**恶意加密流量检测方案 说明文档**

# 方案概述

据报道，截止到2019年10月全球采用HTTPS加密的Web流量的比例已经超过了九成。HTTPS加密传输协议几乎已经接近普及。然而HTTPS在保 护用户隐私的同时，恶意攻击者也开始利用TLS加密来隐蔽恶意命令，从而有效地绕过了传统的检测平台。由于加密流量载荷不可观测、难以解密且流量数据大，我们的方案采用机器学习技术并结合恶意流量相关知识，基于LightGBM模型和所构造的TLS证书内容特征、数据包大小特征实现了较高精度的恶意流量分类。下面我们将介绍所选取的特征，以及选取的动机。

# 2. 特征工程

## 2.1 原始特征分析

|  |  |  |  |
| --- | --- | --- | --- |
| 标准字段字段 | 字段含义 | 样例 | 备注 |
| eventId | 事件ID | d93d6ffd-55ae-4e65-bf28-a90bcf90b824 | 事件唯一id |
| srcAddress | 来源地址 | 172.16.0.108 | 已脱敏 |
| srcPort | 来源端口 | 63415 |  |
| destAddress | 目的地址 | 10.193.90.150 | 已脱敏 |
| destPort | 目的端口 | 443 |  |
| appProtocol | 应用协议 | tls |  |
| tlsSubject | TLS证书主题 | C=US, ST=WA, L=Redmond, O=Microsoft, OU=WSE, CN=settings-win.data.microsoft.com |  |
| tlsIssuerDn | TLS证书持有人唯一标识符 | C=US, ST=Washington, L=Redmond, O=Microsoft Corporation, CN=Microsoft Secure Server CA 2011 |  |
| tlsSni | TLS证书标记 | settings-win.data.microsoft.com |  |
| tlsVersion | TLS版本 | TLS 1.2 |  |
| bytesOut | 发送到客户端字节数 | 9832 |  |
| bytesIn | 发送到服务端字节数 | 2760 |  |
| pktsIn | 发送到客户端数据包数 | 24 |  |
| pktsOut | 发送到服务端数据包数 | 20 |  |
| label | 是否是恶意加密流量 | 1 | 0表示正常流量，1表示恶意流量 |

原始特征中，我们丢弃了eventId、appProtocol 2个字段，eventId是标注流量连接的唯一id，数据集中appProtocol仅有唯一值tls，对模型分类无影响故统一丢弃。

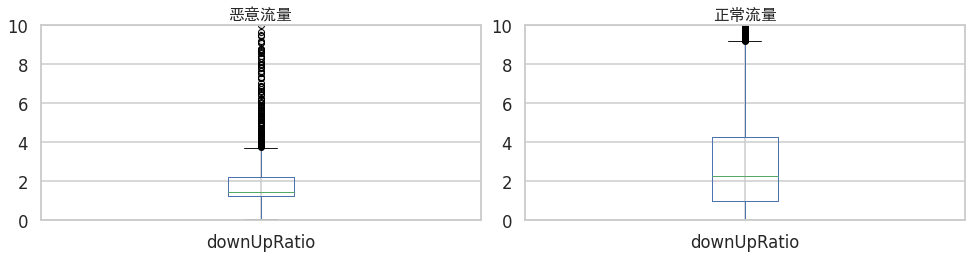
此外，原始特征中主要是包括tls证书内容特征tlsSubject、tlsIssuerDn、tlsSni、tlsVersion，数据包传输地址端口特征:srcAddress、srcPort、destAddress、destPort，数据包传输字节特征:bytesOut、bytesIn、pktsIn、pktsOut。**可进一步细分构造业务特征的字段主要集中在tlsSubject、tlsIssuerDn上，因此我们的特征工程主要聚焦在TLS证书内容上，随后我们会重点对所进行的TLS内容特征工程进行进一步展开介绍。**

