# سید عباس میرقاسمی – 402443180

لینک پروژه:

https://github.com/samirghasemi/-detecting-P2Pbotnets-by-discovering-flow-dependency-

# بخش اول و دوم

خواندن از فایل pcap و ایجاد جریان ها و ذخیره در flow.json :

from scapy.all import rdpcap, IP, TCP, UDP

import json

import base64

def create\_flow\_key(packet):

    """Create a normalized flow key for bi-directional traffic."""

    if IP in packet:

        src\_ip, dst\_ip = packet[IP].src, packet[IP].dst

        src\_port, dst\_port = (packet[TCP].sport, packet[TCP].dport) if TCP in packet else (packet[UDP].sport, packet[UDP].dport) if UDP in packet else (0, 0)

        if (src\_ip > dst\_ip) or (src\_ip == dst\_ip and src\_port > dst\_port):

            src\_ip, dst\_ip = dst\_ip, src\_ip

            src\_port, dst\_port = dst\_port, src\_port

        protocol = packet[IP].proto

        return f"{src\_ip}:{src\_port}-{dst\_ip}:{dst\_port}\_proto\_{protocol}"

    return None

def process\_pcap(file\_path, time\_threshold=5):

    """Process packets from a pcap file and organize them into flows and sessions."""

    packets = rdpcap(file\_path)

    flows = {}

    for packet in packets:

        if IP in packet and (TCP in packet or UDP in packet):

            key = create\_flow\_key(packet)

            # print(1)

            if key:

                if key not in flows:

                    flows[key] = {

                        'start\_time': float("{:.2f}".format(packet.time)),

                        'end\_time': float("{:.2f}".format(packet.time)),

                        'total\_size': len(packet),

                        'sessions': [{

                            'src\_ip': packet[IP].src,

                            'dst\_ip': packet[IP].dst,

                            'src\_port': packet[TCP].sport if TCP in packet else packet[UDP].sport,

                            'dst\_port': packet[TCP].dport if TCP in packet else packet[UDP].dport,

                            'protocol': packet[IP].proto,

                            'start\_time': float("{:.2f}".format(packet.time)),

                            'end\_time': float("{:.2f}".format(packet.time)),

                            'total\_size': len(packet),

                            'number\_of\_packets': 1

                            # 'packet\_summaries': [packet.summary()]  # Store summaries instead of raw packets

                        }]

                    }

                else:

                    flow = flows[key]

                    last\_session = flow['sessions'][-1]

                    if float("{:.2f}".format(packet.time)) - last\_session['end\_time'] > time\_threshold:

                        flow['sessions'].append({

                            'src\_ip': packet[IP].src,

                            'dst\_ip': packet[IP].dst,

                            'src\_port': packet[TCP].sport if TCP in packet else packet[UDP].sport,

                            'dst\_port': packet[TCP].dport if TCP in packet else packet[UDP].dport,

                            'protocol': packet[IP].proto,

                            'start\_time': float("{:.2f}".format(packet.time)),

                            'end\_time': float("{:.2f}".format(packet.time)),

                            'total\_size': len(packet),

                            'number\_of\_packets': 1

                            # 'packet\_summaries': [packet.summary()]  # Store summaries instead of raw packets

                        })

                    else:

                        last\_session['end\_time'] = float("{:.2f}".format(packet.time))

                        last\_session['total\_size'] += len(packet)

                        last\_session['number\_of\_packets'] += 1

                    flow['end\_time'] = float("{:.2f}".format(packet.time))

                    flow['total\_size'] += len(packet)

    return flows

flows = process\_pcap('EX-3.pcap')

save\_flows\_to\_json(flows, '1\_flows.json')

نمونه ذخیره شده:

"192.168.199.134:488-192.168.199.135:49612\_proto\_6": {

        "start\_time": 1476813996.0,

        "end\_time": 1476814008.04,

        "total\_size": 327,

        "sessions": [

            {

                "src\_ip": "192.168.199.135",

                "dst\_ip": "192.168.199.134",

                "src\_port": 49612,

                "dst\_port": 488,

                "protocol": 6,

                "start\_time": 1476813996.0,

                "end\_time": 1476813998.97,

                "total\_size": 210,

                "number\_of\_packets": 2

            },

            {

                "src\_ip": "192.168.199.135",

                "dst\_ip": "192.168.199.134",

                "src\_port": 49612,

                "dst\_port": 488,

                "protocol": 6,

                "start\_time": 1476814008.04,

                "end\_time": 1476814008.04,

                "total\_size": 117,

                "number\_of\_packets": 1

            }

        ]

    },

# بخش سوم

در این بخش با توجه به شرایط خواسته شده، جریان ها را فیلتر می کنیم:

import json

# Define thresholds

SIZE\_THRESHOLD = 5000  # Example threshold for size (bytes)

DURATION\_THRESHOLD = 300  # Example threshold for duration (seconds)

MIN\_OCCURRENCE\_THRESHOLD = 3  # Minimum occurrences (sessions per flow)

def filter\_sessions(flows):

    """

    Filters out sessions that:

    - Have a total size greater than `size\_threshold`.

    - Have a duration longer than `duration\_threshold`.

    - Occur less frequently than `min\_occurrence\_threshold`.

    """

    filtered\_flows = {}

    for flow\_key, flow\_data in flows.items():

        filtered\_sessions = []

        for session in flow\_data['sessions']:

            duration = session['end\_time'] - session['start\_time']

            if session['total\_size'] <= SIZE\_THRESHOLD and duration <= DURATION\_THRESHOLD:

                filtered\_sessions.append(session)

        # Only include flows with enough sessions

        if len(filtered\_sessions) >= MIN\_OCCURRENCE\_THRESHOLD:

            filtered\_flows[flow\_key] = flow\_data.copy()

            # filtered\_flows[flow\_key]['sessions'] = filtered\_sessions

    return filtered\_flows

filtered\_flows = filter\_sessions(flows)

save\_flows\_to\_json(filtered\_flows, '2-filtered\_flows.json')

نمونه فایل ذخیره شده:

"192.168.199.134:445-192.168.199.135:16332\_proto\_6": {

        "start\_time": 1476813986.6,

        "end\_time": 1476814008.73,

        "total\_size": 327,

        "sessions": [

            {

                "src\_ip": "192.168.199.135",

                "dst\_ip": "192.168.199.134",

                "src\_port": 16332,

                "dst\_port": 445,

                "protocol": 6,

                "start\_time": 1476813986.6,

                "end\_time": 1476813986.6,

                "total\_size": 117,

                "number\_of\_packets": 1

            },

            {

                "src\_ip": "192.168.199.134",

                "dst\_ip": "192.168.199.135",

                "src\_port": 445,

                "dst\_port": 16332,

                "protocol": 6,

                "start\_time": 1476814002.72,

                "end\_time": 1476814002.72,

                "total\_size": 93,

                "number\_of\_packets": 1

            },

            {

                "src\_ip": "192.168.199.135",

                "dst\_ip": "192.168.199.134",

                "src\_port": 16332,

                "dst\_port": 445,

                "protocol": 6,

                "start\_time": 1476814008.73,

                "end\_time": 1476814008.73,

                "total\_size": 117,

                "number\_of\_packets": 1

            }

        ]

    },

## بخش چهارم

در این بخش تعداد تکرار جریان های مختلف را بدست می آوریم و در فایل ذخیره میکنیم:

def compute\_occurrences(flows):

    """Count occurrences of each flow and store the count."""

    occurrences = {}

    for flow\_key, flow\_data in flows.items():

        # occurrences[flow\_key] = sum(session['number\_of\_packets'] for session in flow\_data['sessions'])

        occurrences[flow\_key] = len(flow\_data['sessions'])

    return occurrences

occurrences = compute\_occurrences(filtered\_flows)

save\_flows\_to\_json(occurrences, '3-1-occurrences.json')

نمونه فایل ذخیره شده:

