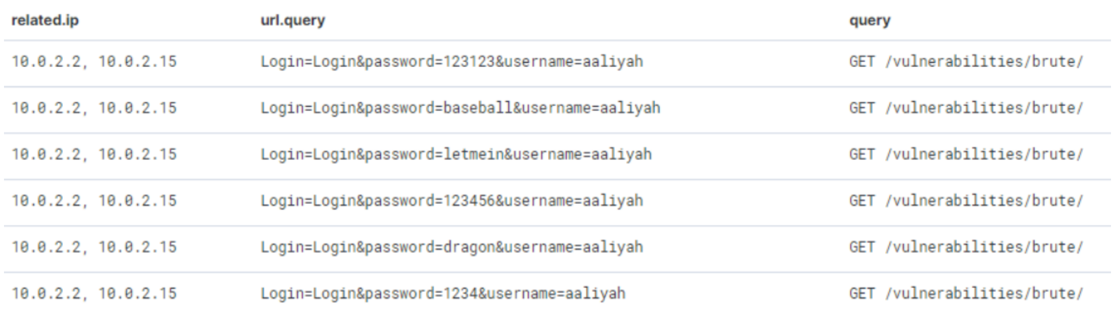
Network Security Project2 Report

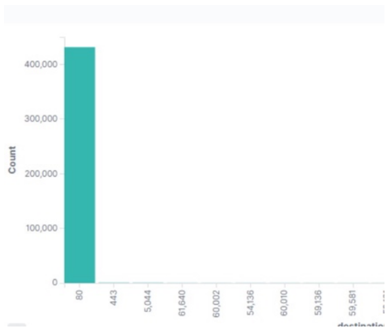
- Observation
  - Attack_1 & Test_1 -> Brute force attack
    - In the column query of packetbeat.json, there exists “GET /vulnerabilities/brute/”
    - The url query shows that the attacker tried to login to an account
    - Evidence



related.ip	url.query	query
10.0.2.2, 10.0.2.15	Login=Login&password=123123&username=aaliyah	GET /vulnerabilities/brute/
10.0.2.2, 10.0.2.15	Login=Login&password=baseball&username=aaliyah	GET /vulnerabilities/brute/
10.0.2.2, 10.0.2.15	Login=Login&password=letmein&username=aaliyah	GET /vulnerabilities/brute/
10.0.2.2, 10.0.2.15	Login=Login&password=123456&username=aaliyah	GET /vulnerabilities/brute/
10.0.2.2, 10.0.2.15	Login=Login&password=dragon&username=aaliyah	GET /vulnerabilities/brute/
10.0.2.2, 10.0.2.15	Login=Login&password=1234&username=aaliyah	GET /vulnerabilities/brute/

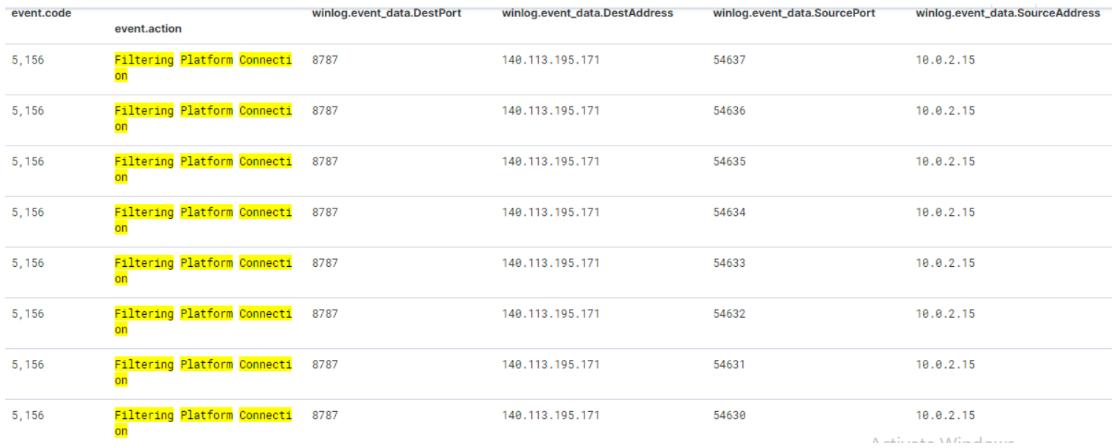
The picture above shows that the query detected brute force attack and the url query appears some login information.

