Traffic Analysis of Spotify

Kameron Dawson, Antony Hernandez, Tomisin Ogunfunmi, Tejal Reddy

May 11, 2022

Abstract

As encryption techniques become increasingly sophisticated and standardized, one might think that it would be impossible for an adversary to obtain information about user actions from an encrypted stream. However, traffic analysis attacks — attacks that rely purely on observing the number and timing of packets sent along with metadata — have proven to be a strong adversarial tool for predicting user actions for audio and video streaming services. In this paper, we investigate the efficacy of traffic analysis attacks on the Spotify Web Player to predict the song a user is streaming. By training a variety of neural networks on the non-encrypted payload size and timing of packets being sent between the client and Spotify server, we are able to predict the correct song that a user is playing from a song-set size of three with an accuracy of up to seventy-five percent. Taken together, our results compare the performances of different neural networks to provide a heuristic of what classification models worked the best for performing a traffic analysis attacks on music streaming services. This paper suggests that there is potential promise in the realm of user action prediction for audio streaming services.

1 Introduction

Traffic analysis has been a long researched topic. From this research, considerable improvements have been made to improve the general security of packets. Today, many packets being sent from devices are heavily encrypted to prevent adversaries from gathering information and compromising the security of the packet sender and receiver. However, even with this encryption, adversaries are still able to learn relevant information just from the timing of the packets [5]. When packets are sent, they are often sent in a stream or sequence which is the case for many video streaming services, websites, and requests. Using this, adversaries do not have to decrypt the packet content to obtain vital and identifying information about the users at the endpoints of communication.

Some of this identifying information includes knowing when an individual is in their home, unmasking relationships, identifying preferences, and general monitoring [6]. By observing when packets are being sent and where they are being sent to, an adversary can find all of this information by simply observing packets through a network. As more and more individuals conduct their work in areas with unprotected networks, this has become an even larger problem.

In this paper, we describe an experiment where we simulate an adversary analyzing packets on a network. Specifically, we will be analyzing packets sent through the encrypted channel created when a user accesses Spotify's services [9]. In this experiment, we collected data from a set of songs and inputted this data into a variety of machine learning models to help identify the patterns in various songs. Using these, we determined which model resulted in the highest accuracy in identifying when a user is listening to a specific song based on packets sent through an encrypted stream. Before starting the experiment, we hypothesize that we will be able to successfully identify which song a user is listening to based purely on the packet's timing, payload size, and metadata.

2 Related Works

Our work draws the most on the Beauty and the Burst: Remote Identification of Encrypted Video Streams paper [8]. This paper analyzes video streaming from Youtube, Netflix, Amazon, and Vimeo

and presents accurate packet burst analysis to accurately identify the video being streamed. They measured the bursty, on-off patterns of encrypted streams and identified a strong correlation between segment requests from clients and the size of underlying segments. For example, during the buffering period of a video, segment-files are found to be fetched at a higher rate (to buffer the actual presentation), which shows a potential way to predict patterns of burst. They exploit the prevalence of these burst patterns, also known as the "fingerprint" of the video stream, to identify the videos streamed. To help with identification, they created a machine learning model using a convolutional neural network to classify video streaming packets.

3 Permissions

This experiment was performed using personal, paid Spotify accounts. Additionally, all computers used for this project were the personal property of the authors, thus all the data sent to these devices through legal practices is analyzable. As it is common knowledge that packet data across a network is available and readable by all computers, our project does not attempt to access restricted data from Spotify servers. Rather, we simply look to read and find patterns within the data that Spotify sends to our computers.

4 Assumptions

There are several assumptions we are making as we conduct this experiment.

(1) We are assuming that we are receiving every packet being transmitted between our personal computers and Spotify. In the code for our model, we are collecting IP addresses to sniff packets from using the Network tab in Chrome DevTools. Although we thoroughly searched this tab for all IP addresses relating to Spotify, there is still a chance we may have missed a few.

(2) Since we are replicating an eavesdropping adversary in this experiment, we are assuming that the adversary will be observing the packets of an individual in a "closed-world" setting. Being in a "closed-world" means that we can assume the adversary knows the small list of songs being streamed from Spotify for the experiment.

