

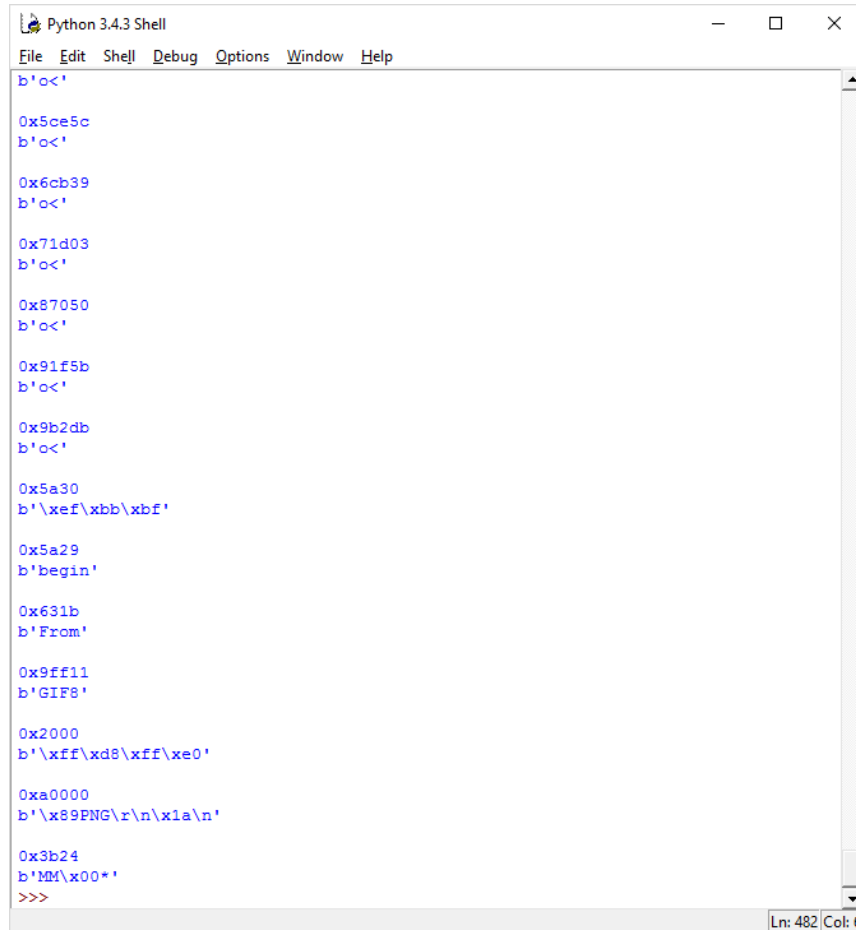
MICT1

EXERCISE WEEK 4

Joeri van Grimbergen (1244825) | Nursize Bilen (1260235) | Rick van Gorp (1328417)
ZUYD HOGESCHOOL | HEERLEN | GROEP 4

