|  |
| --- |
| for i in range(slot\_num):          #生成group中每个包的随机时间          #如果时间片超过了X.mal[i][0]，nxt\_mal\_no就+1          slot\_time = i \* slot\_itv + last\_end\_time          if slot\_time >= X.mal[nxt\_mal\_no][0]:              nxt\_mal\_no += 1              if nxt\_mal\_no == grp\_size:                  break          #有crafted\_pkt\_prob百分比的mal（100\*2）或者矩阵第二维会变成max\_cft\_pkt（l\_c）==5跳过          if (not decide\_has\_pkt(crafted\_pkt\_prob)) or X.mal[nxt\_mal\_no][1] == max\_cft\_pkt:#不变异或者越界              continue          #这个包本身就会改变，且会新加变异包          #变异的流量处理，初始cft\_no=0，表明变异一次，cft\_no越高越少          cft\_no = int(round(X.mal[nxt\_mal\_no][1]))          if proto\_max\_lmt[nxt\_mal\_no] == 3.:              X.craft[nxt\_mal\_no][cft\_no][1] = random.choice([1., 2., 3.])              #mtu最大传输单元              mtu = 1460          elif proto\_max\_lmt[nxt\_mal\_no] == 2.:              X.craft[nxt\_mal\_no][cft\_no][1] = random.choice([1., 2.])              mtu = 1480          elif proto\_max\_lmt[nxt\_mal\_no] == 1.:              X.craft[nxt\_mal\_no][cft\_no][1] = 1.              mtu = 1500          else:              continue          X.craft[nxt\_mal\_no][cft\_no][0] = X.mal[nxt\_mal\_no][0] - slot\_time#超过时间片的时间          X.craft[nxt\_mal\_no][cft\_no][2] = random.uniform(0, mtu)#最大传输单元          #print(X.craft[nxt\_mal\_no][cft\_no][2])          X.mal[nxt\_mal\_no][1] += 1.#表明+新变异包数 |

从代码上看，新增的包应该没啥用。crafted\_pkt\_prob表示变异率，max\_cft\_pkt表示新增包数量/原始包数量。

也就是说，有crafted\_pkt\_prob概率，且针对每个原始包的新增包数量<max\_cft\_pkt时候，

原始包发生三个参数（时间，payload大小，协议层级）的改变，且默认会产生一个新增包。故每一个原始包最多能产生max\_cft\_pkt个新增包。

再由后面的rebuild函数，对于新增包，它是完全复制原始包的状态，但能够自由修改三个参数。在字节编码中，因为三个参数没有影响，也就是全部复制原始包的特征。

举个例子：

假设将原始包经过字节编码，每一个方框代表一个包：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |

被textCNN识别的结果为：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 |

经过新增包后，假设max\_craft==2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 2 | 3 | 4 | 4 | 4 | 5 |

识别结果为：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

Rebuild

|  |
| --- |
| for i in range(grp\_size):          for j in range(int(round(X.mal[i][1]))):              #对于新增的包，他是通过复制上一个的状态，然后进行改变的，本质上是要发生分裂的包改变了三个参数              pkt = copy.deepcopy(groupList[i])              #print('X.mal[i][1])',X.mal[i][1])              #print(X.craft[i][j][1])              if round(X.craft[i][j][1]) == 1:                  if groupList[i].haslayer(Ether):                      pkt[Ether].remove\_payload()                  else:                      raise RuntimeError("Error in rebuilder!")              elif round(X.craft[i][j][1]) == 2:                  if groupList[i].haslayer(IP):                      pkt[IP].remove\_payload()                  elif groupList[i].haslayer(IPv6):                      pkt[IPv6].remove\_payload()                  elif groupList[i].haslayer(ARP):                      pkt[ARP].remove\_payload()                  else:                      raise RuntimeError("Error in rebuilder!")              elif round(X.craft[i][j][1]) == 3:                  if groupList[i].haslayer(ICMP):                      pkt[ICMP].remove\_payload()                  elif groupList[i].haslayer(TCP):                      pkt[TCP].remove\_payload()                  elif groupList[i].haslayer(UDP):                      pkt[UDP].remove\_payload()                  else:                      raise RuntimeError("Error in rebuilder!")              else:                  raise RuntimeError("Error in rebuilder!")              #print(int(round(X.craft[i][j][2])))              pkt.add\_payload(random\_bytes(int(round(X.craft[i][j][2]))))              pkt.time = X.mal[i][0] - X.craft[i][j][0]              newList.append(pkt)          mal\_pkt = copy.deepcopy(groupList[i])          mal\_pkt.time = X.mal[i][0]          newList.append(mal\_pkt) |

## 实验结果证明

参数设置：

|  |
| --- |
| +----Highlight----+  (iter,swarm,delay,mimic) (1,1,6.0,1000)  ------------------------------------------------------------------------------------------------  Feature extractor: 字节编码  ------------------------------------------------------------------------------------------------  (Manipulator Params)  grp\_size: 100  min\_time\_extend: 3.0  max\_time\_extend: 6.0  max\_cft\_pkt: 2  min\_cft\_pkt\_prob:0  max\_cft\_pkt\_prob:0.02  (PSO Params)  pso\_iter: 1  pso\_num: 1  pso\_size: 1  (Particle Params)  w: 0.7298  c1: 1.49618  c2: 1.49618  ------------------------------------------------------------------------------------------------ |

STA\_feature\_list.shape: (1000, 64, 256) #原始变异后的包

STA\_all\_feature\_list.shape: (1017, 64, 256) #加上新增之后的总

STA\_feature\_list是从STA\_all\_feature\_list上扣下来的

以test=1000条数据为例子

**原始feature:**

Python model.py -tf ../dataPGA/t1000.npy

|  |
| --- |
| test feature shape : (1000, 64, 256)  test time = 0.1660473346710205  TN = 558 TP= 437 FN= 1 FP= 4  F1-score = 0.9943117178612059 Precision = 0.9909297052154195 Recall = 0.997716894977169 FPR= 0.0071174377224199285 |

**攻击后feature：**

Python model.py -tf ../dataPGA/t1000\_STA\_feature.npy

|  |
| --- |
| Warning: under EXECUTE mode!  test feature shape : (1000, 64, 256)  test time = 0.1640157699584961  TN = 558 TP= 437 FN= 1 FP= 4  F1-score = 0.9943117178612059 Precision = 0.9909297052154195 Recall = 0.997716894977169 FPR= 0.0071174377224199285 |

结果一模一样

**对于all\_features,我们看一下PDR**

   print("PDR",(1-(craft\_malicious/origin\_malicious)))

效果更不好了，PDR为负值，根据上文的解释，应该和随机复制包有关系

|  |
| --- |
| test feature shape : (1010, 64, 256)  test time = 0.14901399612426758  origin\_benign : 562  origin\_malicious: 438  craft\_benign : 559  craft\_malicious: 441  PDR -0.006849315068493178 |