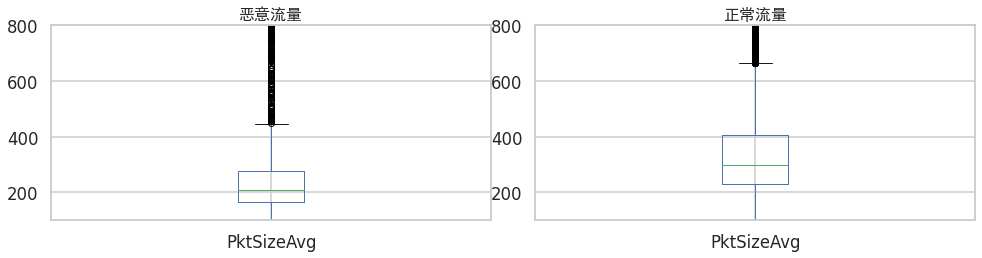
## 2.2 数据包传输特征

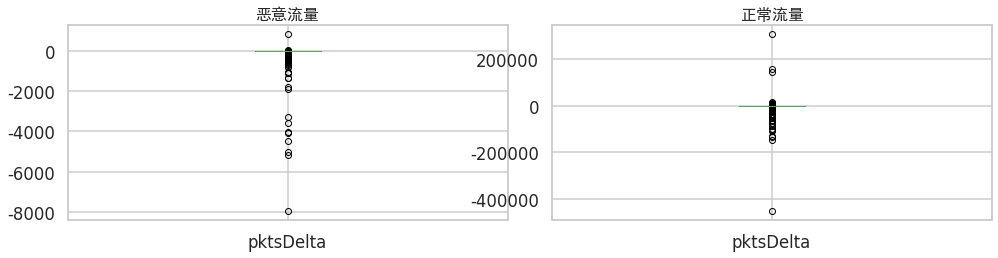
我们在数据包传输字节分布特征方面主要构造的特征见下表:

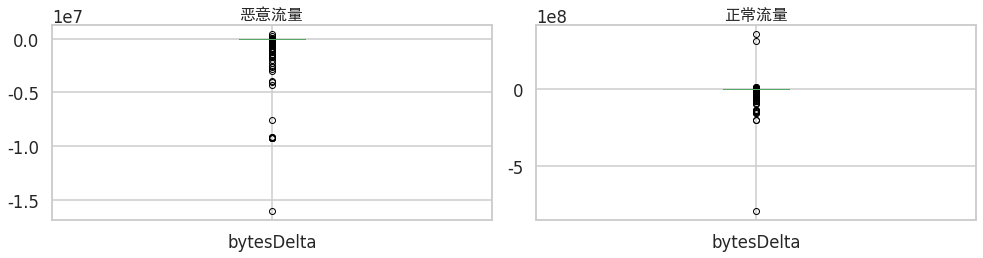
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 标准字段字段 | 字段含义 | (类型) | 构造方法 | 备注 |
| downUpRatio | 字节上下行比 | float64 | data['bytesOut'] / data['bytesIn'] |  |
| PktSizeAvg | 数据包平均大小 | float64 | (data['bytesIn'] + data['bytesOut']) / (data['pktsIn']+data['pktsOut']) | 上下行总传输字节数和数据包数的比值 |
| downPktSizeAvg | 下行数据包平均大小 | float64 | data['bytesOut'] / data['pktsIn'] |  |
| UpPktSizeAvg | 上行数据包平均大小 | float64 | data['bytesIn'] / data['pktsOut'] |  |
| pktsDelta | 上下行数据包个数差值 | int64 | data['pktsIn'] - data['pktsOut'] |  |
| bytesDelta | 上下行数据包字节大小差值 | int64 | data['bytesIn'] - data['bytesOut'] |  |
| AvgSizeDelta | 上下行平均数据包大小差值 | float64 | data['downPktSizeAvg'] - data['upPktSizeAvg'] |  |

