A Hybrid Method for TCP Connection Attack Prediction

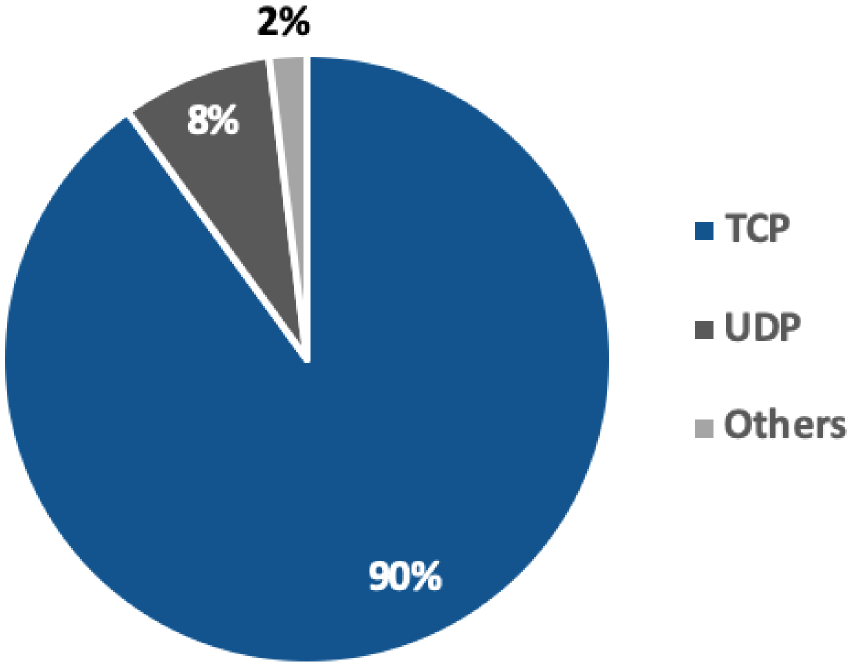
20194185 Fanchen Bu

20205646 Guoyuan An

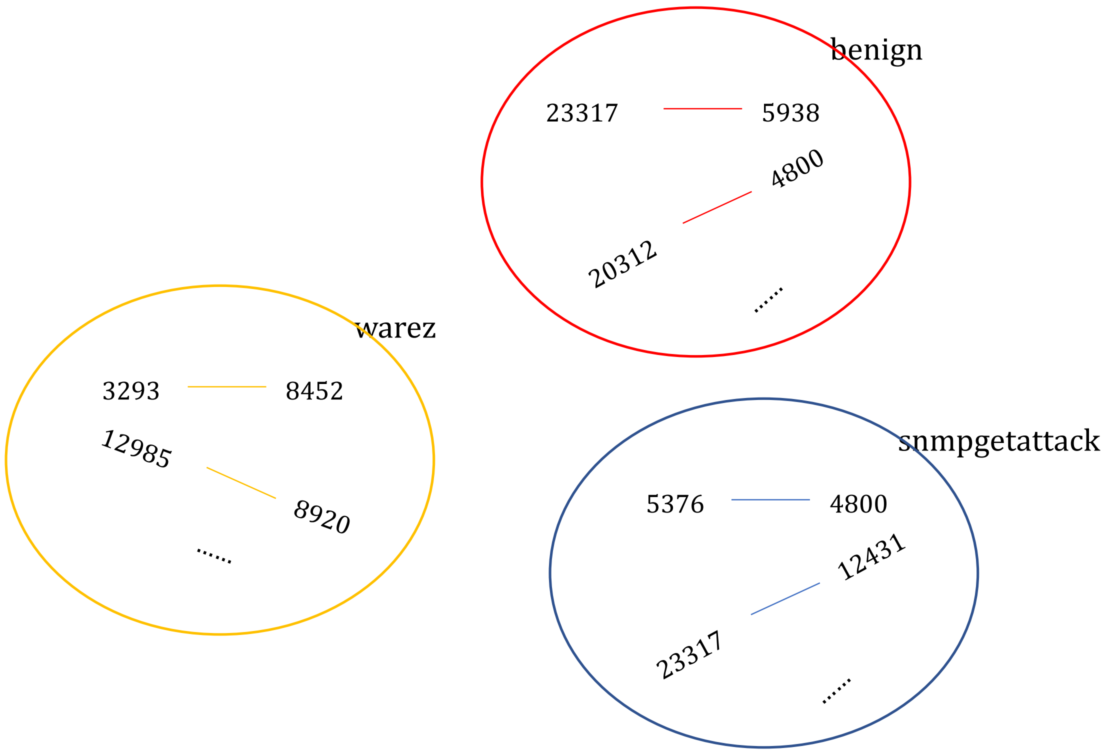
Contents

Introduction

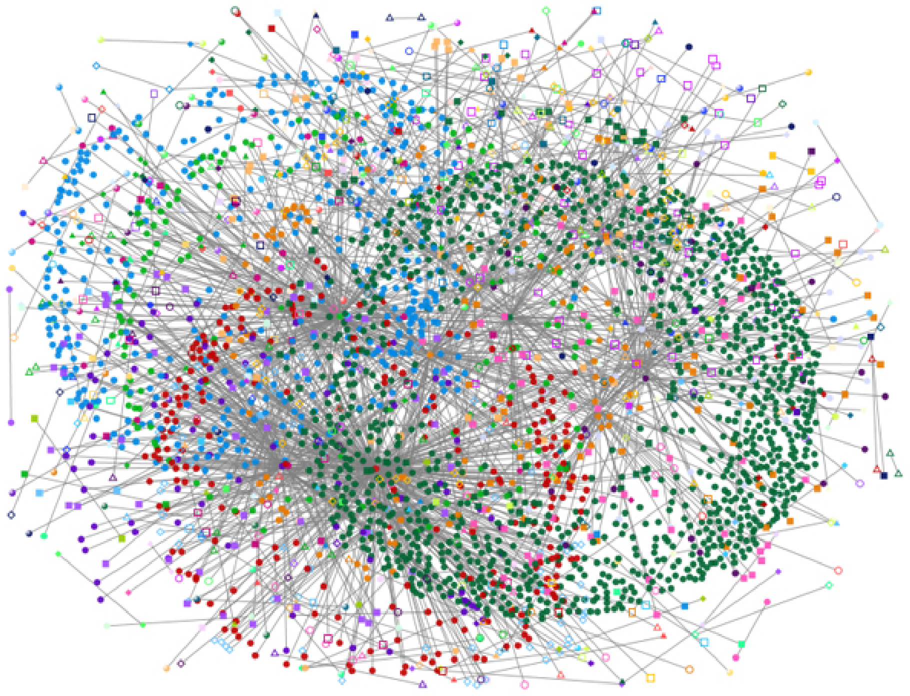
- TCP Connection Attack Prediction



Overview of our model

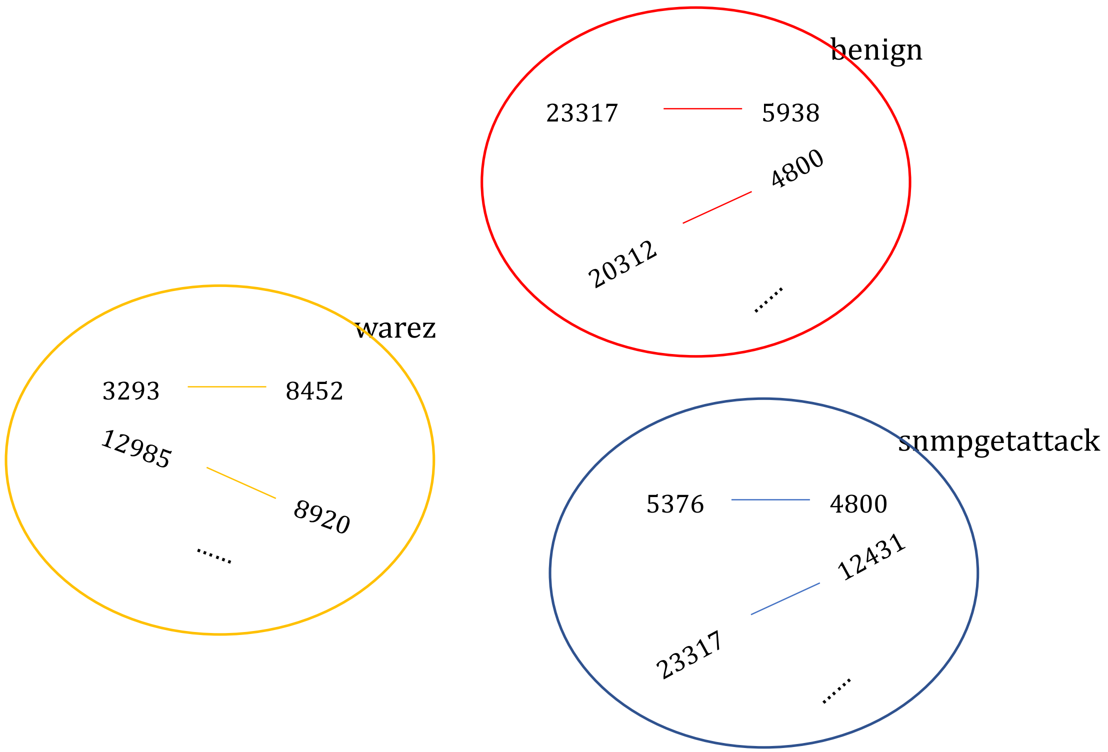


Stat-Filter



SE-GNN

Overview of our model



Stat-Filter

SE-GNN

Stat-Filter

- A statistics-based algorithm using only rules
- Fast and straightforward
- Small dataset
- Simple information

Data preprocessing

- Input: Source id, Destination id, Port, Timestamp
- Output: Connection type [stored in a list in dictionary order: ('-', 'apache2', …, 'warezclient')]
- The tuple sets are collected and the corresponding type tables are built
- We also collect the timestamps for each SDP tuple

```python
for i in range(TRAIN_SIZE):
    with open(os.path.join('train', 'train_%03d.txt' % i)) as f:
        data = f.readlines()
    for line in data:
        if line[-1] == '\n':
            line = line[:-1]
        info_list = line.split('\t')
        s, d, p, t = [int(info_list[j]) for j in range(4)]
        c = attack_types.index(info_list[4])
        train_sdp_total.add((s, d, p))
        train_sd_total.add((s, d))
        train_dp_total.add((d, p))
        train_sp_total.add((s, p))
        train_s_total.add(s)
        train_d_total.add(d)
        train_p_total.add(p)

        type_table_sdp[(s, d, p)].add(c)
        type_table_sd[(s, d)].add(c)
        type_table_dp[(d, p)].add(c)
        type_table_sp[(s, p)].add(c)
        type_table_s[s].add(c)
        type_table_d[d].add(c)
        type_table_p[p].add(c)

        time_list_sdp[(s, d, p)].add(t)
```

