IDENTIFICATION OF USERS VIA SSH TIMING ATTACK

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Thomas Flucke

July 2020

COMMITTEE MEMBERSHIP

TITLE: Identification of Users via SSH Timing Attack

AUTHOR: Thomas Flucke

DATE SUBMITTED: July 2020

COMMITTEE CHAIR: Bruce DeBruhl, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Phillip Nico, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Dongfeng (Phoenix) Fang, Ph.D.
Professor of Computer Science

ABSTRACT

Identification of Users via SSH Timing Attack

Thomas Flucke

Secure Shell, a tool to securely access and run programs on a remote machine, is an important tool for both system administrators and developers alike. The technology landscape is becoming increasingly distributed and reliant on tools such as Secure Shell to protect information as a user works on a system remotely. While Secure Shell accounts for the abuses the security of older tools such as telnet overlook, it still has fundamental vulnerabilities which leak information about both the user and their activities through timing attacks. The OpenSSH client, the implementation included in all Linux, Mac, and Windows computers, sends each keystroke entered to the server as soon as it becomes available. As a result, an attacker can observe the network patterns to know when a user presses a key and draw conclusions based on that information such as what a user is typing or who they are. In this thesis, we demonstrate that such an attack allows a malicious observer to identify a user with a concerning level of accuracy without having direct access to either the client or server systems.

Using machine learning classifiers, we identify individual users in a crowd based solely on the size and timing of packets traveling across the network. We find that our classifiers were able to identify users with 20% accuracy using as little as one hour of network traffic. Two of them promise to scale well to the number of users.

ACKNOWLEDGMENTS

Thanks to Ethan Goldfarb for his contributions to writing the machine learning algorithms.

Thanks to Tedd Akdag for his assistence deploying and maintaining the data collection environment.

Thanks to Andrew Guenther for creating a LaTeXtemplate for Cal Poly Thesises.

Thanks to my family for their supporting me earning my degrees.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Secure Shell is a core part of the computing landscape. According to one survey, 68% of respondents said that Secure Shell is important or critical to their organization [26]. With the internet increasingly moving towards cloud hosting and distributed computing, there is a distinct need for the ability to perform system maintenance remotely. Over years, several protocols have existed to remotely launch programs, but many of them did not account for potential abuses and provided little to no security [46, 43]. Secure Shell was developed in part as a response to this lack of security and attempts to protect the confidentiality and authenticity of the transmitted data. In comparison to the prior iterations of these remote access protocols, Secure Shell does this job quite well and has come to replace these out-dated tools almost entirely.

But as the computing industry becomes more security conscious, attackers become more clever. Rather than attacking a system directly, timing attacks become increasingly common [32, 6, 25, 51]. This is an attack surface which is not covered in the Secure Shell standards, which neglect the implementation details and allow the client implementation to decide how to control the flow of data for performance reasons [45]. Unfortunately, this allows room of insecure implementations after the authentication process has completed [43]. In this study we examine one possible implication of this lack of definition.

In the OpenSSH implementation of Secure Shell, the implementation included in all Linux, Mac, and current Windows computers by default, the client optimizes performance by sending each key or signal as soon as it is received [10]. As a result, a

malicious attacker who is able to observe the network traffic can identify with varying levels of precision when a victim presses a key. On first notice, knowing when a user presses a key may seem harmless, but it reveals private information about both the user and what they are typing [32, 25, 43, 3, 51, 42, 38, 27]. While other researchers have demonstrated that information such as what password a user has entered into the terminal, we examine a more subtly dangerous information leak. In this paper, we use machine learning demonstrate that an attacker can use the information that OpenSSH sends over insecure networks to identify the user typing at the terminal.

While less obviously dangerous than obtaining passwords, this breach of confidentiality has serious implications. An attacker can combine this attack with the aforementioned password attack to obtain both an identity and a password. An oppressive government could use this method to identify citizens over what it intended to be a confidential channel. Or an organization could use this method to covertly track the movements of individuals in a direct violation of their privacy.

Studies have already shown that a program can identify a user based on their typing patterns [42, 38, 27]. Another study has found that software reveals identifying information about itself over secure tunnels which machine learning algorithms can reliably detect. [25]. Other studies have already exploited such weaknesses in the Secure Shell protocol [43]. This study will be the first attempt to combine these methodologies to identify a human individual without every having direct access to either the client or server systems.

We find several machine learning classifiers which were able to classify many people in a crowd of connections with up to 27% accuracy. In addition to this, we find that the method scales well with the number of potential victims implying this attack could be used on a large population without significant loss. We also examine the importance

the amount of data plays in the accuracy of the system, the source from which the data was gathered, and the comparative effectiveness of several learning algorithms.

