Hertentamen SMP-Python 05-07-2022

Belangrijk

Voor deze opgave krijg je 8 uur de tijd. Gedurende deze acht uur mag je uit een veelvoud aan bronnen putten om tot een juiste oplossing te komen: de slides van dit en vorig jaar, de (opgenomen) hoorcolleges, en zelfs digitale bronnen zoals de Python documentatie. Alles mag, zolang je zelf én zelfstandig de opgave maakt - samenwerken en simpelweg code kopiëren en plakken is niet toegestaan.

Zodra je de opdracht inlevert verklaar je ook dat dit je **eigen** werk is. Bij twijfel zullen we een nader onderzoek doen, en zul je je programma mondeling moeten toelichten. We stellen dan vast of je de stof beheerst, en bepalen je cijfer op basis van dat mondeling. **Mochten we constateren dat je (delen van) bronnen hebt overgenomen, of op andere wijze niet je eigen werk hebt ingeleverd, dan volgt er een melding van plagiaat bij de examencommissie.**

Voor deze toets heb je van 10.00 tot 18.00 de tijd. Zorg dat je vóór 18.00 iets inlevert! Studenten met toetstijdverlenging kunnen hun werk inleveren tot 19.00. Van studenten *zonder* toetstijdverlenging zal het werk worden nagekeken wat het laatst vóór 18.00 is ingeleverd. Mocht je bestand om 18.00 in de Codegrade-queue staan ter controle, en pas na 18.00 klaar zijn, dan kijken we die versie van het bestand na.

Opdracht

In deze opgave ga je een programma schrijven wat aan de hand van individuele netwerkpakketten alle communicatie tussen twee hosts kan samenvoegen en uitlezen.

Context

Het is moeilijk om te bevatten hoe veel informatie er elke dag via het Internet wordt uitgewisseld. Dit jaar zullen zo'n 26 miljard devices verbonden zijn via het Internet. Deze devices versturen in totaal 7.7 exabytes aan data per dag, oftewel 7.700.000.000 gigabytes. Per dag wordt er zo'n 430.000 uur aan beeldmateriaal op YouTube gezet, goed voor bijna 50 jaar onafgebroken kijkplezier. En tussen al die data die wordt uitgewisseld, opgeknipt en verborgen in minieme pakketjes, bevindt zich malware.

Voor de gewone mens is het bijna onmogelijk om door die bergen aan gegevens heen te ploegen op zoek naar tekenen van malafide webverkeer. Naast dat het simpelweg niet kan gezien de enorme hoeveelheid, is er ook nog het probleem dat kwaadwillenden uitermate goed zijn in het verbergen van hun snode daden. Ze misbruiken protocollen, verstoppen hun requests tussen andere data, of knippen de malware op in stukjes zodat elk individueel berichtje onschuldig lijkt.

Tijdens Network Engineering hebben jullie kennis gemaakt met een aantal manieren om deze malafide pakketjes toch te detecteren. Vandaag gaan jullie aan de slag met signature-based analysis: het analyseren van headers, packet payloads, en packetstreams om te bepalen of er sprake is van malicious network traffic (zie ook: Network Forensics, hoofdstuk 7, Kurose & Ross paragraaf 8.9.2). Wat voor mensen moeizaam is, is een makkelijke taak voor een goed geschreven computerprogramma.

Uitleg

Het doel van de opdracht is om een netwerkpakketjes-scanner te schrijven die uit een datadump al het netwerkverkeer tussen hosts uitleest, opschoont, én samenvoegt zodat te zien is wat er precies tussen 2 individuele hosts is uitgewisseld. De laatste stap is om te analyseren of er in die communicatie mogelijkerwijs sprake is van malafide webverkeer.

Een belangrijk concept voor deze opdracht is het idee van netwerksessies. Dat houdt in dat er tussen 2 hosts op het netwerk over-en-weer berichten worden uitgewisseld, en dat de inhoud van die berichten even belangrijk is als de context waarin de berichten worden uitgewisseld. Oftewel, soms is het niet alleen belangrijk wat er in een bericht staat, maar ook wat er in de berichten voor en na staat, hoe laat het verstuurd wordt, en op welke poorten er gecommuniceerd wordt. Zo is een GET-request op poort 80 (http) redelijk gebruikelijk, maar is datzelfde request op poort 22 (ssh) toch reden om de zaak nader te bekijken.