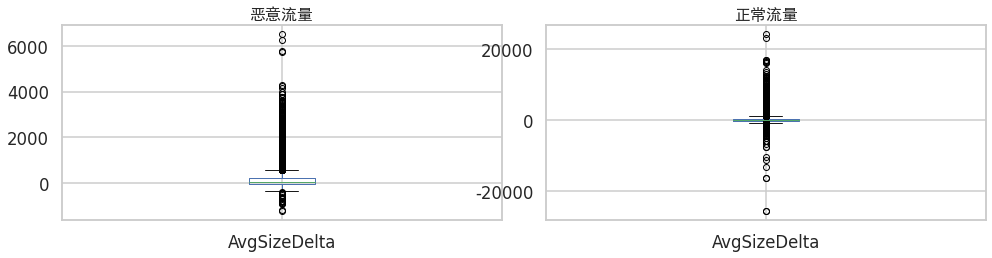
统计性图表数据支撑：











从图表中可以看出，恶意流量、正常流量在所构造的的数据包传输特征上有一定区分度，除根据离群值几乎可以锁定一部分恶意流量样本外，恶意流量在各字段数据的数量级(除pktsDelta外)都有一定差异，pktsDelta表示发送到客户端的数据包个数与发送到服务端的数据包个数之差，正常样本的这一特征远高于恶意样本，具有较大区分度，这也与我们常识相符合。

## 2.3 TLS证书内容特征

1. data['state'] = data['tlsSubject'].str.extract('ST=(.\*?)([,/\s]|$)', expand=True)
2. data['organization'] = data['tlsSubject'].str.extract('O=(.\*?)([,/]|$)', expand=True) #持有组织
3. data['OU'] = data['tlsSubject'].str.extract('OU=(.\*?)([,/]|$)', expand=True)
4. data['CN'] = data['tlsSubject'].str.extract('CN=(.\*?)([,/]|$)', expand=True)
6. data['issueCountry'] = data['tlsIssuerDn'].str.extract('C=(.\*?)([,/\s]|$)', expand=True)
7. data['issueState'] = data['tlsIssuerDn'].str.extract('ST=(.\*?)([,/\s]|$)', expand=True)
8. data['issueOrganization'] = data['tlsIssuerDn'].str.extract('O=(.\*?)([,/]|$)', expand=True) #签发组织
9. data['issueOU'] = data['tlsIssuerDn'].str.extract('OU=(.\*?)([,/]|$)', expand=True)
10. data['issueCN'] = data['tlsIssuerDn'].str.extract('CN=(.\*?)([,/]|$)', expand=True)

### 2.3.1 TLS证书原始特征

TLS证书原始特征指的是直接处理tlsSubject、tlsIssuerDn、tlsSni、tlsVersion字段得到的特征。下面介绍从证书持有者、签发者信息提取到的特征。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 标准字段字段 | 字段含义 | (类型) | 构造方法 | 备注 |
| tlsSni | TLS证书标记 | object | #原始特征 | 通常代表证书签发人域名信息 |
| tlsVersion | TLS版本 | object | #原始特征  类别特征，主要有TLS1.2、TLS1.3、TLSv1、UNDETERMINED等，其他低频值我们统一处理为OTHER |  |
| country | 主体国家 | object | data['country'] = data['tlsSubject'].str.extract('C=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书持有人国家信息 |
| state | 主体省份 | object | data['state'] = data['tlsSubject'].str.extract('ST=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书持有人省份信息 |
| organization | 主体组织 | object | data['organization'] = data['tlsSubject'].str.extract('O=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书持有人组织信息 |
| OU | 主体的其他信息 | object | data['OU'] = data['tlsSubject'].str.extract('OU=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书持有人其他内容信息(OU字段) |
| CN | 主体的公用名称 | object | data['CN'] = data['tlsSubject'].str.extract('CN=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书持有人的公用名称(通常为域名) |
| issueCountry | 证书签发人主体国家 | object | data['issueCountry'] = data['tlsIssuerDn'].str.extract('C=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书签发者国家信息 |
| issueState | 证书签发人主体省份 | object | data['issueState'] = data['tlsIssuerDn'].str.extract('ST=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书签发者省份信息 |
| issueOrganization | 证书签发人主体组织 | object | data['issueOrganization'] = data['tlsIssuerDn'].str.extract('O=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书签发者组织信息 |
| issueOU | 证书签发人主体的其他信息 | object | data['issueOU'] = data['tlsIssuerDn'].str.extract('OU=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书签发者其他内容信息(OU字段) |
| issueCN | 证书签发人主体的公用名称 | object | data['issueCN'] = data['tlsIssuerDn'].str.extract('CN=(.\*?)([,/\s]|$)', expand=True) | 正则匹配证书签发者的公用名称(通常为域名) |

