实验5：网络安全 - 端口扫描

启动Jupyter Notebook服务

在虚拟机中运行以下命令：`sudo jupyter notebook &`。这将在后台启动一个新的 Jupyter 笔记本服务器。即使它在后台运行，它有时也会在终端上打印有关性能的信息。您可以在每次收到消息时按 Enter 键以获得 shell 提示符。要关闭笔记本，请运行 `fg`，然后按两次 Control-C（一次获取确认消息，另一次跳过确认）。

在jupyter运行时，在主机机器上打开您的浏览器，然后在地址栏中键入 `localhost:8888`。这将进入 Jupyter 笔记本文件选择窗口。Jupyter Notebook 实际上是在 Vagrant VM 上的端口 8888 上运行的，但您可以通过主机浏览器访问它，因为端口在 VM 和主机之间进行了转发。

在文件选择窗口中，进入 `assignment5` 目录，然后打开 `Assignment5_Notebook.ipynb`。这将打开一个包含作业其余部分说明的笔记本。按照笔记本的顺序从上到下完成标有 "TODO" 的部分。

记得在离开笔记本或关闭选项卡之前进行 "保存并检查点"（在 "文件" 菜单中）。

实验

网络运营商积极监视其网络，以防范各种入侵攻击。网络攻击者经常对IP地址进行随机“端口扫描”，以找到易受攻击的机器。网络入侵检测系统（NIDS）试图检测这种行为，并将这些进行端口扫描的主机标记为恶意。

在这个作业中，您将分析NetFlow网络测量数据，以识别发送到校园网络的潜在恶意流量。然后，您将模拟用于识别恶意远程主机的在线算法。

这个笔记本有几个部分。每个部分都包含标有 TODO 的部分，您需要完成这些部分。

解析IPFIX数据

`netflow.csv` 文件包含来自普林斯顿校园网络边界路由器的预处理的NetFlow数据。该数据是 "未抽样" 的，即它为通过边界路由器上的任何接口传输的每个数据包编制了流统计信息。我们使用了 `nfdump` 工具来处理路由器收集的原始NetFlow数据。`netflow.csv` 文件的每一行（除了顶部的标题行）记录了一个流的以下信息：

```
Date first seen, Time first seen (m:s), Date last seen, Time last seen (m:s),
Duration (s), Protocol,
Src IP addr, Src port, Dst IP addr, Dst port, Packets, Bytes, Flags, Input
interface, Output interface
```

为了分析这些数据，我们首先需要将其读入Python数据结构。以下代码使用内置的 `csv` 库将 `netflow.csv` 读入一个字典列表中。如果您感兴趣，可以查看 `csv` 库的文档：https://docs.python.org/2/library/csv.html

```
import csv

with open('netflow.csv', 'r') as netflow_file:
    netflow_reader = csv.DictReader(netflow_file)
    netflow_data = list(netflow_reader)

print "Number of flow records: {}".format(len(netflow_data))
print
print "Sample flow entry: {}".format(netflow_data[0])
```

Part A: TCP SYN扫描检测

在这部分作业中，您将分析TCP流记录以查找TCP SYN扫描攻击。TCP SYN扫描涉及向许多不同的目标发送TCP SYN数据包，以发现网络上具有开放端口的易受攻击的计算机。

背景

在先前的作业中，我们通过流的数量和整体流量分析了热门目标。这类更高级别的统计信息提供了对网络异常的粗略视图，但是这些统计显示的关于攻击和入侵尝试的有用信息较少。

在大型企业网络中，恶意主机的异常活动通常很难在大量合法流量中检测到。恶意主机可能不会出现在最高发送者的列表中（无论是按字节还是按流量），而且在事先确定攻击流量的确切性质可能会很困难。

然而，某些类型的攻击具有特征性特征，简化了它们的检测。例如，许多攻击者通过使用TCP SYN数据包扫描地址空间来检测运行易受攻击服务的主机。如果NetFlow记录表明一个流包含了TCP SYN数据包*但没有设置ACK标志的数据包*，我们可以得出结论该流从未完成，因此可能是TCP SYN扫描的一部分——特别是如果有很多这样的流。

当攻击者发送TCP SYN数据包作为TCP SYN扫描的一部分时，有三种可能的结果：

- 目标IP地址上有一个处于活动状态的主机，该主机正在监听TCP SYN数据包的目标端口。在这种情况下，NetFlow记录将在流记录中同时包含TCP SYN和ACK标志。这些流难以与合法流量区分开，因为在这种情况下，主机实际上回答了最初的SYN。
- 目的IP地址所在网络中没有活动主机。在这种情况下，我们将只看到TCP SYN标志。这与合法流量行为非常不同。应该能够通过识别这些流量来识别恶意扫描程序。
- 目标是活动的，但发送SYN的端口已关闭。如果客户端连接到服务器的非监听端口，服务器将发送一个RST/ACK数据包。根据正常的TCP实现指南，一旦接收到RST，扫描程序将立即停止任何TCP连接尝试。与前一种情况一样，NetFlow记录将显示来自扫描程序主机到目标主机的SYN请求，但没有响应的SYN/ACK数据包。