1.1 Ethical Implications

Before we begin this study, as is always the case when testing the security of a system, we must consider the ethics of the study. This in study, we attempt an attack which thus far has not been demonstrated to be possible. Malicious entities reading this paper may use the information contained within to use this attack on victims outside of a laboratory setting.

The risk of an attacker using the information in this paper is not new though. Others have already presumed that such an attack is possible and a determined attacker may already attempt it [43, 3]. One of these studies have already been able to pull password information from the vulnerability. We are not the first to believe that such an attack is possible nor are we the only ones with the resources to attempt this. All of the software used in this study are open source and therefore any person with the technical skill could do what we have done in this research. There are many malicious actors with more skill and resources than we have who could likely replicate these results more reliably. We therefore must conclude that without our involvement, potential attackers might already exploit this breach of confidentiality. We hope that by publishing this research we can impress the significance of this vulnerability which has already been proven to be exploitable by other papers [43].

Chapter 2

BACKGROUND

2.1 Terms and Technologies

This thesis makes reference to a number of networking tools and technologies and machine learning techniques which are imperative to understanding this research and its significance. This section will make clear the details of these terms.

2.1.1 Secure Shell

Secure Shell (SSH) is a protocol which provides secure network services over an insecure network [44]. The details of what services this protocol accommodates are flexible, allowing the SSH server to supplement a default set of procedures. On Linux and most Unix-like systems, the program `ssh` will use the SSH protocol to open an interactive command shell on a remote server [49]. The implementation details of these procedures after requesting a service are not defined in the Request for Comment (RFC) defining the service beyond a few required messages and services which must be supported by all implementations and a few reserved service names [44].

Secure Shell is possibly one of the most critical tools for system maintenance. SSH is pervasive in industry with 82% of survey respondents reporting that they use it and 68% said SSH was important or critical to their business [26]. OpenSSH (a specific implementation of SSH) is used in all Linux, Mac, and latest Windows computers at the time of writing, as well as many more systems [10]. Because of the wide use of this tool and the intended use of transporting sensitive information over an insecure

network, information leaks resulting from implementation details have the potential to jeopardize the confidentiality of data the user believes to be secure [44, 26].

Secure Shell is built on top of the Transmission Control Protocol to ensure that all signals sent from the client to the server are delivered correctly.

2.1.1.1 Transmission Control Protocol

The Transmission Control Protocol (TCP) is a network protocol which ensures data send from one host to another arrives at its destination and are ordered correctly [35]. This is often implemented as a module inside the the operating system kernel. The module will buffer inputs and outputs separately, sending segments at a time to create the appearance of a continuous stream of data and saving them on the receiving end. Important to this study, TCP does not guarantee that data will be sent immediately when the sending process provides it [35]. Rather, the protocol will wait until the buffer is filled unless the sending process associates the PUSH flag with the data. In the case that the PUSH flag is associated with outgoing data, the output buffer will be sent to the receiving host immediately, regardless of the buffer's current capacity.

2.1.2 Attack Models and Defenses

This thesis seeks to expose information which SSH should protect. To this end, we make reference to several models of attack and defense and the counterplay between them.

2.1.2.1 Man in the Middle Attacks

A Man in the Middle (MitM) attack is a type of attack that is conditional on the access of the attacker. Specifically, the attacker places themselves in the path of communication between two hosts such that all network traffic between them pass through the attacker's system [5]. Figure 2.1 visualizes this. This allows the attacker to observe, modify, replace, or drop packets. This process can be thought of as similar to a malicious mailman who opens up the letters of the families he delivers letters to or writes fraudulent letters in the name of a friend or family member. Protocols which use strong encryption and hashing algorithms are resistant to MitM attacks attempting to modify or replace the communication because of the difficulty of creating a new, valid encrypted message without also having the encryption secret [46]. Observing the communication directly would require breaking the encryption which is difficult using industry standard encryption methods.