  - Attack_2 & Test_2 -> DDos
    - A lot of port 80, more than 80% of port number in json file is 80.
    - Evidence



      - ✧ There are 744500 out of 907322 (84%) of port 80 in Test_2 packetbeat.json destination.dest_port
      - ✧ There are 431294 out of 510000 (82%) of port 80 in Attack_2 packetbeat.json

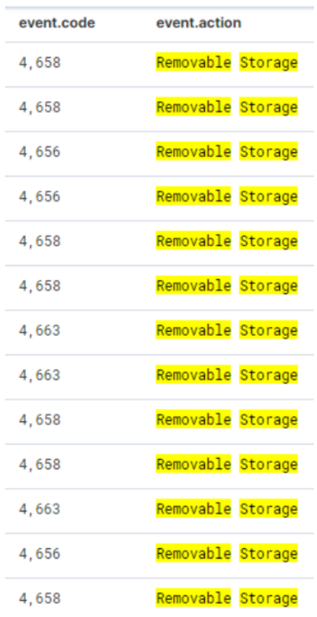
  - Attack_3 & Test_3 -> Port scanning
    - The windows security code 5156 (Filtering platform connection) occurs. And it indicates that the attackers try continuous port to attack the victim.
    - Evidence



event.code	event.action	winlog.event_data.DestPort	winlog.event_data.DestAddress	winlog.event_data.SourcePort	winlog.event_data.SourceAddress
5,156	Filtering Platform Connection	8787	140.113.195.171	54637	10.0.2.15
5,156	Filtering Platform Connection	8787	140.113.195.171	54636	10.0.2.15
5,156	Filtering Platform Connection	8787	140.113.195.171	54635	10.0.2.15
5,156	Filtering Platform Connection	8787	140.113.195.171	54634	10.0.2.15
5,156	Filtering Platform Connection	8787	140.113.195.171	54633	10.0.2.15
5,156	Filtering Platform Connection	8787	140.113.195.171	54632	10.0.2.15
5,156	Filtering Platform Connection	8787	140.113.195.171	54631	10.0.2.15
5,156	Filtering Platform Connection	8787	140.113.195.171	54630	10.0.2.15

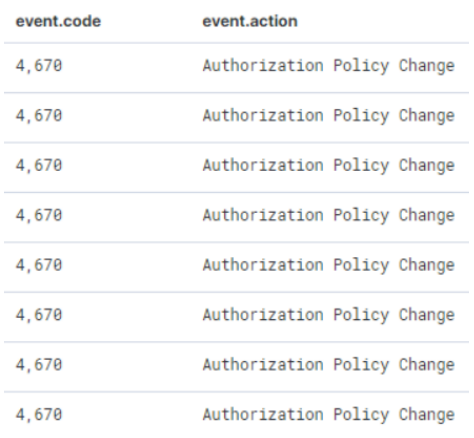
The victim is 140.113.195.171:8787, and the attacker is 10.0.2.15. The attacker had tried many ports(54630~54637…a lot more) to attack.

- Attack_4 & Test_4 -> Phishing attachment
  - There exists tar.exe and cmd.exe in winlogbeat.json file
  - "Authorization policy changed" (event.action) occurs in winlogbeat.json ,and its event code is 4670.
  - "Removable Storage" (event.action) occurs in winlogbeat.json.
  - Evidence



