# AEIDS

## 概述

- 开发语言：Python2.7（稍加修改可用python3.3运行）
- 深度学习框架：Keras
- 深度学习算法：Autoencoder
- 数据集：UNSW-NBI5
- 数据库：PostgreSQL 9.5
- python依赖
    - keras
    - pcapy：处理pcap数据包
    - psycopg2：数据库交互
    - numpy
    - impacket：网络协议工具包

## 使用方法

- 安装依赖库与相关软件
- 在PostgreSQL中建立数据库，并导入aeids.sql中的模式，并修改aeids.py中的open_conn()函数
- 修改aeids.conf中的root_directory与training_filename，其中pcap的tcp连接数可以用wireshark查看
- 调用aeids.py开始训练 调用方法： `python aeids.py <training|predicting|testing|counting> <tcp|udp> <port> <hidden_layers> <activation_function> <dropout> <training filename> [batch_size] [testing filename]`

## 核心思想

- 分析pcap包，将其分为若干个tcp流，tcp流由若干数据包构成
- 用形如src_addr-src_port-dst_addr-dst_port-protocol唯一确定tcp流
- 用字节出现的频率表示一个tcp流，维度为256
- 将字节频率的256维向量作为模型的输入
- 预测时，对后续的tcp流同样计算字节频率，计算差值，超过阈值则异常

## 模块功能

### 主函数调用

程序入口为aeids.py中的main函数，由使用方法中的调用方式传入参数。

```
def main(argv):
    phase = argv[1]
    protocol = argv[2]
    port = argv[3]

    if phase != "counting":
        hidden_layers = argv[4].split(",")
        activation_function = argv[5]
        dropout = float(argv[6])
        batch_size = int(argv[8])
        filename = argv[7]

        if phase == "testing":
            aeids(phase, filename, protocol, port, hidden_layers,
activation_function, dropout, argv[8])
        else:
            aeids(phase, filename, protocol, port, hidden_layers,
activation_function, dropout, batch_size=batch_size)
    else:
        count_byte_freq(argv[4], protocol, port)
```

当phase为training时，执行以下逻辑

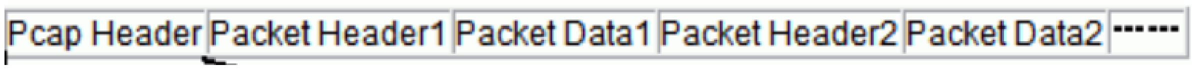
```
# 读取aeids.conf
read_conf()

# 初始化模型
autoencoder = init_model(hidden_layers, activation_function, dropout)

# 决定steps_per_epoch
steps_per_epoch = conf["training_filename"]["{}-{}".format(filename, port)] /
batch_size

# 使用tensorflow时，用tensorboard记录
if tensorboard_log_enabled and backend == "tensorflow":
    tensorboard_callback = TensorBoard(log_dir="./logs", batch_size=10000,
write_graph=True, write_grads=True,
                                        histogram_freq=1)
    # 训练模型，byte_freq_generator为读取pcap的generator
    autoencoder.fit_generator(byte_freq_generator(filename, protocol, port,
batch_size), steps_per_epoch=100,
                                  epochs=100, verbose=1, callbacks=
[tensorboard_callback])
    # 保存模型，方便下次load
    autoencoder.save("models/{}/aeids-with-log-{}-hl{}-af{}-
do{}.hdf5".format(filename, protocol + port, ",".join(hidden_layers),
activation_function, dropout), overwrite=True)

print("Training autoencoder finished. Calculating threshold...")
predict_byte_freq_generator(autoencoder, filename, protocol, port, hidden_layers,
activation_function, dropout, phase)
done = True
prt.cleanup_all_buffers()
prt = None
print("\nFinished.")
```

模型初始化，在层与层之间加入Dropout防止过拟合

```
def init_model(hidden_layers = [200, 100], activation_function ="relu", dropout =
0):
    input_dimension = 256
    # 实例化Keras向量
    input = Input(shape=(input_dimension,))

    for i in range(0, len(hidden_layers)):
        if i == 0:
            encoded = Dense(int(hidden_layers[i]), activation=activation_function)
(input)
        else:
            encoded = Dense(int(hidden_layers[i]), activation=activation_function)
(encoded)

        encoded = Dropout(dropout)(encoded)

    for i in range(len(hidden_layers) - 1, -1, -1):
        if i == len(hidden_layers) - 1:
            decoded = Dense(int(hidden_layers[i]), activation=activation_function)
(encoded)
        else:
            decoded = Dense(int(hidden_layers[i]), activation=activation_function)
(decoded)

        decoded = Dropout(0.2)(decoded)

    # 如果是二分类任务，最后一层一般是sigmoid激活函数，对应loss为binary_crossentropy;
    # 如果是多分类任务，最后一层一般是softmax激活函数，对应loss为
categorical_crossentropy。
    if len(hidden_layers) == 1:
        decoded = Dense(input_dimension, activation="sigmoid")(encoded)
    else:
        decoded = Dense(input_dimension, activation="sigmoid")(decoded)
    autoencoder = Model(outputs=decoded, inputs=input)
    autoencoder.compile(loss="binary_crossentropy", optimizer="adadelta")

    return autoencoder
```

