Hertentamen SMP-Python 05-06-2022

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.

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.

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, nr, timestamp, payload):
        self.nr = nr
        self.timestamp = timestamp
        self.payload = payload

class NetworkSession():
    def __init__(self, src_ip, src_port, dst_ip, dst_port, messages):
        self.src_ip = src_ip
        self.src_port = src_port
        self.dst_ip = dst_ip
        self.dst_port = dst_port
        self.messages = messages
```

Beoordeling

Je cijfer wordt bepaald aan de hand van de volgende criteria. Voor een 6 moeten de volgende functionaliteiten werken:

1. Het scheiden van ip-adressen en port-nummers.
2. Het vervangen van alle alternatieve punctuatie door punten in de ip-adressen.
3. Het weghalen van whitespaces rondom de gegevens bij alle string-waarden behalve de payload.
4. Het verzamelen en omzetten van alle berichten in één netwerksessie naar een lijst met SessionMessage-objecten.
5. 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. Het analyseren van NetworkSession-objecten om te kijken of er sprake is van malafide netwerkverkeer.
2. 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.

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.