De netwerkdata die je gaat verwerken wordt aangeleverd als JSON-bestand. In je skeleton-bestand staat een functie waarmee je dit bestand kan inlezen. De netwerkdata staat, onder ideale omstandigheden, opgeslagen in het volgende format:

```
[ {“src_ip”: “192.168.4.1”,
    “dest_ip”: “145.18.11.151”,
    “src_port”: 80,
    “dest_port”: 80,
    “timestamp”: “25-06-2022 13:27.05”,
    “msg_no”:0,
    “payload”: “Hello there”},
  {“src_ip”: “145.18.11.151”,
    “dest_ip”: “192.168.4.1”,
    “src_port”: 80,
    “dest_port”: 80,
    “timestamp”: “25-06-2022 13:28.16”,
    “msg_no”:1,
    “payload”: “General Kenobi”} ]
```

Elk pakketje bestaat uit een dictionary met 7 key:value pairings. De volgende keys worden gebruikt in het pakket:

- `src_ip`: het IP-adres dat het bericht heeft verzonden.
- `dest_ip`: het IP-adres dat het bericht heeft ontvangen.
- `src_port`: het poortnummer vanuit waar het bericht is verzonden.
- `dest_port`: het poortnummer waarop het bericht is ontvangen.
- `timestamp`: het tijdstip waarop het bericht is ontvangen.
- `msg_no`: het reeksnummer van het bericht. **LET OP**: dit reeksnummer hoort bij de sessie. Dus als host 1 het eerste bericht stuurt met `msg_no` = 1, dan verstuurt host 2 een reply met `msg_no` = 2.
- `payload`: het daadwerkelijk verzonden bericht in het pakketje.

Belangrijk: Let op het datatype van de verschillende values. Hieronder staat per key welk datatype de value heeft:

- `src_ip`: string
- `dest_ip`: string
- `src_port`: integer
- `dest_port`: integer
- `timestamp`: string
- `msg_no`: integer
- `payload`: string

Een netwerksessie wordt in deze dataset gedefinieerd als *alle* communicatie tussen 2 hosts op een vast poortnummer per host, ongeacht de tijdsduur. Stel dat host A via poort 60 berichten stuurt naar host B op poort 98, dan zijn alle berichten die heen en weer gaan tussen poort 60 van host A en poort 98 van host B onderdeel van dezelfde netwerksessie, ongeacht of die sessie 2 minuten of 2 jaar duurt.

Zoals eerder beschreven is dit het ideale format van de netwerkdata. Toch kan het gebeuren dat gegevens minder netjes worden uitgelezen uit het netwerkverkeer. Het volgende bericht is óók mogelijk:

```
{“source”: "115,105,116,104:66   ”,
 “destination”:“   106.101.100.105:66”,
 “timestamp”: “   04-05-2022 20:06.18   ”,
 “msg_no”:0,
 “payload”: “Execute Order 66”}
```

Dit ziet er al een stuk minder net uit. Om dit goed te herstellen moet je gegevens éérst normaliseren: zorgen dat al je informatie in hetzelfde format staat. De eerste 3 functies die je moet afmaken zorgen gezamenlijk ervoor dat alle pakketjes in hetzelfde format staan.

Na het normaliseren kan je het netwerkverkeer tussen 2 hosts gaan samenvoegen. In eerste instantie ga je alle berichten die bij één sessie horen verzamelen. Zodra de individuele berichten zijn verzameld combineer je ze tot één geheel wat je kan uitlezen.

Om je hierbij te helpen zijn er in het skeleton-bestand twee classes gedefinieerd. De class `SessionMessage` slaat gegevens op over een enkel pakketje. De class `NetworkSession` slaat gegevens op over de netwerksessie. Hieronder staat de code die je hiervoor krijgt op de toets:

```
class SessionMessage():
    def __init__(self, msg_no, timestamp, payload):
        self.msg_no = msg_no
        self.timestamp = timestamp
        self.payload = payload
```

Een instance van deze heeft 3 instance-variabelen:

