size: 0x0
[IMAGE_DIRECTORY_ENTRY_IAT]
0x158 0x0 VirtualAddress: 0x2000
0x15C 0x4 Size: 0x8
[IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT]
0x160 0x0 VirtualAddress: 0x0
0x164 0x4 Size: 0x0
[IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR]
0x168 0x0 VirtualAddress: 0x2008
0x16C 0x4 Size: 0x48
[IMAGE_DIRECTORY_ENTRY_RESERVED]
0x170 0x0 VirtualAddress: 0x0
0x174 0x4 Size: 0x0

----------Version Information----------

```
[VS_VERSIONINFO]
0x1B490    0x0   Length:                        0x32C
0x1B492    0x2   ValueLength:                   0x34
0x1B494    0x4   Type:                          0x0

[VS_FIXEDFILEINFO]
0x1B4B8    0x0   Signature:                     0xFEEF04BD
0x1B4BC    0x4   StrucVersion:                  0x10000
0x1B4C0    0x8   FileVersionMS:                 0x10000
0x1B4C4    0xC   FileVersionLS:                 0x0
0x1B4C8    0x10  ProductVersionMS:              0x10000
0x1B4CC    0x14  ProductVersionLS:              0x0
0x1B4D0    0x18  FileFlagsMask:                 0x3F
0x1B4D4    0x1C  FileFlags:                     0x0
0x1B4D8    0x20  FileOS:                        0x4
0x1B4DC    0x24  FileType:                      0x1
0x1B4E0    0x28  FileSubtype:                   0x0
0x1B4E4    0x2C  FileDateMS:                    0x0
0x1B4E8    0x30  FileDateLS:                    0x0

[VarFileInfo]
0x1B4EC    0x0   Length:                        0x44
0x1B4EE    0x2   ValueLength:                   0x0
0x1B4F0    0x4   Type:                          0x1

  [Var]
  0x1B50C    0x0   Length:                        0x24
  0x1B50E    0x2   ValueLength:                   0x4
  0x1B510    0x4   Type:                          0x0
    Translation: 0x0000 0x04b0

[StringFileInfo]
0x1B530    0x0   Length:                        0x28C
0x1B532    0x2   ValueLength:                   0x0
0x1B534    0x4   Type:                          0x1

  [StringTable]
  0x1B554    0x0   Length:                        0x268
  0x1B556    0x2   ValueLength:                   0x0
  0x1B558    0x4   Type:                          0x1
  LangID: 000004b0

    Assembly Version: 1.0.0.0
    Comments:
    CompanyName:
    FileDescription: TelegramRAT
    FileVersion: 1.0.0.0
    InternalName: TelegramRAT.exe
    LegalCopyright: Copyright \xc2\xa9  2020
    LegalTrademarks:
    OriginalFilename: TelegramRAT.exe
    ProductName: TelegramRAT
    ProductVersion: 1.0.0.0

----------Imported symbols----------

[IMAGE_IMPORT_DESCRIPTOR]
0x1B385    0x0   OriginalFirstThunk:           0x1D1AD
0x1B385    0x0   Characteristics:              0x1D1AD
0x1B389    0x4   TimeDateStamp:                0x0        [Thu Jan  1 00:00:00 1970 UTC]
0x1B38D    0x8   ForwarderChain:               0x0
0x1B391    0xC   Name:                        0x1D1C7
0x1B395    0x10  FirstThunk:                   0x2000

mscoree.dll._CorExeMain Hint[0]

----------Resource directory----------

[IMAGE_RESOURCE_DIRECTORY]
0x1B400    0x0   Characteristics:              0x0
0x1B404    0x4   TimeDateStamp:                0x0        [Thu Jan  1 00:00:00 1970 UTC]
0x1B408    0x8   MajorVersion:                 0x0
0x1B40A    0xA   MinorVersion:                 0x0
0x1B40C    0xC   NumberOfNamedEntries:         0x0
0x1B40E    0xE   NumberOfIdEntries:           0x2
  Id: [0x10] (RT_VERSION)
  [IMAGE_RESOURCE_DIRECTORY_ENTRY]
```

```
  [IMAGE_RESOURCE_DIRECTORY_ENTRY]
  0x1B410    0x0   Name:                          0x10
  0x1B414    0x4   OffsetToData:                  0x80000020
    [IMAGE_RESOURCE_DIRECTORY]
    0x1B420    0x0   Characteristics:              0x0
    0x1B424    0x4   TimeDateStamp:                0x0        [Thu Jan  1 00:00:00 1970 UTC]
    0x1B428    0x8   MajorVersion:                0x0
    0x1B42A    0xA   MinorVersion:                0x0
    0x1B42C    0xC   NumberOfNamedEntries:        0x0
    0x1B42E    0xE   NumberOfIdEntries:           0x1
      Id: [0x1]
      [IMAGE_RESOURCE_DIRECTORY_ENTRY]
      0x1B430    0x0   Name:                        0x1
      0x1B434    0x4   OffsetToData:                0x80000038
        [IMAGE_RESOURCE_DIRECTORY]
        0x1B438    0x0   Characteristics:            0x0
        0x1B43C    0x4   TimeDateStamp:              0x0        [Thu Jan  1 00:00:00 1970 UTC]
        0x1B440    0x8   MajorVersion:              0x0
        0x1B442    0xA   MinorVersion:              0x0
        0x1B444    0xC   NumberOfNamedEntries:      0x0
        0x1B446    0xE   NumberOfIdEntries:         0x1
        \--- LANG [0,0][LANG_NEUTRAL,SUBLANG_NEUTRAL]
          [IMAGE_RESOURCE_DIRECTORY_ENTRY]
          0x1B448    0x0   Name:                      0x0
          0x1B44C    0x4   OffsetToData:             0x80
            [IMAGE_RESOURCE_DATA_ENTRY]
            0x1B480    0x0   OffsetToData:              0x1E090
            0x1B484    0x4   Size:                      0x32C
            0x1B488    0x8   CodePage:                  0x0
            0x1B48C    0xC   Reserved:                  0x0

  Id: [0x18] (RT_MANIFEST)
  [IMAGE_RESOURCE_DIRECTORY_ENTRY]
  0x1B418    0x0   Name:                          0x18
  0x1B41C    0x4   OffsetToData:                  0x80000050
    [IMAGE_RESOURCE_DIRECTORY]
    0x1B450    0x0   Characteristics:              0x0
    0x1B454    0x4   TimeDateStamp:                0x0        [Thu Jan  1 00:00:00 1970 UTC]
    0x1B458    0x8   MajorVersion:                0x0
    0x1B45A    0xA   MinorVersion:                0x0
    0x1B45C    0xC   NumberOfNamedEntries:        0x0
    0x1B45E    0xE   NumberOfIdEntries:           0x1
      Id: [0x1]
      [IMAGE_RESOURCE_DIRECTORY_ENTRY]
      0x1B460    0x0   Name:                        0x1
      0x1B464    0x4   OffsetToData:                0x80000068
        [IMAGE_RESOURCE_DIRECTORY]
        0x1B468    0x0   Characteristics:            0x0
        0x1B46C    0x4   TimeDateStamp:              0x0        [Thu Jan  1 00:00:00 1970 UTC]
        0x1B470    0x8   MajorVersion:              0x0
        0x1B472    0xA   MinorVersion:              0x0
        0x1B474    0xC   NumberOfNamedEntries:      0x0
        0x1B476    0xE   NumberOfIdEntries:         0x1
        \--- LANG [0,0][LANG_NEUTRAL,SUBLANG_NEUTRAL]
          [IMAGE_RESOURCE_DIRECTORY_ENTRY]
          0x1B478    0x0   Name:                      0x0
          0x1B47C    0x4   OffsetToData:             0x3BC
            [IMAGE_RESOURCE_DATA_ENTRY]
            0x1B7BC    0x0   OffsetToData:              0x1E3CC
            0x1B7C0    0x4   Size:                      0x1EA
            0x1B7C4    0x8   CodePage:                  0x0
            0x1B7C8    0xC   Reserved:                  0x0


----------Debug information----------

[IMAGE_DEBUG_DIRECTORY]
0x1B2EC    0x0   Characteristics:            0x0
0x1B2F0    0x4   TimeDateStamp:              0xA123ADCA [Thu Sep  2 03:53:46 2055 UTC]
0x1B2F4    0x8   MajorVersion:              0x0
0x1B2F6    0xA   MinorVersion:              0x0
0x1B2F8    0xC   Type:                      0x2
0x1B2FC    0x10  SizeOfData:                0x61
0x1B300    0x14  AddressOfRawData:          0x1D124
0x1B304    0x18  PointerToRawData:          0x1B324
Type: IMAGE_DEBUG_TYPE_CODEVIEW
```

```
    [CV_INFO_PDB70]
    0x1B324     0x0   CvSignature:                  0x53445352
    0x1B328     0x4   Signature_Data1:              31BD56E0
    0x1B32C     0x8   Signature_Data2:              352D
    0x1B32E     0xA   Signature_Data3:              4D07
    0x1B330     0xC   Signature_Data4:              998C76A1EB1727C7
    0x1B338     0x14  Age:                          0x1
    0x1B33C     0x18  PdbFileName:
C:\Users\User\Downloads\ToxicEye\TelegramRAT\obj\Release\TelegramRAT.pdb

[IMAGE_DEBUG_DIRECTORY]
0x1B308     0x0   Characteristics:              0x0
0x1B30C     0x4   TimeDateStamp:                0x0        [Thu Jan  1 00:00:00 1970 UTC]
0x1B310     0x8   MajorVersion:                 0x0
0x1B312     0xA   MinorVersion:                 0x0
0x1B314     0xC   Type:                         0x10
0x1B318     0x10  SizeOfData:                   0x0
0x1B31C     0x14  AddressOfRawData:             0x0
0x1B320     0x18  PointerToRawData:             0x0
Type: IMAGE_DEBUG_TYPE_REPRO

----------Base relocations----------

[IMAGE_BASE_RELOCATION]
0x1BA00     0x0   VirtualAddress:               0x1D000
0x1BA04     0x4   SizeOfBlock:                  0xC
    0001D1DCh HIGHLOW
    0001D000h ABSOLUTE
```

