基于应用层用户密码对的简单流量分类

胡崇新 2021201328

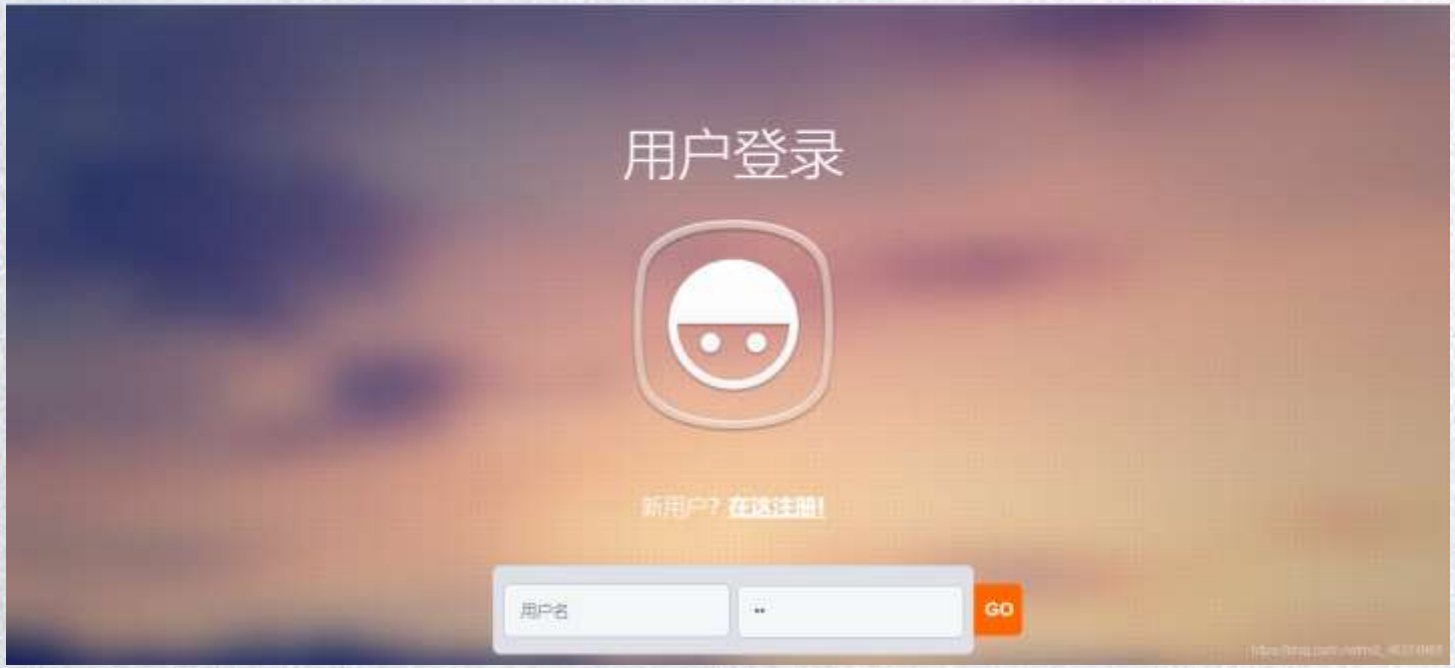
基于应用层用户密码对的简单流量分类

提纲

- 题目背景
- 实验目标
- 环境搭建
- 代码实现
- 实验结果
- 反思展望

题目背景

- 本选题主要研究流量分类问题，web应用的注册和登陆页面容易遭受恶意攻击者的青睐，因为经常存在弱口令、sql注入、xss注入等常见web漏洞，往往是渗透攻击的前哨。
- 正常ip和恶意ip的用户密码对的特征明显不同，可以用来进行ip分类



实验目标

- 1.为了模拟真实环境，使用虚拟机模拟公网上暴露的服务器，启动普通的web应用，使用主机模拟大量正常和恶意用户，向虚拟机发包，使用wireshark保存流量数据。

- 2.主机发包使用python脚本构造，生成的随机ip先由主机随机打上正常和恶意的标签，根据标签来进行不同登陆包的构造。

- 3.从保存的pcapng流量数据中用pyshark库提取ip和对应的所有请求中的用户密码对进行深度学习模型训练和测试，由于第2步中有ip和标签的映射，所以是有监督学习。

环境搭建

- Wireshark 4.2.0
- 启智平台（模型训练和测试）
- Kali linux（受害机）
- Windows（用户）

环境搭建

- 数据集详情
  - 2000个ip，50%概率为恶意，恶意ip中1/3在爆破，1/3在sql注入，1/3在xss注入
  - 共86557次请求，平均每个ip请求43.3次
  - Wireshark截获的pcapng流量数据为71.4MB