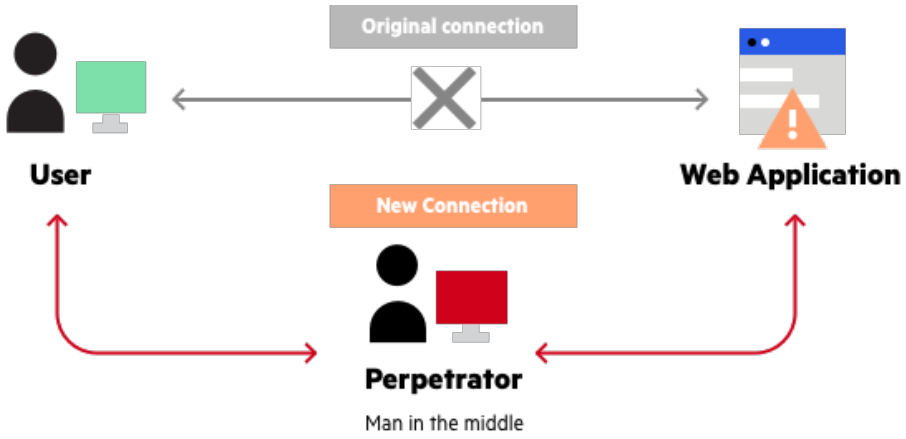


Figure 2.1: Visualization of a Man-in-the-Middle Attack [13]

2.1.2.2 Encryption

Encryption is the process of encoding information using a secret value, often called a secure key or secret, such that only those who know the secret can read the in-

formation [1]. This protects against direct manipulation of the data by a MitM as described above because without knowing the secure key the attacker cannot create a new message which the secure key will decrypt correctly. Having this protection is useful when communicating over an untrusted network where bad actors might access private information. Additionally, encryption provides some defense against altering the message because without knowing the secret the attacker cannot know how their changes would translate in the decryption process. More advanced cryptographic techniques can provide further security against certain types of nuanced attacks.

2.1.2.3 Timing Attacks

Timing attacks seek to use information about the timing of data to infer information about it [6, 25, 32]. Encryption provides no protection against a timing attack because timing attacks do not attempt to directly access the data, rather learn about it by inference. For example, if an attacker knows that a bank statement takes on average 1 second to load, but a money transfer takes 7 seconds, the attacker can measure the time between when the server receives the request and when it sends a response to estimate what type of operation the client requested. The attacker never needs to see the contents of the communication between the client and the server to make this inference assuming it has a prebuilt lookup table mapping timings to events.

There are many types of timing attacks, but in this thesis we focus on network based timing attacks.

2.1.3 Machine Learning Classifiers

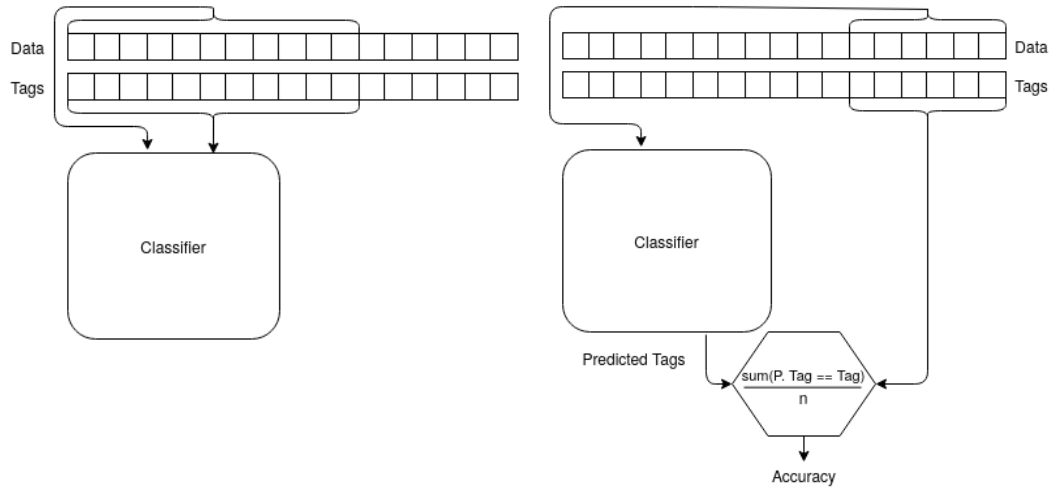
Machine learning (ML) is a class of algorithms that take data as an input and use mathematical models to find patterns in the data [31]. These patterns are incor-

porated into a resulting system which can provide insights into new data. The significance of this is that the programmer does not need to tell the algorithm what patterns to look for as the algorithm will discover these patterns on its own. This is very useful for discovering patterns in complex, multifaceted data which may not have an intuitive algorithm to solve. While ML can be either supervised, unsupervised, or semi-supervised, this thesis only makes use of supervised learning. In this thesis, several machine learning algorithms are used to interpret the complex data obtained through the MitM.

2.1.3.1 Supervised Learning

Supervised Learning is the process of training an ML algorithm using known data [23, 37]. An existing set of data is pre-tagged, usually manually, with the desired output for the algorithm. The ML system is then feed the data and tags which it uses to find the patterns, a period known as training. Once the patterns are discovered, the system can take untagged input and output a predicted tag.