Observations on type tables

- Almost all SDP tuples in the training set correspond to a unique type
- → We assume that each SDP tuple in the validation set or the test set corresponds to a unique type
- What about other types of tuples?
- unique type in training set $\overset{?}{\rightarrow}$ same type in validation or test set

```
Counter(len(v) for v in type_table_sdp.values())
# Counter({1: 308996, 2: 77})
Counter(tuple(v) for v in type_table_sdp.values() if len(v) > 1)
# Counter({(0, 13): 34, (0, 7): 16, (16, 6): 7, (0, 3): 5, (0, 17): 4,
 # (24, 0): 3, (0, 15): 2, (0, 4): 2, (18, 7): 1, (8, 0): 1, (0, 16): 1, (0, 23): 1})
Counter(len(v) for v in type_table_sd.values())
# Counter({1: 54422, 2: 448, 3: 56, 4: 16, 6: 2})
Counter(len(v) for v in type_table_dp.values())
# Counter({1: 247259, 2: 4587, 3: 23, 4: 1})
Counter(len(v) for v in type_table_sp.values())
# Counter({1: 185023, 2: 3971})
Counter(len(v) for v in type_table_s.values())
# Counter({1: 9198, 2: 214, 4: 21, 3: 20, 5: 11, 6: 8, 7: 5, 9: 1})
Counter(len(v) for v in type_table_d.values())
# Counter({1: 20982, 2: 285, 3: 33, 4: 32, 5: 9, 7: 3, 6: 2, 13: 1, 18: 1})
Counter(len(v) for v in type_table_p.values())
# Counter({1: 16530, 2: 6386, 3: 1257, 4: 378, 5: 105, 6: 18, 7: 1})
```

Reliabilities of unique-type tuples

- SDP: 530/530
- DP: 524/525
- SP: 709/709
- SD: 60702/60799
- S: 11185/11185
- D: 32801/32801
- P: 1338/1344

```python
valid_unique_type_sdp = [(s, d, p) for (s, d, p) in valid_sdp_total if len(type_table_sdp[(s, d, p)]) == 1]
valid_unique_type_dp = [(s, d, p) for (s, d, p) in valid_sdp_total if len(type_table_dp[(d, p)]) == 1]
valid_unique_type_sp = [(s, d, p) for (s, d, p) in valid_sdp_total if len(type_table_sp[(s, p)]) == 1]
valid_unique_type_sd = [(s, d, p) for (s, d, p) in valid_sdp_total if len(type_table_sd[(s, d)]) == 1]
valid_unique_type_s = [(s, d, p) for (s, d, p) in valid_sdp_total if len(type_table_s[s]) == 1]
valid_unique_type_d = [(s, d, p) for (s, d, p) in valid_sdp_total if len(type_table_d[d]) == 1]
valid_unique_type_p = [(s, d, p) for (s, d, p) in valid_sdp_total if len(type_table_p[p]) == 1]
print(len(valid_unique_type_sdp),
      sum((type_table_sdp[(s, d, p)].issubset(possible_answers[(s, d, p)])) for (s, d, p) in valid_unique_type_sdp))
# 530 530 (all sdp tuples that can be found in the training set correspond to a unique type)
print(len(valid_unique_type_dp),
      sum((type_table_dp[(d, p)].issubset(possible_answers[(s, d, p)])) for (s, d, p) in valid_unique_type_dp))
# 525 524
print(len(valid_unique_type_sp),
      sum((type_table_sp[(s, p)].issubset(possible_answers[(s, d, p)])) for (s, d, p) in valid_unique_type_sp))
# 709 709
print(len(valid_unique_type_sd),
      sum((type_table_sd[(s, d)].issubset(possible_answers[(s, d, p)])) for (s, d, p) in valid_unique_type_sd))
# 60799 60702
Counter(min(type_table_sd[(s, d)]) for s, d, p in valid_sdp_total if
        len(type_table_sd[(s, d)]) == 1 and min(type_table_sd[(s, d)]) not in possible_answers[(s, d, p)])
# Counter({18: 81, 7: 6, 23: 6, 17: 3, 19: 1})
print(len(valid_unique_type_s),
      sum((type_table_s[s].issubset(possible_answers[(s, d, p)])) for (s, d, p) in valid_unique_type_s))
# 11185 11185
print(len(valid_unique_type_d),
      sum((type_table_d[d].issubset(possible_answers[(s, d, p)])) for (s, d, p) in valid_unique_type_d))
# 32801 32801
print(len(valid_unique_type_p),
      sum((type_table_p[p].issubset(possible_answers[(s, d, p)])) for (s, d, p) in valid_unique_type_p))
# 1344 1338
Counter((p, min(type_table_p[p])) for s, d, p in valid_sdp_total
        if len(type_table_p[p]) == 1 and min(type_table_p[p]) not in possible_answers[(s, d, p)])
# Counter({(43502, 18): 6})
```

Possible answers

- The possible types of an SDP tuple that appears in a query file is the set of attack types listed in the corresponding answer file + non-attack
- If an SDP tuple appears in multiple query files, we take the intersection

```python
possible_answers = dict.fromkeys(valid_sdp_total, set())
for i in range(VALID_SIZE):
    with open(os.path.join('valid_query', 'valid_query_%03d.txt' % i)) as f:
        query_data = f.readlines()
    with open(os.path.join('valid_answer', 'valid_answer_%03d.txt' % i)) as f:
        answer_data = f.readlines()
    if not answer_data:
        ground_truth = {0}
    else:
        ground_truth = set([attack_types.index(t)
                            for t in answer_data[0].split('\t')]) | {0}
    for line in query_data:
        if line[-1] == '\n':
            line = line[:-1]
        info_list = line.split('\t')
        s, d, p = [int(info_list[j]) for j in range(3)]
        if not possible_answers[(s, d, p)]:
            possible_answers[(s, d, p)] = ground_truth.copy()
        else:
            possible_answers[(s, d, p)].intersection_update(ground_truth)
```

Prediction on the unique-type tuples

- Q1: What about the 72 SDP tuples whose corresponding unique-type SD tuples fail to match?
- Q2: What about the tuples not containing unique-type tuples?
- Q3: What about the tuples never seen?

```
prediction_valid = dict.fromkeys(valid_sdp_total)
for sdp in valid_sdp_total:
    s, d, p = sdp
    sd = (s, d)
    dp = (d, p)
    sp = (s, p)
    if s not in train_s_total:  # s, never seen
        prediction_valid[sdp] = {0}
    elif sdp in train_sdp_total:  # sdp, seen before
        prediction_valid[sdp] = type_table_sdp[sdp].copy() \
            if len(type_table_sdp[sdp]) == 1 else {0}
    elif unique_type_s(s):  # s, unique type
        prediction_valid[sdp] = type_table_s[s].copy()
    elif unique_type_d(d):  # d, unique type
        prediction_valid[sdp] = type_table_d[d].copy()
    elif unique_type_p(p):  # p, unique type
        prediction_valid[sdp] = type_table_p[p].copy()
    elif unique_type_dp(s, d, p):  # dp, unique type
        prediction_valid[sdp] = type_table_dp[dp].copy()
    elif unique_type_sp(s, p):  # sp, unique type
        prediction_valid[sdp] = type_table_sp[sp].copy()
    elif unique_type_sd(s, d):  # sd, unique type
        prediction_valid[sdp] = type_table_sd[sd].copy()
```

Vague types

- Recall:

```
Counter(min(type_table_sd[(s, d)]) for s, d, p in valid_sdp_total if
        len(type_table_sd[(s, d)]) == 1 and min(type_table_sd[(s, d)]) not in possible_answers[(s, d, p)])
# Counter({18: 81, 7: 6, 23: 6, 17: 3, 19: 1})
Counter(tuple(type_table_sd[(s, d)]) for s, d, p in valid_sdp_total if
        len(type_table_sd[(s, d)]) > 1 and type_table_sd[(s, d)].isdisjoint(possible_answers[(s, d, p)]))
# Counter({(24, 23): 8, (24, 11, 23): 1, (1, 18): 1})
```

- 7: ipsweep, 17: satan, 18: smurf, 19: smurfttl, 23: warez, 24: warezclient
- Our strategy is dealing with these “vague types” first and then ignoring them in the following processes
- After dealing with them, we can assume that we can always pick the answer in the types of each sd tuple as long as it can be found in the training set

7: ipsweep

```python
Counter(tuple(v) for v in type_table_dp.values() if 7 in v)
# Counter({(7,): 11132, (0, 7): 34, (13, 7): 1, (12, 7): 1, (18, 7): 1, (11, 7): 1})
any(type_table_dp[(d, p)] == {7} for s, d, p in valid_sdp_total)
# False
Counter(tuple(v) for v in type_table_p.values() if 7 in v)
# Counter({(0, 7): 285, (0, 20, 7): 92, (0, 13, 7): 39, (0, 11, 20, 7): 32, (7,): 8,
# (0, 11, 7): 6, (0, 20, 23, 7): 6, (0, 7, 11, 20, 23): 4, (0, 12, 7): 1,
# (0, 18, 20, 7): 1, (0, 3, 20, 7): 1})
Counter(tuple(v) for v in type_table_s.values() if 7 in v)
# Counter({(0, 7): 11, (0, 16, 7): 5, (0, 12, 7): 2, (0, 7, 18, 23, 24): 2,
# (0, 6, 7, 16, 18, 23, 24): 1, (0, 7, 13, 22, 23): 1, (0, 2, 6, 7, 10, 23): 1,
# (0, 16, 3, 7): 1, (0, 7, 15, 17, 18): 1, (0, 5, 6, 7, 13, 16, 17, 23, 24): 1,
# (0, 3, 7): 1, (0, 24, 23, 7): 1, (0, 17, 7): 1, (0, 7, 12, 18, 23, 24): 1,
# (7, 18, 19, 23, 24): 1, (0, 3, 7, 18, 23, 24): 1, (0, 2, 7, 18, 23, 24): 1,
# (0, 7, 17, 18, 23, 24): 1})
Counter(tuple(v) for v in type_table_d.values() if 7 in v)
# Counter({(7,): 4305, (16, 7): 247, (0, 9, 12, 7): 14, (0, 9, 7): 7, (9, 7): 6,
# (0, 6, 7, 9, 16): 5, (9, 12, 7): 4, (0, 7): 2, (24, 23, 7): 2,
# (0, 3, 5, 7, 23, 24): 1, (0, 7, 9, 10, 13, 17, 22): 1, (0, 6, 7, 16, 18, 23, 24): 1,
# (0, 1, 2, 3, 6, 7, 9, 12, 13, 15, 17, 18, 22): 1, (0, 7, 23): 1,
# (0, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 22, 23, 24): 1,
# (0, 9, 16, 7): 1, (0, 23, 7): 1, (24, 18, 23, 7): 1, (0, 7, 17, 18, 23, 24): 1,
# (7, 12, 18, 23, 24): 1, (7, 9, 11, 12, 19): 1, (9, 18, 12, 7): 1, (16, 9, 6, 7): 1,
# (16, 9, 7): 1})
sum(type_table_d[d] == {7} for s, d, p in valid_sdp_total)
# 1169
sum(type_table_d[d] == {7} and 7 in possible_answers[(s, d, p)]
    for s, d, p in valid_sdp_total)
# 1169
sum(type_table_d[d] == {7, 16} for s, d, p in valid_sdp_total)
# 474
sum(type_table_d[d] == {7, 16} and 7 in possible_answers[(s, d, p)]
    for s, d, p in valid_sdp_total)
# 474
any(type_table_d[d] == {7, 16} and 16 in type_table_s[s]
    for s, d, p in valid_sdp_total)
# False
```