探索数据各字段分布时，我们还发现在state、issueState字段存在大量值不同但代表意义仍然一致的情况，例如这两个字段出现了大量'北京市','北京','beijing','BeiJing','BJ','北京帿','北京眿'值，实际上均表示了省/市位于北京，我们在这里进行了进一步离散化处理，将实际含义一致的特征值归一合并为同一类，例如上例中我们将特征值统一合并为'BeiJing'。

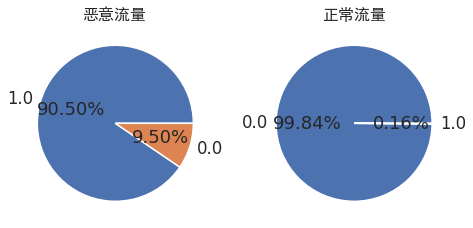
### 2.3.2 根据业务知识构造的TLS证书类别特征

在本节中，我们首先列表给出根据业务知识所构造的所有TLS证书内容类别特征，然后将对其中的一部分特征进行重点介绍。

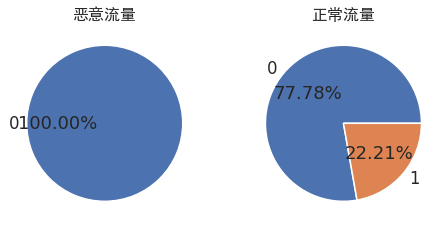
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 标准字段字段 | 字段含义 | (类型) | 构造方法 | 备注 |
| selfCA | 自签名标志 | int | #略，详见代码 | 判断签发人组织和持有人组织或是公用名称是否相同，相同则为1，否则为0，如果这两个字段不存在则为NaN |
| organization\_is\_DefaultProb | 组织名称是否在常见的恶意流量TLS证书组织默认名单内 | int | #略，详见代码 | 判断组织名称是否在常见的恶意组织名单内，是则为1否则为0，如果组织名称字段为空则设为NaN |
| srcPortFlag | 判断srcPort是否小于49152 | int | data["srcPortFlag"] = data["srcPort"]<49152 | 根据《Detecting Malware in TLS Traffic》[1]报告指出大部分恶意流量的源端口处于49152~65535范围内 |

**统计性图表数据支撑：**

organization\_is\_DefaultProb在恶意样本和正常样本内的分布情况



srcPortFlag在恶意样本和正常样本内的分布情况



根据上面2个饼状图的信息，可以反过来验证数据集分布是否与我们构造organization\_is\_DefaultProb和srcPortFlag的动机一致。

可以看到在srcPort端口分布方面，所有的恶意流量都落在了49152~65535范围内，而正常样本中有77.8%落在了49152~65535范围内；在organization\_is\_DefaultProb方面，恶意流量有10%为1，而在正常流量中只有0.16%，因此这也具有一定参考价值。

**进一步处理构造的类别型特征：**

1. tlsSni、CN

由于tlsSni、CN数据均表示域名信息，可以进行进一步处理，例如①<https://www.qq.com>和②<https://news.qq.com>实际上表示的是同一主体机构，我们认为①、②通常具有类似的行为。**因此在我们的处理中，我们提取这两个字段的根域名赋值回tlsSni、CN字段。**将高度离散化的特征进行一次整合，A榜测试结果表明这样的处理可以使分数有较大提升。