Afterwards, a testing period verifies that the system has trained correctly. This is done by segregating the pre-tagged data set into a training set and a testing set [23, 37]. The algorithm is only allowed to train on the training set. Once training completes, the system is used to predict the tags of the testing set without seeing the tag. The predicted tags are then compared to the actual tags to determine the accuracy of the system's outputs. Figure 2.2 visualizes this whole process.



(a) Training (b) Testing

Figure 2.2: Visualization of Training and Testing

2.1.3.2 K-Fold Validation

A risk in the tradition ML training/testing split is that due to anomalies or unevenness in the distribution of data points in the training and testing set, the algorithm will seem to produce results which do not represent the general population. For example, if the data points of one tag are disproportionately placed into the testing set it will artificially weight the accuracy of the final test to algorithm's ability to identify this tag. K-Fold Validation mitigates this problem.

Using K-Fold validation, all the samples are divided in K separate divisions [37]. Then the algorithm is trained and tested K times, each time disregarding the previously learned patterns and each time using a different division as the testing set as can be seen in Figure 2.3. After all the runs are complete, the results of each of the testing phases are averaged into a final accuracy score. This method ensures that each tag is represented in the final accuracy score in relation to its representation in the data.

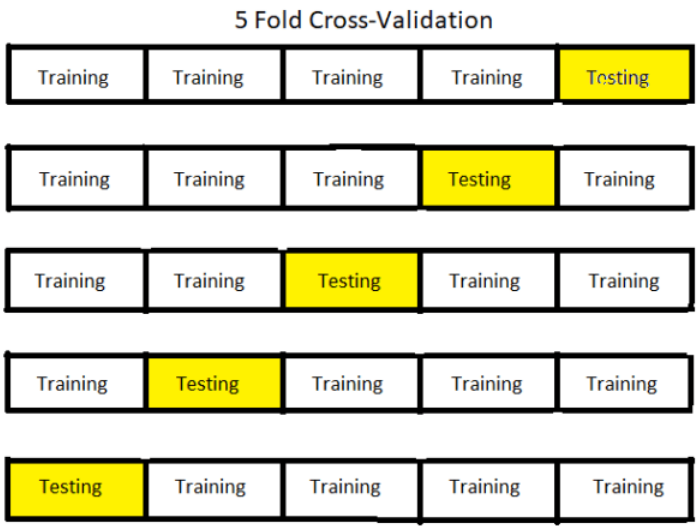


Figure 2.3: A visualization of 5-fold validation [41].

2.1.3.3 Hypertuning

ML Algorithms are not one-size-fits-all. Most algorithms have configuration parameters which determine how the program runs [23]. The exact parameters which yield the best results are often domain specific and can be hard to determine without experimentation. The process of hypertuning, also called parameter tuning, involves running the classifier with each set of parameters to determine whether or not a particular configuration provides better or worse performance than the others. The ultimate goal of hypertuning is to find the optimal set of parameters. The developer can hypertune for any variable they prefer, but in this thesis we hypertune exclusively to maximize accuracy.

2.1.3.4 Principal Component Analysis

Principal Component Analysis (PCA) is the process of transforming data using linear analysis into a new dataset which preserves the differences [23]. The important dif-

ferences in the new dataset are that each of the components are uncorrelated, which hence reduces the redundancy of features. Additionally, the analysis orders the principal components by their descriptive power such that the components most useful are first. Means that depending on the dataset, a small handful of features can be chosen which contain the same power as almost all of the features. This is useful for reducing the number of features which can help algorithms train more quickly by not having to discover which features are not useful.

Importantly, it is impossible to both reduce the feature space and retain the full descriptive power of the original feature space. Therefore, using PCA will require either losing some descriptive power or transforming the data into a descriptively identical feature set of the same length.

2.1.3.5 Classifiers

A classifier is an algorithm which takes in input and assigns it to one of several classifications. The goal of this thesis is to use classifiers to classify SSH connections according to who is using it. In this section, we will examine several classifiers which we use.

2.1.3.6 K-Nearest Neighbors Classifier

The K-Nearest Neighbors Classifier (K-NN) is a very simple algorithm. It takes each of the data points in the training set and plots them in N-dimensional space [23]. When asked to predict a new tag for a sample, the classifier algorithm finds where the new sample lands in the N-dimensional space and finds the K training points closest to it. Each of those training points then votes that the new point be classified according to their own tag. See Figure 2.4 for a visualization of this process.