(3) We are assuming that the songs being streamed from Spotify are deterministic. This means that the packets being sent from Spotify when streaming the song two different times are similar. No defense mechanisms (like random noise) have been used to treat the packet bursts.

5 Goals

Our goal for this study was to prove that performing a traffic analysis on encrypted Spotify packets would allow for an adversary to break a CPA security-like problem and have a strong chance of predicting the song that a user is listening to by observing the packet data. The main difference is that instead of attempting to break Spotify's song encryption, our goal is to predict a song based off of an analysis of the packet traffic that corresponds to a song with a higher than random accuracy. That is, the adversary wants to win the game in figure 1.

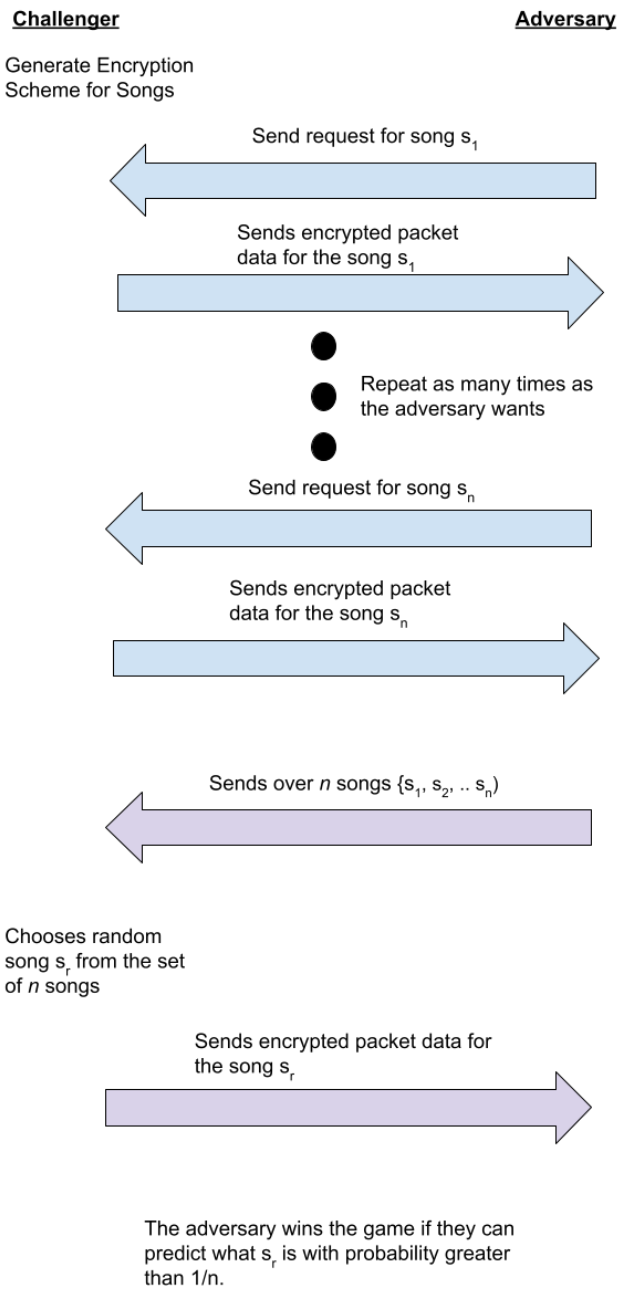

Figure 1: Spotify CPA Game

6 Security Policy

The security policy of Spotify is detailed below. Our goal is to compromise part (a) of the confidentiality goal.

(1) **Confidentiality:**

  (a) Only the Spotify user can stream a specific song and only they will know exactly what song they are streaming.

  (b) The user is the only entity that can add or remove songs from their music library.

  (c) When a user begins streaming, an encrypted channel is created between Spotify and the user's streaming device.

(2) **Integrity**

  (a) The song being streamed should be correct, and its packets should not be tampered with.

  (b) The state of the user's music library should be correct and reliable.

  (c) No adversary is able to insert packets into the encrypted stream.

(3) **Availability**

  (a) The user can stream songs at any time.

  (b) The user can add or remove songs from their library at any time.