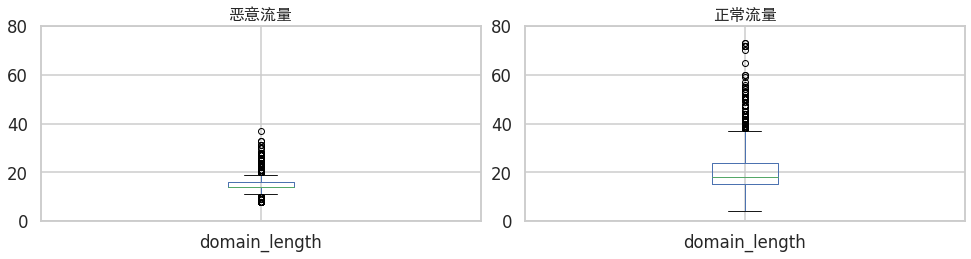
1. tlsVersion、country

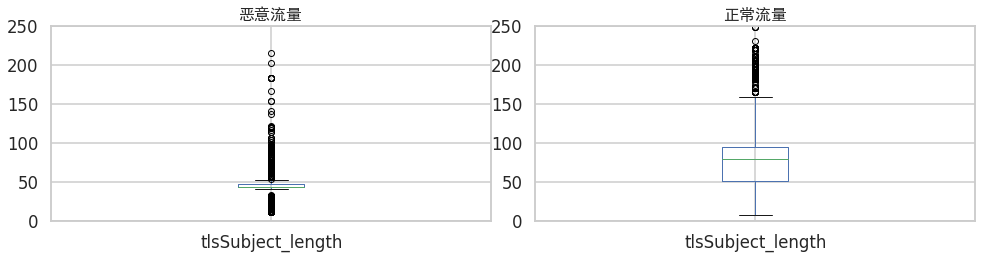
tlsVersion较高频的值包括'TLS 1.2','TLS 1.3','TLSv1','UNDETERMINED'，对于其他类别我们同一标记为“OTHER”；同样的，我们对country字段也有类似处理，'CN','US','XX','AU','GB','--'为较高频的值，对于其他国家我们统一标记为”OTHER”。