- msg_no: het nummer van het bericht
- timestamp: de timestamp van het bericht
- payload: de tekst in de payload van het bericht

```
class NetworkSession():
    def __init__(self, src_ip, src_port, dest_ip, dest_port, messages):
        self.src_ip = src_ip
        self.src_port = src_port
        self.dest_ip = dest_ip
        self.dest_port = dest_port
        self.messages = messages
```

Een instance van deze class heeft 5 instance-variabelen:

- `self.src_ip`: het IP-adres dat het eerste bericht in de netwerksessie heeft verzonden.
- `self.dest_ip`: het IP-adres dat het eerste bericht in de netwerksessie heeft ontvangen.
- `self.src_port`: het poortnummer vanuit waar het eerste bericht in de netwerksessie is verzonden.
- `self.dest_port`: het poortnummer waarop het eerste bericht in de netwerksessie is ontvangen.
- `self.messages`: een lijst met `SessionMessage`-instances.

Opgave

Voor dit tentamen moet je zeven functies voorzien van code. Hieronder staan ze beschreven. Per functie is ook aangegeven of deze onderdeel is van de basisfunctionaliteiten, of de aanvullende functionaliteiten. Bij de basisfunctionaliteiten moet je voor 90% van de Codegrade-tests slagen om een voldoende te krijgen. Zodra jouw programma aan deze eis voldoet wordt er pas gekeken naar de aanvullende functionaliteiten.

Tip:
Lees de opgave goed door. Bepaal per functie wat de parameters zijn, wat het datatype van die parameter is, de gewenste return-waarde(n) van de functie, en wat je moet veranderen aan de parameters om tot de gewenste return-waarden te komen. Ga dan pas aan de slag met programmeren.

`normalize_source_and_destination(log_entry)`

basisfunctionaliteit

Een deel van de log entries heeft nog geen gesplitste waarden voor poort-nummer en IP-adres. Deze functie zorgt ervoor dat deze gegevens alsnog worden gesplitst.

Deze functie heeft één parameter: een individuele entry uit de lijst met berichten. De return-waarde van deze functie is diezelfde entry uit de lijst met berichten. Je past dus de bestaande lijst met log-entries aan.

Het enige verschil is dat, mocht er sprake zijn van een ongesplitst IP-adres en poort-nummer voor source en destination adressen in de log entry, deze worden alsnog worden gesplitst en worden opgeslagen onder de juiste keys. De oude keys, met de ongesplitste waarden, worden vervolgens uit de log_entry verwijderd.

Tip: Kijk in het voorbeeld onder *Uitleg* om te zien welke keys er worden gebruikt als de source en destination adressen nog niet gesplitst zijn.

Outputvoorbeeld:

```
>>> from opgh import *
>>> networkdata = load_jsonfile("networkdata.json")
>>> print(networkdata[0])
{'source': '192.168.4.1:20', 'destination': '145,18,11,151:443', 'timestamp': '
2022-07-04 16:07:39   ', 'msg_no': 0, 'payload': 'Beautiful is better than
ugly.\nExplicit is better than implicit.\n'}
>>> networkdata[0]=normalize_source_and_destination(networkdata[0])
>>> print(networkdata[0])
{'timestamp': '   2022-07-04 16:07:39   ', 'msg_no': 0, 'payload': 'Beautiful is
better than ugly.\nExplicit is better than implicit.\n', 'src_ip': '192.168.4.1',
'src_port': 20, 'dest_ip': '145,18,11,151', 'dest_port': 443}
>>>
```

`normalize_whitespace(log_entry)`

basisfunctionaliteit

Deze functie zorgt ervoor dat voor één log-entry eventuele whitespace rondom de tekst van alle string-waarden behalve de payload wordt verwijderd.

Deze functie heeft één parameter: een individuele entry uit de lijst met berichten. De return-waarde van deze functie is diezelfde entry uit de lijst met berichten. Je past dus de bestaande lijst met log-entries aan.

Het enige verschil is dat voor elke value van het datatype string, met uitzondering van de value voor `payload`, eventuele whitespace voorafgaand of volgend op de tekst is verwijderd.

