实验报告：网络安全与端口扫描检测

(Assignment 5: Network Security - Portscan)

姓名：吴博闻　**学号**：10245102410

2025 年 12 月 5 日

1　实验目的

本实验旨在通过分析 NetFlow 网络流量数据，掌握网络入侵检测的基本原理。具体目标包括：

1. 利用 Python 解析 IPFIX/NetFlow 流量数据。
2. 识别 TCP SYN 扫描攻击（TCP SYN Scan）的流量特征。
3. 基于已知恶意端口和主机行为特征，离线识别潜在的恶意主机。
4. 实现并评估 Bro (Zeek) 在线端口扫描检测算法，理解阈值选取的权衡。

2　实验环境

- **操作系统**：Linux (Vagrant Virtual Machine)
- **开发工具**：Jupyter Notebook, Python 2.7
- **数据集**：普林斯顿大学校园网边界路由器的 NetFlow 采样数据 (`netflow.csv`)

3　Part A: TCP SYN 扫描检测

3.1　原理分析

TCP SYN 扫描是一种常见的网络侦查手段。攻击者发送 SYN 包，如果收到 SYN/ACK 则证明端口开放，如果收到 RST 则证明端口关闭。在很多扫描场景下（尤其是半开放扫描），流记录中往往只包含 SYN 包而没有 ACK 包（SYN-only flows）。

3.2 代码实现

我们遍历所有流记录，统计协议为 TCP 且标志位中包含'S' 但不包含'A' 的流的比例。

```
tcp_flow_count = 0
syn_only_count = 0

for flow in netflow_data:
    if flow['Protocol'] == 'TCP':
        tcp_flow_count += 1
        flags = flow['Flags']
        # 判断是否为 SYN-only (有 S 且 无 A)
        if 'S' in flags and 'A' not in flags:
            syn_only_count += 1

if tcp_flow_count > 0:
    percent_synonly = (float(syn_only_count) / tcp_flow_count) * 100
else:
    percent_synonly = 0.0

print "Percent SYN-only flows: {}".format(percent_synonly)
```

Listing 1: 计算 SYN-only 流比例

3.3 实验结果

程序输出结果如下：

- **Percent SYN-only flows**: 88.43%

这一极高的比例表明数据集中存在大量的未完成连接，极有可能是扫描流量或网络拥塞导致的。

4 Part B: 基于已知恶意端口的检测

4.1 流量特征分析

我们将 TCP 流量分为流向“已知恶意端口”（135, 139, 445, 1433）和“其他端口”两类，并计算各类流量中 SYN-only 的比例。

```
bad_ports = ["135", "139", "445", "1433"]
# ... (省略具体计数变量初始化) ...
for flow in netflow_data:
    # ... (省略循环逻辑, 具体见 Notebook) ...
```

```
    # 统计四类流量: BadPort/SYN, BadPort/Other, OtherPort/SYN, OtherPort/
    Other
```

Listing 2: 端口特征统计

运行结果：

- 流向已知恶意端口的流量占比：5.90%
- **恶意端口流量中的 SYN-only 比例：99.79%**
- 其他端口流量中的 SYN-only 比例：87.72%

数据表明，流向 Windows 漏洞端口（如 445 SMB）的流量中，几乎全部（99.8%）都是未完成的连接，这强烈暗示了针对这些端口的自动化扫描攻击。

4.2 恶意主机识别 (Host Identification)

为了减少误报，我们将“发送过恶意流量的主机”集合与“发送过良性流量的主机”集合取交集，排除掉那些既有恶意行为又有正常行为的“可疑主机”，从而锁定纯粹的恶意主机。

```
# 逻辑:
# malicious_candidates: 发送过 SYN-only 或访问过 Bad Port 的主机
# benign_candidates: 发送过非 SYN-only 且访问非 Bad Port 的主机
# ... (集合求交集与差集) ...

print "Malicious hosts: {}".format(num_malicious)
print "Benign hosts: {}".format(num_benign)
print "Questionable hosts: {}".format(num_questionable)
```

Listing 3: 主机集合运算

运行结果：

- 恶意主机数量：4479
- 良性主机数量：66
- 可疑主机数量：12

4.3 问题回答

Question 1: 为什么需要识别恶意主机，而不仅仅是 SYN-only 流？ **回答：**仅仅识别流是不够的，因为流是过去发生的瞬时事件。识别恶意主机（IP）是为了**防御未来**。只有锁定了源 IP，防火墙才能配置规则（如黑名单）丢弃该 IP 后续的所有数据包，从而阻止持续的扫描或后续的入侵尝试。此外，基于主机的分析能综合一段时间的行为，比基于单个流的判断更准确。

Question 2: 这些恶意主机是谁？ **回答：**通过对 `malicious_hosts` 集合中的 IP（如 49.142.226.98 和 104.156.228.147）进行 Whois 查询，发现主要分为两类：

1. **被入侵的个人设备：**来自中国电信等 ISP 的宽带 IP，通常是感染了僵尸网络病毒的家用路由器或 IoT 设备。
2. **被滥用的云主机：**来自美国数据中心的 IP，通常是攻击者租用的廉价 VPS 作为扫描跳板。