分析

让我们首先探索NetFlow跟踪，以确定普林斯顿大学校园网络是否存在具有TCP SYN标志但没有ACK标志的流量（即，具有TCP SYN但没有ACK的流）。

在下面的单元格中，确定`netflow_data`中具有SYN数据包但没有ACK数据包的TCP流的百分比。如果在 "Flags" 字段中有"S"，则流具有SYN数据包。如果在flags字段中有"A"，则流具有ACK数据包。确保仅考虑TCP流量（使用 "Protocol" 字段）。

打印您在单元格末尾找到的百分比，并附上信息标签（例如`print "Percent SYN-only flows: {}".format(percent_synonly)`）。

```
# TODO: determine the percentage of TCP flows in netflow_data that had a SYN
packet but no ACK packet
```

通过将答案传递给下面的`check_synonly_percentage()`函数来检查您的答案（作为介于0和100之间的浮点数）。这会将您的答案的md5哈希值（四舍五入到最接近的整数）与正确答案的md5哈希值进行比较。

```
from testing import check_synonly_percent

# TODO: pass the percentage of SYN-only TCP flows to the following function
check_synonly_percent(TODO)
```

属于这一类别的每个流都是TCP SYN扫描的候选流。

Part B: 使用已知的恶意端口进行端口扫描检测

另一种识别攻击的方法涉及使用TCP端口，这些端口与特别容易受到攻击的服务相对应。对于这项任务，我们将考虑端口135、139、445和1433作为已知的恶意端口。这些端口对应于Windows RPC、NetBIOS、SMB和SQL-Snake攻击。我们期望大多数到"已知错误"端口的流量都是扫描器，但这并不一定适用于所有到这些端口的流量。

在下面的单元格中，通过识别NetFlow数据中流向网络上已知恶意端口的流，而不考虑IP地址，来测试这一假设。然后，看看这些流是否更有可能是TCP SYN扫描，而不是流向其他端口的流：

1. 将`netflow_data`中的所有TCP流分为以下几类：
    1. 仅SYN流向已知恶意端口
    2. 其他流向已知恶意端口
    3. 仅SYN流向其他端口
    4. 其他流向其他端口
2. 从这些集合中，计算以下三个百分比，将它们存储在变量中，并打印出带有信息标签的百分比：
    1. 在所有TCP流中，流向已知坏端口的流的百分比
    2. 在所有TCP流中，仅SYN流向已知恶意端口的百分比
    3. 在所有TCP流中，仅SYN流向其他端口的百分比

```
# TODO write code as described above
bad_ports = ["135", "139", "445", "1433"]
```

通过将您在上一个单元格中找到的3个百分比传递给下面的相应`check`函数来检查它们。

```
from testing import check_percent_knownbad, check_percent_synonly_knownbad,
check_percent_synonly_other

# TODO: pass percent values to following check functions
check_percent_knownbad(TODO)
check_percent_synonly_knownbad(TODO)
check_percent_synonly_other(TODO)
```

除了识别可能是恶意流的情况外，网络操作员还希望识别可能是恶意主机的情况。然后，操作员可以阻止来自这些主机的未来流量。

问题 1

在实际流量中停止端口扫描，为什么需要识别恶意主机，而仅仅识别SYN-only流不足够呢？

回答 1

TODO: your answer here.

然而，网络运营商希望避免阻塞良性流量。阻塞良性流量的潜在性取决于发送看似恶意流量和发送正常流量的主机之间的重叠。

完成以下代码以找到此重叠：

1. 找到在记录的NetFlow数据中向学校发送了SYN-only或已知的恶意端口流的所有主机（IP地址）的集合。我们将这些称为“恶意主机”。
2. 找到将非SYN-only流发送到其他（非已知的恶意）端口的所有主机的集合。我们将这些称为“良性主机”。
3. 找到“恶意主机”与“良性主机”的交集。我们将这个交集称为“可疑主机”。
4. 从“恶意主机”和“良性主机”集中移除这些“可疑主机”。
5. 计算并打印带有信息性标签的3个值：
    1. 剩余的恶意主机数量
    2. 剩余的良性主机数量
    3. 可疑主机的数量

只包括源IP地址在普林斯顿网络外部的流量（不包括128.112.*.*）。我们提供了一个`internalIP()`函数，如果参数IP地址来自普林斯顿网络，则返回True。

Python中的`set`类型将非常有用，文档在此处：https://docs.python.org/2/library/stdtypes.html#set-types-set-frozenset

```
# InternalIP returns true if the argument IP address is within the Princeton
network
def internalIP(ip):
    s = ip.split('.')
    if s[0] == "128" and s[1] == "112":
        return True
    return False

# TODO: write code as described above
```

将您找到的数据传递给以下的`check`函数以进行验证。

```
from testing import check_num_malicious_hosts, check_num_benign_hosts,
check_num_questionable_hosts

# TODO: pass percent values to following check functions
check_num_benign_hosts(TODO)
check_num_malicious_hosts(TODO)
check_num_questionable_hosts(TODO)
```

问题 2

所有这些可能是恶意主机的是谁？从“恶意主机”集合中选择2个IP地址，并使用 `whois` 和在线搜索查找有关它们的信息。在下面报告您发现的最有趣的部分。

回答 2

TODO: your answer here.