- It is highly concentrated on destination ids with unique type (7) or types 7 and 16
- All SDP tuples in the validation set containing SD tuples with types 7 and 16 actually have 7 as one of its possible answers
- Checking type tables of source ids is a possible way to provide statistical evidence

```python
elif unique_type_d(d):  # d, unique type
    prediction_valid[sdp] = type_table_d[d].copy()
elif type_table_d[d] == {7, 16}:
    prediction_valid[sdp] = {0} if 16 in type_table_s[s] else {7}
```

17: satan

```
Counter(tuple(v) for v in type_table_s.values() if 17 in v)
# Counter({(0, 3, 13, 14, 17): 1, (0, 7, 15, 17, 18): 1,
# (0, 5, 6, 7, 13, 16, 17, 23, 24): 1, (0, 17, 7): 1,
# (0, 3, 17, 18, 23, 24): 1, (1, 11, 17, 18, 23, 24): 1,
# (24, 17, 0, 23): 1, (0, 13, 17, 18, 23, 24): 1,
# (24, 0, 17, 23): 1, (0, 17, 18, 23, 24): 1,
# (0, 3, 17, 18, 20, 23, 24): 1,
# (0, 15, 17, 20, 21, 23, 24): 1, (0, 7, 17, 18, 23, 24): 1})
Counter(tuple(v) for v in type_table_d.values() if 17 in v)
# Counter({(24, 17, 18, 23): 2, (0, 17, 5): 1,
# (0, 7, 9, 10, 13, 17, 22): 1,
# (0, 1, 2, 3, 6, 7, 9, 12, 13, 15, 17, 18, 22): 1,
# (0, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 22, 23, 24): 1,
# (0, 17, 18, 23, 24): 1, (24, 17, 0, 23): 1, (0, 7, 17, 18, 23, 24): 1,
# (0, 15, 17, 20, 21, 23, 24): 1, (17,): 1})
[(s, d, p, possible_answers[(s, d, p)], type_table_s[s]) for s, d, p in valid_sdp_total
  if 17 in type_table_sd[(s, d)] and 17 in possible_answers[(s,d,p)]]
# []
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
    for sdp in train_sdp_total if type_table_sdp[sdp] == {17})
# Counter({(1, 0): 7, (15, 55): 2, (4, 0): 1, (3, 4): 1, (10822, 33): 1, (63, 28): 1,
# (4, 8): 1, (448, 28): 1, (4, 7): 1, (1126, 59): 1, (27, 12): 1, (2, 20): 1,
# (4, 28): 1, (3, 36): 1, (2, 5): 1, (3612, 59): 1, (2, 2): 1, (3, 26): 1,
# (10, 28): 1, (3, 5): 1, (2922, 42): 1})
```

- It is concentrated neither on source ids nor destination ids
- Besides, no obvious temporal patterns can be observed
- We just use some strong conditions

```
elif 17 in type_table_sd[sd]:
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {17} if
     connection_times > 1000 and
     span <= 0.1 * connection_times else {0}
```

18: smurf & 19: smurfttl

- Basically we need to check cases when source ids have type 18 and 19
- As 19 is much rarer than 18 and the temporal patterns are similar, we add a strict condition to make the algorithm tend to predict 18 in such cases

```python
Counter(tuple(v) for v in type_table_s.values() if 18 in v)
# Counter({(18,): 8700, (18, 19): 196, (24, 0, 18, 23): 5, (0, 7, 18, 23, 24): 2,
# (24, 18, 12, 23): 2, (0, 6, 7, 16, 18, 23, 24): 1, (0, 18, 13): 1,
# (0, 18, 4, 13): 1, (0, 7, 15, 17, 18): 1, (0, 1, 6, 13, 18, 23, 24): 1,
# (0, 3, 17, 18, 23, 24): 1, (2, 9, 12, 13, 18, 23, 24): 1,
# (1, 11, 17, 18, 23, 24): 1, (0, 10, 18, 23, 24): 1, (0, 7, 12, 18, 23, 24): 1,
# (7, 18, 19, 23, 24): 1, (0, 24, 18, 23): 1, (0, 3, 7, 18, 23, 24): 1,
# (24, 18, 10, 23): 1, (0, 13, 17, 18, 23, 24): 1, (0, 17, 18, 23, 24): 1,
# (0, 2, 7, 18, 23, 24): 1, (24, 18, 13, 23): 1, (0, 3, 17, 18, 20, 23, 24): 1,
# (0, 7, 17, 18, 23, 24): 1, (0, 13, 18, 23, 24): 1, (0, 18, 23): 1, (1, 18, 23): 1})
Counter(tuple(v) for v in type_table_d.values() if 18 in v)
# Counter({(18,): 7251, (24, 0, 18, 23): 6, (24, 18, 23): 5, (0, 18): 2,
# (0, 24, 18, 23): 2, (24, 17, 18, 23): 2, (0, 6, 7, 16, 18, 23, 24): 1,
# (0, 1, 2, 3, 6, 7, 9, 12, 13, 15, 17, 18, 22): 1, (0, 18, 6): 1,
# (0, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 22, 23, 24): 1,
# (0, 17, 18, 23, 24): 1, (24, 18, 23, 7): 1, (24, 18, 12, 23): 1,
# (0, 7, 17, 18, 23, 24): 1, (0, 18, 20, 23, 24): 1, (7, 12, 18, 23, 24): 1,
# (9, 18, 12, 7): 1, (24, 18, 13, 23): 1})
Counter(tuple(v) for v in type_table_d.values() if 19 in v)
# Counter({(0, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 22, 23, 24): 1, (7, 9, 11, 12, 19): 1})
Counter(tuple(v) for v in type_table_s.values() if 19 in v)
# Counter({(19,): 472, (18, 19): 196, (7, 18, 19, 23, 24): 1, (19, 22): 1})
len([(s, d, p) for s, d, p in valid_sdp_total if type_table_s[s] == {18}])
# 3294
len([(s, d, p) for s, d, p in valid_sdp_total if type_table_s[s] == {19}])
# 0
len([(s, d, p) for s, d, p in valid_sdp_total if type_table_s[s] == {18, 19}])
# 347
any((s, d, p) for s, d, p in valid_sdp_total if 19 in possible_answers[(s, d, p)])
# False
```

```python
elif unique_type_s(s):  # s, unique type
    prediction_valid[sdp] = type_table_s[s].copy()
elif type_table_s[s] == {18, 19}:
    prediction_valid[sdp] = {19} if 19 in type_table_dp[dp] else {18}
```

23: warez & 24: warezclient

- They are the most difficult two types in our analysis, which usually appear together and have very similar temporal patterns.

```python
Counter(tuple(type_table_sd[(s, d)]) for (s, d) in valid_sd_total if not type_table_sd[(s, d)].isdisjoint({23, 24}))
# Counter({(24, 23): 51, (24, 0, 23): 21, (24, 13, 23): 5, (24, 0, 18, 23): 3, (0, 24, 23): 3, (24, 18, 23): 2, (0, 24): 2,
# (23,): 2, (0, 6, 7, 16, 23, 24): 2, (0, 23, 7): 2, (0, 23): 1, (24, 8, 0, 23): 1, (24, 0, 23, 7): 1, (0, 24, 5, 23): 1,
# (24, 0, 13, 23): 1, (24, 17, 23): 1, (24, 11, 23): 1, (24, 9, 23): 1, (0, 23, 6, 7): 1, (0, 17, 24, 23): 1, (24, 12, 23): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {23})
# Counter({(1, 0): 322, (2, 0): 310, (2, 1): 11, (2, 3): 4})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {24})
# Counter({(1, 0): 282, (2, 0): 104, (4, 16): 44, (2, 1): 27, (2, 4): 13, (2, 3): 10, (4, 17): 8, (3, 13): 2, (3, 16): 1, (3, 12): 1})
Counter((len(valid_time_list_sdp[(s, d, p)]), valid_time_list_sdp[(s, d, p)][-1] - valid_time_list_sdp[(s, d, p)][0])
        for s, d, p in valid_sdp_total if type_table_sd[(s, d)] in ({23}, {24}, {23, 24}) and 23 in possible_answers[(s, d, p)])
# Counter({(2, 0): 65, (1, 0): 35, (2, 1): 3})
Counter((len(valid_time_list_sdp[(s, d, p)]), valid_time_list_sdp[(s, d, p)][-1] - valid_time_list_sdp[(s, d, p)][0])
        for s, d, p in valid_sdp_total if type_table_sd[(s, d)] in ({23}, {24}, {23, 24}) and 24 in possible_answers[(s, d, p)])
# Counter({(1, 0): 58, (2, 0): 35, (4, 16): 12, (2, 1): 10, (2, 4): 4, (2, 5): 2, (3, 9): 1, (4, 17): 1, (3, 8): 1})
```

- Besides, it’s observed that each validation file contains at most one of them.

```python
elif type_table_sd[sd] in ({23}, {24}, {23, 24}):  # 23, 24
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    if connection_times == 1 or connection_times >= 5:
        prediction_valid[sdp] = {0}
    else:
        prediction_valid[sdp] = {24} if span >= 1 else {23}
```