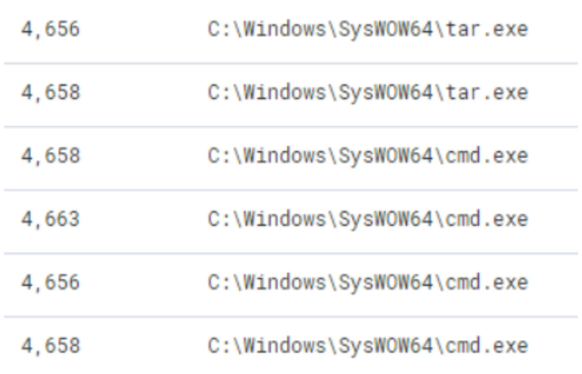
event.code	event.action
4,658	Removable Storage
4,658	Removable Storage
4,656	Removable Storage
4,656	Removable Storage
4,658	Removable Storage
4,658	Removable Storage
4,663	Removable Storage
4,663	Removable Storage
4,658	Removable Storage
4,658	Removable Storage
4,663	Removable Storage
4,656	Removable Storage
4,658	Removable Storage

According to windows website (link), the attacker may use removable storage device to enter a system.



event.code	event.action
4,670	Authorization Policy Change
4,670	Authorization Policy Change
4,670	Authorization Policy Change
4,670	Authorization Policy Change
4,670	Authorization Policy Change
4,670	Authorization Policy Change
4,670	Authorization Policy Change
4,670	Authorization Policy Change

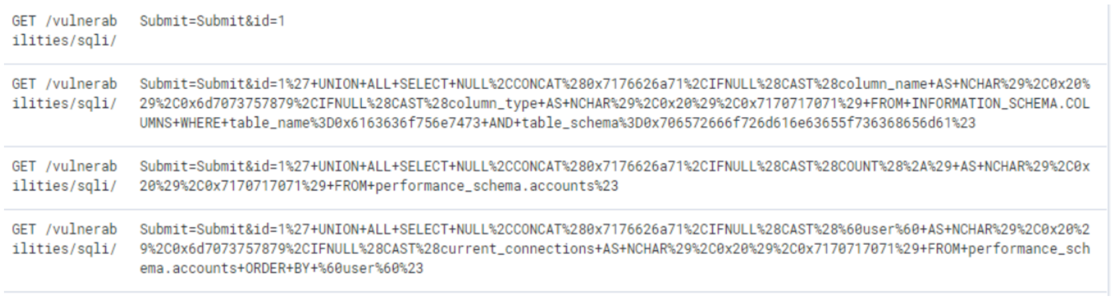
According to mitre attack website (link), the attacker may modify the file or directory permissions in order to evade access control lists. Besides, once the DACLs are modified, windows security log with event_id 4670 will occur.



4,656	C:\Windows\SysWOW64\tar.exe
4,658	C:\Windows\SysWOW64\tar.exe
4,658	C:\Windows\SysWOW64\cmd.exe
4,663	C:\Windows\SysWOW64\cmd.exe
4,656	C:\Windows\SysWOW64\cmd.exe
4,658	C:\Windows\SysWOW64\cmd.exe

We can see that there exists “tar.exe” and “cmd.exe” in winlog.event_data.ProcessName.

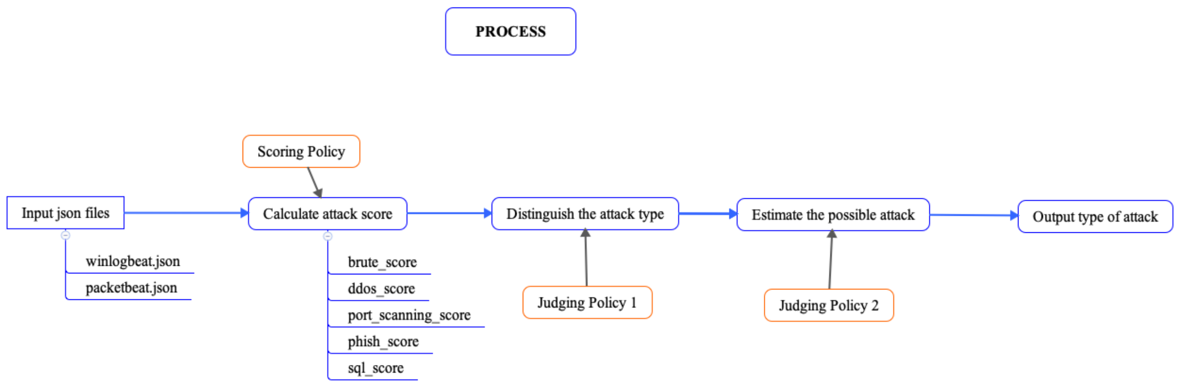
- Attack_5 & Test_5 -> SQL Injection
  - There exists SQL syntax keywords such as “SELECT” “WHERE” “FROM” “Submit=Submit&id=1” in column url of packetbeat.json.
  - In the column query of packetbeat.json, there exists “GET /vulnerabilities/sqli/”.
  - Evidence