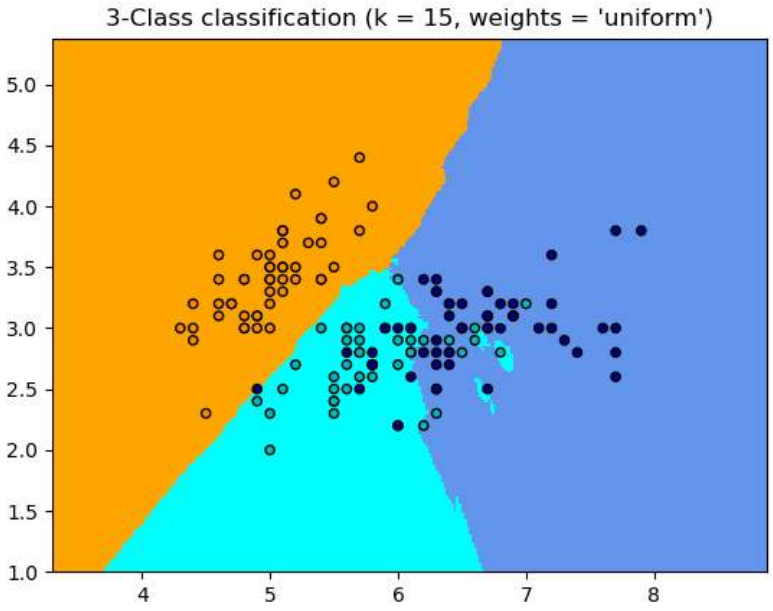


Figure 2.4: Visualization of a K-Nearest Neighbors Classifier
This visual uses a 2-dimensional feature space and K = 15 [37]. The dots represent training samples, the color of the dots represent the training tag, and the colored space represents the predicted tag for a testing sample at that location.

This method supports many voting schemes, although the most common are uniform where each point casts one vote and linear distance where the votes of training points count more the closer it is to the new point [23].

Additionally the K value is flexible, although which is best tends to vary on the type of data [23].

Finally, the K-Nearest Neighbor has the unique feature of offering a hierarchy of classifications. Unlike most ML classifiers which output a single tag, K-NN can output a list of possible tags ordered by percentage of the vote received.

2.1.3.7 Random Forest Classifier

A Random Forest (RF) Classifier is a model which uses an array of Random Decision Trees (RDT). Each tree is composed of nodes, branches and leaves [23]. To classify data, a sample is fed into the root node. Each node contains a feature to examine and a set of branches corresponding to each of the possible values or ranges of values. Each branch leads to either another decision node which repeats this process, or a leaf which determines the class for the sample. See Figure 2.5 for an example.

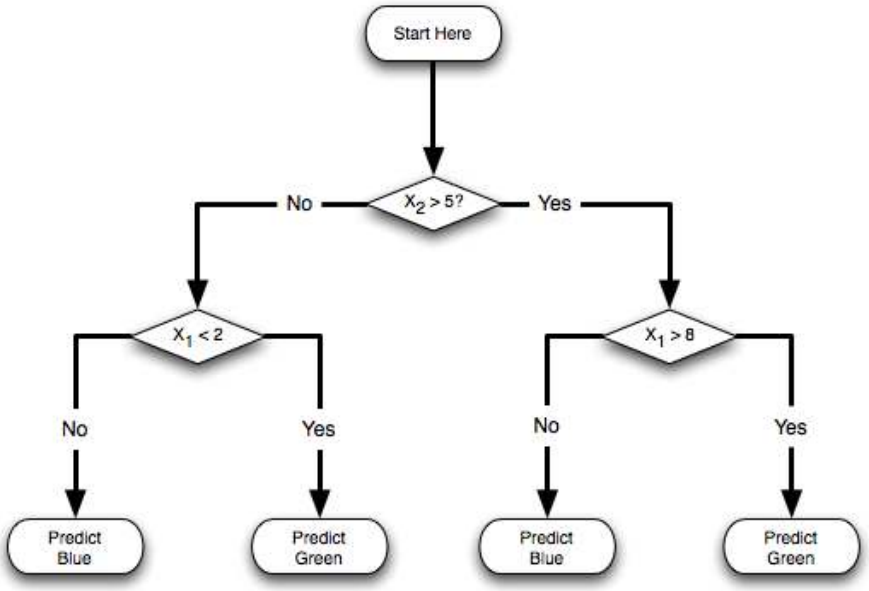


Figure 2.5: A sample RDT [47].

The RDT is built up from algorithms which examine training data and decide which features best characterize the data [23].