7 Experimental Design

7.1 Overview

There are many publicly available pieces of information included with a packet that we could use in a side-channel attack of Spotify, but we decided to look specifically at packet payload size and timing because this is what has been done for similar attacks on video streaming services (see related work). Packet payload size and timing allows us to identify and learn patterns based on the compression algorithm that Spotify uses. The process for collecting and analyzing this data is described below.

We decided to use the Spotfy web player on the browser, rather than the desktop or mobile apps because it allowed us to more easily see what IP addresses the HTTPS requests were being sent to and integrate with the other tools we used in our experiment (more details below).

The first step in the packet sniffing, a technique whereby packet data flowing across a network is detected and observed, was to find the IP address from which packets were sent from the Spotify web player in order to setup an IP address capture filter to only capture packets from the web player. At first we ran a traceroute command on the Spotify Web Player Hostname. However, we noticed that we hardly received any data in our captures when we should have received much more. After closer examination of the web player, we realized Spotify uses Akamai's CDNs, which means that the web player uses the IP address of their closest edge server, so the IP address could be different for each streaming session. Thus, we looked at the network tab on the browser's developer tools to find the actual IP addresses the HTTPS requests were being sent to and built a list of possible IP addresses.

Another issue we encountered is that we realized that Spotify heavily relies upon the browser cache, so in order to capture all new data we have to clear the browser cache first; otherwise we will be streaming cached data rather than receiving new data over the network. We were now able to capture all new packets by filtering by all the possible IP addresses.

In our first version of the data analysis of packet captures we manually looked at the data to identify characteristics and patterns that allow us to infer information about the Spotify user. We did so by exporting Wireshark captures to an Excel spreadsheet and graphing the result. This was useful for identifying easy pieces of information, including:

- Whether a song is currently being streamed, which is easily identifiable because significantly more data is received than when not streaming a song.
- If a user is a free or premium user, which is easily identifiable using the fact that free users stream at a lower bitrate than premium users (128 kbit/s versus 256 kbit/s on the web player), so the amount of data received for a premium song stream is double that of a free song stream.
- The selected audio quality as more data is received for higher audio quality.

However, we wanted to be able to identify hard pieces of information, including what song is being played, if an advertisement or a song is being played, what advertisement is being played, and other user actions (e.g. creating playlists, looking at someone's profile, etc). Thus we decided to build and train machine learning models to better identify patterns in the data. To limit the scope of this project due to the short time frame we had to work, we decided to only try to identify what song is being played based on the packet captures, rather than looking at many different identifiable pieces of information.

7.2 Tools

Detailed below are the tools we used to collect our data and build and train our machine learning models.

7.2.1 Wireshark

Wireshark is a network analyzer tool that allows a user to observe what packets are being sent and received on their network [11]. The parts of a packet are the body (typically encrypted) and the header. The header includes the source IP address, destination IP address, time to live, network layers and protocols, length of the body, and more. Additionally, included with a packet captured through Wireshark is the time it was received. An example session capture can be seen in figure 2. Wireshark helped us visualize and understand the different parts of packets and what information is included with a capture, as well as testing out different capture filters to isolate packets sent from Spotify.

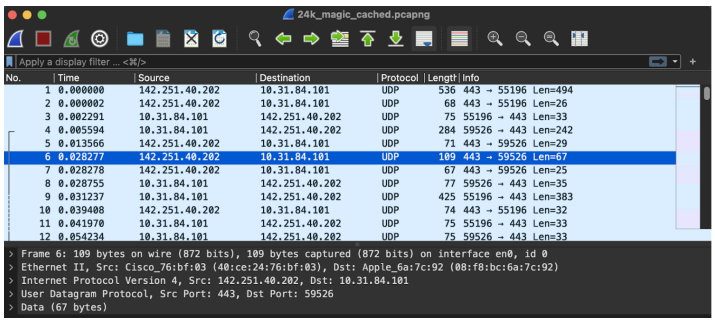

Figure 2: A sample live packet capture session for a song on Spotify.

7.2.2 Spotipy

Spotipy is a Python library that allows the use of the Spotify Web API [3] in Python scripts. We used Spotipy in our Python code to automate the streaming of songs from Spotify, rather than manually streaming songs using the Spotify web player in the browser, which is more time-consuming, less precise for data captures, and not feasible for building a large dataset.