23: warez & 24: warezclient

- We also tried to analysis the inter-connection temporal patterns, but couldn't get any results

```python
sd_23_24 = [sd for sd in train_sd_total if type_table_sd[sd] == {23, 24}]
time_type_23_24 = defaultdict(SortedList)
for t, s, d, p, c in tsdpc_list:
    if (s, d) in sd_23_24:
        time_type_23_24[(s, d)].add((t, c))
for s, d, p in valid_sdp_total:
    if (s, d) in sd_23_24 and \
            (23 in possible_answers[(s, d, p)] or 24 in possible_answers[(s, d, p)]):
        c = min(possible_answers[(s, d, p)].intersection({23, 24}))
        for t in valid_time_list_sdp[(s, d, p)]:
            time_type_23_24[(s, d)].add((t, c + 0.1))
for s, d, p in test_sdp_total:
    if (s, d) in sd_23_24:
        # c = min(possible_answers[(s,d,p)].intersection({23, 24}))
        for t in test_time_list_sdp[(s, d, p)]:
            time_type_23_24[(s, d)].add((t, 23.5))
```

```
(4323, 8452)
140 TEST
290 23
293 TEST
306 24(V)
482 23
633 TEST
784 23
811 24
920 TEST
1108 TEST
1195 23
1258 23
1272 23(V)
1284 23
1489 23
1518 24(V)
1544 23
1601 24
1707 23
1758 23

(3293, 8452)
191 23
248 23
344 23
428 TEST
939 23
1090 23
1128 24
```

```
1142 24
1256 23
1324 TEST
1373 23
1480 23
1554 23
1571 23
1735 23
1770 24

(12985, 12653)
47 24
201 24
201 24
304 23
304 23
376 23(V)
376 23(V)
390 24
390 24
402 24
406 24
529 23
529 23
580 23
580 23
592 TEST
592 TEST
595 24
595 24
```

```
647 23
647 23
669 23
669 23
736 TEST
736 TEST
754 TEST
812 TEST
812 TEST
841 23
841 23
922 23
922 23
981 23
981 23
1005 23
1005 23
1059 23
1059 23
1097 24
1146 23
1146 23
1335 24
1335 24
1347 24
1348 TEST
1348 TEST
1351 24
1494 24
1498 23
1498 23
```

```
(12985, 15235)
19 24
19 24
263 24
444 TEST
444 TEST
499 24
593 TEST
593 TEST
605 TEST
609 TEST
706 24
707 23
707 23
799 24
799 24
824 24
824 24
837 24
841 24
1022 TEST
1022 TEST
1033 24
1085 23(V)
1085 23(V)
1290 23
1290 23
1317 23
1317 23
```

```
1355 24
1355 24
1367 24
1371 24
1482 TEST
1482 TEST
1528 24
1529 24
1675 23
1675 23
1692 24(V)
1715 TEST
1715 TEST

(12985, 22570)
25 23
25 23
54 TEST
54 TEST
66 TEST
70 TEST
98 23
98 23
214 24
214 24
276 24
277 24
298 24
299 24
459 23(V)
```

```
459 23(V)
584 TEST
584 TEST
616 23(V)
616 23(V)
681 23
681 23
747 TEST
747 TEST
774 TEST
775 TEST
813 TEST
1032 24
1032 24
1091 24(V)
1234 24
1253 24
1254 24
1412 24(V)
1412 24(V)
1424 24(V)
1428 24(V)
1576 24
1576 24
1589 24
1593 24
1594 24
1594 24
1714 TEST
1714 TEST
1776 24
```

```
1776 24

(1013, 8452)
7 23
81 23
138 23
155 23(V)
304 23
392 24
400 24
484 24
720 24
803 23
866 23
940 TEST
990 TEST
1038 24
1090 TEST
1413 24
1426 24
1437 23
1470 23
1528 23
1619 TEST

(12985, 12678)
81 23
81 23
176 TEST
176 TEST
```

```
188 TEST
192 TEST
326 24
326 24
338 24
342 24
371 24(V)
372 24(V)
382 23
382 23
470 TEST
471 TEST
474 24
475 24
589 23
589 23
796 23
796 23
852 23(V)
852 23(V)
990 23
990 23
1029 23
1029 23
1094 23
1094 23
1284 24
1368 24
1446 24
1568 23(V)
1568 23(V)
```

TODO: Non-unique-type tuples

- Now, we need to deal with SD tuples with multiple types.

```python
if s not in train_s_total:  # s, never seen
    prediction_valid[sdp] = {0}
elif sdp in train_sdp_total:  # sdp, seen before
    prediction_valid[sdp] = type_table_sdp[sdp].copy() \
    if len(type_table_sdp[sdp]) == 1 else {0}
elif unique_type_s(s):  # s, unique type
    prediction_valid[sdp] = type_table_s[s].copy()
elif unique_type_d(d):  # d, unique type
    prediction_valid[sdp] = type_table_d[d].copy()
elif type_table_d[d] == {7, 16}:
    prediction_valid[sdp] = {0} if 16 in type_table_s[s] else {7}
elif unique_type_p(p):  # p, unique type
    prediction_valid[sdp] = type_table_p[p].copy()
elif 17 in type_table_sd[sd]:
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {17} if connection_times > 1000 \
    and span <= 0.1 * connection_times else {0}
elif unique_type_dp(s, d, p):  # dp, unique type
    prediction_valid[sdp] = type_table_dp[dp].copy()
elif unique_type_sp(s, p):  # sp, unique type
    prediction_valid[sdp] = type_table_sp[sp].copy()
```

```python
elif type_table_sd[sd] in ({23}, {24}, {23, 24}):  # 23, 24
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    if connection_times == 1 or connection_times >= 5:
        prediction_valid[sdp] = {0}
    else:
        prediction_valid[sdp] = {24} if span >= 1 else {23}
elif unique_type_sd(s, d):  # sd, unique type
    prediction_valid[sdp] = type_table_sd[sd].copy()
```

Which types need our attention?

- Based on the prediction results on the validation set, we found that types 3, 6, 12, 13, 16, 20, 21 can be easily missed
- Besides, there are also some types can never be covered by unique-type SD tuples, because none has a unique type as one of them
- Note that some types don't appear in the validation set

```
sorted(set([min(v) for v in type_table_sd.values() if len(v) == 1]))
# [0, 3, 6, 7, 9, 10, 11, 12, 13, 16, 17, 18, 19, 20, 22, 23]
# i.e., the remaining types are [1, 2, 4, 5, 8, 14, 15, 21]
```

12: pod & 13: portsweep

```
Counter(tuple(type_table_sd[(s, d)]) for s, d, p in valid_sdp_total if 12 in type_table_sd[(s, d)] and 12 in possible_answers[(s, d, p)])
# Counter({(12,): 21, (24, 12, 23): 1, (12, 13): 1})
Counter(tuple(type_table_sd[(s, d)]) for s, d, p in valid_sdp_total if 13 in type_table_sd[(s, d)] and 13 in possible_answers[(s, d, p)])
# Counter({(24, 0, 13, 23): 315, (0, 13, 7): 231, (0, 18, 13): 129, (0, 13): 125, (0, 4, 13): 124, (13,): 69, (0, 2, 13): 30, (24, 13, 23): 8, (0, 3, 13): 2, (12, 13): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {13})
# Counter({(1, 0): 991, (2, 38): 450, (3, 40): 353, (2, 39): 116, (3, 41): 60, (2, 40): 47, (2, 20): 42, (3, 42): 29, (2, 41): 27, (29, 58): 23, (29, 59): 15, (28, 58): 15,
# (28, 57): 14, (2, 42): 11, (2, 46): 10, (2, 44): 10, (2, 45): 9, (2, 21): 8, (3, 44): 8, (27, 57): 8, (2, 43): 8, (3, 43): 7, (2, 36): 6, (2, 49): 5, (2, 48): 5, (28, 59): 5,
# (3, 46): 4, (2, 37): 4, (28, 56): 4, (2, 26): 4, (3, 53): 4, (2, 22): 4, (2, 53): 4, (2, 31): 3, (2, 32): 3, (3, 47): 3, (2, 28): 3, (27, 58): 3, (30, 59): 3, (2, 47): 3,
# (2, 24): 3, (3, 45): 3, (2, 51): 3, (30, 58): 2, (3, 52): 2, (2, 27): 2, (25, 51): 2, (2, 54): 2, (2, 33): 2, (2, 57): 2, (2, 50): 2, (3, 55): 2, (2, 1762): 2, (29, 57): 2 ...
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {12})
# Counter({(45, 0): 112, (1, 0): 43, (45, 1): 6, (447, 1): 1, (2, 0): 1, (40, 1): 1, (26, 1): 1, (445, 0): 1, (17, 0): 1, (450, 0): 1, (448, 1): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {7})
# Counter({(1, 0): 6648, (2, 3): 4441, (10, 54): 55, (9, 48): 3, (2, 6): 3, (8, 42): 2, (5, 24): 2, (10, 55): 1, (8, 51): 1, (7, 51): 1, (4, 18): 1, (5, 19): 1, (8, 31): 1,
# (10, 1795): 1, (7, 30): 1, (8, 50): 1, (4, 12): 1, (2, 0): 1, (3, 12): 1, (7, 36): 1, (4, 20): 1, (8, 37): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {18})
# Counter({(1, 0): 39192, (3, 2): 681, (8, 6): 270, (7, 6): 235, (63, 59): 175, (3, 1): 146, (62, 59): 134, (64, 59): 120, (7, 5): 110, (8, 7): 108, (61, 59): 87, (2, 2): 86,
# (2, 1): 86, (63, 58): 71, (6, 6): 70, (65, 59): 66, (61, 58): 61, (62, 58): 61, (8, 5): 57, (9, 6): 49, (6, 5): 46, (60, 59): 45, (60, 58): 39, (64, 58): 34, (7, 7): 29,
# (66, 59): 23, (59, 58): 23, (199, 24): 19, (197, 24): 17, (4, 2): 17, (9, 7): 16, (7, 4): 16, (62, 57): 15, (59, 59): 15, (58, 58): 13, (200, 24): 13, (195, 24): 12, (479, 59): 12,
# (480, 59): 12, (481, 59): 12, (201, 24): 11, (482, 59): 11, (167, 22): 11, (65, 58): 10, (163, 22): 10, (166, 22): 10, (478, 59): 10, (165, 22): 10, (167, 21): 10, (6, 4): 10 ...
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {2})
# Counter({(91, 59): 2, (92, 59): 2, (80, 59): 2, (66, 59): 1, (49, 29): 1, (21, 17): 1, (88, 59): 1, (77, 59): 1, (73, 21): 1, (70, 58): 1, (69, 59): 1, (94, 59): 1, (91, 58): 1,
# (41, 12): 1, (20, 15): 1, (78, 59): 1, (47, 36): 1, (65, 58): 1, (87, 59): 1, (83, 58): 1, (1, 0): 1, (97, 59): 1, (85, 59): 1, (72, 59): 1, (10, 7): 1, (90, 59): 1, (83, 59): 1,
# (14, 5): 1, (14, 7): 1, (79, 59): 1, (90, 58): 1, (2, 0): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {3})
# Counter({(1, 0): 218, (13, 54): 29, (14, 57): 26, (14, 58): 24, (13, 53): 21, (13, 55): 19, (14, 56): 16, (13, 56): 12, (14, 55): 10, (14, 59): 9, (13, 57): 4,
# (13, 52): 3, (12, 56): 3, (12, 51): 3, (12, 53): 3, (12, 57): 2, (2, 3): 2, (12, 52): 2, (12, 55): 2, (12, 54): 2, (11, 54): 2, (12, 47): 1, (12, 58): 1, (4, 10): 1,
# (13, 1794): 1, (2, 4): 1, (10, 41): 1, (13, 1796): 1, (15, 1797): 1, (14, 1797): 1, (11, 53): 1, (15, 57): 1, (13, 58): 1, (15, 59): 1, (11, 52): 1, (12, 48): 1,
# (5, 18): 1, (11, 45): 1, (2, 7): 1})
```