代码实现

- send.py(数据集构造)
  - 正常ip
    - 用户密码对大部分相同，少量不同（试错）
    - Peter-Pt@35666, Peter-Pt@35666, Peter-Pt@35666, Peter-Pt@3566，Peter-P@35666,Pter-Pt@35666
  - 恶意ip
    - 爆破
      - 用字典遍历用户名或密码
      - Admin-qwerty, admin-password,ADMIN-123456
    - SQL注入
      - 含有特殊字符和sql关键字
      - Peter- 1’ or ‘1’ = ‘1, Peter-' and if(length(database())>8,sleep(2),0) --+
    - XSS注入
      - 含有html标签
      - Peter-<script>alert(1)</script>, Peter-<img src=1 onerror=alert(1)>

代码实现

- send.py(数据集构造)
  - 正常ip
    - 使用github上收集的普通用户和密码字典
    - 使用mutate_string函数对字符串进行变异

```python
def mutate_string(s, mutation_chance=0.1):
    """
    对输入的字符串进行变异处理。

    参数:
        s (str): 输入的字符串。
        mutation_chance (float): 变异发生的概率，默认为0.1（即10%的概率）。

    返回:
        str: 可能被变异后的字符串。
    """
    if random.random() > mutation_chance:
        return s  # 如果不发生变异，则直接返回原字符串

    # 在字符串中随机选择一个位置
    position = random.randint(0, len(s))

    # 随机选择一种操作：插入、删除或替换
    operation = random.choice(['insert', 'delete', 'replace'])
    mutated_string=""
    if operation == 'insert':
        # 随机选择一个字符插入到随机位置
        char_to_insert = chr(random.randint(32,
                                            126))  # ASCII printable characters
        mutated_string = s[:position] + char_to_insert + s[position:]
    elif operation == 'delete':
        # 删除随机位置的一个字符
        if position < len(s):
            mutated_string = s[:position] + s[position + 1:]
        else:
            mutated_string = s  # 如果位置超出范围，则不进行删除操作
    elif operation == 'replace':
        # 替换随机位置的一个字符
        char_to_replace = chr(random.randint(
            32, 126))  # ASCII printable characters
        mutated_string = s[:position] + char_to_replace + s[position + 1:]

    return mutated_string
```

代码实现

- send.py(数据集构造)
  - 恶意ip
    - 爆破
      - 用github上爆破字典随机选择用户名或密码
    - SQL注入
      - 用github上sql字典随机选择用户名或密码
    - XSS注入
      - 用github上xss字典随机选择用户名或密码

```
    choice = random.choice(['brute', 'sql', 'xss'])
    if choice == 'brute':
        user_cnt = random.randint(5, 10)
        for _ in range(user_cnt):
            username = random.choice(LargeDict["BruteUser"])
            pwd_cnt = random.randint(5, 10)
            for _ in range(pwd_cnt):
                password = random.choice(LargeDict["BrutePwd"])
                # data1 = data.format(username, password)
                # data1 = data
                data1 = data.copy()

                data1["username"] = username
                data1["password"] = password
                response = session.post(url,
                                        headers=headers,
                                        data=data1)
                logging.info(
                    f"Sent brute force request from {ip} - Status: {response.status_code}"
                )
                time.sleep(0.1)
    elif choice == 'sql':
        atk_cnt = random.randint(20, 30)
        for _ in range(atk_cnt):
            usr_sql = random.choice(LargeDict["SQL"])
            pwd_sql = random.choice(LargeDict["SQL"])
            # data1 = data.format(usr_sql, pwd_sql)
            # data1 = data
            data1 = data.copy()

            data1["username"] = usr_sql
            data1["password"] = pwd_sql
            response = session.post(url, headers=headers, data=data1)
            logging.info(
                f"Sent SQL injection request from {ip} - Status: {response.status_code}"
            )
            time.sleep(0.1)
    elif choice == 'xss':
        atk_cnt = random.randint(20, 30)
        for _ in range(atk_cnt):
            usr_xss = random.choice(LargeDict["XSS"])
            pwd_xss = random.choice(LargeDict["XSS"])
            # data1 = data.format(usr_xss, pwd_xss)
            # data1=data
            data1 = data.copy()

            data1["username"] = usr_xss
            data1["password"] = pwd_xss
            response = session.post(url, headers=headers, data=data1)
            logging.info(
                f"Sent XSS request from {ip} - Status: {response.status_code}"
            )
            time.sleep(0.1)
```