}

  "192.168.199.134:445-192.168.199.135:49612\_proto\_6": 9

{

# بخش پنجم

در این بهش وابستگی جریان های دولایه ای رو استخراج میکنیم:

import math

def extract\_dependencies(flows, T\_dep, N\_dep, S\_dep\_th):

    """Extract two-level dependencies based on temporal proximity and occurrence similarity."""

    occurrences = compute\_occurrences(flows)

    dependencies = {}

    Sdep\_scores = {}

    # Prepare flows for processing by sorting them based on the start time of their sessions

    for flow\_key, flow\_data in flows.items():

        flow\_data['sessions'].sort(key=lambda x: x['start\_time'])

    # Compare each flow with every other flow

    for fi\_key, fi\_data in flows.items():

        for fj\_key, fj\_data in flows.items():

            if fi\_key != fj\_key:

                for fi\_session in fi\_data['sessions']:

                    for fj\_session in fj\_data['sessions']:

                        if abs(fi\_session['start\_time'] - fj\_session['start\_time']) <= T\_dep:

                            Ni = occurrences[fi\_key]

                            Nj = occurrences[fj\_key]

                            if abs(Ni - Nj) < N\_dep:

                                pair\_key = (fi\_key, fj\_key)

                                if pair\_key in dependencies:

                                    dependencies[pair\_key] += 1

                                else:

                                    dependencies[pair\_key] = 1

    # Calculate Sdep scores for all identified dependencies

    for (fi, fj), Tij in dependencies.items():

        Ni = occurrences[fi]

        Nj = occurrences[fj]

        Sdep = math.sqrt(Tij\*\*2 / (Ni \* Nj))

        if Sdep > S\_dep\_th:

            Sdep\_scores[f"{fi}, {fj}"] = Sdep

    return Sdep\_scores

# Define thresholds

T\_dep = 30  # Maximum time difference between flow starts

N\_dep = 5   # Maximum difference in occurrences

S\_dep\_th = 0.5  # Minimum score threshold for a dependency to be considered significant

# Assuming `flows` is your data structure loaded from somewhere as described

dependencies = extract\_dependencies(filtered\_flows, T\_dep, N\_dep, S\_dep\_th)

# print("Dependencies with scores:", dependencies)

save\_flows\_to\_json(dependencies, '3-2-dependencies.json')

نمونه فایل ذخیره شده:

{

    "192.168.199.134:445-192.168.199.135:49612\_proto\_6, 192.168.199.134:445-244.168.199.135:49612\_proto\_6": 0.7453559924999299,

{

# بخش ششم

در این بخش تلاش میکنیم وابستگی جریان های چند لایه ای را استخراج کنیم:

def parse\_dependencies(dependencies):

    """Parse the dependencies to a more accessible structure."""

    parsed\_dependencies = {}

    for key, score in dependencies.items():

        flows = key.split(", ")

        for i in range(len(flows) - 1):

            if flows[i] not in parsed\_dependencies:

                parsed\_dependencies[flows[i]] = []

            parsed\_dependencies[flows[i]].append((flows[i + 1], score))

    return parsed\_dependencies

def find\_multi\_layer\_dependencies(parsed\_dependencies):

    """Construct multi-layer dependencies from two-layer dependencies."""

    multi\_layer\_dependencies = {}

    for source\_flow, targets in parsed\_dependencies.items():

        for target\_flow, score in targets:

            if target\_flow in parsed\_dependencies:  # Check if the target has further dependencies

                for next\_target, next\_score in parsed\_dependencies[target\_flow]:

                    multi\_layer\_key = f"{source\_flow}, {target\_flow}, {next\_target}"

                    multi\_layer\_dependencies[multi\_layer\_key] = min(score, next\_score)  # Use the min score as the dependency strength

    return multi\_layer\_dependencies

# Example data

parsed\_dependencies = parse\_dependencies(dependencies)

multi\_layer\_dependencies = find\_multi\_layer\_dependencies(parsed\_dependencies)