12: pod & 13: portsweep

```python
elif type_table_sd[sd] == {12, 13}:
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    if (2 <= connection_times <= 3 and 35 <= span <= 50) or \
            (27 <= connection_times <= 30 and 56 <= span >= 59) or \
            (connection_times == 2 and span == 20):
        prediction_valid[sdp] = {13}
    elif connection_times >= 10 and span <= 1:
        prediction_valid[sdp] = {12}
    else:
        prediction_valid[sdp] = {0}
elif 13 in type_table_sd[sd]:
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {13} if (2 <= connection_times <= 3 and 35 <= span <= 50) or \
                                    (27 <= connection_times <= 30 and 56 <= span >= 59) else {0}
elif 12 in type_table_sd[sd]:
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {12} if span <= 1 and connection_times >= 10 else {0}
```

3: dict

- Connections that connect only once are indistinguishable by temporal patterns
- We need "special" temporal patterns of each type

```
Counter(tuple(v) for v in type_table_sd.values() if 3 in v)
# Counter({(0, 3): 3, (0, 18, 3): 2, (0, 3, 13): 1, (18, 3): 1, (3,): 1})
Counter(tuple(type_table_sd[(s, d)]) for s, d, p in valid_sdp_total if
        3 in type_table_sd[(s, d)] and 3 in possible_answers[(s, d, p)])
# Counter({(0, 18, 3): 104, (0, 3): 68, (18, 3): 61, (3,): 52, (0, 3, 13): 6})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {3})
# Counter({(1, 0): 218, (13, 54): 29, (14, 57): 26, (14, 58): 24, (13, 53): 21,
# (13, 55): 19, (14, 56): 16, (13, 56): 12, (14, 55): 10, (14, 59): 9, (13, 57): 4,
# (13, 52): 3, (12, 56): 3, (12, 51): 3, (12, 53): 3, (12, 57): 2, (2, 3): 2,
# (12, 52): 2, (12, 55): 2, (12, 54): 2, (11, 54): 2, (12, 47): 1, (12, 58): 1,
# (4, 10): 1, (13, 1794): 1, (2, 4): 1, (10, 41): 1, (13, 1796): 1, (15, 1797): 1,
# (14, 1797): 1, (11, 53): 1, (15, 57): 1, (13, 58): 1, (15, 59): 1, (11, 52): 1,
# (12, 48): 1, (5, 18): 1, (11, 45): 1, (2, 7): 1})
Counter((len(valid_time_list_sdp[(s, d, p)]),
         valid_time_list_sdp[(s, d, p)][-1] - valid_time_list_sdp[(s, d, p)][0])
        for s, d, p in valid_sdp_total if 3 in type_table_sd[(s, d)]
        and 3 in possible_answers[(s, d, p)])
# Counter({(1, 0): 288, (2, 3): 2, (3, 9): 1})
```

```
elif type_table_sd[sd] == {3, 18}:
    prediction_valid[sdp] = {0} if 18 in type_table_dp[dp] else {3}
elif 3 in type_table_sd[sd]:
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {3} if 51 <= span <= 59 and 11 <= connection_times <= 14 else {0}
```

20: snmpgetattack & 21: snmpguess

```python
Counter(tuple(v) for v in type_table_sd.values() if 20 in v)
# Counter({(20, 21): 2, (20,): 2})
Counter(tuple(v) for v in type_table_sd.values() if 21 in v)
# Counter({(20, 21): 2})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {20})
# Counter({(12, 55): 3339, (12, 56): 2727, (11, 51): 705, (11, 55): 264,
# (11, 56): 223, (11, 50): 179, (10, 51): 76, (10, 55): 75, (10, 56): 69,
# (1, 0): 67, (9, 55): 67, (2, 5): 63, (12, 1795): 52, (8, 51): 50,
# (9, 56): 48, (10, 46): 42, (9, 51): 38, (8, 50): 38, (8, 55): 36,
# (10, 50): 34, (9, 50): 32, (12, 1796): 26, (10, 45): 26, (12, 1794): 26,
# (8, 46): 26, (8, 45): 24, (9, 46): 22, (7, 50): 18, (7, 45): 18, (7, 51): 16,
# (8, 56): 14, (11, 1795): 14, (7, 30): 12, (12, 57): 12, (9, 45): 12, (9, 40): 12,
# (11, 1796): 10, (4, 15): 9, (11, 1794): 8, (2, 10): 8, (7, 55): 8, (8, 35): 8,
# (12, 1797): 8, (13, 57): 7, (3, 10): 7, (13, 58): 6, (12, 1793): 6, (11, 52): 6...
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {21})
# Counter({(45, 58): 140, (45, 59): 33, (44, 57): 16, (56, 58): 15, (44, 58): 13,
# (54, 57): 12, (43, 57): 11, (1, 0): 11, (56, 59): 10, (42, 57): 9, (43, 58): 9,
# (55, 58): 7, (6, 6): 4, (55, 59): 4, (43, 56): 3, (45, 1799): 2, (37, 49): 2,
# (53, 57): 2, (43, 1798): 2, (49, 50): 1, (42, 58): 1, (48, 50): 1, (39, 50): 1,
# (52, 57): 1, (44, 59): 1, (8, 6): 1, (45, 1797): 1, (10, 12): 1, (57, 58): 1,
# (48, 51): 1, (42, 55): 1})
```

```python
elif type_table_sd[sd] == {20, 21}:
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    if connection_times == 1:
        prediction_valid[sdp] = {0}
    elif connection_times <= 13 and span >= 20:
        prediction_valid[sdp] = {20}
    elif connection_times >= 37 and 49 <= span <= 59:
        prediction_valid[sdp] = {21}
    else:
        prediction_valid[sdp] = {0}
```

6: ignore

```python
Counter(tuple(v) for v in type_table_sd.values() if 6 in v)
# Counter({(6,): 13, (0, 16, 6, 7): 4, (0, 6, 7, 16, 23, 24): 2, (16, 6, 7): 2, (0, 2, 6, 7): 1, (0, 23, 6, 7): 1, (0, 6): 1, (0, 1, 18, 6): 1})
Counter(tuple(type_table_sd[(s, d)]) for s, d, p in valid_sdp_total if 6 in type_table_sd[(s, d)] and 6 in possible_answers[(s, d, p)])
# Counter({(0, 6): 92})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {6})
# Counter({(1, 0): 130, (2, 38): 43, (2, 40): 30, (2, 39): 22, (2, 41): 4, (2, 42): 3, (2, 52): 1, (2, 47): 1, (2, 45): 1})
```

```python
elif 6 in type_table_sd[sd]:
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {6} if connection_times == 2 and 38 <= span <= 42 else {0}
```

16: saint

```
Counter(tuple(v) for v in type_table_sd.values() if 16 in v)
# Counter({(16, 7): 249, (0, 16, 6, 7): 4, (0, 16, 7): 3, (0, 16): 2, (0, 6, 7, 16, 23, 24): 2, (16, 6, 7): 2, (16, 0, 7): 1, (16,): 1})
Counter(tuple(type_table_sd[(s, d)]) for s, d, p in valid_sdp_total if 16 in type_table_sd[(s, d)] and 16 in possible_answers[(s, d, p)])
# Counter({(0, 16, 6, 7): 4, (0, 16): 4, (0, 16, 7): 3})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0]) for sdp in train_sdp_total if type_table_sdp[sdp] == {16})
# Counter({(2, 10): 344, (1, 0): 71, (3, 20): 43, (2, 9): 30, (3, 19): 12, (2, 0): 4, (8, 0): 4, (3, 9): 2, (4, 35): 2, (4, 25): 2, (12, 8): 1,
# (9, 42): 1, (9, 30): 1, (4, 29): 1, (3, 25): 1, (3, 24): 1, (3, 27): 1})
Counter((time_list_sdp[sdp][1] - time_list_sdp[sdp][0], time_list_sdp[sdp][2] - time_list_sdp[sdp][1])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {16} and len(time_list_sdp[sdp]) == 3)
# Counter({(10, 10): 43, (10, 9): 6, (9, 10): 6, (3, 6): 2, (10, 15): 1, (10, 14): 1, (10, 17): 1})
```

```
elif 16 in type_table_sd[sd]:
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    if connection_times == 2 and span == 10:
        prediction_valid[sdp] = {16}
    elif connection_times == 3 and \
            (timestamps[1] - timestamps[0], timestamps[2] - timestamps[1]) in ((10, 10), (9, 10), (10, 9)):
        prediction_valid[sdp] = {16}
    else:
        prediction_valid[sdp] = {0}
```

1: apache2

- Note that type 1 doesn't appear in the validation set.

```
Counter(tuple(v) for v in type_table_s.values() if 1 in v)
# Counter({(0, 1, 6, 13, 18, 23, 24): 1, (1, 11, 17, 18, 23, 24): 1, (1, 18, 23): 1})
Counter(tuple(v) for v in type_table_d.values() if 1 in v)
# Counter({(0, 1, 2, 3, 6, 7, 9, 12, 13, 15, 17, 18, 22): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {1})
# Counter({(48, 10): 2, (10, 6): 2, (216, 11): 1, (305, 59): 1, (37, 9): 1,
# (276, 28): 1, (185, 1797): 1, (52, 24): 1, (18, 26): 1, (463, 37): 1,
# (853, 25): 1, (31, 1): 1})
```

```
elif 1 in type_table_sd[sd]:  # 1
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {1} if connection_times >= 100 and span <= connection_times / 5 else {0}
```

2: back

- Type 2 has fairly special temporal patterns.

```
any((s, d) for (s, d, p) in valid_sdp_total if 2 in type_table_sd[(s, d)]
    and 2 in possible_answers[(s, d, p)])
# False
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {2})
# Counter({(91, 59): 2, (92, 59): 2, (80, 59): 2, (66, 59): 1, (49, 29): 1,
# (21, 17): 1, (88, 59): 1, (77, 59): 1, (73, 21): 1, (70, 58): 1, (69, 59): 1,
# (94, 59): 1, (91, 58): 1, (41, 12): 1, (20, 15): 1, (78, 59): 1, (47, 36): 1,
# (65, 58): 1, (87, 59): 1, (83, 58): 1, (1, 0): 1, (97, 59): 1, (85, 59): 1,
# (72, 59): 1, (10, 7): 1, (90, 59): 1, (83, 59): 1, (14, 5): 1, (14, 7): 1,
# (79, 59): 1, (90, 58): 1, (2, 0): 1})
```

```
elif 2 in type_table_s[s]:  # 2
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {2} if 60 <= connection_times <= 100 and 58 <= span <= 59 else {0}
```

4: guest & 15: rootkit

- As most connections with these types connect only once, and there are not enough samples for us to get more information, we don't do positive predictions for them

```
Counter(tuple(v) for v in type_table_sd.values() if 4 in v)
# Counter({(0, 4, 13): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {4})
# Counter({(1, 0): 68, (2, 3): 1})
Counter(tuple(v) for v in type_table_sd.values() if 15 in v)
# Counter({(0, 17, 15): 2})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {15})
# Counter({(1, 0): 6, (3, 5): 1})
```

5: httptunnel-e

- Most connections with type 5 have the same temporal pattern, we use that as our only prediction reference

```
Counter(tuple(v) for v in type_table_sd.values() if 5 in v)
# Counter({(0, 24, 5, 23): 1, (0, 5): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {5})
# Counter({(2, 30): 41, (3, 30): 2, (1, 0): 1, (2, 33): 1, (2, 0): 1, (3, 33): 1})
```

```
elif 5 in type_table_sd[sd]:  # 5
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {5} if connection_times == 2 and span == 30 else {0}
```