Part C: 在线端口扫描检测

在之前的部分，我们使用离线分析预先记录的流量来识别潜在的恶意主机。真实的网络操作员不希望等待离线分析以检测潜在的威胁。相反，他或她将配置一个网络入侵检测系统（NIDS），使用在线端口扫描检测算法对主机进行分类，实时更新这些分类以响应流量流的发生。

Bro的端口扫描检测算法

Bro系统中使用的端口扫描检测算法于1999年由Vern Paxson等人发表。

Bro的检测算法建立在同样的直觉上，即连接尝试失败（仅SYN或SYN/RST流）是识别扫描的良好指标。

该算法根据连接所要连接的服务（由端口确定）对连接尝试进行不同处理。Bro仅跟踪对一组特定的“已知良好”服务的连接尝试失败（仅SYN或SYN/RST）。

对于所有其他服务，Bro跟踪所有连接尝试，无论尝试是成功还是失败。

对于每个发送主机，Bro计算与跟踪的连接尝试相对应的不同目标地址的数量。如果该数量超过某个阈值 `T`，Bro将主机标记为扫描器。

默认情况下，只考虑失败的服务集，包括HTTP（80）、SSH（22）、Telnet（23）、SMTP（25）、IDENT（113）、FTP（20）和Gopher（70）。这些服务是“已知良好”的，即我们预期会对这些服务发起许多连接，因此我们不应将成功的连接尝试计入发送主机。

任务: 实现Bro的端口扫描检测算法

完成下面的Bro类，以实现Bro的端口扫描检测算法。

```
from collections import defaultdict

# TODO: complete the methods marked TODO in the Bro class to implement the Bro
algorithm
class Bro:
```

```
    # class constructor saves the parameter threshold value and sets instance
variables
    def __init__(self, threshold):
        self.T = threshold

        # self.good_services is the list of port numbers to which successful
connections
        #     (SYN and ACK) should not be tracked
        self.good_services = [80, 22, 23, 25, 113, 20, 70]

        # self.tracked maps sending hosts to a set of destination addresses
        #     from tracked connection attempts
        self.tracked = defaultdict(set)


    # TODO: block_connection() takes a sending host IP address of an incoming
connection
    #       returns True if the connection should be blocked (because the host
has surpassed
    #       the threshold of destination addresses in tracked connection
attempts) or False otherwise
    def block_connection(self, host_ip):
        pass


    # TODO: process_flow() takes a netflow record of a TCP flow that was not
blocked
    #       (as a netflow record dict) and should update self.tracked
    #       with the contents of the record
    def process_flow(self, netflow_record):
        pass
```

请注意，Bro检测算法的完整实现还将包含一个移动时间窗口，用于记住跟踪的连接尝试。这可以防止良性主机被阻止，如果它被动态分配一个以前属于攻击者的IP地址。然而，我们在此任务中使用的NetFlow数据仅包含约5分钟的流量，因此我们不需要担心时间窗口。

下面的`run_bro()`函数接受一个阈值，并在NetFlow数据上运行您的Bro算法，返回在算法过程中被阻止的所有地址的集合。此函数已经完成。

```
def run_bro(threshold):
    blocked_hosts = set()
    bro = Bro(threshold)
    for flow in netflow_data:
        src_ip = flow["Src IP addr"]
        if flow["Protocol"] != "TCP":
            continue
        if internalIP(src_ip):
            continue
        block = bro.block_connection(src_ip)
        if block:
            blocked_hosts.add(src_ip)
            continue
        bro.process_flow(flow)
    return blocked_hosts
```

在在其网络入侵检测系统上运行Bro之前，网络操作员可能希望了解算法对阈值变量`T`的敏感性。

在下面的单元格中，创建一个包含120个元素的列表，使得第`i`个元素包含由`run_bro(i)`返回的被阻止主机的数量。然后将列表传递给`plot_bro()`函数以绘制敏感性曲线。

```
%matplotlib inline
from plotting import plot_bro

# TODO: generate the list described above


# TODO: pass the list to the plot_bro function
plot_bro(TODO)
```

问题 3

Bro算法的默认阈值是100。根据上述图的结果，你认为为什么选择了这个值？选择更高或更低的阈值的优缺点是什么？

回答 3

TODO: your answer here

问题 4

一个阈值为100的Bro算法将阻止大约115个主机（如果你的实现没有，请检查是否有错误）。这比在第B部分找到的恶意主机数量要少得多。Bro算法和第B部分中使用的方法之间的哪些差异导致了这种差异？你认为哪种方法更可能是准确的？为什么？

回答 4

TODO: your answer here.

除了Bro之外，还有许多其他在线端口扫描检测算法。例如，Threshold Random Walk（TRW）算法使用最大似然比来调整阈值参数，以尽可能阻止恶意流量，同时不将良性主机列入黑名单。

如果你感兴趣，TRW算法在这篇论文中有详细描述：http://www.icir.org/vern/papers/portscan-oak04.pdf。