RDT are powerful because they provide conditional classification and relate features together while still being inspectable. One failing is that they can be fickle and small changes in how the graph is constructed can significantly change the results. To solve this problem, the Random Forest uses many trees in aggregate. The RF builds a

large number of RDTs using its training data. When asked to classify a sample, the RF feeds it to each of the RDTs. Each RDT then votes for a proposed classification and whichever tag gets the most votes is the output of the RF.

2.1.3.8 Support Vector Machine Classifier

A Support Vector Machine (SVM) is a relatively new ML algorithm [23]. It works by plotting the training points on an N-dimensional space and drawing divisions to create a, 'margin', between the classifications. Untagged data is then plotted in the N-dimensional space and associated to the class it most closely fits. In order to force the distinct classes to be non-overlapping, the input space is mapped onto an N-dimensional feature space.

The transformation allows for a non-linear separation in the input space to become a linear separation in the feature space as seen in Figure 2.6.

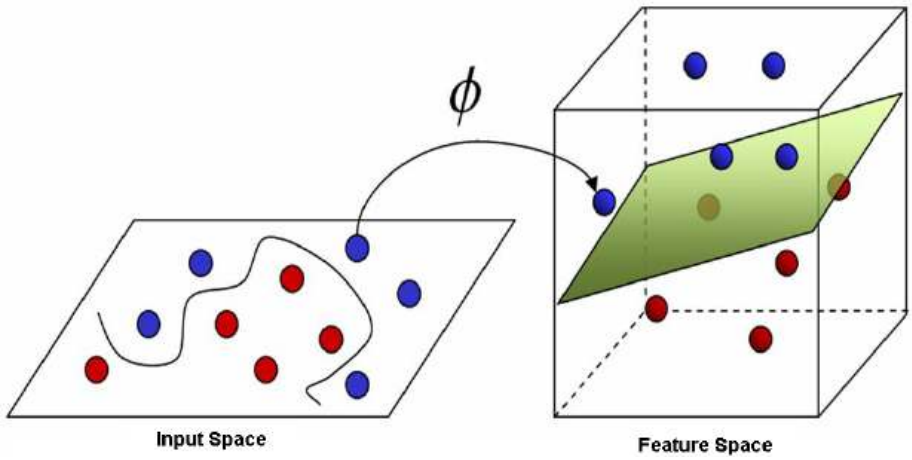


Figure 2.6: Conversion from input space to feature space in a SVM [50]

Because SVMs uses n-dimensional vectors rather than the original input data, the computations can be faster and the model scales more efficiently [23]. At the same

time, the algorithm is very sensitive to its configuration making good hypertuning crucial. Additionally, training an SVM is slow which complicates hypertuning.

2.1.3.9 Perceptron Classifier

A perceptron is a simple model which outputs either a 1 or a 0 according to equation 2.1 where $x_{1..N}$ is an array of inputs, $w_{1..N}$ is an array of weights, and b is a bias weight. [23].

$$b + \sum_{i=1}^{N} w_i x_i > Threshold \tag{2.1}$$

A single perceptron or array of perceptrons by themselves can classify according to a linear combination of features. For more complex classifications, perceptrons can be arranged into layers. Most commonly, one layer will receive an input, pass it on to a hidden layer which then can pass it on to more hidden layers until reaching an output layer which classifies the input based on the previous computations. This model simulates the processes of the biological brain, with each perceptron representing a neuron which either fires or does not. Each neuron which fires influences the potential for further neurons to fire. The result is a system that classifies values according to non-linear combinations of inputs, as is visualized in Figure 2.7.

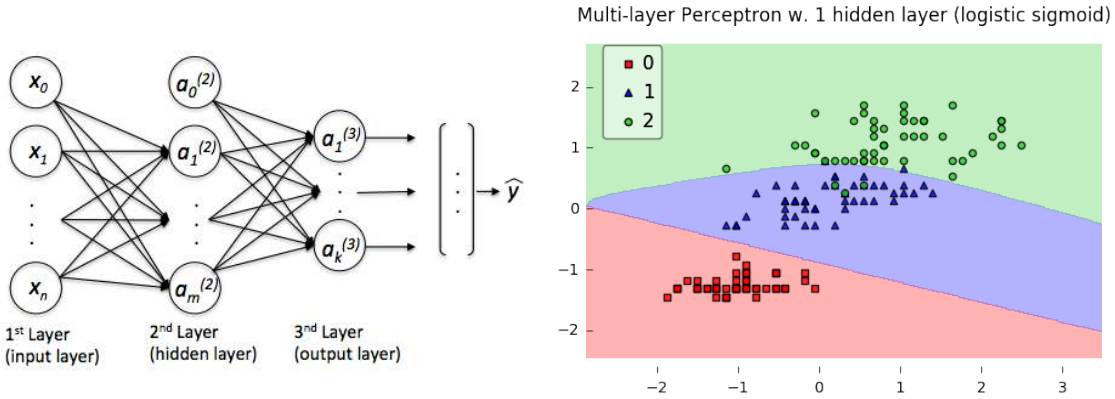


Figure 2.7: Visualization of an ANN [39].