8: mailbomb & 14: processtable

```python
Counter(tuple(v) for v in type_table_sd.values() if 8 in v)
# Counter({(24, 8, 0, 23): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {8})
# Counter({(48, 58): 19, (48, 59): 14, (47, 58): 11, (47, 57): 8, (49, 59): 7,
# (48, 1799): 1, (6, 7): 1, (43, 59): 1, (47, 1799): 1, (42, 58): 1})
Counter(tuple(v) for v in type_table_sd.values() if 14 in v)
# Counter({(0, 14): 1})
Counter((len(time_list_sdp[sdp]), time_list_sdp[sdp][-1] - time_list_sdp[sdp][0])
        for sdp in train_sdp_total if type_table_sdp[sdp] == {14})
# Counter({(15, 56): 27, (15, 57): 5, (14, 53): 4, (7, 24): 2, (3, 8): 1,
# (14, 52): 1, (9, 34): 1, (13, 48): 1})
```

```python
elif 8 in type_table_sd[sd]:  # 8
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {8} if 47 <= connection_times <= 49 \
        and 57 <= connection_times <= 59 else {0}
elif 14 in type_table_sd[sd]:  # 14
    timestamps = valid_time_list_sdp[sdp]
    connection_times = len(timestamps)
    span = timestamps[-1] - timestamps[0]
    prediction_valid[sdp] = {14} if connection_times == 15 and \
        56 <= connection_times <= 57 else {0}
```

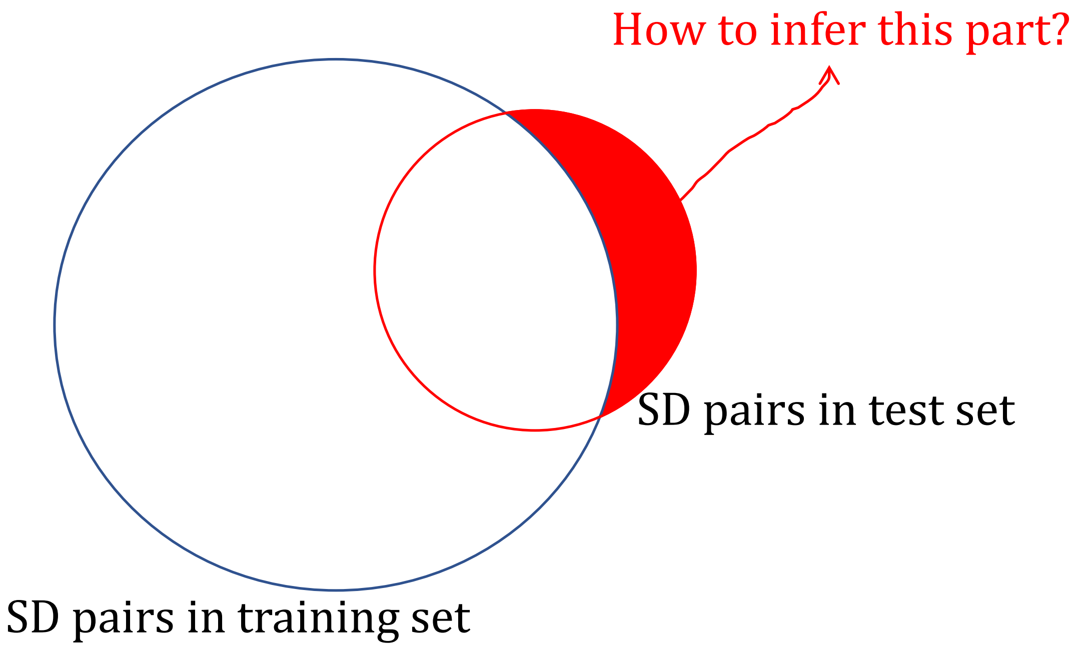
Final algorithm

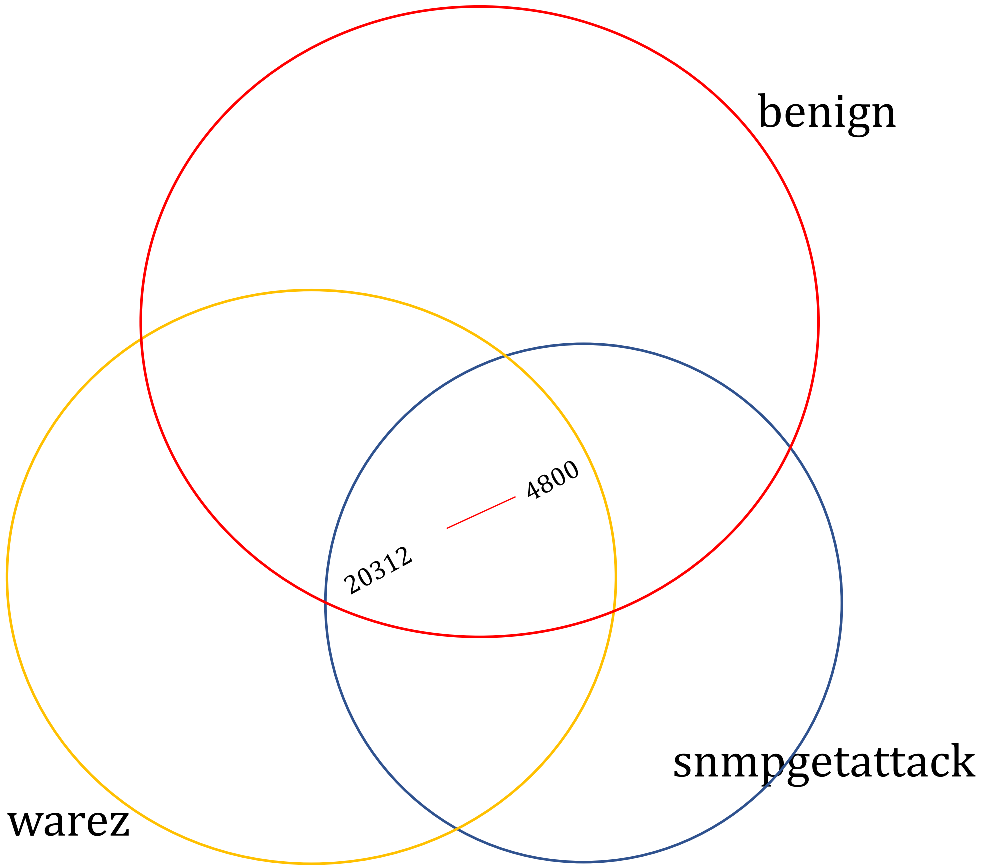
- It is a highly “if-else” algorithm based on rules
- It works well on the validation set but is hard to be generalized

Algorithm 1: Stat-Filter

```
input  : An SDP tuple x = (s, d, p) in V_SDP or E_SDP
output: Prediction of its type c(x)
CT ← |T(x)|
SP ← max T(x) − min T(x)
if s ∉ T_S then
    return 0
else if x ∈ T_SDP then
    if |A_SDP(x)| = 1 then
        return the only element in A_SDP(x)
    else
        return 0
    end
else if |A_S(s)| = 1 then
    return the only element in A_S(s)
else if A_S(s) = {18, 19} then
    if 19 ∈ A_DP(d, p) then
        return 19
    else
        return 18
    end
else if |A_D(d)| = 1 then
    return the only element in A_D(d)
else if A_D(d) = {7, 16} then
    if 16 ∈ A_S(s) then
        return 0
    else
        return 7
    end
else if |A_P(p)| = 1 then
    return the only element in A_P(p)
else if 17 ∈ A_SD(s, d) then
    if CT > 1000 and SP ≤ CT/10 then
        return 17
    else
        return 0
    end
else if |A_DP(d, p)| = 1 then
    return the only element in |A_DP(d, p)
else if |A_SP(s, p)| = 1 then
    return the only element in |A_SP(s, p)
else if A_SD ⊊ {23, 24} then
    if CT = 1 or CT ≥ 5 then
        return 0
    else if SP ≥ 1 then
        return 24
    else
        return 23
    end
else if |A_SD(s, d)| = 1 and A_SD contains no vague
  types then
    return the only element in A_SD(s, d)
else if A_SD(s, d) = {12, 13} then
    if 2 ≤ CT ≤ 3 and 35 ≤ SP ≤ 50 OR
      27 ≤ CT ≤ 30 and 56 ≤ SP ≤ 59 OR
      (CT, SP) = (2, 20) then
        return 13
    else if CT ≥ 10 and SP ≤ 1 then
        return 12
    else
        return 0
    end
else if 13 ∈ A_SD(s, d) then
    if 2 ≤ CT ≤ 3 and 35 ≤ SP ≤ 50 OR
      27 ≤ CT ≤ 30 and 56 ≤ SP ≤ 59 then
        return 13
    else
        return 0
    end
else if A_SD(s, d) = {3, 18} then
    if 18 ∈ A_DP(d, p) then
        return 0
    else
        return 3
else if 3 ∈ A_SD(s, d) then
    if 51 ≤ SP ≤ 59 and 11 ≤ CT ≤ 14 then
        return 3
    else
        return 0
else if 12 ∈ A_SD(s, d) then
    if CT ≥ 10 and SP ≤ 1 then
        return 12
    else
        return 0
else if A_SD(s, d) = {20, 21} then
    if CT = 1 then
        return 0
    else if CT ≤ 13 and SP ≥ 20 then
        return 20
    else if CT ≥ 37 and 49 ≤ SP ≤ 59 then
        return 21
    else
        return 0
else if 6 ∈ A_SD(s, d) then
    if CT = 2 and 38 ≤ SP ≤ 42 then
        return 6
    else
        return 0
else if 16 ∈ A_SD(s, d) then
    if (CT, SP) = (2, 10) OR CT = 3 and
      (T(x)[2] − T(x)[1], T(x)[1] − T(x)[0]) ∈
      {(10, 10), (9, 10), (9, 9)} then
        return 16
    else
        return 0
else if 1 ∈ A_SD(s, d) then
    if CT ≥ 100 and SP ≤ CT/5 then
        return 1
    else
        return 0
else if 2 ∈ A_S(s) then
    if 60 ≤ 100 and 58 ≤ SP ≤ 59 then
        return 2
    else
        return 0
else if 5 ∈ A_SD(s, d) then
    if (CT, SP) = (2, 30) then
        return 5
    else
        return 0
else if 8 ∈ A_SD(s, d) then
    if 47 ≤ CT ≤ 49 and 57 ≤ CT ≤ 59 then
        return 8
    else
        return 0
else if 14 ∈ A_SD(s, d) then
    if CT = 15 and 59 ≤ CT ≤ 57 then
        return 14
    else
        return 0
else
    return 0
```