Una volta in possesso del file PE di ToxicEye, esso viene sottoposto alla procedura di estrazione delle features da studiare.

In [ ]:

```
toxiceye = extract_features_from_executable("TelegramRAT.exe",
"CyberSecStaticMalwareAnalysis/TelegramRAT.exe")
toxiceye['Malware'] = 1
toxiceye.head()
```

Out[ ]:

	Name	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp	e_csum	e_ip	e_cs	e_lfarlc	e_c
0	TelegramRAT.exe	23117	144	3	0	4	0	65535	0	184	0	0	0	64	

Successivamente il dataset, in formato .csv, viene caricato in un DataFrame della libreria Pandas, in modo da semplificarne l'utilizzo. Il record relativo a ToxicEye viene introdotto nel dataset.

In [ ]:

```
malwares_dframe = pd.read_csv('CyberSecStaticMalwareAnalysis/dataset_malwares.csv')
malwares_dframe = malwares_dframe.append(toxiceye, ignore_index=True)
malwares_dframe
```

Out[ ]:

	Name	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp
0	VirusShare_a878ba26000edaac5c98eff4432723b3	23117	144	3	0	4	0	65535	0	184
1	VirusShare_ef9130570fddc174b312b2047f5f4cf0	23117	144	3	0	4	0	65535	0	184
2	VirusShare_ef84cdeba22be72a69b198213dada81a	23117	144	3	0	4	0	65535	0	184
3	VirusShare_6bf3608e60ebc16cbcff6ed5467d469e	23117	144	3	0	4	0	65535	0	184
4	VirusShare_2cc94d952b2efb13c7d6bbe0dd59d3fb	23117	144	3	0	4	0	65535	0	184
...	...	...	...	...	...	...	...	...	...	...
19607	VNC-Server-6.2.0-Windows.exe	23117	144	3	0	4	0	65535	0	184
19608	Microsoft.GroupPolicy.Management.ni.dll	23117	0	0	0	0	0	0	0	0
19609	cryptuiwizard.dll	23117	144	3	0	4	0	65535	0	184

	Name	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp
19610	winhttp.dll	23117	144	3	0	4	0	65535	0	184
19611	TelegramRAT.exe	23117	144	3	0	4	0	65535	0	184

19612 rows × 79 columns

Dallo studio dei valori univoci della colonna 'Malware' possiamo osservare come il dataset sia composto per la maggiore da record relativi a eseguibili malevoli.