7.2.3 Scapy

Scapy is a Python library built on top of Wireshark that allows the capture of packets from within Python scripts [4]. We used Scapy in our Python code to automate the process of sniffing for packets received from Spotify.

7.2.4 Selenium

Selenium is a Python library that allows for the automation of web browser interaction from Python scripts [2]. We used Selenium to automate the process of opening the browser, clearing the cache, and beginning playback of a song on the Spotify web player.

7.2.5 PyTorch

PyTorch is a machine learning framework for Python that offers many tools and resources to build and train machine learning models [1]. We used PyTorch to train our models to classify a song capture.

7.2.6 Seaborn

Seaborn is a python visualization library for making statistical graphics [10]. We used this library to plot the heatmap in Figure 7.

7.3 TSAI

TSAI (Time Series AI) is a Python library built on top of PyTorch that includes the implementations for many state-of-the-art machine learning models for time series classification, regression, forecasting, and more [7]. We used TSAI to test different models for song classification.

7.4 Data Generation

In order to develop a proof of concept before spending significant resources and work-hours on developing a model, and due to the constraint of our limited compute power, we decided to only include three songs in our classification. Additionally, our limited compute power and the variable lengths of songs led us to only capture data for the first 60 seconds of a song.

The first step to creating a machine learning model with high accuracy is to create a high quality data set with many data points using these constraints. Fortunately, the nature of our data allows for the automatic generation of data as the data consists of packets from streaming a song from Spotify. The process for data generation is as follows:

(1) Use the Selenium web driver library to open Google Chrome and clear the browser cache.

(2) Use Selenium to navigate to the Spotify web player in browser and click on the play button to start a playback.

(3) For each of the three songs, use the Spotipy library to begin playback of the song on the web player.

(4) Use the Scapy library to begin packet sniffing for 60 seconds, filtering for all of the Spotify web player IPs.

(5) Store the data from each song's packet sniffing as a list of lists, where each list element is one song capture and contains tuples for each packet with the tuple elements being the arrival time and payload size of the packet.

(6) Once one capture is collected for each song, open the dataset pickle file if it exists and append the new data to the existing data; otherwise create a new pickle file to store the new data.

This generates and stores one capture for each song in our classification, which will each be one data point for our model, so we repeat the process as often as possible to generate many data points. The limiting factors in this generation process are that generating one song capture takes 60 seconds and that we can only stream one song per Spotify account, so using one Spotify account we can generate 60 data points per hour. Additionally, periodically the Spotify web player uses IP addresses that we previously did not include in our filter, so we have to manually add it to our list as currently we could not find a way to automatically find the IP address the web player was using.

7.5 Data Pre-Processing

Collecting many high-quality data points is an important factor affecting a machine learning model's performance, but just as important, if not more so, is the pre-processing of the data that encodes the data to be inputted to the model.

In our first iteration of the data pre-processing, each data point inputted to the model was just a list of packet payload sizes for one song capture. This led to poor performance with an accuracy of just 33.00% on the test data set, which is not any better than if given a song capture we had just guessed the song. The reason this feature encoding is a poor choice is because it only takes into consideration the relative order of packet arrivals, but does not encode that the packets arrive at different times. This is a crucial piece of information to leave out because the time difference from one packet to the next could be a few microseconds or 10 seconds. Therefore, we had to revise our view of the data and find a better feature encoding.

We realized that a song capture is basically a time series with packet payload size mapped over arrival time. Therefore, each data point is constructed to have one feature (packet payload size), which is a list where each list entry represents the total payload size received at that time step. For example, for a 60 second capture with a time step of 0.1 seconds there are 600 list entries.

The arrival time of each packet is in the format of a Unix timestamp, which is the number of seconds that have elapsed since January 1, 1970, so we made the relative arrival time for the first packet received for a song capture be zero seconds and for all subsequent packets be the packet arrival time minus the arrival time for the first packet received. We count a packet payload size towards the time step in which its relative arrival time is closest to. We construct the feature for a data point by looping through the total number of time steps for the 60 second capture and if there were packets received at that time step we add the total packet payload size to the data point's list; otherwise it adds zero for that time step.