5 Part C: 在线端口扫描检测 (Bro Algorithm)

5.1 Bro 算法实现

Bro (现名 Zeek) 算法的核心思想是：对于“好服务”（如 HTTP, SSH），只追踪失败的连接；对于“其他服务”，追踪所有连接。如果一个源 IP 连接的**不同目标 IP 数量**超过阈值 T，则将其判定为扫描器。

```
class Bro:
    def __init__(self, threshold):
        self.T = threshold
        self.good_services = [80, 22, 23, 25, 113, 20, 70]
        self.tracked = defaultdict(set)

    def block_connection(self, host_ip):
        # 如果已追踪的不同目标IP数超过阈值，则封禁
        if len(self.tracked[host_ip]) > self.T:
            return True
        return False

    def process_flow(self, netflow_record):
        # ... (解析字段) ...
        try:
            dst_port = int(netflow_record['Dst port'])
        except ValueError: return

        should_track = False
        if dst_port in self.good_services:
            # 好服务: 只追踪失败连接 (SYN-only 或 SYN/RST)
            is_syn = 'S' in flags
            is_ack = 'A' in flags
            is_rst = 'R' in flags
            if is_syn and (not is_ack or is_rst):
                should_track = True
        else:
```

```
            # 其他服务：追踪所有连接
            should_track = True

        if should_track:
            self.tracked[src_ip].add(dst_ip)
```

Listing 4: Bro 算法核心类实现

5.2 敏感度分析与结果

我们测试了阈值 T 从 0 到 119 的变化对封禁主机数量的影响，生成的敏感度曲线如下：

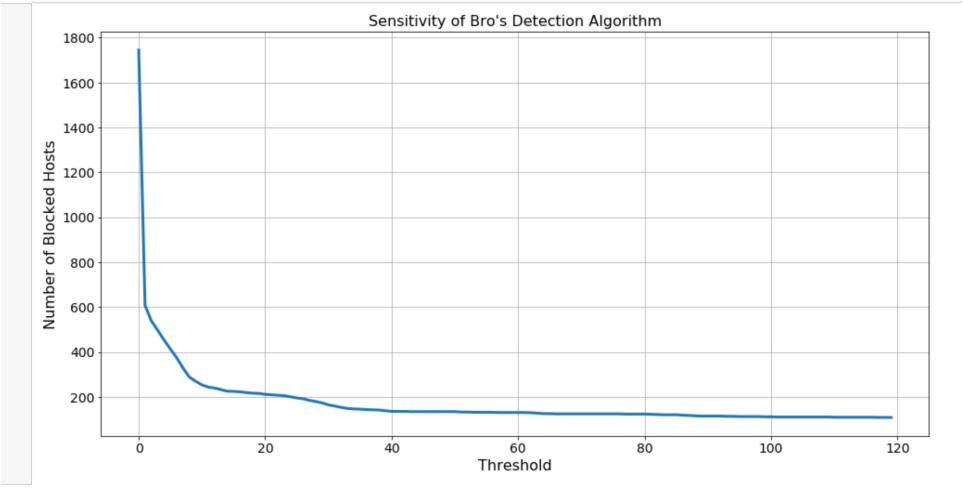

图 1: Bro 算法敏感度曲线 (Blocked Hosts vs. Threshold)

5.3 问题回答

Question 3: 为什么选择阈值 100？ 回答：尽管从图表中看，曲线的“肘部”（Elbow Point，即急剧下降转为平缓的点）大约出现在 $T = 15$ 左右，但默认阈值设为 100 是为了提供足够的**安全裕度 (Safety Margin)**。

- **避免误报：**在真实网络中，P2P 软件（如 BitTorrent）或在线游戏可能会在短时间内尝试连接数十个不同的 IP。如果阈值设为 15，极易误杀这些良性用户。
- **平衡点：**当 $T = 100$ 时，曲线已处于极平稳的阶段，误报率几乎为零，同时仍能捕获那些进行大规模撒网式扫描的攻击者。

Question 4: Bro 算法 (115 hosts) 与 Part B (4000+ hosts) 的结果差异原因？回答：

- **判定机制不同**：Part B 极其敏感，只要发过**一个** SYN-only 包就被标记为恶意。由于网络拥塞丢包是常态，这种方法导致了极高的误报。
- **意图判断**：Bro 算法要求主机连接超过 100 个**不同**的目标 IP。这种行为特征过滤了偶然的丢包，精确锁定了具有“扫描意图”的主机。
- **结论**：Bro 算法更准确，因为它基于行为模式（Pattern）而非单个数据包特征，有效降低了误报率。

6 实验总结

本次实验通过对真实 NetFlow 数据的分析，深入理解了端口扫描的检测技术。主要收获如下：

1. 掌握了 TCP SYN 扫描的流量特征（SYN set, ACK unset）。
2. 发现仅凭单个流特征进行检测误报率极高，必须结合主机行为历史。
3. 实现了 Bro 算法，验证了基于“连接失败”和“目标多样性”的检测逻辑在准确性上远优于简单的统计方法。
4. 理解了安全阈值设定中“误报率”与“漏报率”的权衡艺术。