GET /vulnerabilities/sqli/	Submit=Submit&id=1
GET /vulnerabilities/sqli/	Submit=Submit&id=1%27+UNION+ALL+SELECT+NULL%2CCONCAT%280x7176626a71%2CIFNULL%28CAST%28column_name+AS+NCHAR%29%2C0x20%29%2C0x6d7073757879%2CIFNULL%28CAST%28column_type+AS+NCHAR%29%2C0x20%29%2C0x7170717071%29+FROM+INFORMATION_SCHEMA.COLUMNS+WHERE+table_name%3D0x6163636f756e7473+AND+table_schema%3D0x706572666f726d616e63655f736368656d61%23
GET /vulnerabilities/sqli/	Submit=Submit&id=1%27+UNION+ALL+SELECT+NULL%2CCONCAT%280x7176626a71%2CIFNULL%28CAST%28COUNT%28%2A%29+AS+NCHAR%29%2C0x20%29%2C0x7170717071%29+FROM+performance_schema.accounts%23
GET /vulnerabilities/sqli/	Submit=Submit&id=1%27+UNION+ALL+SELECT+NULL%2CCONCAT%280x7176626a71%2CIFNULL%28CAST%28%60user%60+AS+NCHAR%29%2C0x20%29%2C0x6d7073757879%2CIFNULL%28CAST%28current_connections+AS+NCHAR%29%2C0x20%29%2C0x7170717071%29+FROM+performance_schema.accounts+ORDER+BY+%60user%60%23

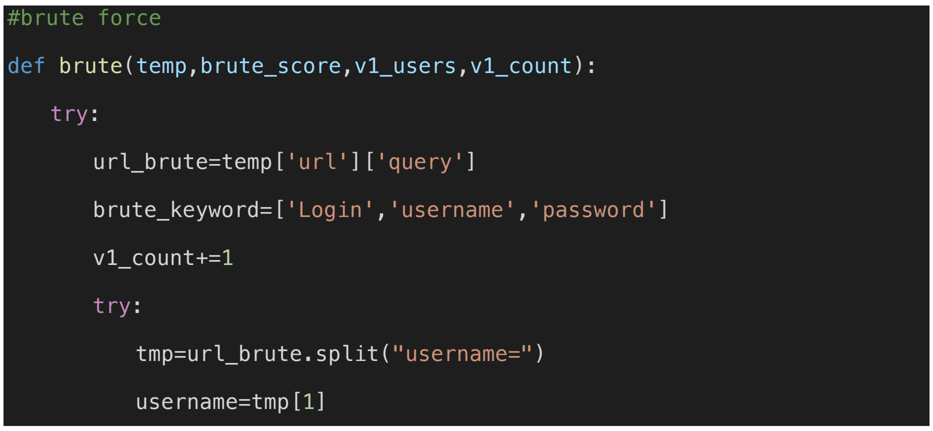
The picture above shows that the query detected sql injection and the url query indeed appear some syntaxes of sql.