Given the input constraints of the models we are using, the final formatted input data is a three-dimensional numpy array, where the first dimension is the number of data points, the second dimension is the number of features for a data point, and the third dimension is the number of time steps for a feature. In our case our dimensions were (111, 1, # time steps) because we generated 111 data points (37 for each song), had one feature (packet payload size), and the time steps were one of the hyperparameters that we experimented with during training. The final pre-processing done was to just remove any duplicate or empty data points from the data set that may have accidentally been added during the data generation since we were generating data on multiple machines so we had to employ version control to combine our data sets.

7.6 Machine Learning Models

Since the focus of this paper is not machine learning, but rather the security analysis of a side channel attack on Spotify using machine learning as a data analysis technique, we decided to not develop our own models. Instead we used the TSAI library because it contains state-of-the-art time series classification models implemented in Python.

We experimented using different models from the library with different model hyper-parameters, as well as varying the number of time steps in our data pre-processing to see what combination produced the model with the highest accuracy on the test data set. Our test data set was constructed from randomly selecting 20% of the data points from our overall data set, with the rest being used for the train data set. We used 25 epochs to reduce the training time.

8 Results

The purpose of the experiment was to classify a song capture (collection of packets received) of the first 60 seconds of a song as one of three possible songs. We used 9 different time series classification models from the TSAI library to complete this task. The model accuracy results with different time steps used

in data pre-processing are shown in the tables below. Note that tables 3 and 4 use fewer number of data points, this is due to our computing power and it will be further explain in the discussion section.

Model	Number of Data Points	Accuracy
ResCNN	111	50.00%
ResNet	111	40.91%
OmniScaleCNN	111	40.91%
FCN	111	36.36%
LSTM_FCN	111	36.36%
XceptionTime	111	36.36%
InceptionTime	111	36.36%
mWDN	111	36.36%
LSTM	111	36.36%

Table 1. Time step of 1 second.

Model	Number of Data Points	Accuracy
ResNet	111	54.55%
FCN	111	45.45%
LSTM_FCN	111	40.91%
OmniScaleCNN	111	40.91%
XceptionTime	111	36.36%
InceptionTime	111	36.36%
ResCNN	111	36.36%
mWDN	111	36.36%
LSTM	111	36.36%

Table 2. Time step of 0.1 seconds.

Model	Number of Data Points	Accuracy
XceptionTime	24	75.00%
FCN	111	50.00%
LSTM_FCN	111	45.45%
InceptionTime	36	42.86%
ResNet	60	41.67%
ResCNN	21	41.67%
mWDN	18	33.33%
LSTM	111	31.82%
OmniScaleCNN	None	None

Table 3. Time step of 0.01 seconds.

Model	Number of Data Points	Accuracy
LSTM	15	33.00%
FCN	None	None
LSTM_FCN	None	None
ResNet	None	None
ResCNN	None	None
InceptionTime	None	None
XceptionTime	None	None
OmniScaleCNN	None	None
mWDN	None	None

Table 4. Time step of 0.001 seconds.

The model column is the name of the model run, the number of data points is the amount of data used in training the model, and the accuracy is how well the model performed on a test data set. The accuracy was determined by first training the model on the training data set, then inputting data points from the test data set to the trained model, outputting a classification, and checking if the

classification was correct. For some model-time step combinations we use less data points, and in some cases are not able to train a model at all, because of our limited compute power. This is explained in greater detail in the discussion section.

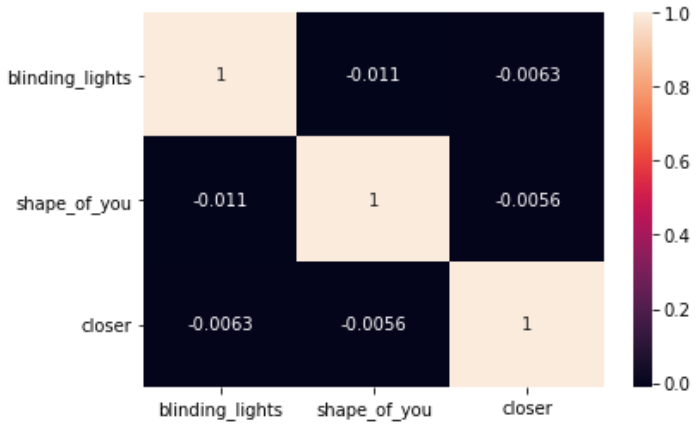

Figure 3: Cross correlation heatmap obtained for captured data.