Outputvoorbeeld:

```
>>> from opgh import *
>>> networkdata = load_jsonfile("networkdata.json")
>>> print(networkdata[0])
{'source': '192.168.4.1:20', 'destination': '145,18,11,151:443', 'timestamp': '
2022-07-04 16:07:39   ', 'msg_no': 0, 'payload': 'Beautiful is better than
ugly.\nExplicit is better than implicit.\n'}
>>> networkdata[0]=normalize_whitespace(networkdata[0])
>>> print(networkdata[0])
{'source': '192.168.4.1:20', 'destination': '145,18,11,151:443', 'timestamp':
'2022-07-04 16:07:39', 'msg_no': 0, 'payload': 'Beautiful is better than
ugly.\nExplicit is better than implicit.\n'}
>>>
```

`normalize_ip(log_entry)`

basisfunctionaliteit

Soms worden IP-adressen niet goed uitgelezen uit het bronbestand, en daardoor opgeslagen met het verkeerde scheidingsteken. Bij deze netwerkberichten wordt soms de punt (`.`) verwisseld voor een komma (`,`). Deze functie zorgt ervoor dat deze uitleesfout wordt gecorrigeerd.

Deze functie heeft één parameter: een individuele entry uit de lijst met berichten. De return-waarde van deze functie is diezelfde entry uit de lijst met berichten. Je past dus de bestaande lijst met log-entries aan.

Het enige verschil is dat overal waar een `src_ip` of `dest_ip` initieel gescheiden is door komma's in plaats van punten, dat de komma's zijn vervangen door punten.

Outputvoorbeeld:

```
>>> from opgh import *
>>> networkdata = load_jsonfile("networkdata.json")
>>> print(networkdata[1])
{'src_ip': '192,168,4,1   ', 'dest_ip': '  145,18,11,151  ', 'src_port': 20,
'dest_port': 20, 'timestamp': '2022-07-04 16:07:39', 'msg_no': 1, 'payload':
'Simple is better'}
>>> networkdata[1]=normalize_ip(networkdata[1])
>>> print(networkdata[1])
{'src_ip': '192.168.4.1   ', 'dest_ip': '  145.18.11.151  ', 'src_port': 20,
'dest_port': 20, 'timestamp': '2022-07-04 16:07:39', 'msg_no': 1, 'payload':
'Simple is better'}
>>>
```

Tussenstand

De bovenstaande functies zorgen ervoor dat alle berichten in de lijst in hetzelfde format worden gezet. Alle ip-adressen en poortnummers zijn opgesplitst, elke waarde is omgezet naar het juiste datatype, eventuele ongewenste spaties zijn verwijderd, en ip-adressen worden allemaal op dezelfde wijze opgeschreven. Dit proces heet normaliseren: je corrigeert afwijkingen van de norm.

Het normaliseren van data is een noodzakelijke stap als je gegevens wilt analyseren. Waar wij zien dat met `src_ip = "192.168.4.1"` en `src_ip = " 192.168.4.1 "` hetzelfde wordt bedoeld, kan een computer die context niet interpreteren. Datzelfde geldt voor de waarden `dest_ip = "192,168,4,1"` en `dest_ip = "192.168.4.1"`. Als wij uit de context begrijpen dat allebei ip-adressen zijn, dan is het zeer waarschijnlijk dat ze ook allebei hetzelfde ip-adres zijn. Normaliseren zorgt ervoor dat die dubbelzinnigheid wordt verwijderd, zodat het ook zonder context duidelijk is welke waarde wordt bedoeld.

`network_session_messages(src_ip, src_port, dest_ip, dest_port, log_entries)`

basisfunctionaliteit

De volgende stap in het proces is het verzamelen van individuele berichten uit één netwerksessie. Deze functie zorgt hiervoor.

De functie heeft 5 parameters: `src_ip`, het ip-adres vanuit waar een bericht is verzonden, `src_port`, de poort vanuit waar een bericht is verzonden, `dest_ip`, het ip-adres waar het bericht naartoe gaat, `dest_port`, de poort waar het bericht naartoe gaat, en `log_entries`, de hele lijst met netwerkberichten. De return-waarde van deze functie is een lijst met `SessionMessage`-objecten.

Bij het opstarten van de functie geef je mee voor welke `src_ip`, `src_port`, `dest_ip`, en `dest_port` je de berichten wilt verzamelen. Vervolgens loopt het programma alle berichten langs die over het netwerk zijn verzonden. Voor elk bericht controleert het programma of het onderdeel is van *alle* communicatie tussen host A en host B. Als dit het geval is, maakt het programma een nieuw `SessionMessage` object aan voor dat bericht en slaat het op in de lijst met `SessionMessage`-objecten.