- Model or algorithm
  - Process of my model



  - Model type
    - Rule-based
    - Based on the observation above, I set some rules and give the related score in order to distinguish five attacks.
    - The score is based on the frequency and the specificity of the feature
      - Since port 80 is pretty common, the score I gave to ddos_score will only increase 0.01 when a port 80 appears.
      - “Get /vulnerabilities/sqli/” only occurs in SQL injection, thus, I gave 20 points to sql_score each time the query is detected.
  - Scoring policy
    - Brute-Force attack (brute_score)

Behavior	Score/time
“Get /vulnerabilities/brute/” in “query”	20
“Login” or “username” or “password” in url.query	1



```
#brute force
def brute(temp,brute_score,v1_users,v1_count):
    try:
        url_brute=temp['url']['query']
        brute_keyword=['Login','username','password']
        v1_count+=1
        try:
            tmp=url_brute.split("username=")
            username=tmp[1]
            if username not in v1_users:
                v1_users.append(username)
        except:
            pass
        for keywords in brute_keyword:
            if keywords in url_brute:
                brute_score+=1
        query_brute=temp['query']
        match=re.search("GET.*vulnerabilities.*brute",query_brute)
        if(match):
            brute_score+=20
    except:
        brute_score+=0
    return brute_score,v1_users,v1_count
```

◆ DDoS (ddos_score)

Behavior	Score/time
Port 80 occurs	0.01

```
#ddos port 80
def ddos(temp,ddos_score,v2):
    try:
        port=temp['destination']['port']
        v2+=1
        if port==80:
            ddos_score+=0.01
    except:
        ddos_score+=0
    return ddos_score,v2
```

◆ Port Scan (port_scanning_score)

Behavior	Score/time
Winlog.event.code is 5156	1

```
#port scanning trait
def port_scanning(temp,port_scanning_score,dest_port,source_port,src_dest_port):
    try:
        event_code=temp['event']['code']
```

```
      if event_code==5156:
         port_scanning_score+=1
         try:
            tmp_dest_port=temp['winlog']['event_data']['DestPort']
            if tmp_dest_port not in dest_port:
               dest_port.append(tmp_dest_port)
               src_dest_port[tmp_dest_port]=[]
            tmp_source_port=temp['winlog']['event_data']['SourcePort']
            if tmp_source_port not in src_dest_port[tmp_dest_port]:
               src_dest_port[tmp_dest_port].append(tmp_source_port)
         except:
            pass
   except:
       port_scanning_score+=0
   return port_scanning_score,dest_port,source_port,src_dest_port
```

Since the feature is too weak, I wrote other function to judge port scanning attack, the followings are the function code:

```
def port_scanning_2(temp,src_dest_ip_port):
    try:
        dest_ip=temp['destination']['ip']
        dest_port=temp['destination']['port']
        src_ip=temp['source']['ip']
        src_port=temp['source']['port']

        keys_dest=dest_ip
        if keys_dest not in src_dest_ip_port:
            src_dest_ip_port[keys_dest]=dict()
        if src_ip not in src_dest_ip_port[keys_dest]:
            src_dest_ip_port[keys_dest][src_ip]=[]
        if src_port not in src_dest_ip_port[keys_dest][src_ip]:
            src_dest_ip_port[keys_dest][src_ip].append(src_port)
    except:
        pass
    return src_dest_ip_port
```

The dictionary I created in this function will be sent to judging policy 2 (judge_2 function). And judge_2 to will send src_dest_ip_port to Attack 3 filtering function (double_verify_port_scan).

- Phishing Email (phish_score)

Behavior	Score/time
“tar.exe” or “cmd.exe” appears in winlog.event_data.ProcessName	60
Event.action is “Authorization Policy Change” or “Removable Storage”	80