In addition to the machine learning models, we calculated the cross correlation between the three different songs that we used. We did this by first gathering all of the capture data into single lists for each song, and then used the heatmap function from the Seaborn library to create the heatmap in Figure 3. The heatmap showed us that the packet captures between the different songs have a very low correlation and that packet captures for the same song have a very high correlation. This is promising since this potentially means that the machine learning models can be fine-tuned and trained with more data to produce even higher accuracy results. This is because the low correlations among different songs found in this heatmap could signify that the models can perform well in classifying different songs, and the high correlation within a song could signify that the models can perform well in classifying captures from the same song as the same song.

9 Discussion

9.1 Results

The results of our experiment show that the accuracy of the models generally increase as the time step gets more precise. This is because as the number of decimal places of the time step increases we are able to encode more information about the packet arrival patterns. This is a significant factor on the model performance because, for example, Spotify sends about 325kB of data for premium users every 10 seconds, which means that there is a large cluster of packets within a one second time frame every 10 seconds. If we clump packets together by the second or even the tenth of a second, then all the captures just look the same, so more precision is needed to differentiate the captures.

The maximum available precision of the arrival times is 0.000001 seconds, so we did not encode the maximum amount of precision possible in our model. This is due to compute power constraints. We ran the models on a personal computer with a GPU with 4GB of RAM, and with every increase in precision we have a ten-fold increase in time steps for each data point. In Tables 3 and 4 we can see that we had to reduce the number of data points used in order to actually be able to complete the training of the model given the memory constraints. In some cases, like the OmniScaleCNN model in Table 3 and all the models except for LSTM in Table 4, we were not able to train the model at all, regardless of the number of data points.

For some models (e.g. ResNet from a time step of 0.1 to 0.001 seconds) the accuracy decreased even though the time step precision increased, which was caused by the decrease in the number of

data points used due to memory constraints, as described in the previous paragraph. Some models had a small decrease in accuracy even though the number of data points used stayed the same and the precision of the time step increased (e.g. LSTM from a time step of 0.1 to 0.001 seconds), which may be attributable to the small data set used, so there are fluctuations in accuracy from one training to the next. However, in these cases the decreases were small, so they do not detract away from the success of this attack. The model LSTM did not significantly increase in accuracy from a time step of 0.01 to 0.001 seconds, but this may be because of the small number of data points used.

The benchmark we used to determine if the accuracy of a model was good is whether or not it was better than just guessing. Given a song capture and knowing it could be one of three possible songs, then by guessing we will have on average an accuracy of 33.00%. Applying this benchmark, we can see that the LSTM model with a time step of 0.01 seconds performed worse than guessing, but only by 1.18%. The LSTM model with a time step of 0.001 seconds and the mWDN model with a time step of 0.01 seconds performed the same as guessing. The other 24 model-time step combinations performed better than guessing.

In summary, of the 27 model-time step combinations we ran, 1 performed worse than guessing, 2 performed the same as guessing, and 24 performed better than guessing. The best performing model was XceptionTime with a time step of 0.01 seconds at 75.00% accuracy on the test data set, but it was on a smaller data set, so more testing would be needed to determine if this accuracy is repeatable. However, with 88% of the models performing better than guessing and the general increase in accuracy as the precision of the time step increases, the results show that this network analysis attack on Spotify is possible and can be successful.

9.2 Defense Mechanisms

There are many possible defenses against this type of attack, but we will only discuss the two most popular and easiest to implement. The first strategy is to pad packets to be of uniform sizes or to send packets at fixed timing intervals to blur the traffic features. This approach works by making all song captures look the same, so a machine learning model would not be able to distinguish between the capture from one song and another song. The second strategy is to randomly send dummy packets to obscure any patterns with noise. This approach works by making the captures for the same song look different, so a machine learning model would not be able to learn to classify all captures for one song as that song.

We tested the effectiveness of these strategies by artificially constructing fake song captures. For the first strategy, we took 10 data points from our data set and padded every packet to be the same size, then inputted it into our most successful model XceptionTime. The result was that it only correctly classified one of the 10 data points for an accuracy of 10.00%, which was significantly worse than guessing and worse than the model accuracy of 75.00%. For the second strategy, we also took 10 data points and randomly inserted packets of random size into the capture, then inputted it into XceptionTime. This had an accuracy of 20.00%, which was also worse than guessing and the model accuracy.