1. What we found and where we found it

Voor dit onderzoek is een bestand genaamd “data” geleverd. Dit bestand is representatief voor een onbekend bestandssysteem dat bestaat uit 162 clusters met een grootte van 4k. Het zoeken naar bestanden in het bestand heeft de volgende resultaten opgeleverd:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
b'c<'
0x5ce5c
b'c<'
0x6cb39
b'c<'
0x71d03
b'c<'
0x87050
b'c<'
0x91f5b
b'c<'
0x9b2db
b'c<'
0x5a30
b'\xef\xbb\xbf'
0x5a29
b'begin'
0x631b
b'From'
0x9ff11
b'GIF8'
0x2000
b'\xff\xd8\xff\xe0'
0xa0000
b'\x89PNG\r\n\x1a\n'
0x3b24
b'MM\x00*'
>>>
```

In bovenstaande afbeelding zijn drie bekende bestandsheaders inclusief offset te zien:

- GIF8(9/7);
- FFD8FFE0 (JFIF);
- PNG.

Het grootste bestand in “data” was het JPG bestand met JFIF als magic value. Volgens de specificaties van JFIF eindigt een JPG-bestand met FF D9. Dit JPG-bestand is geschreven op het begin van een cluster dus is **geen** onderdeel van de RAM/Drive-slack. Het bestand met GIF8 als magic value werd geschreven op het resterende deel van het cluster waarin het JPG-bestand eindigt. Daarnaast overlapt het GIF-bestand de sectorgrootte van 512 bytes niet (gezien vanaf het einde van het JPG-bestand). Dit houdt dus in dat het GIF-bestand onderdeel is van de RAM-slack. Het PNG-bestand is geschreven aan het begin van het cluster dus is **geen** onderdeel van de RAM/Drive-slack.

De posities en lengtes van de gevonden data wordt weergegeven in een tabel op de volgende pagina.

Bestand	Begin (Offset hex)	Eind (Offset hex)	Lengte (Dec)	Cluster
JPG-bestand	0x2000	0x9ff11	646929	3 – 159
GIF-bestand	0x9ff11	0x9ff36	37	159
PNG-bestand	0xa0000	0xa0043	67	160

De inhoud van de JPG-afbeelding is onderstaand weergegeven:



De GIF-afbeelding en de PNG-afbeelding bevatten weinig image data. Als deze worden weergegeven bevatten deze enkel een wit puntje.

Daarnaast is het bestand *"data"* bewerkt door de JPG-afbeelding eruit te halen. Dit resulteerde in een kleinere grootte van het bestand *"data"*. Van het gefilterde bestand, genaamd *DATAFILTERED*, werd gecontroleerd of hashes gemaakt van delen van het bestand overeenkomen met de data (hashes) die door de National Software Reference Library is gepubliceerd in haar RDS. Bij het uitvoeren van dit Python script zijn geen matches weergegeven. Er zijn geen resultaten bekend bij gebruik van een *"partial file hashes"*-database. Dit is een database waarbij delen van bestanden zijn gehashed en waarbij de hash gebruikt kan worden ter identificatie van een gedeelte van het bestand.

2. How we found it

Om tot de hierboven beschreven resultaten te komen is er een Python script ontworpen door groep 4. In dit gedeelte zal beschreven worden hoe dit Python script tot stand is gekomen en het zal een korte handleiding bevatten om het Python script te bedienen.

2.1 Handelingen voor resultaten

In dit gedeelte worden de uitgevoerde handelingen beschreven om tot de hierboven beschreven resultaten te komen.

2.1.1 National Software Reference Library

Om dit onderzoek uit te voeren is onder andere gebruik gemaakt van een Python script en de data die wordt geleverd door de National Software Reference Library (RDS). De data die door deze instantie wordt geleverd omvat hashes en metadata van bestanden om te gebruiken voor bestandsidentificatie.

Aan de hand van het Python script en de bestandsgrootte die gedefinieerd werd in het bestand *NSRLFile.txt* van de RDS is per bytegroep die gelijk is aan de bestandsgrootte, per cluster bepaald of er een overeenkomst te vinden is tussen de gegenereerde hashes van de bytegroepen en de hash die bij een specifiek bestand hoort in de RDS. Dit Python script staat uitgelegd in hoofdstuk 2.3.

De referentiedata uit de National Software Reference Library is gedownload via de volgende URL: http://www.nsrl.nist.gov/RDS/rds_2.51/RDS_251.iso. Deze ISO bevat 6 GB aan hashes en andere informatie. Om het bestand te prepareren voor gebruik is de *NSRLFile.txt* waarin de hashes staan geplaatst in dezelfde map als het Python script.

Deze methode heeft echter geen resultaten opgeleverd.

2.1.2 Python script

Om de resultaten in het Python script weer te geven is het Python script uitgevoerd. Het bestand "data" moet in dezelfde map staan als het Python script. In 2.3 Uitleg Python script wordt beschreven hoe het Python script is opgebouwd. In 2.2 Controle uitvoer Python script wordt de controle van de uitvoer van het Python script beschreven. Let op: Er zijn twee python scripts geschreven, namelijk één voor het zoeken van magic values en één voor het vergelijken van hashes.

2.2 Controle uitvoer Python script

De uitvoer van het Python script wordt gecontroleerd aan de hand van een hex-editor (XVI32). De offsets die worden weergegeven door Python moeten representatief zijn voor de locatie van de eerste waarde van de bestandsheader die is gevonden.

Eerst is de bekendste uitvoer gecontroleerd (JPG, GIF, PNG), waarna de overige uitvoer is gecontroleerd. Bij controle van deze uitvoer bleek dat ze onderdeel waren van de drie hierboven genoemde bestanden. Er zijn dus geen additionele bestanden gevonden door dit Python script naast het JPG-, GIF- en PNG-bestand.

Bij het tweede python script is het niet mogelijk om de uitvoer goed te controleren. Er kan een steekproef genomen worden van enkele MD5-hashes uit het bestand *DATAFILTERED*. Dit komt doordat de database van het National Software Reference Library ongeveer 70 – 100 miljoen hashes bevat. Het kost veel tijd om dit manueel door te werken.

2.3 Uitleg Python script

Het eerste Python script is te vinden in de bijbehorende Github-repository van groep 4. Onderstaand worden delen van het script kort uitgelegd.

```
HeaderFinder.py - C:\Users\Rick\Documents\MICT1\week4\HeaderFinder.py (3.4.3)
File Edit Format Run Options Window Help
import re
import binascii

file = open("data", "rb")
content = file.read()

with open("ALLheaders.txt", "r") as f:
    arrHeaders = f.read().splitlines()

for i in range(0, len(arrHeaders)):
    arrHeaders[i] = binascii.unhexlify(arrHeaders[i])

print("Database with known Magic Values loaded:\n" + str(arrHeaders))