Current challenges

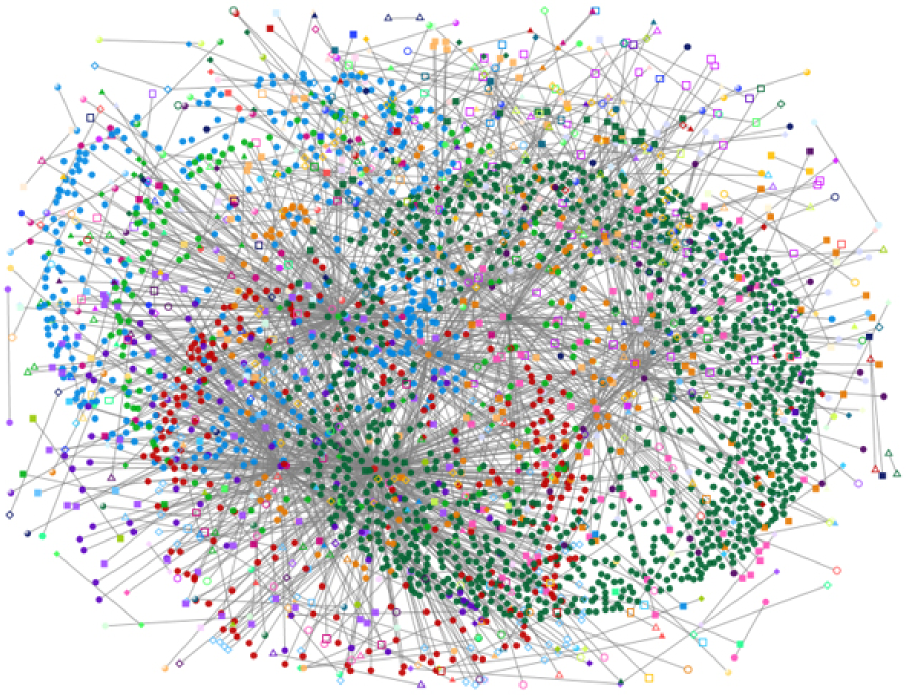




- Some SD pairs never appear in training set.
- Some SD pairs have many attack types

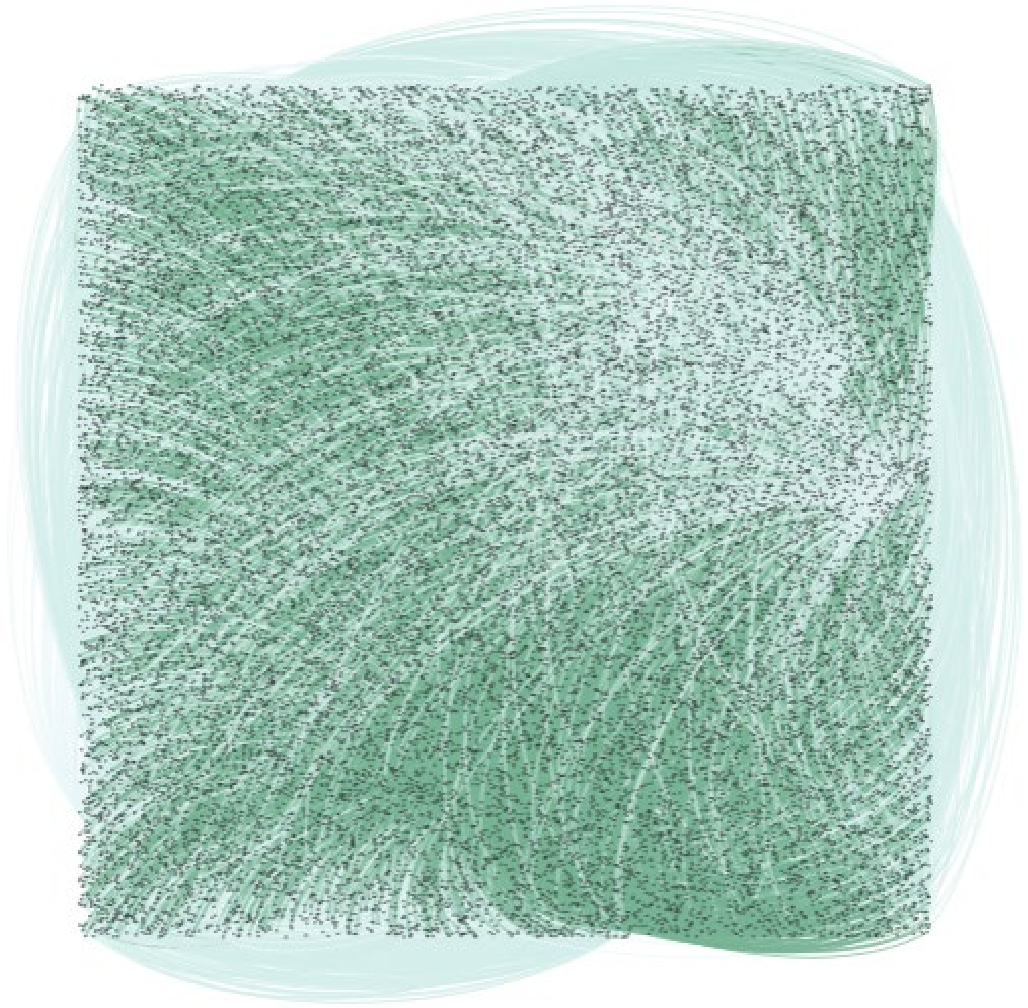
Overview of our model

Stat-Filter



SE-GNN

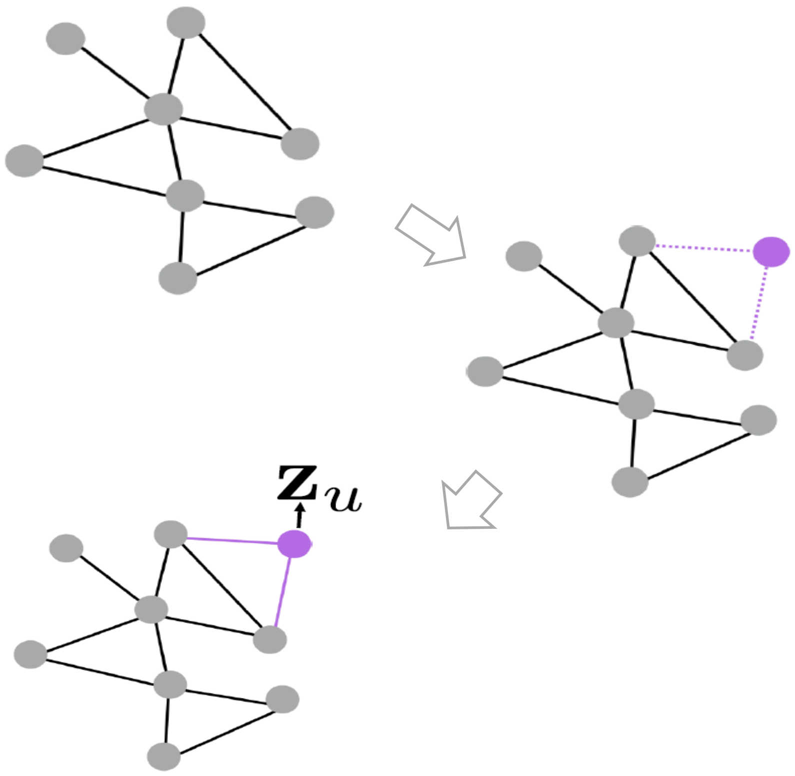
Solution: Graph



Drawn using Dephi

GraphSage

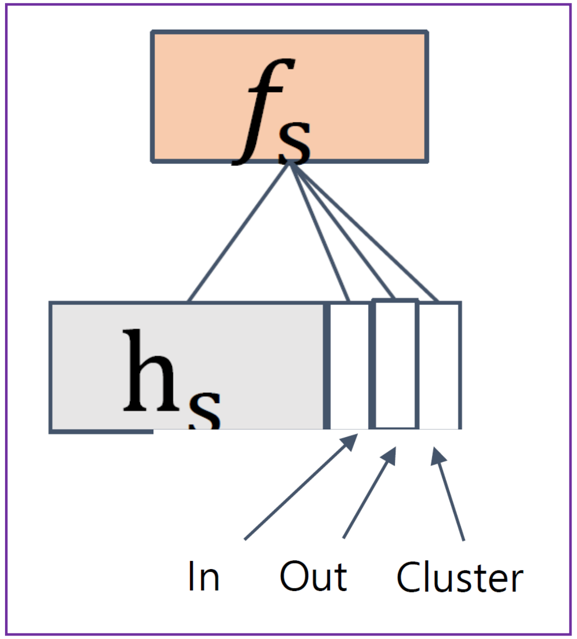
$$\mathbf{h}_v^k = \sigma\left(\left[W_k \cdot \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}, \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right) \qquad (1)$$

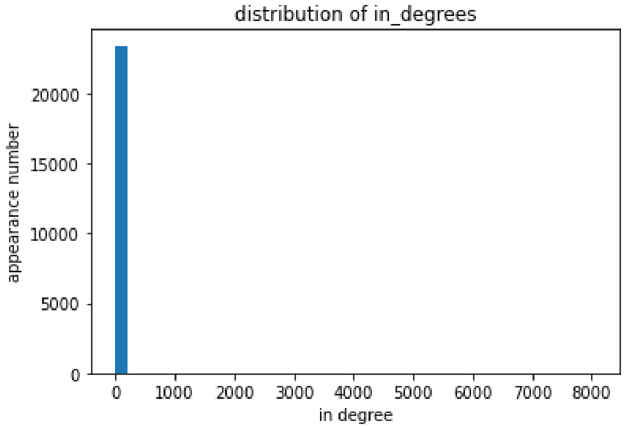


Spatial Information

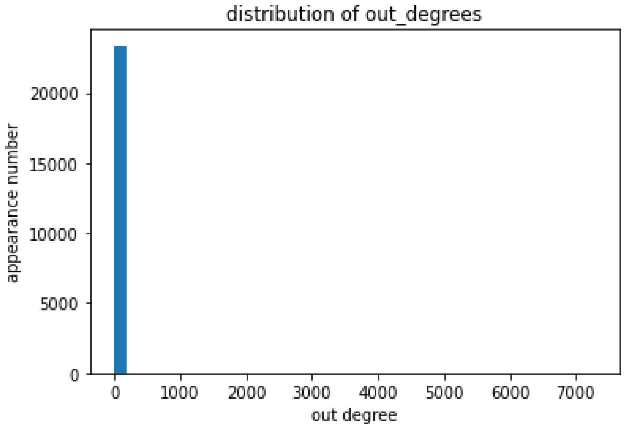
- We mainly consider in-degree, out-degree and clustering coefficient of each node.

$$\mathbf{f}_v = MLP_1\left([\mathbf{h}_v, In_v, Out_v, Cluster_v]\right), \tag{2}$$

