سید عباس میرقاسمی – ۴۰۲۴۴۳۱۸۰

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:

```python
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 = {}
```

```python
    # 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)
```