To train the perceptron network, also known as an artificial neural network, neural network, or ANN, training data is fed through the network as if it was being classified. Once a result tag is determined, it is deemed either correct or incorrect. If correct, all perceptrons involved in the computation are, 'rewarded', and given stronger weights. If the output is incorrect, the perceptrons involved are, 'punished', and given weaker weights.

Perceptrons are very powerful, but are hard to tune properly and take a long time to train [23].

2.1.4 Statistical Accuracy

Any algorithm has a chance of getting some predictions correct by random chance [9]. In some cases, the algorithm may outperform the statistical average by sheer luck. In order to be confident that the model is performing according to its merits, we employ several statistical models.

2.1.4.1 P-Values

When determining whether or not a result is significant, we calculate a p-value. Given a null hypothesis (H_0), a p-value is the probability of observing the provided sample [9]. This idea is visualized in Figure 2.8

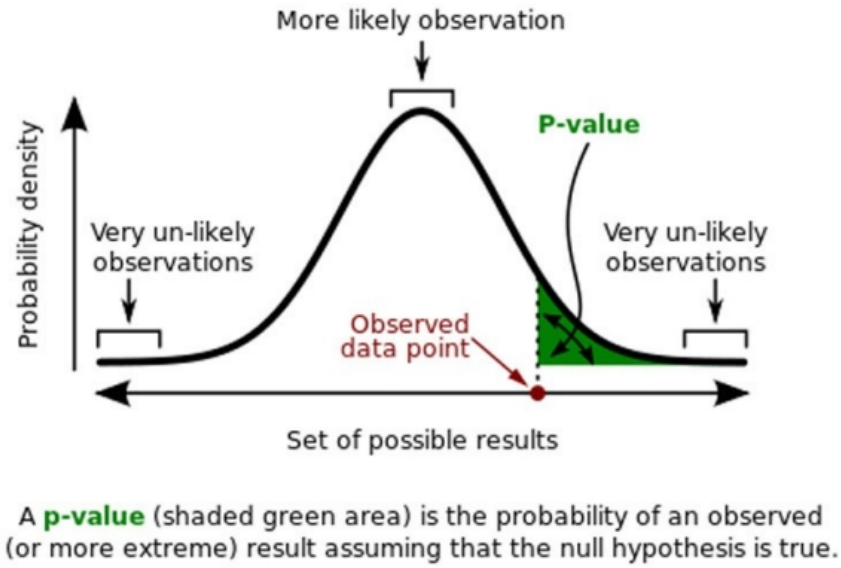


Figure 2.8: Visualization of a P-Value [48]

A p-value is important because of its use in comparing to the significance level of results. Before a significance test, a threshold significance level is predetermined for the p-value. When the p-value is below the significance level, in other words the sample is a below a certain likelihood of being observed, the results are considered significant enough to reject the null hypothesis. Common significance levels are 0.05 or 0.01.

2.1.4.2 Confidence Intervals

A confidence interval is a range of values derived from the sample in which the true value of the measure parameter is expected to be [9]. If a test were repeated 100 times and a 95%-confidence interval calculated for each one, 95 of the resulting ranges would be expected to contain the true value. For example, if someone were to measure the height of 50 men in the United States and calculate a 95%-confidence interval based

on that sample, there would be a 95% chance that the average height of men in the U.S. would be within that interval.

2.1.4.3 F1-Score

The F_1-Score is a measure of a classifier's accuracy. The F_1-Score works by computing the harmonic mean of the recall and precision as seen in Equation 2.2 [12].

[!ht]

$$F_1 = 2\frac{pr}{p+r} \tag{2.2}$$

Figure 2.9: The F_1-Score Equation.
p represents precision and r represents recall.

The precision is defined as the ratio of predictions which are accurate to total predictions. The recall is defined as the ratio of predictions which are accurate to all the predictions which should have received the tag. For example, a classifier which always predicts the tag A for a dataset which contains an even split of A, B, and C will have perfect recall for A but 1/3 precision.

2.2 Related Works

Although some researchers have suggested that studies have proven that an attacker can identify a user based on their packet timings, they failed to provide a source which demonstrates this and we could not find any corroborating evidence in scholastic databases [3].Other researchers have proposed that such an attack is hypothetically possible, this particular attack has thus far been outside the scope of their research [43]. Although this attack has not explicitly been tested, there have been several studies in related areas that indicate its potential.