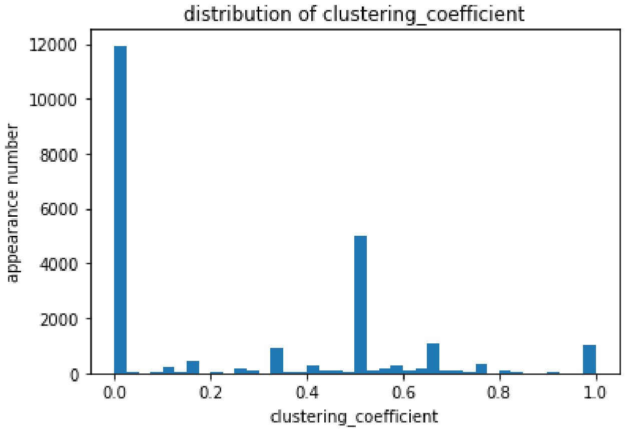




in-degree



out-degree

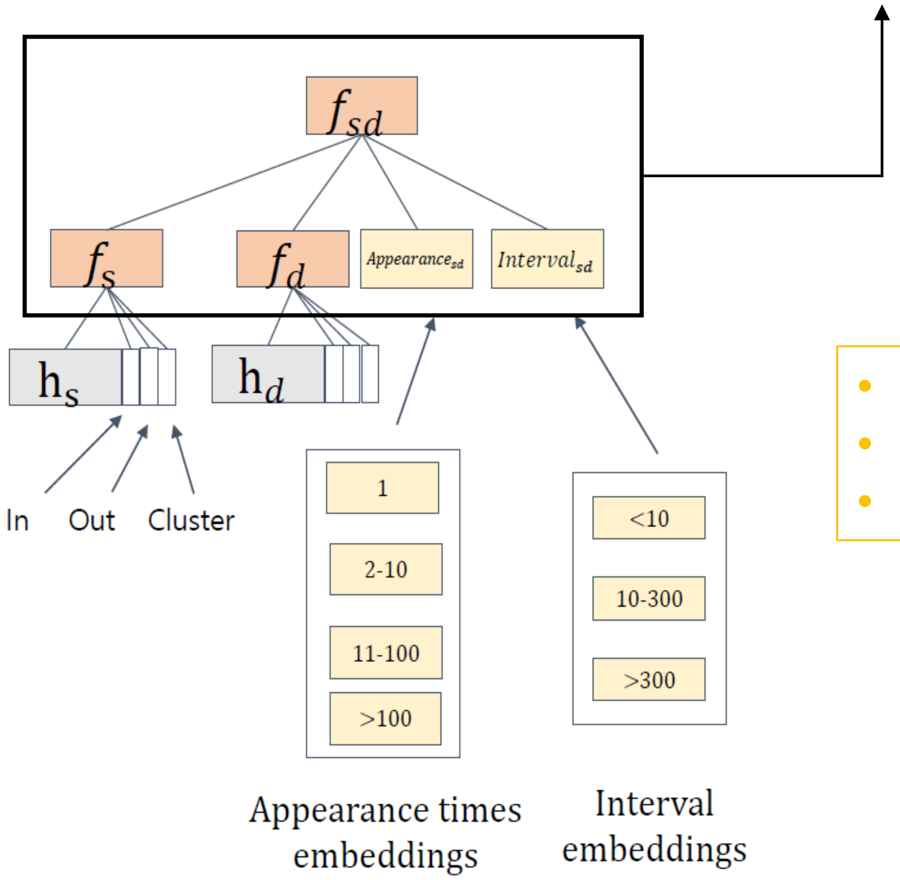


clustering coefficient

Temporal Information

- We mainly consider the appearance times and time interval of each SD pair in a limited time window.

$$\mathbf{f}_{sd} = MLP_2\left([\mathbf{f}_s, \mathbf{f}_d, \mathbf{Appearance}_{sd}, \mathbf{Interval}_{sd}]\right). \quad (3)$$



- For SD pair (5736,4800),
- Appearance time ==5,
- Time interval == 30s.

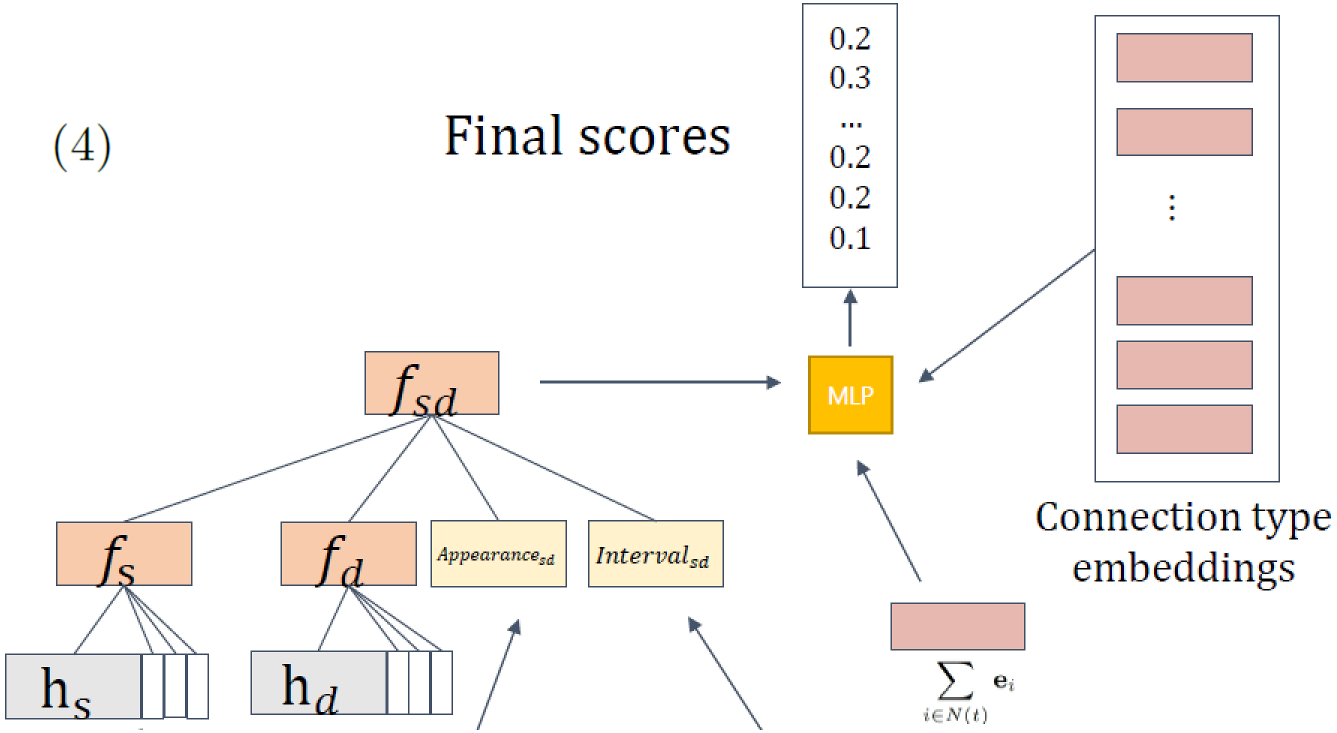
5376	4800	42629	243	snmpgetattack
23317	12431	42629	243	snmpgetattack
20312	4800	42629	243	-
23317	5938	42629	243	-
15394	4800	6870	245	-
23317	882	6870	245	-
3293	8452	6870	248	warez
12985	8920	6870	249	warez
12985	8920	6870	249	warez
20312	4800	6870	253	-
23317	5938	6870	253	-
5376	4800	6870	253	snmpgetattack
23317	12431	6870	253	snmpgetattack
5376	4800	6870	258	snmpgetattack
23317	12431	6870	258	snmpgetattack
20312	4800	6870	258	-
23317	5938	6870	258	-
20312	4800	6870	263	-
23317	5938	6870	263	-
5376	4800	6870	263	snmpgetattack
23317	12431	6870	263	snmpgetattack
20312	4800	6870	268	-
23317	5938	6870	268	-
5376	4800	6870	268	snmpgetattack
23317	12431	6870	268	snmpgetattack
20312	4800	6870	273	-
23317	5938	6870	273	-
5376	4800	6870	273	snmpgetattack
23317	12431	6870	273	snmpgetattack
20312	4800	6870	278	-

Relations among connection types

- We mainly consider the appearance times and time interval of each SD pair in a limited time window.

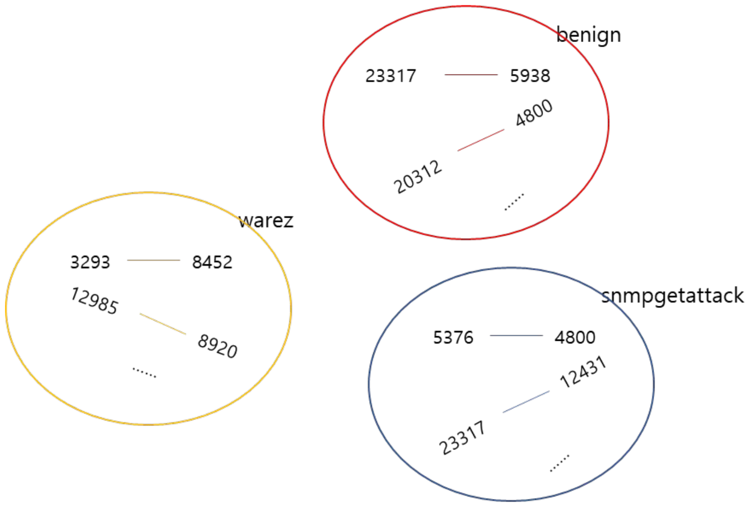
$$\hat{p}_{sd,t} = MLP_3\left(\left[\mathbf{f}_{sd}, \sum_{i\in N(t)} \mathbf{e}_i, \mathbf{e}_t\right]\right), \qquad (4)$$

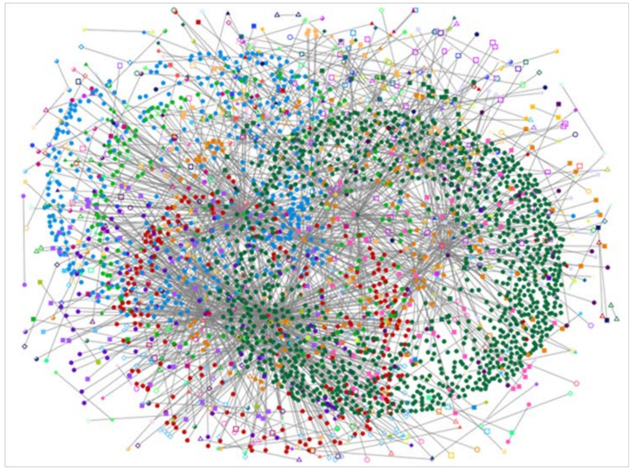
- Negative sampling is used.



Result and Conclusion

- We gave an extensive statistical analysis, even detailed into the appearance pattern of each attack type.
- We combined both the local information and global graph information.
- We achieved an amazing weighted F1 score in Valid set: 0.9663.





weighted F1 score in Valid set:
0.9663.

Stat-Filter

SE-GNN