In [ ]:

```
malwares_dframe.Malware.value_counts()
```

Out[ ]:

```
1    14600
0     5012
Name: Malware, dtype: int64
```

Come anticipato precendentemente, quasi tutti i valori della feature *SuspisciousNameSection* sono uguali a 0 (con media 0.02).

In [ ]:

```
malwares_dframe["SuspiciousNameSection"].describe()
```

Out[ ]:

```
count    19612.000000
mean         0.018152
std          0.183109
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          4.000000
Name: SuspiciousNameSection, dtype: float64
```

Per facilitare l'addestramento futuro del riconoscitore, il dataset viene sottoposto a standardizzazione tramite l'utilizzo della classe StandardScaler, fornita dalla libreria *SciKitLearn*.

In [ ]:

```
from sklearn.preprocessing import StandardScaler

X = malwares_dframe.drop(["Name","Malware"], axis=1)
scaler = StandardScaler().fit(X)
X_standardized = pd.DataFrame(scaler.transform(X), columns=X.columns)
X_standardized
```

Out[ ]:

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp	e_csum	e_ip	e_cs
0	0.0	-0.035065	-0.047510	-0.040544	-0.038600	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868
1	0.0	-0.035065	-0.047510	-0.040544	-0.038600	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868
2	0.0	-0.035065	-0.047510	-0.040544	-0.038600	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868
3	0.0	-0.035065	-0.047510	-0.040544	-0.038600	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868
4	0.0	-0.035065	-0.047510	-0.040544	-0.038600	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868
...	...	...	...	...	...	...	...	...	...	...	...	...
19607	0.0	-0.035065	-0.047510	-0.040544	-0.038600	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868
19608	0.0	-	-	-	-0.043228	-0.040436	-7.044645	-	-	-	-	-

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp	e_csum	e_ip	e_cs	
		0.180939	0.049586	0.040544				0.016353	0.181226	0.029242	0.058929	0.055868	0
19609	0.0	-0.035065	-0.047510	-0.040544	-0.038600	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868	0
19610	0.0	-0.035065	-0.047510	-0.040544	-0.038600	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868	0
19611	0.0	-0.035065	-0.047510	-0.040544	-0.038600	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868	0

19612 rows × 77 columns

Il dataset viene infine diviso in una porzione dedicata unicamente all'addestramento e ad una tramite la quale valutare la capacità di generalizzazione del classificatore addestrato con la prima parte. Il record relativo a ToxicEye è stato introdotto nella porzione di addestramento e, per poter replicare in futuro la divisione, è stato definito un seme predefinito per la funzione che fornisce casualità.

In [ ]:

```
from sklearn.model_selection import train_test_split

train_X, test_X, train_y, test_y = train_test_split(
    X_standardized, malwares_dframe["Malware"], test_size=.3, random_state=42)
```

Per poter operare più comodamente su ToxicEye, lo StandardScaler usato in precedenza viene applicato sul relativo record.

In [ ]:

```
toxiceye_drop = toxiceye.drop(["Name", "Malware"], axis=1)
toxiceye_standardized = pd.DataFrame(scaler.transform(toxiceye_drop),
columns=toxiceye_drop.columns)
toxiceye_standardized.head()
```

Out[ ]:

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp	e_csum	e_ip	e_cs	e_lfa
0	0.0	-0.035065	-0.04751	-0.040544	-0.0386	-0.040436	0.148864	-0.016353	-0.033981	-0.029242	-0.058929	-0.055868	0.0532

Esplorazione del dataset

Per avere un'idea più precisa del dataset e della sua composizione, sono state applicate alcune procedure. Un'analisi di questo tipo permette di formare un'idea qualitativa, ad alto livello di astrazione, del dataset e di poter, sulla base di questa, effettuare tutte le necessarie future considerazioni.

Usiamo *Principal Component Analysis* (*PCA*) per poter rappresentare l'intero dataset in uno spazio bidimensionale e, quindi, osservare, in generale, come si comporta *ToxicEye* in relazione agli altri eseguibili. Tramite una rappresentazione di questo tipo è possibile, in alcuni particolari casi, individuare alcuni pattern e, quindi, semplificare il lavoro.

In [ ]:

```
import sklearn
import seaborn as sns
```

In [ ]:

```
from sklearn import decomposition
pca = decomposition.PCA(n_components = 2, random_state = 39)
pca_X = pca.fit_transform(train_X)

pca_toxic = pca.transform(toxiceye_standardized)
```

In [ ]:

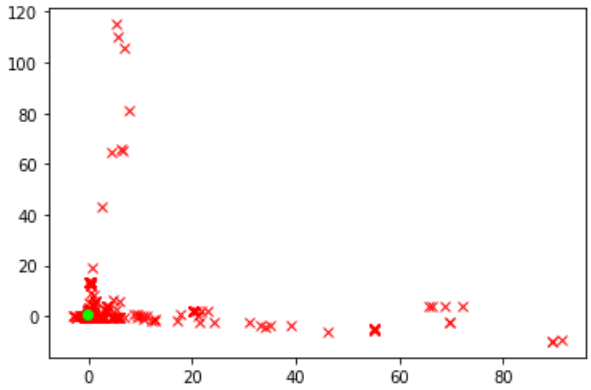
```
import matplotlib.pyplot as plt

plt.plot(pca_X[:, 0], pca_X[:, 1], 'x', color=[1, 0, 0])
```

```
plt.plot(pca_X[:,0], pca_X[:,1], 'x', color=[1,0,0])
plt.plot(pca_toxic[:,0], pca_toxic[:,1], 'o', color=[0,1,0]) #ToxicEye è in verde
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fc751c1db10>]
```



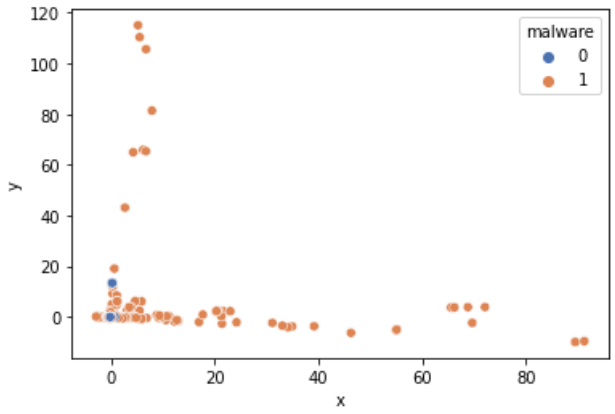
In [ ]:

```
tmp_df = pd.DataFrame({
    'x': pca_X[:, 0],
    'y': pca_X[:, 1],
    'malware': train_y,
})

sns.scatterplot(data=tmp_df, x='x', y='y', hue="malware", palette='deep')
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc75c7aaf90>
```