Outputvoorbeeld:

```
>>> from opgh import *
>>> networkdata = load_jsonfile("networkdata.json")
>>> for i in range(len(networkdata)):
...     networkdata[i] =
normalize_ip(normalize_whitespace(normalize_source_and_destination(networkdata[i])
))
# Bovenstaande code normaliseert de netwerkdata, zodat we deze kunnen gebruiken in
de functie.

>>> print(networkdata[0])
{'timestamp': '2022-07-04 16:07:39', 'msg_no': 0, 'payload': 'Beautiful is better
than ugly.\nExplicit is better than implicit.\n', 'src_ip': '192.168.4.1',
'src_port': 20, 'dest_ip': '145.18.11.151', 'dest_port': 443}
# We laten zien dat de gegevens netjes genormaliseerd zijn.

>>> network_session_messages("192.168.4.1", 20, "145.18.11.151", 443, networkdata)
[<opgh_ref.SessionMessage object at 0x000001E65194A7C0>, <opgh_ref.SessionMessage
object at 0x000001E65194A700>, <opgh_ref.SessionMessage object at
0x000001E65194A760>, <opgh_ref.SessionMessage object at 0x000001E65194A550>,
<opgh_ref.SessionMessage object at 0x000001E65194A820>]
# We roepen de functie aan. Deze geeft een lijst met SessionMessage-objecten
terug.
>>>
```

`network_sessions(log_entries)`

basisfunctionaliteit

Dit is de laatste stap van het basisprogramma. Deze functie zorgt ervoor dat alle communicatie uit het bestand `log_entries` wordt gegroepeerd per netwerksessie.

Deze functie heeft één parameter: `log_entries`, het complete bestand met netwerkberichten. De return-waarde van deze functie is een lijst met `NetworkSession`-objecten.

Deze functie loopt alle log entries één voor één langs. Eerst controleert de functie of er al een `NetworkSession`-object voor deze netwerksessie is aangemaakt. Zo ja, dan wordt het bericht genegeerd. Zo nee, dan worden eerst alle berichten uit die netwerksessie opgehaald via de functie `network_session_messages`. Daarna wordt er een nieuw `NetworkSession`-object toegevoegd aan de lijst met `NetworkSession`-objecten. Als argumenten worden de `src_ip`, `dest_ip`, `src_port`, `dest_port`, en de lijst met `SessionMessage`-objecten meegegeven bij het aanmaken van het object.

Outputvoorbeeld:

```
>>> from opgh import *
>>> networkdata = load_jsonfile("networkdata.json")

>>> for i in range(len(networkdata)):
...     networkdata[i] =
normalize_ip(normalize_whitespace(normalize_source_and_destination(networkdata[i])
))

>>> print(networkdata[0])

{'timestamp': '2022-07-04 16:07:39', 'msg_no': 0, 'payload': 'Beautiful is better
than ugly.\nExplicit is better than implicit.\n', 'src_ip': '192.168.4.1',
'src_port': 20, 'dest_ip': '145.18.11.151', 'dest_port': 443}

>>> network_sessions(networkdata)

[<opgh_ref.NetworkSession object at 0x000001E651895D60>, <opgh_ref.NetworkSession
object at 0x000001E6518CC940>, <opgh_ref.NetworkSession object at
0x000001E6518CC1F0>, <opgh_ref.NetworkSession object at 0x000001E6518A7880>,
<opgh_ref.NetworkSession object at 0x000001E6518CFB20>, <opgh_ref.NetworkSession
object at 0x000001E6518CFF70>, <opgh_ref.NetworkSession object at
0x000001E6518CF490>, <opgh_ref.NetworkSession object at 0x000001E65193B070>,
<opgh_ref.NetworkSession object at 0x000001E65193B580>, <opgh_ref.NetworkSession
object at 0x000001E65193B6A0>]
>>>
```

`detect_suspicious_activity(self, suspicous_activity_db)`

aanvullende functionaliteit

Deze methode wordt gedefinieerd binnen de class `NetworkSession`, bovenaan het skeleton-bestand, en wordt gebruikt om te controleren of er eventueel sprake is van malafide communicatie tussen twee hosts.