The question is if these strategies are successful, then why does Spotify not employ them. The reason may be because of the performance hit of these defense mechanisms. Looking at one song capture, the largest packet size was 22469 bytes and the smallest was 80 bytes, so the first strategy of padding all packets to be the same size would introduce a significant performance hit. One solution to this problem could be to revise the compression algorithm and way that Spotify sends data to better partition the data, making all the packet sizes uniform without having to pad. The second strategy also has a performance cost since it would require sending more packets with data that is not actually being used, and there are strategies to filter out the noise to determine which packets are actually useful data, so may be a less effective strategy in the face of advanced adversaries.

In summary, both defense mechanisms are successful in preventing our attack, but the most viable strategy for Spotify would most likely be to revise how they partition their data for transmission so all packets are of the same size without needing to pad, which would avoid the performance cost. It could be the case that Spotify may determine that this attack does not pose a significant security risk as

the ability to know what song a user is listening to is not a leak of sensitive information, so a defense mechanism is not needed.

9.3 Future Work

Due to our limited computing power and time, we limited our pool of songs to three, capture length to 60 seconds, features to one (payload size), data points to 111, and time step precision to at most 0.001. Future work will focus on getting more compute power (most likely from the cloud) to train models with more songs, full song lengths for captures, more features than just payload size, more data points, and greater time step precision. The first focus would be on time step precision and number of data points as the results showed these to be the most promising ways to increase model accuracy.

In addition to these changes, we will spend more time fine-tuning the model hyper-parameters in order to achieve higher model accuracy. Once we have built and trained a reliable and accurate model for song classification, the next steps would be to analyze Spotify packets on the open Internet to test how well this model might perform if we are not able to capture every single packet. In this experiment we were able to capture almost all of the packets because we were packet sniffing on our own device, which may not be the case on the open Internet.

Finally, we would like to develop models to identify other user actions and behaviors on Spotify, with the most interesting being what ad is being played as this could have huge security and monetary implications if successful.

10 Conclusion

This paper describes the machine learning models created to analyze the packets sent through the encrypted channel created between a personal computer and Spotify. In our experiment, we preprocessed data using a variety of Python packages and tools. We then inputted this data into a variety of machine learning models to determine which model yielded the highest accuracy with predicting a song based on the packets being sent through the encrypted channel. Through this experiment, we learned just how much information can be found through simple traffic analysis when there is no protection regarding the timing of packets.

11 Author Contributions

Each teammate contributed to each part of the project, however, we would like to acknowledge Kameron Dawson and Antony Hernandez for their work on the experiment and technical analysis, as well as Tejal Reddy and Tomisin Ogunfunmi for their research on similar experiments and projects.

12 Acknowledgements

We would like to thank Professor Ron Rivest and Professor Yael Kalai for teaching us this semester. We would also like to thank the 6.857 TAs for their feedback and help in the ideation of our project.

References

[1] Pytorch. https://pytorch.org/.

[2] Selenium. https://pypi.org/project/selenium/.

[3] Welcome to spotipy!. https://spotipy.readthedocs.io/en/2.19.0/.

[4] Philippe Biondi and the Scapy community. Scapy. https://scapy.net/.

[5] Saman Feghhi and Douglas J. Leith. A web traffic analysis attack using only timing information. *IEEE Transactions on Information Forensics and Security*, 11(8):1747–1759, 2016.

[6] S. Kadloor, X. Gong, N. Kiyavash, T. Tezcan, and N. Borisov. Low-cost side channel remote traffic analysis attack in packet networks. In *2010 IEEE International Conference on Communications*, pages 1–5, 2010.

[7] Ignacio Oguiza. tsai - a state-of-the-art deep learning library for time series and sequential data. Github, 2022.

[8] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1357–1374, Vancouver, BC, August 2017. USENIX Association.

[9] Spotify. https://www.spotify.com/us/.

[10] Michael Lawrence Waskom. seaborn: statistical data visualization, April 2021.

[11] Wireshark. https://www.wireshark.org/.