代码实现

- 数据预处理
  - Tokenizer的字符级处理（很重要）
  - 转换为序列
  - Padding

```
# Tokenizer的字符级处理
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(all_usernames_passwords)

# 转换为序列
X = tokenizer.texts_to_sequences(all_usernames_passwords)

# Padding
max_length = max([len(seq) for seq in X])
X = pad_sequences(X, maxlen=max_length)

# 标签
y = np.array(labels)
```

代码实现

- 构建深度学习模型
  - Sequential
  - Embedding
  - LSTM(128 个隐藏单元)
  - Dropout(用于正则化，防止模型过拟合)
  - Dense(全连接层,进行二分类)

```
# 4. 构建深度学习模型
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=64, input_length=X_train.shape[1]))
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Binary classification (normal or malicious)
```

代码实现

- 模型训练、测试和预测
  - 损失函数：binary_crossentropy
  - 优化器：Adam
  - 评估指标：accuracy
  - Epoch:10
  - batch_size:32
  - EarlyStopping:
    - 提前停止训练，防止过拟合

```
# 5. 训练模型
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)
# model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test), callbacks=[early_stopping])

# 6. 评估模型
score = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {score[1]:.4f}")


texts_test = all_usernames_passwords[len(X_train):]  # 获取测试集对应的原始文本

random_index = np.random.choice(len(X_test))  # 随机选择一个测试集的索引
sample_data = X_test[random_index]
true_label = y_test[random_index]
original_text = texts_test[random_index]  # 获取该样本的原始文本数据

# 使用模型进行预测
prediction = model.predict(np.array([sample_data]))  # 预测单条数据

# 获取预测的概率值
prediction_prob = prediction[0][0]

# 输出真实标签、预测标签、原始文本和预测概率
print(f"原始文本: {original_text}")
print(f"真实标签: {'恶意' if true_label == 1 else '正常'}")
print(f"预测结果: {'恶意' if prediction_prob >= 0.5 else '正常'}")
print(f"预测概率: {prediction_prob:.4f}")
```

实验结果

- 测试集上最终准确率
  - 99.7%

- 预测单条结果正确

```
数据已保存到 'usernames_passwords_labels.csv' 文件中。
Epoch 1/10
50/50 [==============================] - 10s 172ms/step - loss: 0.4570 - accuracy: 0.8206 - val_loss: 0.1523 - val_accuracy: 0.9625
Epoch 2/10
50/50 [==============================] - 7s 133ms/step - loss: 0.1949 - accuracy: 0.9469 - val_loss: 0.2804 - val_accuracy: 0.8800
Epoch 3/10
50/50 [==============================] - 6s 129ms/step - loss: 0.1637 - accuracy: 0.9688 - val_loss: 0.0548 - val_accuracy: 0.9900
Epoch 4/10
50/50 [==============================] - 6s 129ms/step - loss: 0.1452 - accuracy: 0.9600 - val_loss: 0.0508 - val_accuracy: 0.9950
Epoch 5/10
50/50 [==============================] - 6s 129ms/step - loss: 0.0989 - accuracy: 0.9781 - val_loss: 0.0322 - val_accuracy: 0.9975
Epoch 6/10
50/50 [==============================] - 6s 126ms/step - loss: 0.0829 - accuracy: 0.9837 - val_loss: 0.0355 - val_accuracy: 0.9950
Epoch 7/10
50/50 [==============================] - 6s 127ms/step - loss: 0.0709 - accuracy: 0.9856 - val_loss: 0.0202 - val_accuracy: 0.9975
Epoch 8/10
50/50 [==============================] - 6s 126ms/step - loss: 0.0661 - accuracy: 0.9844 - val_loss: 0.0818 - val_accuracy: 0.9825
13/13 [==============================] - 1s 51ms/step - loss: 0.0322 - accuracy: 0.9975
Test Accuracy: 0.9975
1/1 [==============================] - 0s 337ms/step
原始文本：xia e-mail169 xia e-mail169 xia e-mail=169 xia e-mail169 xia e-mail169 xia e-mail169 xia e-mail169 xia e-mail169 xia e-mail169 xia
真实标签：正常
预测结果：正常
```

反思展望

- 实验中要注意对输入数据进行字符级处理，才能捕捉语义信息。

- 实验中最难的一步可能是数据集构造，由于网络上无此类公开数据集，我构造的数据集结构化特征较明显，导致最终分类准确率过高，将来希望找到更贴近现实的数据集。

谢谢！