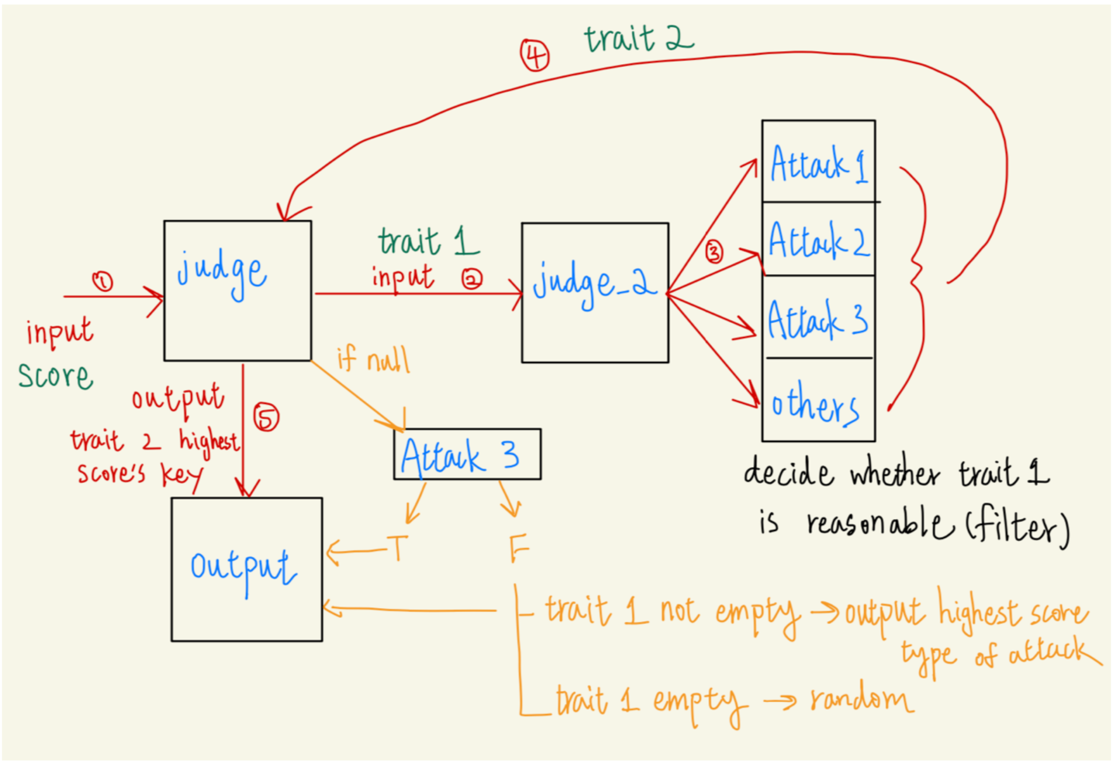
```
#phishing trait
def phish(temp,phish_score):
   try:
      suspicious=["cmd.exe","tar.exe"]
      process_name=temp['winlog']['event_data']['ProcessName']
      for words in suspicious:
         if words in process_name:
            phish_score+=60
   except:
      phish_score+=0
   try:
      event_action=["Removable Storage","Authorization Policy
 Change"]
      win_task=temp['event']['action']

      if win_task in event_action:
         phish_score+=80
   except:
      phish_score+=0
   return phish_score
```

- SQL Injection (sql_score)

Behavior	Score/time
“Get /vulnerabilities/sqli/” in “query”	20
“From” or “SELECT” or “Where” or “Submit=Submit&id=1” occurs in url.query	5

```
#sql rules
def sql(temp,sql_score):
    try:
        url_sql=temp['url']['query']
        sql_keyword=["FROM","SELECT","WHERE","Submit=Submit&id=1"]
        for keywords in sql_keyword:
            if keywords in url_sql:
                sql_score+=5
        query_sql=temp['query']
        match=re.search("GET.*vulnerabilities.*sql",query_sql)
        if(match):
            sql_score+=20
    except:
        sql_score+=0
    return sql_score
```

- Judging policy (judge+judge_2)
    - Process



④ trait 2
① input Score
judge
trait 1 input ②
judge_2
③
Attack 1
Attack 2
Attack 3
others
decide whether trait 1 is reasonable (filter)
if null
output trait 2 highest score's key
⑤
Attack 3
T
F
Output
trait 1 not empty → output highest score type of attack
trait 1 empty → random

    - Score-> dictionary with 5 attacks(keys) and its corresponding score(values).
        - {“Attack 1”:12345, “Attack 2”:666, “Attack 3”:0, “Attack 4”:0, “Attack 5”:0}

- Trait_1-> dictionary with 5 attacks(keys) and its corresponding score(values) larger than 0.
  - {"Attack 1":12345, "Attack 2":666}
- Trait_2->filtering result of Trait_1
  - The functions (double_verify_xxx) will determine whether the key and value pairs in Trait_1 can go to Trait_2
- Code:

  Function judge

```python
def judge(score,v1_users,src_dest_port,v2):

    trait_1=dict()
    for k,v in score.items():
        if v>0:
            trait_1[k]=v
    answer=""

    if len(trait_1)==0:
        if double_verify_port_scan(src_dest_port)==True:
                answer="Attack 3"
    else:
        tmp_1=sorted(trait_1.items(),key=lambda x:x[1],reverse=True)
        ans_1=tmp_1[0][0]
        trait_2=judge_2(trait_1,v1_users,src_dest_port,v2)
        tmp=sorted(trait_2.items(),key=lambda x:x[1],reverse=True)
        try:
            answer=tmp[0][0]
        except:
            answer=ans_1
            if double_verify_port_scan(src_dest_port)==True:
                if answer=="Attack 2":
                    answer="Attack 3"
    if answer=="":
        attacks=["Attack 1","Attack 2","Attack 3","Attack 4","Attack 5"]
        answer=random.choice(attacks)

    return answer
```

Function judge_2 (dispatcher)

```
def judge_2(trait_1,v1_users,src_dest_port,v2):
    trait_2=dict()
    for k,v in trait_1.items():
        if(k=="Attack 1"):
            if double_verify_brute(v,v1_users)==True:
                trait_2[k]=v
        elif(k=="Attack 2"):
            if double_verify_ddos(v,v2)==True:
                trait_2[k]=v
        elif(k=="Attack 3"):
            if double_verify_port_scan(src_dest_ip_port)==True:
                trait_2[k]=v
        else:
            trait_2[k]=v
    return trait_2
```

Decide whether to append trait_1 to trait_2

Attack 1 filtering function

```
def double_verify_brute(v,v1_list):
    if len(v1_list)>0:
        return True
    elif v>1000:
        return True
    return False
```

If no username in all url queries or score lower than 1000,don't append it to trait_2.

Attack 2 filtering function

```
def double_verify_ddos(v,v2):
    ratio=v*100/v2
    if ratio>0.6:
        return True
    else:
        return False
```

If the amount of port 80 is smaller than 60%, do not append it to trait_2.

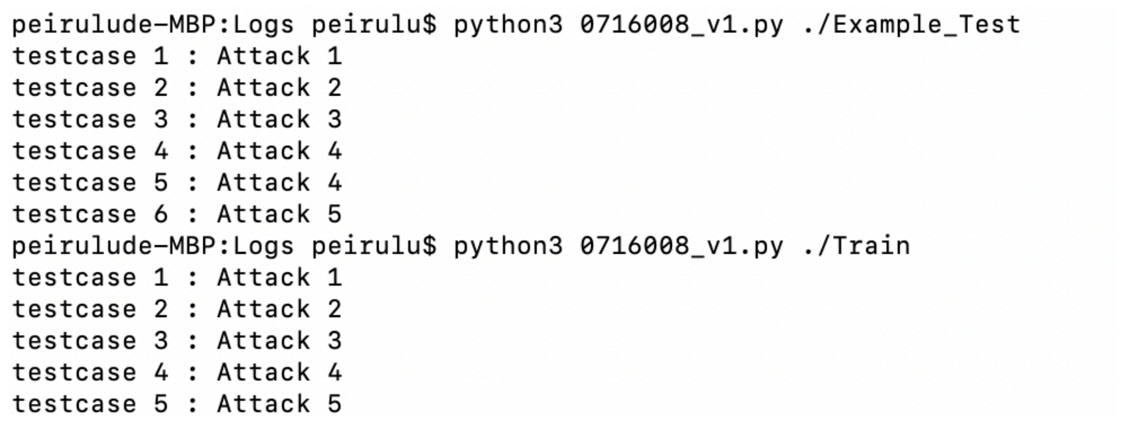
Attack 3 filtering function

```
def double_verify_port_scan(src_dest_ip_port):
    difference=dict()
    for key1,value1 in src_dest_ip_port.items():
        for key2,value2 in value1.items():
            if len(value2)>10:
                value2.sort()
                for i in range(1,len(value2)):
                    keyd=value2[i]-value2[i-1]
                    if keyd not in difference:
                        difference[keyd]=1
                    else:
                        difference[keyd]+=1
                try:
                    difference_value=0
                    try:
                        difference_value+=difference[1]
                        try:
                            difference_value+=difference[2]
                        except:
                            pass
                    except:
                        pass

                    rate=(difference_value/(len(value2)-1))
                    difference={}
                except:
                    rate=0
                if rate>0.5:
                    return True
    return False
```