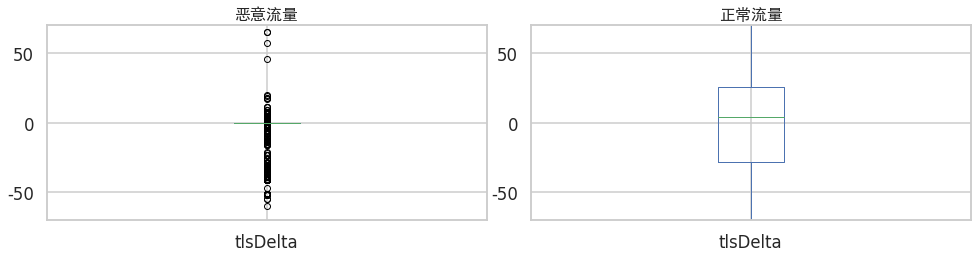
### 2.3.3 根据业务知识构造的TLS证书数值型特征

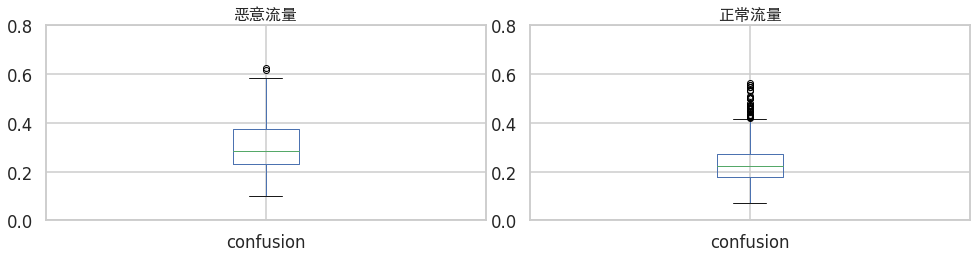
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 标准字段字段 | 字段含义 | (类型) | 构造方法 | 备注 |
| domain\_length | tlsSni证书标记域名长度 | int64 | data['domain\_length'] = data.tlsSni.str.len() |  |
| tlsSubject\_length | 证书持有人内容tlsSubject长度 | int64 | data['tlsSubject\_length'] = data.tlsSubject.str.len() |  |
| tlsIssuer\_length | 证书签发人内容tlsIssuerDn长度 | int64 | data['tlsIssuer\_length'] = data.tlsIssuerDn.str.len() |  |
| tlsDelta | 证书持有人、签发人证书内容长度之差 | int64 | data['tlsDelta'] = data['tlsSubject\_length'] - data['tlsIssuer\_length'] |  |
| CN\_domain\_length | 证书持有人公用名称长度 | int64 | data['CN\_domain\_length'] = data['CN'].str.len() |  |
| confusion | 描述tlsSni字段域名的混乱程度 | int64 | #实现较复杂，详见代码 | 对tlsSni字段进行分词，定义分词得到的单词数量长度-1与域名总长度的比值为混乱程度，我们假设正常流量的域名通常是人类可读的，具有较小的confusion值，例如apple.com分词后有2个单词，总长度为9，confusion约为0.22，而sdafapple.com分词后为5个单词，cofusion为0.56。 |
| confusion\_CN | 描述CN字段域名的混乱程度 | int64 | #实现较复杂，详见代码 | #对CN字段进行处理，过程与上面类似 |
| SubjectItemCount | 描述tlsSubject具有的非空元素个数，元素在这里指CN、O、ST、OU等字段。 | int64 | data['SubjectItemCount'] = data['tlsSubject'].str.count("=") |  |
| IssuerItemCount | 描述tlsIssuerDn具有的非空元素个数，元素在这里指CN、O、ST、OU等字段。 | int64 | data['IssuerItemCount'] = data['tlsIssuerDn'].str.count("=") |  |
| tlsSni\_pointCnt | 对tlsSni字段中出现的小数点个数进行计数 | int64 | data['tlsSni\_pointCnt'] = data[data['tlsSni'].notna()].tlsSni.apply(**lambda** x: x.count('.')) | 这基于我们的一个假设:恶意样本来源域名通常没有子域名，即小数点个数较少。我们通过记录tlsSni中小数点的个数以描述网站的层级。 |
| tlsSni\_pointCnt\_rate | 计算tlsSni字段中出现的小数点个数与域名长度的比值 | float64 | data['tlsSni\_point\_rate'] = data[data['tlsSni'].notna()].tlsSni.apply(**lambda** x: x.count('.') / len(x)) | 动机同上 |
| tlsSni\_alp\_rate | 计算tlsSni字段中出现的字母个数与域名长度的比值 | float64 | data['tlsSni\_alp\_rate'] = data[data['tlsSni'].notna()].tlsSni.apply(**lambda** x: len(re.findall('[a-zA-Z]', x)) / len(x)) | 我们认为正常流量来源域名不包含较多的数字，例如apple.com不存在较多数字，而1255appl5e4.com很可能是一个恶意域名。因此可以用字母占域名长度的比例描述。 |
| tlsSni\_num\_rate | 计算tlsSni字段中出现的数字个数与域名长度的比值 | float64 | data['tlsSni\_num\_rate'] = data[data['tlsSni'].notna()].tlsSni.apply(**lambda** x: len(re.findall('[0-9]', x)) / len(x)) | 动机同上 |
| CN\_pointCnt | 对CN字段中出现的小数点个数进行计数 | int64 | data['CN\_pointCnt'] = data[data['CN'].notna()].CN.apply(**lambda** x: x.count('.')) | 这基于我们的一个假设:恶意样本来源域名通常没有子域名，即小数点个数较少。我们通过记录CN中小数点的个数以描述网站的层级。 |
| CN\_pointCnt\_rate | 计算CN字段中出现的小数点个数与域名长度的比值 | float64 | data['CN\_point\_rate'] = data[data['CN'].notna()].CN.apply(**lambda** x: x.count('.') / len(x)) | 动机同上 |
| CN\_alp\_rate | 计算CN字段中出现的字母个数与域名长度的比值 | float64 | data['CN\_alp\_rate'] = data[data['CN'].notna()].CN.apply(**lambda** x: len(re.findall('[a-zA-Z]', x)) / len(x)) | 我们认为正常流量来源域名不包含较多的数字，例如apple.com不存在较多数字，而1255appl5e4.com很可能是一个恶意域名。因此可以用字母占域名长度的比例描述。 |
| CN\_num\_rate | 计算CN字段中出现的数字个数与域名长度的比值 | float64 | data['CN\_num\_rate'] = data[data['CN'].notna()].CN.apply(**lambda** x: len(re.findall('[0-9]', x)) / len(x)) | 动机同上 |
| tlsSni\_alp\_rate\_tlsSni\_point\_rate | tlsSni\_alp\_rate与tlsSni\_point\_rate相乘得到的值 | float64 | data['tlsSni\_alp\_rate\_tlsSni\_point\_rate'] = data['tlsSni\_alp\_rate'] \* data['tlsSni\_point\_rate'] | 通过特征交互构造的多项式特征，筛选得到的较好特征。 |
| confusion\_tlsSni\_pointCnt | confusion与tlsSni\_pointCnt相乘得到的值 | float64 | data['confusion\_tlsSni\_pointCnt'] = data['confusion'] \* data['tlsSni\_point\_rate'] | 通过特征交互构造的多项式特征，筛选得到的较好特征。 |