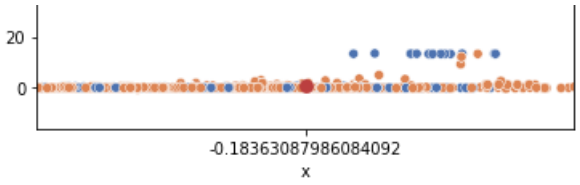


In [ ]:

```
sns.scatterplot(data=tmp_df, x='x', y='y', hue="malware", palette='deep')
sns.pointplot(data=pd.DataFrame({'x': pca_toxic[:,0], 'y': pca_toxic[:,1], 'malware': [1]}),
  x='x', y='y', color='#bb3f3f'
)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc750bdec10>
```



Per quanto è possibile osservare non sembrano essere presenti evidenti relazioni fra eseguibili malevoli e non in uno spazio bidimensionale.

Metriche

Per la valutazione dei vari modelli viene utilizzato il punteggio f1 (media armonica fra precision e recall). Di seguito vengono riportate le definizioni delle metriche e ne viene descritta l'implementazione.

- F1-score: $2\frac{\text{precision} \cdot \text{recall}}{\text{precision}+\text{recall}}$
- Precision: $\frac{tp}{tp+fp}$
- Recall (sensitivity): $\frac{tp}{tp+fn}$

In [ ]:

```
# From https://datascience.stackexchange.com/questions/45165/how-to-get-accuracy-f1-precision-and-recall-for-a-keras-model
# on 10th May 2020
from keras import backend as K

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Costruzione del classificatore

Il modello scelto per la classificazione dei malware è una rete neurale profonda, composta da un'alternanza di livelli densi e di livelli di dropout. Per l'implementazione della rete è stata utilizzata la libreria *Keras*, utilizzante *Tensorflow* come backend. Il classificatore è stato pensato per classificare in modo corretto e deciso il record relativo a ToxicEye ed è stato ottenuto come conseguenza di molteplici design. Tramite un processo di questo tipo è stato possibile osservare, tramite Tensorboard, come la gran parte delle operazioni fosse ottimizzabile tramite l'utilizzo di TPU, processori pensati per un'esecuzione efficiente delle operazioni su tensori.

In [ ]:

```
load = True
if load:
  dnn = tf.keras.models.load_model(
      'CyberSecStaticMalwareAnalysis/dnn',
      custom_objects = {
          'f1_m': f1_m,
          'precision_m': precision_m,
          'recall_m': recall_m,
        }
    )
```

```
else:
  dnn = tf.keras.Sequential([
                            tf.keras.layers.Dense(256, activation = 'relu', input_dim = train_X.sha
e[1]),
                            tf.keras.layers.Dropout(rate=.2),
                            tf.keras.layers.Dense(256, activation = 'relu'),
                            tf.keras.layers.Dropout(rate=.2),
                            tf.keras.layers.Dense(256, activation = 'relu'),
                            tf.keras.layers.Dropout(rate=.2),
                            tf.keras.layers.Dense(1, activation = 'sigmoid'),
  ])
  dnn.compile(
      loss='binary_crossentropy',
      optimizer='adam',
      metrics=[
               'acc',
               f1_m,
               precision_m,
               recall_m
               ]
    )

  dnn.fit(
      train_X, train_y,
      batch_size=64, epochs=50,
      validation_data = (test_X, test_y),
      callbacks = [
                    tf.keras.callbacks.EarlyStopping(
                        monitor='val_loss',
                        patience=3,
                        restore_best_weights=False,
                    ),
                    tensorboard_callback,
                  ],
  )
```

In [ ]:

```
#%tensorboard --logdir logs/fit
```

Verifichiamo che il modello classifichi il nostro malware come tale.

In [ ]:

```
dnn.predict(toxiceye_standardized), np.round(dnn.predict(toxiceye_standardized))
```

Out[ ]:

```
(array([[0.99999475]], dtype=float32), array([[1.]], dtype=float32))
```

Di seguito l'istanza viene salvata in modo persistente, garantendo di poter lavorare sempre con lo stesso classificatore.

In [ ]:

```
save = False
if save:
  dnn.save('CyberSecStaticMalwareAnalysis/dnn')
```

Fast Gradient Sign Method

Una volta definito il modello, in grado di classificare correttamente ToxicEye come malware, questo è stato sottoposto all'attacco **Fast Gradient Sign Method** (**FGM**), la cui implementazione è stata resa disponibile dalla libreria Adversarial-Robustness-Toolbox. Al fine di costruire una versione elusiva del malware ToxicEye, il risultato di questo metodo fornisce delle importantissime indicazioni. Consente, infatti, di studiare il comportamento del classificatore a partire da una funzione d'errore e di individuare le direzioni da seguire e i valori che le feature dovranno assumere durante la fase di costruzione. Dato l'utilizzo di solamente due classi da parte del classificatore, la funzione d'errore utilizzata è la Binary Cross-Entropy.

In [ ]:

```
from art.estimators.classification import TensorFlowV2Classifier
from art.attacks.evasion import FastGradientMethod
classifier = TensorFlowV2Classifier(model=dnn, nb_classes=2, input_shape=train_X.shape[1], loss_obj
ect=tf.keras.losses.binary_crossentropy)

attacker = FastGradientMethod(estimator = classifier, minimal=True, eps=.5)

toxiceye_adv = attacker.generate(x = toxiceye_standardized.to_numpy())
```

Una volta ottenuta l'istanza per mezzo dello studio del gradiente, questa è stata riportata ad una versione precedente la standardizzazione, in modo da poterne studiare i valori reali.

In [ ]:

```
toxiceye_adv_reverted = scaler.inverse_transform(toxiceye_adv)
toxiceye_adv_reverted = pd.DataFrame(data=toxiceye_adv_reverted, columns=toxiceye.drop(['Name',
'Malware'], axis=1).columns)
toxiceye_adv_reverted
```

Out[ ]:

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp	e_csum	e_ip
0	23116.5	-349.575211	-719.559684	-606.07008	-428.235678	-457.893239	60979.851771	-318.541891	-440.808322	-507.625835	-913.451455

Una volta ottenuto il record avente valori reali, questo è stato confrontato con il record originale, in modo da avere indicazioni sul segno effettivo del gradiente su ogni singola feature.

In [ ]:

```
delta = toxiceye_adv_reverted - toxiceye.drop(['Name', 'Malware'], axis=1)
delta
```

Out[ ]:

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp	e_csum	e_ip
0	-0.5	-493.575211	-722.559684	-606.07008	-432.235678	-457.893239	-4555.148229	-318.541891	-624.808322	-507.625835	-913.451455

Tramite attacco Fast Gradient Sign è stato generato un record classificato come non malevolo a partire da dati relativi ad un malware.

In [ ]:

```
dnn.predict(toxiceye_adv) # deve essere classificato come NON malware
```

Out[ ]:

```
array([[1.7849335e-09]], dtype=float32)
```

Questa prima istanza elusiva è stata ottenuta supponendo di poter modificare in modo arbitrario ogni valore di un dato file PE. In generale, questo non è assolutamente il caso; da una procedura di questo tipo vorremo avere non solo informazioni attendibili sulla natura della modifica di una data feature ma anche dei valori che essa debba assumere. Per fare questo abbiamo sottoposto ogni feature ad analisi, al fine di studiare la possibilità effettiva di modifica secondo la direzione indicata dal gradiente e l'estrazione di un sottoinsieme di features, facilmente modificabili, tramite le quali sia comunque possibile costruire un'istanza in grado di illudere il classificatore. In quest'analisi abbiamo tenuto conto dei vincoli imposti dal formato PE e, in generale, delle informazioni date dalla prima istanza di attacco. L'obiettivo ultimo dell'analisi è la costruzione di una maschera che possa identificare le features sulle quali montare effettivamente un attacco realistico.

Analisi di modificabilità delle features

Come descritto in precedenza, lo scopo di quest'analisi è la costruzione di una maschera, array contenente valori booleani, che permetta di filtrare features per le quali la modifica risulti essere irrealizzabile o estremamente complessa, in modo da concentrare gli

sforzi del *FGM* su quelle features effettivamente modificabili.

L'analisi delle features viene effettuata a partire dal loro valore nel record originale e del valore nel record ottenuto tramite FGM, nonchè dal loro significato. Dai due valori numerici vengono estratte informazioni relative alla dimensione del cambiamento e al segno generale del gradiente. Di seguito viene costruito un report in modo da poter accedere a queste informazioni in modo immediato. Informazioni sul significato dei vari campi sono state ricavate dalla documentazione fornita da Microsoft.

Durante l'analisi sono state effettuate alcune considerazioni sul segno del gradiente, dipendente dalla funzione d'errore. Queste considerazioni sono state effettuate supponendo che il gradiente calcolato su una data feature mantenga lo stesso comportamento al variare dell'insieme di features sulle quali l'attacco viene montato. Non potendo dare garanzie su questo tipo di comportamento, queste considerazioni sono di natura altamente speculativa.

In [ ]:

```
feature_analysis = dict()
report = ['ToxicEye evasion: Changes']

for i, c in enumerate(toxiceye.drop(['Name', 'Malware'], axis=1).columns):
  report.append(f'Column: {c}')
  feature_analysis[c] = dict(
      name = c,
      original_value = toxiceye.loc[0, (c)],
      adversarial_value = toxiceye_adv_reverted.loc[0, (c)],
      delta = toxiceye_adv_reverted.loc[0, (c)] - toxiceye.loc[0, (c)],
      gradient_sign = "+" if toxiceye_adv[0][i] - toxiceye_standardized.loc[0, (c)] > 0 else "-",
  )

  report.append(f'\tOriginal version: {toxiceye.loc[0, (c)]}, Evasion version:
{toxiceye_adv_reverted.loc[0, (c)]}')
  report.append(f'\tDelta (change to original version): {toxiceye_adv_reverted.loc[0, (c)] -
toxiceye.loc[0, (c)]}')
  report.append(f'\tGradient sign : {"+" if toxiceye_adv[0][i] - toxiceye_standardized.loc[0, (c)]
> 0 else "-"}')
  if toxiceye_adv_reverted.loc[0, (c)] < 0:
    report.append(f'\t! - ValueError for Evasion version: restore the default value')

  report.append('-'*40)

with open('changes_report.txt', 'w') as f:
  f.write('\n'.join(report))
```

Le prime features oggetto di analisi sono quelle per le quali il valore nell'istanza elusiva risulta essere minore di 0, per le quali il gradiente assume segno negativo. Ricordando il metodo tramite il quale viene formata una nuova istanza secondo *FGM*,

$$X_* = X + \epsilon * sign(\nabla_x J(X, y_{true}))$$

si consideri come sono stati ottenuti i valori delle features: la prima istanza elusiva è stata generata usando un ϵ relativamente basso (fattore di moltiplicazione del gradiente studiato a partire dalla funzione d'errore), pari a $0,5$. Inoltre, i vincoli sul formato PE non consentono l'inserimento di valori negativi. Sicuramente altre features non potranno essere soggette a modifica, causa vincoli imposti dal formato PE; è quindi ragionevole supporre che, per ottenere una nuova istanza elusiva, escludendo le features non modificabili, sarà necessario dover aumentare il fattore moltiplicativo del gradiente, fornito al *FGM*. Seguendo la direzione del gradiente e osservando come per ϵ relativamente esigui i valori assunti risultino irrealistici, è possibile convenire che, al crescere di ϵ, si otterranno comunque valori la cui modifica risulta essere irrealizzabile.

Alla luce di queste considerazioni, abbiamo deciso di non tenere conto di queste features per attacchi futuri. La maschera assumerà, per queste features, valore False.

L'analisi tiene ora conto delle features ottenute dagli header MS-DOS. Il cambiamento del valore di questi campi può avvenire solamente tramite modifica diretta dell'eseguibile, la quale deve essere tale da garantirne il funzionamento. La maggior parte di queste features ricade nel caso descritto in precedenza (valori irrealizzabili e gradiente negativo). Particolare è la features *e_magic*: il formato PE richiede essa abbia sempre valore fisso, pari a *0x5A4D* (*23117*), mentre l'istanza elusiva suggerisce di modificarne il valore. Abbiamo quindi deciso di non considerare queste features in attacchi futuri.

Così come *e_magic*, anche la feature *Machine* può assumere valori pre-stabiliti, vincolati dal formato PE. *Machine* indica il tipo di macchina per la quale è stato compilato l'eseguibile, nel nostro caso *0x14C* (Processori Intel 386 e compatibili). Modifiche di questa feature comprometterebbero fortemente il corretto funzionamento dell'eseguibile.

L'analisi ha reso evidente la presenza di campi con valori predefiniti. Eventuali modifiche per questi campi verranno attuate tenendo conto dei potenziali valori ammissibili e della direzione del gradiente, nonchè del significato implicito.

SectionMinEntropy indica l'entropia minore tra tutte le sezioni. Voler introdurre cambiamenti di valore per questa feature comporterebbe una combinazione di modifiche a sezioni già presenti e l'introduzione di nuove sezioni. Se il codice fosse scritto in C++ questo non sarebbe un problema, si potrebbe infatti usare la direttiva del compilatore