for i in range(0, len(arrHeaders)):
    for m in re.finditer(re.compile(re.escape(arrHeaders[i])), content):
        print("\n" + hex(m.start()))
        print(arrHeaders[i])
```

Bovenstaand is het gebruikt Python-script weergegeven. Eerst worden er modules geladen:

- Module re: Deze wordt gebruikt om op basis van reguliere expressies overeenkomsten te vinden in het geladen bestand;
- Module binascii: Deze wordt gebruikt om de hexadecimale data uit het bestand *allheaders* te converteren naar binaire data, leesbaar voor Python.

Vervolgens worden beide bestanden die het betreft geopend en worden de waarden die geladen worden uit het bestand *allheaders.txt* geconverteerd naar binaire data. Vervolgens wordt per *header* uit het bestand *allheader* gecontroleerd of deze header bestaat in het bestand *data*. Als dit het geval is wordt de hexadecimale offset weergegeven inclusief de bijbehorende binaire data.

Het bestand *allheaders.txt* bevat alle hexadecimale waarden van magic values die worden gebruikt om een specifiek type bestand te identificeren. Het bestand gebruikt de volgende syntax per regel: *[Hexadecimal Value]\n*. In deze regel is *\n* representatief voor een nieuwe regel, dit is dus niet zichtbaar als karakter.

Het tweede Python script is te vinden in de bijbehorende Github-repository van groep 4. Onderstaand worden delen van het script kort uitgelegd.

```
import threading
import time
import hashlib
```

In bovenstaande afbeelding wordt weergegeven dat het Python script enkele modules importeert. Dit betreft de volgende modules:

- Threading: Deze module wordt gebruikt om multithreading in Python te gebruiken. Dit houdt in dat meerdere activiteiten op hetzelfde moment kunnen worden uitgevoerd zonder dat de main thread hierdoor onderbroken wordt;
- Time: Deze module wordt gebruikt om de main thread te laten wachten voor enkele seconden. Specifiek voor de *sleep()*-functie;
- Hashlib: Deze module wordt gebruikt om binaire data om te zetten naar een MD5-digest. Deze module is in staat om ook andere hashfuncties toe te passen op binaire data.

```
-
def CheckHash(line):
    try:
        arrHashes = []
        if int(line.split(",")[4]) < 4096: #Size of file must be smaller than 4096 in order to be in RAM/Drive Slack
            with open("DATAFILTERED", "rb") as newfile:
                number = 1
                number2 = 0
                while number < 4:
                    count = int(line.split(",")[4])
                    while count+(number2*4096) < number*4096:
                        newfile.seek(count+(number2*4096))
                        toHash = newfile.read(int(line.split(",")[4]))
                        arrHashes.append(hashlib.md5(toHash).hexdigest().upper())
                        count = count + 1
                        number2 = number2 + 1
                    number = number + 1
                for i in range(0, len(arrHashes)):
                    if arrHashes[i] in line.split(",")[1]:
                        print(line.split(",")[1])
                        print(arrHashes[i])
    except:
        print("Error in thread")
    #Done with thread
```

In bovenstaande afbeelding is de functie *CheckHash* weergegeven. Deze functie heeft de variabele *line* als input. Deze variabele is representatief voor een regel uit het bestand *NSRLFile.txt*. Aan de hand van deze variabele worden de volgende bewerkingen uitgevoerd:

- Er wordt gecontroleerd of de bestandsgrootte kleiner is dan de gedefinieerde clustergrootte van 4096 (4k);
- Per cluster wordt gecontroleerd aan de hand van de bestandsgrootte uit *NSRLFile.txt* of de hash van een bytegroep met dezelfde grootte als de bestandsgrootte overeenkomt met de hash van het bestand dat gedefinieerd staat in *NSRLFile.txt*. Deze controle wordt herhaaldelijk uitgevoerd door steeds één byte verder te verschuiven in het bestand *DATAFILTERED* en daar een hash van te genereren. Deze controle stopt per cluster als de huidige positie + bestandsgrootte groter worden dan de clustergrootte van 4096;
- Er wordt gecontroleerd of de gegenereerde hashes uit het bestand *DATAFILTERED* overeenkomen met de hash uit *NSRLFile.txt*. Als dit het geval is wordt de hash weergegeven in het programma.

```

threads = []

with open("NSRLFile.txt", "r", encoding="latin1") as f:
    for line in f:
        blnActive = False
        while blnActive == False:
            if threading.active_count() > 200:
                blnActive = False
                time.sleep(3)
            else:
                blnActive = True
                t = threading.Thread(target=CheckHash, args=(line,))
                threads.append(t)
                t.start()

#Wait for all threads to finish
for x in threads:
    x.join()

```

In bovenstaand gedeelte worden de zogenoemde *threads* gestart. Per regel uit het bestand *NSRLFile.txt* wordt een thread gemaakt om een vergelijking te maken met de gegenereerde hashes van het bestand *DATAFILTERED*. Het is echter toegestaan om maximaal 200 threads tegelijk uit te voeren. Als dit maximum wordt overschreden wacht de main thread 3 seconden met het vervolg uitvoeren. Vervolgens wordt opnieuw gecontroleerd of er nog steeds meer dan 200 threads actief zijn. Als dit niet het geval is wordt er een nieuwe thread gestart. Threads worden toegepast, zodat er meerdere bewerkingen tegelijkertijd kunnen plaatsvinden.

In de onderste *for-loop* wordt gecontroleerd of alle threads beëindigd zijn. Als hier sprake van is wordt de main thread beëindigd.