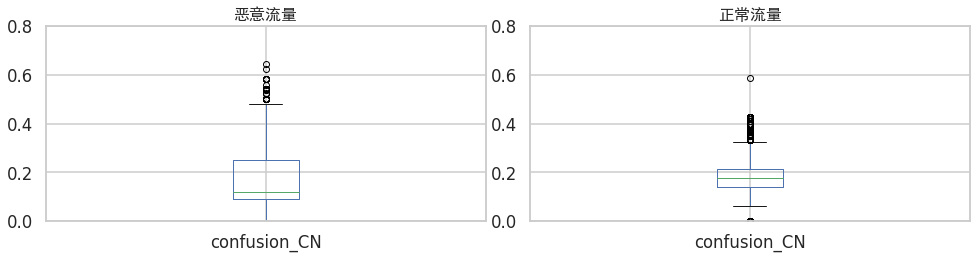
**统计性图表数据支撑(部分)：**

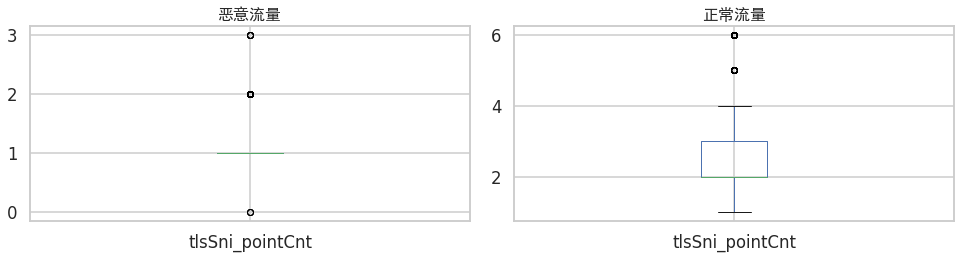












从上面的数据可视化图中可以看出，数据在所构造的特征上的分布与预期假设基本一致，例如tlsSni\_pointCnt字段很明显地出现了恶意流量基本上为1，而正常流量在1~4之间摆动，与我们认为恶意流量域名基本不包含子域名的假设一致；在confusion字段的分布中，也可以看出恶意流量的域名混淆程度略高。

