Machine Learning to Identify Running Android Applications in Real Time

Samantha Comeau, David Larson

Abstract

The goal of this project was to use a machine learning model to identify running applications on an Android VM in real time. The project consisted of 4 phases, (1) setting up a development environment, (2) sniffing packets of five different applications, (3) create a machine learning model on the collected data, and (4) classifying packet data in real time.

System Design

Phase 1 - System Setup

This phase was used to set up the Android and gateway VMs. For the Android x86 VM an ISO file was imported into Oracle VirtualBox and set up to only be connected to an internal network. A gateway VM running TinyCore Linux was also set up on the same internal network. The Android VM communicates through the gateway VM to gain an IP address, resolve DNS, and connect to the internet in any way.

To setup the internet through the gateway we set them both up on an internal network which we called *IntNet*. We set up the gateway VM to also connect to the NAT router so it could connect to the internet through our host machine. Using *UDHCDP* we configured the DNS server, DHCP server, and the router on the gateway VM.

```
start 192.168.12.100
end 192.168.12.200
interface eth1
option subnet 255.255.255.0
option router 192.168.12.1
option lease 43200
option dns 8.8.8.8
option domain local
```

Code Snippet 1. /etc/udhcpd.conf

We had to use filetool in order to backup files to a persistent storage location otherwise we would lose files on reboot from the gateway VM. The commands we had run on reboot are

shown in Code Snippet 2. SSH and a git repository were also set up to easily develop both collaboratively and efficiently, since we didn't install a GUI for TinyCore.

```
/user/local/etc/init.d/openssh start
/opt/eth1.sh
sysctl -w net.ipv4.ip_forward=1
iptables -t nat -A POSTROUTING -o etho0 -j MASQUERADE
sudo pip install pyshark==0.3.8
sudo pip install scikit-learn
```

Code Snippet 2. /opt/bootlocal.sh

In setting up these VMs, we came across a few reliability problems. If left idle for more than about 5 minutes, the Android VM would become unresponsive and need to be restarted. We experienced lagging issues when running it on machines that did not have at least 16GB of RAM. Another issue we experienced was needing to increase the TinyCore memory significantly from the original 1024 MB in order to collect the testing packets.

Phase 2 - Packet Capture and Definitions

The goal of this phase was to sniff packets and segregate them into bursts and flows from files and live, while the Android VM is running. In order to collect packets for file testing we used TCP dump using the following command, this filters based on eth1 so we don't get background noise from the gateway VM while collecting packets from the Android VM. The information printed included source IP, source port, destination IP, destination port, protocol, number of packets sent, and number of bytes sent.

```
sudo tcpdump -vv -x -i eth1 -w
sample.pcap
```

Code Snippet 3. TCPDUMP to collect PCAPs.

We chose to use pyshark's file capture and live capture methods to collect pcap data. It was easy to interface with these methods because they return a collection of packet objects to us. To generate flows and bursts from the PCAP, we separate packets based on whether the last packet occurred within a second of the previous packet or not. If yes, a new burst object is created to store the packets. Each flow is defined as the packets within a burst with the same source port / ip to destination port / ip.

Initially, we used tcpdump to generate a .pcap file containing the network flows over a brief period of time. We then wrote our code to parse that file and print statistics correctly. Once that was working, we then converted it to be able to receive packet flows from the network in real time and print statistics as they were received. Packet flows were divided into bursts, which are defined as groups of network packets separated by 1 second of network silence. Then, all of the

packets in each burst were divided into flows based on the source and destination IP and ports. Then we simply printed the statistics of each flow.

Phase 3 - Machine Learning Classifier

The goal of this phase was to train a machine learning model with around 75% accuracy on the following applications which run on the Android VM.

Fruit Ninja
Google News
Weather Channel
Wikipedia
Youtube

Table 1. Applications

In order to accurately train our model, we collected 50 traces for each application. To do this we ensure that no applications are running on the Android VM. Then we start the TCPDUMP command given in Code Snippet 3, and open the application we want to trace. For each, we started tcpdump, opened the app, simulated use (making various searches, playing the game, browsing through pages, etc.) in order to generate network traffic. We then stop the TCPDUMP command. This gave us in total 250 traces to train our model on.

For our feature vectors on the flows, we chose to use number of packets sent, number of bytes sent, and the protocol used. Ideally, we would have added more features to our model, however due to time constraints and difficulties with the development environment, we stuck with these features. To create our feature vectors we added to the script created from part 1 to have it write the features of the packets to a CSV file. This CSV took about 5-10 minutes to generate given the 250 PCAP files we had as traces. After we generated our CSV file, it was used to train a Decision Tree model. We found that the Decision Tree took the least amount of time to train, and also had the highest accuracy. We used scikit-learn for machine learning purposes.

Our model had around 75-90% accuracy when training with our own testing PCAP files, which was done one at a time because it was easier to label them this way. We found that oftentimes Fruit Ninja and Weather Channel were confused in our model. Another limitation of our program is that during demonstrations we found that it doesn’t work on certain PCAP files. We think this is a limitation of pyshark but we are unsure what is causing pyshark to fail to return packet objects from certain PCAP files.

There are several ways in which our model could have been improved. For one thing, we could have cleaned the data we collected better. The operating system surely generates some kind of traffic in the background unrelated to the running application. These packets were captured in our data files and were used to train the classifier in spite of the fact that those packets actually weren't from the application. This probably caused a slightly lower accuracy for our actual model.

Phase 4 - Live Classification

The goal of this phase was to round out all the previous phases to classify traffic in real time. Using pyshark's live capture and the CSV file of our training data, we live classify the traffic. During this phase we got the same accuracy on average as previously. A limitation of this phase is that we do not have an accurate way to find the accuracy of our model, so we had to eyeball the results of the classification. Pyshark generates results at a slightly delayed pace, which also limits our ability to accurately state how precise our model is.

Conclusion

This project involved a wide variety of skills and concepts in which we gained knowledge we did not previously have. First, we understood how to implement a gateway VM which allowed us to man-in-the-middle all traffic for our research purposes. We also learned how to capture packet information automatically and parse that packet information in python. Finally, we learned that machine learning is a useful tool when it comes to network traffic analysis and that even when we have the constraints of not accessing any raw packet data, we can still easily classify the source of that data. Hence, even strong end to end encryption cannot prevent a man-in-the-middle from understanding the network traffic that is being generated from different applications.

If we were to do the project again, we would likely spend more time iteratively improving our machine learning model. Unfortunately we didn't have as much time as we would have liked to have to do so, and doing so could have improved our accuracy.