## pcap包处理

### pcap包结构



#### Packet Header

- Timestamp(4B)：时间戳高位，精确到seconds，这是Unix时间戳。捕获数据包的时间一般是根据这个值
- Timestamp(4B)：时间戳低位，能够精确到microseconds
- Caplen(4B)：当前数据区的长度，即抓取到的数据帧长度，由此可以得到下一个数据帧的位置。
- Len(4B)：离线数据长度，网路中实际数据帧的长度，一般不大于Caplen，多数情况下和Caplen值一样

### 代码实现

因为pcap包数据量较大，所以针对keras.fit_generator构建一个generator，其中实现了线程类StreamReaderThread和TcpStream，负责读取pcap包，核心逻辑如下

#### StreamReaderThread

线程启动

```
prt = StreamReaderThread(get_pcap_file_fullpath(filename), protocol, port)
prt.start()
```

解析每一个数据包

```
def run(self):
    while not self.done:
        (header, frame) = self.pcap.next()
        if not header:
            break
        self.parse_packet(header, frame)

    self.empty_buffer()
    print("waiting for all threads to finish")
    while len(self.tcp_buffer) > 0:
        time.sleep(0.0001)
    print("main loop finished")
    self.done = True
```

对于每一个数据包，根据id判断其属于哪个tcp流，将其加入tcp流中。

```
# self.tcp_buffer = {}
# self.pcap = pcapy.open_offline(filename)
def parse_packet(self, header, frame):
    # 根据链路类型选取decoder，一般为以太网
    datalink = self.pcap.datalink()
    if datalink == pcapy.DLT_EN10MB:
        decoder = ImpactDecoder.EthDecoder()
    elif datalink == pcapy.DLT_LINUX_SLL:
        decoder = ImpactDecoder.LinuxSLLDecoder()
    else:
        raise Exception("Datalink not supported")
    ether = decoder.decode(frame)

    ts = float(str(header.getts()[0]) + "." + str(header.getts()[1]))
    # 从header中获取时间戳
    self.last_timestamp = ts

    if ether.get_ether_type() == ImpactPacket.IP.ethertype:
        # id: src_addr-src_port-dst_addr-dst_port-protocol
        # rev_id: dst_addr-dst_port-src_addr-src_port-protocol
        # tcp_tuple: (src_addr, src_port, dst_addr, dst_port)
        (id, tcp_tuple) = generate_id(ether)
        if id == False:
            return
        (rev_id, tcp_tuple) = generate_reverse_id(ether)

        self.acquire_lock("parse")

        if id in self.tcp_buffer:
            tcp_stream = self.tcp_buffer[id]
            to_server = True
        elif rev_id in self.tcp_buffer:
            tcp_stream = self.tcp_buffer[rev_id]
            to_server = False
        else:
            # a new stream has appeared
            # 根据id与时间戳生成TcpStream对象
            tcp_stream = TcpStream(id, ts, self)
            self.tcp_buffer[id] = tcp_stream
            to_server = True
            packet = ether.child()
            segment = packet.child()
            tcp_stream.start()

        tcp_stream.add_packet(ts, to_server, ether)

        self.packet_counter += 1
        self.release_lock("parse")
```

**TcpStream**

线程启动后，判断tcp流是否结束，若结束则调用move_stream将tcp流移出队列

```
# self.last_packet_time为上文初始化传入的时间戳
def run(self):
    while self.state not in end_states:
        if self.reader_thread.is_timeout(self.last_packet_time) and self.state not
in end_states:
            self.state = STATE_TIMEOUT
        else:
            time.sleep(0.0001)

    self.reader_thread.move_stream(self.id)
```

将就绪的tcp流放入ready队列

```
def move_stream(self, id):
    self.tcp_buffer[id].finish()
    if self.tcp_buffer[id].client_data_len > 0 or
self.tcp_buffer[id].server_data_len > 0:
        self.ready_tcp_buffer.append(self.tcp_buffer[id])
    del(self.tcp_buffer[id])
```

删去数据，用字节频率表示tcp流

```
def finish(self):
    for segment_tuple in self.server_buffer:
        self.server_data += segment_tuple[2]

    del self.server_buffer

    for segment_tuple in self.client_buffer:
        self.client_data += segment_tuple[2]

    del self.client_buffer
    self.ready = True

    self.client_data_len = len(self.client_data)
    self.server_data_len = len(self.server_data)
    self.client_bf = __calculate_byte_frequency__(self.client_data,
self.client_data_len)
    self.server_bf = __calculate_byte_frequency__(self.server_data,
self.server_data_len)
```

将就绪的tcp流从队列中取出，字节频率表示为256维向量，作为模型的输入

```
def byte_freq_generator(filename, protocol, port, batch_size):
    global prt
    global conf
    global done
    prt = StreamReaderThread(get_pcap_file_fullpath(filename), protocol, port)
    prt.start()
    counter = 0
    done = False

    while not done:
        while not prt.done or prt.has_ready_message():
            if not prt.has_ready_message():
                prt.wait_for_data()
                continue
            else:
                buffered_packets = prt.pop_connection()
                if buffered_packets is None:
                    time.sleep(0.0001)
                    continue
                if buffered_packets.get_payload_length("server") > 0:
                    byte_frequency = buffered_packets.get_byte_frequency("server")
                    X = numpy.reshape(byte_frequency, (1, 256))

                    if counter == 0 or counter % batch_size == 1:
                        dataX = X
                    else:
                        dataX = numpy.r_["0,2", dataX, X]

                    counter += 1

                    if counter % batch_size == 0:
                        yield dataX, dataX

        if dataX.shape[0] > 0:
            yield dataX, dataX

        prt.reset_read_status()
```