# 模型训练&调优过程

在特征处理方面，对于下面这些高维稀疏纯分类特征，我们使用了TargetEncoder[2]对其进行编码，将其转换为float型变量。

1. cat\_features = ['tlsSni', 'tlsVersion', 'country','state','organization','OU','CN','issueCountry','issueState','issueOrganization', 'issueOU','issueCN','srcAddress','destAddress']

在模型选择方面，我们最终选择了5折交叉验证训练的LightGBM作为分类器模型，并将其预测概率简单平均求和，作为最终的预测概率。

在模型调优方面，我们使用了贝叶斯自动寻优，找到了一组效果较好的参数(主要为num\_leaves、max\_depth、min\_data\_in\_leaf、bagging\_fraction、feature\_fraction、bagging\_freq、reg\_lambda、reg\_alpha、min\_split\_gain)后，我们还多次手动调参，验证了以下措施具有较好效果:

* 改变K折取样方式，由sklearn. model\_selection.KFold改为分层抽样sklearn. model\_selection.StratifiedKFold，提高每折训练过程中数据集分布的一致性
* 改变scale\_pos\_weight参数，这个参数是LightGBM在处理不均衡数据集时的可选参数，这个参数代表着正样本的权重大小，可以有效提高召回率，我们在调优过程中手动从1到10进行二分查找，最终确定为1.15在线下验证时具有较好效果
* 加入基于规则的后处理，我们认为在实际业务部署时来自暗网的流量(域名包含onion|tor2web|4tor|sineadhollywoodnutt|d2web.org|civiclink.network等字段[3])应当被拦截或至少告知用户访问暗网域名存在风险，因此我们在模型预测为正常样本的数据中进行了进一步筛选，如果这些数据的tlsSni字段预示着其来自暗网，程序将其预测结果修改为恶意样本
* 改变随机种子，在我们的配置中，random\_seed=42为一个较好的选择
* 不断调整分类阈值，在0.4~0.55之间调整，最终线上表现最好的版本阈值仍然为0.5

# 总结与展望

在特征构造方面，我们认为可能比较有价值的两个方向分别是：第一，对tlsSubject、tlsIssuerDn、(srcAddress、destAddress)等字段进行embedding；第二，爬虫获取Alexa全球流量排行TOP50000的域名作为外部数据表，判断每一个event的tlsSni域名和CN域名是否在TOP50000内作为新的特征。但在线下实现时发现这两个实现效率比较低，耗时开销较大且精度提升有限，故最终舍弃了这两个方向。

在模型选择方面，除了上面提到的LGB 5折验证模型以外，我们还尝试了模型Stacking、模型blending、特征分组后针对不同组的特征训练不同的分类器再vote的方式，其中Stacking、blending方法线下表现下降明显故没有进一步选择线上验证，特征分组再训练不同分类器的模型预测结果在A榜测试结果有约0.001的提升，但线下波动也较大，考虑到模型的稳定性以及B榜提交次数的限制，最终我们并不打算在初赛的B榜测试中提交这个版本。

总的而言，本次比赛中我们通过对数据集进行广泛探索和特征挖掘，进一步了解、学习了TLS/SSL和HTTPS协议，通过特征工程和模型调优，最终在线上下取得了一个较好的结果。同时，本次比赛锻炼了我们使用机器学习方法去解决实际安全问题的能力，并且接触到了实际业务中的数据，提高了我们的数据挖掘实战能力和对恶意流量领域知识的理解，对我们而言是一次宝贵的学习经历。

# 参考文献

[1] Roques, Olivier. *Detecting Malware in TLS Traffic*. Diss. Imperial College London, 2019.

[2] <http://contrib.scikit-learn.org/category_encoders/targetencoder.html>

[3] <https://www.cnblogs.com/royfans/p/12722792.html>