```
#pragma data_section("nome_sezione")
```

per introdurre un gran numero di sezioni e adattarle in modo da portare la media dell'entropia il più vicina possibile al valore richiesto, circa pari ad 1. Il compilatore C# non riconosce questa direttiva, non consentendo di inserire con facilità nuove sezioni. L'approccio alternativo comporterebbe la modifica del codice sorgente in modo da causare modifica indiretta dell'entropia generale. Data la composizione delle sezioni che formano PE standard per codice C#, seguire questa strada è risulta essere estremamente complesso. Abbiamo quindi deciso di escludere questa feature da attacchi futuri.

SectionMaxChar indica il valore di Characteristics maggiore tra tutte le sezioni, il gradiente indica di abbassarlo. La sezione maggiore ha come valore 0x60000020, somma dei seguenti flags: IMAGE_SCN_CNT_CODE, IMAGE_SCN_MEM_EXECUTE, IMAGE_SCN_MEM_READ. Questi flag indicano la presenza di codice nella sezione, se la sezione è eseguibile e se può essere letta. Una modifica di questi valori verrebbe ottenuta modificando questi flag, compromettendo la corretta esecuzione dell'eseguibile.

SectionMaxVirtual fa riferimento al massimo valore dell'indirizzo virtuale all'interno delle sezioni. La direzione del gradiente richiede di incrementarne il valore, rendendo la feature appetibile a modifiche, le quali però risultano non immediata da applicare.

Anche per la feature *SectionMaxPhysical* il gradiente assume valore positivo, richiedento di aumentarne il valore. Una modifica di questo parametro si otterrebbe andando ad introdurre nuove sezioni, aventi VirtualSize maggiore delle altre. Secondo le considerazioni fatte in precedenza, l'introduzione di nuove sezioni risulta essere complessa non potendo fornire direttive direttamente sul compilatore in questo senso.

SectionMinVirtualsize descrive la minima dimensione virtuale assunta dalle sezioni quando queste sono caricate in memoria, secondo il gradiente questa dimensione deve essere aumentata. Ragionando sul suo significato e studiandone il peso sulla successiva classificazione, abbiamo convenuto che eventuali modifiche di questa feature non abbiano peso elevato e possono essere ottenute in modo naturale lungo la dimensione suggerita dal gradiente tramite l'introduzione di nuove istruzione operando dal codice sorgente.

SectionMinRawSize indica la dimensione minima delle sezioni in caso di file descriventi oggetti o, per file immagini, la minima dimensione su disco dei dati inizializzati. Il segno del gradiente suggerisce di aumentarne la dimensione, da alcuni esperimenti sul suo valore abbiamo osservato come mantere il valore originale porti ad una minore confidenza sulla classificazione. Abbiamo quindi deciso di non considerare questa feature come punto di attacco, eventuali cambiamenti a seguito di introduzione di nuove istruzioni seguirebbero comunque la direzione dettata dal segno del gradiente.

Per quanto riguarda *Characteristics*, questa feature contiene flags che descrivono le proprietà dell'eseguibile. Nel malware è 0x0022 (34), cioè somma di due flags: *IMAGE_FILE_LARGE_ADDRESS_AWARE* (0x0020) & *IMAGE_FILE_EXECUTABLE_IMAGE* (0x0002). Il compilatore C# non permette di specificare queste flags.

Magic assume valori interi che identificano lo stato di un'immagine. Il nostro malware ha come valore 0x10b (267), identifica cioè un normale eseguibile.

DLLCharacteristics contiene invece dei flags che identificano le caratteristiche dell'immagine e delle DLL. Non può essere cambiato tramite compilatore C#.

Le features *SizeOfStackCommit* e *SizeOfHeapReserve* indicano rispettivamente la dimensione dello stack e la dimensione dell'heap di riserva. Mentre il compilatore di C++ permette di indicare la dimensione di questi campi, il compilatore di C# non prevede l'assegnamento di un valore da parte di utenti esterni.

DirectoryEntryImport e *DirectoryEntryImportSize* fanno riferimento alla quantità di DLL importate e al numero di funzioni di queste DLL importate. Da quanto abbiamo potuto osservare l'unica DLL importata dall'eseguibile originale era *mscoree.dll* e l'introduzione di nuove .dll sia nel file .csproj, contenente le informazioni di compilazione, che nel codice sorgente non sembrano portare alcun cambiamento a questi parametri.

NumberOfSections e *SectionsLength* indicano il numero di sezioni presenti nel file eseguibile. Ricordando l'analisi della feature *SectionMinEntropy*, C# non permette di richiedere l'introduzione di nuove sezioni.

Per quanto riguarda le features *MajorSubsystemVersion* e *MinorSubsystemVersion*, queste indicano la versione minima di Subsystem e versione maggiore di Subsystem. Una modifica di questi valori potrebbe compromettere la corretta esecuzione dell'eseguibile.

MajorLinkerVersion e *MinorLinkerVersion* indicano la versione minore e maggiore del Linker.

La feature *DirectoryEntryExport* fa riferimento alla sezione .edata, contiene informazioni sui simboli a cui altre immagini possono accedere tramite collegamento dinamico. Vale lo stesso discorso di *DirectoryEntryImport*.

ImageDirectoryEntryImport indica indirizzo della sezione .idata, non facilmente modificabile.

ImageDirectoryEntryResource fa riferimento all'indirizzo della sezione .rsrc, non facilmente modificabile.

ImageDirectoryEntryException contiene l'indirizzo della sezione .pdata, come le precedenti la sua modifica non risulta essere immediata.

La feature *SizeOfCode* indica la dimensione della sezione del codice (testo) o la somma di tutte le sezioni del codice se sono presenti più sezioni. Per modificare questo parametro, basta aggiungere del padding all'interno del codice sorgente.

SizeOfImage indica la dimensione (in byte) dell'immagine, inclusi tutte gli header, quando l'immagine viene caricata in memoria. Aumentando la dimensione del codice SizeOfCode, aumenta di conseguenza anche la dimensione dell'immagine.

TimeDateStamp contiene il timestamp che indica quando il file è stato creato. Facilmente modificabile senza andare ad intaccare il funzionamento dell'eseguibile.

Al termine dell'analisi, *TimeDateStamp* e *SizeOfCode* sono risultate essere le features più proni a modifiche e, quindi, atte ad essere utilizzate nel montare effettivamente l'attacco. Pensiamo che queste features siano ideali data la facilità tramite la quale è possibile introdurre modifiche, siano esse in eccesso o in difetto rispetto al valore originale, e la natura delle modifiche, ottenuta osservando l'istanza elusiva iniziale. A queste due è stata inoltre aggiunta la feature *SizeOfImage*, la quale varia naturalmente al variare di *SizeOfCode*. In base a questa analisi, abbiamo costruito una maschera in modo da montare un attacco sulla base delle sole features effettivamente modificabili.

In [ ]:

```
change_features = ['TimeDateStamp', 'SizeOfCode', 'SizeOfImage' ]

attacker = FastGradientMethod(estimator = classifier, minimal=True, eps=6)

mask = np.array([], dtype='bool')

for f in toxiceye_standardized.columns:
  if f in change_features:
    mask = np.append(mask, True)
  else:
    mask = np.append(mask, False)

toxiceye_adv = attacker.generate(x = toxiceye_standardized.to_numpy(), mask=mask)
```

Di seguito viene creato un record, relativo all'istanza elusiva non standardizzata, in modo da poter applicare le considerazioni fatte fino ad ora e facilitare l'estrazione delle features da analizzare.

In [ ]:

```
toxiceye_adv_reverted = scaler.inverse_transform(toxiceye_adv)
toxic_adv_cured = pd.DataFrame(data=toxiceye_adv_reverted, columns=toxiceye.drop(['Name',
'Malware'], axis=1).columns)
toxic_adv_cured
```

Out[ ]:

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp	e_csum	e_ip	e_cs	e_lfarlc	e_ovno	e_oemid	e_
0	23117.0	144.0	3.0	0.0	4.0	0.0	65535.0	0.0	184.0	0.0	0.0	0.0	64.0	0.0	0.0	

Verifichiamo se sono presenti features a valore negativo, e in tal caso riportiamo i valori all'originale.

In [ ]:

```
for c in toxic_adv_cured.columns:
  if toxic_adv_cured.loc[0, (c)] < 0:
    print("{}: reverted to original ({} -> {})".format(
        c,
        toxic_adv_cured.loc[0, (c)],
        toxiceye.loc[0, (c)],
    ))
    toxic_adv_cured.loc[0, (c)] = toxiceye.loc[0, (c)]
```

```
SizeOfImage: reverted to original (-203860727.3466532 -> 139264)
```

Visualizziamo le restati features da modificare, evidenziando il cambiamento necessario ad ottenere una misclassificazione.

In [ ]:

```
for f in change_features:
  original = toxiceye.loc[0, (f)]
  adversarial = toxic_adv_cured.loc[0, (f)]
  if original != adversarial:
    print(f'{f}: {toxiceye.loc[0, (f)]} -> {round(toxic_adv_cured.loc[0, (f)])}')
```

```
TimeDateStamp: 3213165410 -> 1443841756
SizeOfCode: 111104 -> 309840008
```

Dopo aver riportato i dovuti campi dell' header ai valori originali e aver applicato i cambiamenti suggeriti dal Fast Gradient Sign Method, viene verificata l'effettiva errata classificazione.

In [ ]:

```
toxic_adv_cured_scaled = scaler.transform(toxic_adv_cured.to_numpy())
dnn.predict(toxic_adv_cured_scaled), np.round(dnn.predict(toxic_adv_cured_scaled))
```

Out[ ]:

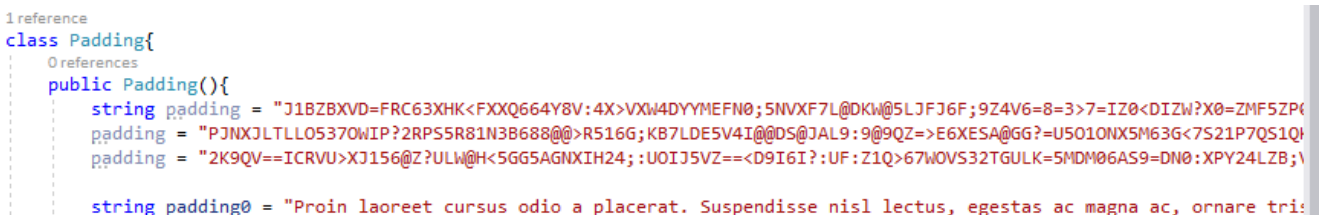
```
(array([[0.38557017]], dtype=float32), array([[0.]], dtype=float32))
```

Costruzione della versione elusiva di Toxic Eye

Individuate le features sulle quali montare l'attacco, il codice sorgente è stato modificato in modo da rispecchiare le modifiche indicate da *FGM*. Abbiamo chiamato l'eseguibile risultato della compilazione come *Toxic Eye Sneaky*, sneaky in quanto in grado di eludere la classificazione a malware. Si noti che, al termine della compilazione, il sistema Windows Defender è stato comunque in grado di identificare correttamente l'eseguibile come malevolo.

Per quanto riguarda la feature *TimeDateStamp*, essa è stata impostata, seguendo le indicazioni fornite da *FGM*, al valore *1443841756*, data corrispondente a `sabato 3 ottobre 2015 05:09:16 GMT+02:00 DST`.

Per quanto riguarda invece *SizeOfCode*, sono state introdotte stringhe randomiche, alfanumeriche, di grandezza variabile, fino all'utilizzo di tutto lo spazio disponibile destinato al salvataggio di stringhe letterali. Per limitare l'impatto sulle prestazioni al momento dell'esecuzione, le variabili sono state introdotte in classi non richiamate dal programma principale, della quale si richiede comunque la compilazione. La randomicità e la diversa grandezza delle stringhe constente di evitare procedure di ottimizzazione da parte del compilatore. Di seguito viene riportata una porzione di codice descrivente le modifiche apportate.