If the continuous port < 50% in every dictionary data, don’t append the it to trait_2.
*The format of dictionary is {"dest ip”:{“source:ip”:[list of port]}}

The threshold of above 3 functions is based on observation, the threshold value isn’t very high since the goal of using these three functions is to filtering out the unreasonable types of attacks.

- Result & Accuracy
  - 100% accuracy in both Train folder and Example_Test folder



```
peirulude-MBP:Logs peirulu$ python3 0716008_v1.py ./Example_Test
testcase 1 : Attack 1
testcase 2 : Attack 2
testcase 3 : Attack 3
testcase 4 : Attack 4
testcase 5 : Attack 4
testcase 6 : Attack 5
peirulude-MBP:Logs peirulu$ python3 0716008_v1.py ./Train
testcase 1 : Attack 1
testcase 2 : Attack 2
testcase 3 : Attack 3
testcase 4 : Attack 4
testcase 5 : Attack 5
```

  - Score distribution in Train folder

	Brute_score	Ddos_score	Port_scan_score	Phish_score	Sql_score
Attack 1	353556	13335.36	0	0	0
Attack 2	0	7445.0	0	0	0
Attack 3	0	23.78	790	0	0
Attack 4	0	0.27	357	45440	0
Attack 5	2	404.58	0	0	28380

  - Score distribution in Example_Test folder

	Brute_score	Ddos_score	Port_scan_score	Phish_score	Sql_score
Test 1	78407	3220.5	0	0	0
Test 2	0	4312.94	0	0	0
Test 3	0	1.47	131	0	0
Test 4	0	0	123	3200	0
Test 4-2	0	0.04	35	6660	0
Test 5	0	48.12	0	0	1955

- Interesting/Problem encounter

This project is the most challenging project I have ever wrote so far. I started to work on the project pretty early but the progression speed of my project is slower than a sloth. At first, I tried to use filebeat to send the json files up to Kibana but some problems occur. The transmission of json file will stop by itself after a while. Thanks for TAs' help, I know that the problem occurs because the disk space of my ubuntu-server is not big enough. So, I install a new ubuntu-server and set its disk space to 100GB and delete the stdout in logstash.conf. On my third installation, my Kibana finally will not occur "Kibana is not loaded properly" after transmitting all the json file!

I have tried to work on writing machine learning python code after my Kibana works fine. I extract the feature which I used in my final version of rule-based model and feed it to a SVM model. The reason I choose SVM model is that it'll create one more dimension than the number of features, and thus it'll separate different types of data more accurately than KNN algorithm. However, things didn't work as I expected. The accuracy percentage of my model is not high at all. It's even lower than the result that I arbitrarily choose the feature (event.action and event.outcome in winlogbeat.json) and send it to the model! I tried to normalized my selected feature and use LabelEncoder() (a tool in sklearn.preprocessing) in order to prevent overfitting. The accuracy indeed rises. But it's still not ideal enough. After trying the models and features and forms of input array for a week, I thought that I got more familiar with machine learning. Unfortunately, the deadline is approaching in the meanwhile. Therefore, I decided to use the rule-based model which has much higher accuracy than my machine learning model.

I considered the most interesting part in this project is the process of finding distinguishing features of each attack. Even though it's like exploring an unknown area in sheer darkness, a beacon light will appear eventually. Just like how I found windows security log event_id 4670 (phishing trait), I found it while I was using python code to see the distribution of event actions of each types of attack. The surprising discovery is quite intriguing to me.