De functie heeft 2 parameters: `self` en `suspicious_activity_db`. `self` geeft aan dat dit een zogeheten instance method is. Voor deze parameter hoef je tijdens het aanroepen van de functie niets in te vullen. `suspicious_activity_db` is een beknopte malware-database waarin een aantal karakteristieken staan van verboden verkeer. Deze functie geeft alleen een return-waarde als het verkeer voldoet aan (minstens) één van de karakteristieken in `suspicious_activity_db`.

Er zijn drie situaties waarin er sprake kan zijn van malafide webverkeer:

1. Als de communicatie begint voor werktijd, of eindigt na werktijd;
2. Als er specifieke termen voorkomen in individuele payloads van de messages;
3. Als er communicatie is tussen 2 specifieke hosts die niet met elkaar mogen communiceren.

Elke situatie heeft zijn eigen foutmelding. Het kan voorkomen dat er meerdere foutmeldingen van toepassing zijn, bijvoorbeeld wanneer er na werktijd wordt gecommuniceerd via verboden poorten. In dat geval moet er voor elke schending van de regels een stuk tekst aan de foutmelding worden toegevoegd.

Outputvoorbeeld

We gaan ervan uit op dit punt dat de netwerkdata genormaliseerd is.

```
>>> netsessions = network_sessions(networkdata)
>>> netsession = netsessions[0]
>>> print(netsession)
<opgh_ref.NetworkSession object at 0x000001E6518C18B0>
# laten zien wat er is opgeslagen in netsession.

>>> netsession.messages
[<opgh_ref.SessionMessage object at 0x000001E6518C16D0>, <opgh_ref.SessionMessage
object at 0x000001E6518C1820>, <opgh_ref.SessionMessage object at
0x000001E6518C1790>, <opgh_ref.SessionMessage object at 0x000001E6518C1490>,
<opgh_ref.SessionMessage object at 0x000001E6518C13D0>]

>>> netsession.detect_suspicious_activity(SUSPICIOUS_ACTIVITY_DB)
['Hosts 192.168.4.1 and 145.18.11.151 not allowed to communicate\n']
```

`generate_report(sessions, suspicious_activity_db)`

extra functionaliteit

Deze functie genereert een rapport met alle waarschuwingen die door de methode `detect_suspicious_activity` worden aangemaakt.

De functie heeft 2 parameters: `sessions`, een lijst met `NetworkSession`-objecten aangemaakt door de functie `network_sessions`, en `suspicious_activity_db`, een beknopte malware-database waarin een aantal karakteristieken staan van verboden verkeer. De return-waarde van deze functie is een string-variabele met daarin de tekst van het rapport.

Voor elk `NetworkSession`-object voert de functie de methode `detect_suspicious_activity` uit, en voegt een eventuele return-waarde toe aan een string-variabele met daarin alle waarschuwingen.

Outputvoorbeeld

```
>>> netsessions = network_sessions(networkdata)
>>> print(generate_report(netsessions, SUSPICIOUS_ACTIVITY_DB))
Network data report

Network Session Report
192.168.4.1:20 -> 145.18.11.151:443
From 2022-07-04 16:07:39 to 2022-07-04 16:07:39 5 message(s) were sent
Flagged because:
- Hosts 192.168.4.1 and 145.18.11.151 not allowed to communicate

Network Session Report
115.105.116.104:1137 -> 98.97.115.101:80
From 2022-07-04 16:07:39 to 2022-07-04 16:07:39 3 message(s) were sent
```

```
Found nothing suspicious

Network Session Report
145.18.11.151:443 -> 127.0.0.1:443
From 2022-07-04 16:07:39 to 2022-07-04 16:07:39 14 message(s) were sent
Flagged because:
- Suspicious payload 'pen-test' in message 5

Network Session Report
192.168.4.1:20 -> 145.18.11.151:20
From 2022-07-04 16:07:39 to 2022-07-04 16:07:39 20 message(s) were sent
Flagged because:
- Hosts 192.168.4.1 and 145.18.11.151 not allowed to communicate

Network Session Report
115.105.116.104:1137 -> 98.97.115.101:1137
From 2022-07-04 16:07:39 to 2022-07-04 16:07:39 7 message(s) were sent
Found nothing suspicious

Network Session Report
115.105.116.104:1137 -> 106.101.100.105:1137
From 2022-07-04 16:07:39 to 2022-07-04 16:07:39 6 message(s) were sent
Found nothing suspicious

Network Session Report
145.18.11.151:443 -> 127.0.0.1:80
From 2022-07-04 16:07:39 to 2022-07-04 16:07:39 8 message(s) were sent
Found nothing suspicious

Network Session Report
115.105.116.104:66 -> 106.101.100.105:66
From 2022-07-04 16:07:39 to 2022-07-04 16:07:39 6 message(s) were sent
Flagged because:
- Suspicious payload '66' in message 9
- Suspicious payload '66' in message 10

Network Session Report
115.105.116.104:1137 -> 106.101.100.105:286
From 2022-07-04 16:07:39 to 2022-07-04 16:07:39 2 message(s) were sent
Found nothing suspicious

Network Session Report
145.18.11.151:66 -> 115.105.116.104:66
From 2022-07-04 16:07:39 to 2022-07-04 23:07:39 3 message(s) were sent
Flagged because:
- Hosts 145.18.11.151 and 115.105.116.104 not allowed to communicate
- Communication outside office hours
```