```
        padding = padding0;

        padding0 = "9G5QYL=T99JZUSZ:BUGB177O1MJME<U;DF@PB6<BUWVQ=8QH@VTRM4@67M4UJXNNEGXEK1ITQ0V>7O<IZO06;OT3L85K;0UO6XHAYF

        padding = padding0 + "123";
        padding0 = "2SPMNISRVUPAZ8XX?SC:4TURU18F9JM6VY0XG?GR377QL3YY;374D90NOCV0V@Y0=T7Z58D0R:SCMLF?UID76T1D?4<C>5HHA3@P9(
        padding0 = padding0 + "123";
        padding0 = "<KNQA<P22NW:=8J2NZV5N95S9OBHP5UY3U<TZ2NC0AV>L>7SZ70?Q9LK:K5B2L;UWWW0SX0R7=V=YEH1GM7;WV5JCMAI=HX@BHF9K:
        padding = padding + "123";
        padding0 = padding0 + "123";

        string padding1 = "3:@0C:;7T2:BRC1ESGMNBQVN=32J6UBON7G8US5X13>6XKIL9LY;IY43WBIQ5PQ0PBMVDIA1UDHJ8PY@OVK00LHLI=QA<1
    }
}
```

Verifichiamo che il Timestamp del nostro nuovo eseguibile sia effettivamente uguale a quello indicato da FSG.

In [ ]:

```
pe_adv = pefile.PE("CyberSecStaticMalwareAnalysis/TelegramRATsneaky.exe")
print(pe_adv.FILE_HEADER.TimeDateStamp)
```

```
1443841756
```

Infine, verifichiamo che il nostro malware modificato, ma che mantiene intatte tutte le sue funzioni, sia classificato come non malware, portando a termine con successo l'attacco di misclassificazione.

In [ ]:

```
toxiceye_sne = extract_features_from_executable("TelegramRATsneaky.exe",
"CyberSecStaticMalwareAnalysis/TelegramRATsneaky.exe")
toxiceye_sne = toxiceye_sne.drop(["Name"], axis=1)
toxiceye_sne = scaler.transform(toxiceye_sne)
dnn.predict(toxiceye_sne), np.round(dnn.predict(toxiceye_sne))
```

Out[ ]:

```
(array([[0.262429]], dtype=float32), array([[0.]], dtype=float32))
```

Il nostro nuovo eseguibile viene effettivamente classificato come non malware, riuscendo ad eludere il classificatore.

Conclusioni

In questo elaborato è stata studiata l'implementazione di un attacco a danni di un sistema di classificazione di malware. Una volta ottenuto l'eseguibile malevolo e costruito il classificatore, formato da una rete neurale profonda, esso è stato sottoposto ad un'istanza di *Fast Gradient Sign Method* che, insieme ad un'analisi delle features del dataset di addestramento, ci ha permesso di ottenere indicazioni sulle modifiche da apportare al codice sorgente per ottenere una classificazione errata. Una volta apportate le varie modifiche si è potuto verificare come il nuovo eseguibile fosse effettivamente in grado di eludere il sistema di classificazione.

La ricerca e manipolazione di codice eseguibile malevolo ci ha permesso di formare una prima esperienza diretta, seppur estremamente limitata, sulle difficoltà della gestione di questi tipi di software. L'implementazione dell'attacco ci ha permesso invece di osservare quanto la robustezza di un sistema di riconoscimento sia una proprietà chiave, di altissimo valore.

In generale, l'attacco implementato è stato un attacco di tipo White Box, dove erano noti a priori sia il modello da attaccare che il dataset di addestramento. La procedura diventa sicuramente più complessa nel momento nel quale non sia noto il modello di riconoscimento: sarebbe infatti necessario costruirne un'approssimazione, cercando di non generare il sospetto da parte di eventuali sistemi di monitoraggio.

Lavorando di prima mano sul dataset adottato abbiamo potuto osservare come alcune delle features in esso presenti non portino alcuna informazione utile, come nel caso di *e_magic*. Pensiamo che la rimozione di tutte queste features poco informative e l'utilizzo di metodi di riduzione di dimensionalità, come *Principal Component Analysis*, consentano di ridurre la superficie di attacco e, di conseguenza, fornire una maggiore robustezza al sistema complessivo. Sempre in relazione al dataset, ricercando informazioni relative alla struttura del file PE e al significato dei suoi campi, abbiamo potuto osservare la struttura generale dei file PE minimi, ovvero dotati solamente dei campi strettamente necessari al funzionamento, ottenuta osservando il comportamento del sistema operativo di fronte a file PE soggetti a fuzzing. Alla luce di questo, pensiamo possa essere estremamente interessante ridurre le features del dataset tenendo conto della presenza o meno dei relativi campi nei file PE minimi. Questo pensiamo possa rendere

features del dataset tenendo conto della presenza o meno dei relativi campi nei file PE minimi. Questo pensiamo possa rendere estremamente ardua la procedura di alterazione del codice sorgente, consentendo di ridurre istanze elusive.

Le ultime considerazioni sono invece relative al modello utilizzato. Come descritto in precedenza, non avere a disposizione il modello del classificatore rende più ardua l'implementazione di un attacco. Per quanto riguarda l'architettura del classificatore pensiamo che, in un ambito di analisi statica, l'utilizzo di reti profonde senza livelli convoluzionali e ricorrenti sia l'approccio migliore. Pensiamo inoltre che l'utilizzo di *Generative Adversarial Network* possa portare migliori prestazioni in relazione alla robustezza. Ulteriore approccio che riteniamo possa essere estremamente interessante è l'utilizzo di un *Autoencoder*, in addestrato con sole istanze di malware e la cui classificazione avviene valutando l'errore di ricostruzione generale sul dataset di addestramento e l'errore della singola istanza. Pensiamo che, tramite quest'ultimo metodo, sia più arduo applicare attacchi basati sullo studio del gradiente, anche se la costruzione di un approssimatore è comunque possibile.