Beoordeling

Je cijfer wordt bepaald aan de hand van de volgende criteria. Je begint met een 1 voor de juiste bestandsnaam. Voor een 6 moeten de volgende functionaliteiten werken:

1. `normalize_source_and_destination`: Het scheiden van ip-adressen en port-nummers.
2. `normalize_whitespace`: Het weghalen van whitespaces rondom de gegevens bij alle string-waarden behalve de payload.
3. `normalize_ip`: Het vervangen van alle alternatieve punctuatie door punten in de ip-adressen.
4. `network_session_messages`: Het verzamelen en omzetten van alle berichten in één netwerksessie naar een lijst met SessionMessage-objecten.
5. `network_sessions`: Het doorlopen van alle verzonden berichten, het identificeren van alle netwerksessies, en per netwerksessie een NetworkSession-object aanmaken. Dit levert een lijst met NetworkSession-objecten op.

Daarnaast zijn er 2 punten te verdienen met de volgende functionaliteiten:

1. `detect_suspicious_activity`: Het analyseren van NetworkSession-objecten om te kijken of er sprake is van malafide netwerkverkeer.
2. `generate_report`: Het genereren van een rapport van de scan en analyse.

De resterende 2 punten bestaan uit:

- Flake8 linter levert geen foutmeldingen op: 0.5 punt
- Vrije punten docenten voor nette code: 1.5 punt

Let op: Codegrade controleert eerst of de eerste 5 functionaliteiten goed geïmplementeerd zijn. Pas wanneer je voor 90% van de tests van de basisfunctionaliteit geslaagd bent krijg je toegang tot de andere testsuites. Dat houdt in dat als de vijf basisfuncties niet voor 90% werken, je automatisch een onvoldoende krijgt en er niet wordt gekeken naar de rest van je code.

Algemene tips

Onthoudt tijdens het tentamen: **save early, save often**. Lever regelmatig een tussentijdse versie van je programma in op CodeGrade. Op deze manier krijg je snel feedback op de functionaliteit van je programma, en weet je waar eventuele foutmeldingen vandaan komen.
CodeGrade werkt met een standaardinstallatie van Python 3.7. Externe modules zullen dus niet werken in de CodeGrade omgeving. Je zal voor deze toets dus moeten vertrouwen op de standaard modules van Python.
Let op het outputvoorbeeld. De tests op CodeGrade controleren of jouw output overeenkomt met het outputvoorbeeld. Ook kleine afwijkingen, zoals spaties of komma's op de verkeerde plek, kunnen er dus voor zorgen dat jouw code niet door de test komt.
Het outputvoorbeeld laat alle situaties zien waar je rekening mee moet houden in je programma. Denk dus niet te groot (of te klein) bij het schrijven van je code.
Zorg dat je het bestand met de juiste bestandsnaam upload. CodeGrade probeert om het bestand 'opgh.py' te openen voor de tests. Als jouw bestand een andere naam heeft, dan zal CodeGrade het niet correct kunnen laden of testen.