2.2.1 Vulnerabilities in Secure Shell

Because Secure Shell is so widely used, there already exists studies which examine possible vulnerabilities in the protocol or the OpenSSH implementation which we discuss here.

2.2.1.1 Secure Shell Timing Attacks on Passwords

In a study by study conducted by Song et al. they were able to extract information about passwords typed on a secure SSH channel using only the timing of packets [43]. By detecting packets which represent keystrokes, the researchers were able to accurately measure the timings of key presses and calculate the time between them. They could then map the timings of the digraphs to a pre-computed statistical model of digraph timings. From this, they were able to determine on average 1 bit of information per keystroke in addition to the length of the password. Using this information, they created a Markov Model which successfully guessed the password 70% of the time within 40 guesses. They only used 4 human participants and found that testing accuracy was hindered by using training data a different user.

In their paper, they suggest that the inter-keystroke timing could be used to identify the user but did not attempt to detect this using the Secure Shell traffic.

2.2.1.2 Attacks on Secure Shell Cryptography

Researchers Albrecht et al. published in their paper a vulnerability working around the cryptography of Secure Shell [1]. By mutating the header of a Secure Shell packet, replacing it with a block from another message, they can distort the length field in the header. This will cause the MAC validation at the end of the packet to fail. The

number of blocks read before the MAC validation fails reveals information about the substituted header. In this way, the researchers were able to extract 18 to 32 bits of cleartext, depending on implementation details, without ever accessing the shared secret.

2.2.2 Timing Attacks

In addition to attacks targeting SSH, there have been studies similar to this in that they use the timing of network traffic to extrapolate sensitive information. While these papers do not have the same targets as this thesis, they do use similar methods and prove some degree of viability.

2.2.2.1 Identifying Encrypted Traffic

Leroux et al. examined the network patterns of several applications to see if they could be identified over an encrypted network [25]. They were able to identify the program from which network traffic originated with great accuracy based solely on the timing and size of packets on the Tor network or a VPN. They were able to accomplish this by measuring the Inter-Arrival Time (IAT), packet size, and 'burst' size (A burst is a grouping of packets in close succession). Feeding these statistics into several machine learning algorithms, each was able to learn the patterns of different applications to some degree. This proves that different applications have significant differences in their network signatures and that the timing differences introduced by redirection and encryption do not significantly obscure these differences.

2.2.2.2 Search Engine Auto-Complete

In a recent study, Monaco was able to use the timing of packets to determine what a user typed into a search engine. [32]. Many search engines now have an auto-complete feature which attempts to guess what a user is typing before they finish. Whenever a user begins to type, the browser sends the partial search string to the server which then sends back a list of possible completed search strings. Using the timing of these signals and the size of the partial search string, Monaco was able to decipher what keys were pressed with near perfect accuracy.

Of note, this study did not use live participants but rather a public database of recordings of participants typing which were replayed into a live web browser.

2.2.2.3 Noise

In a study conducted by Crosby et al., they examined the potential of timing attacks executed over a network and the levels of precision to the timing of events over a global network [6]. They found that the round-trip-time for a packet strongly correlated with the time the server spent processing a request. This would indicate that that after adjusting for networking delays, the variation in noise delay from a network is negligible compared to the primary operations. Contrary to this, they also found that at the lowest quartile for response speed the correlation began to waver significantly.

Notably, neither this paper nor Leroux et al. examined the timing effects of both encryption and network noise on an encrypted payload [6, 25]. Despite that neither of these papers tested the impact of these two variables together, neither alone has a disruptive impact on the timing reliability. This study proceeds with the assump-

tion that these variables combined will not have a significant disturbance. If such a disturbance does emerge, it will show in the analysis of this study.

2.2.3 Programmer Identification

Finally, there has been plenty of research done on identifying users based on how they use the computer. Many of these study focus on identifying intruders or fraudulent accesses, but these techniques could also be used for more malicious purposes by an eavesdropping attacker. Additionally, because programmers interact with a computer differently than those from other fields, some of these studies have focused on this occupation specifically [27].

2.2.3.1 Typing based Identification

A study from the University of Helsinki in Finland found that they could reliably identify programmers based on their typing patterns [27]. In their study, they measured the time between keystrokes, time to press a key, or time to type a specific digraph. Using this data and a simple nearest-neighbor classifier, they were able to successfully identify the participant with a high degree of accuracy. In particular, they found that the time to press a specific key combined with the time to type a digraph were the most useful for their algorithms.

2.2.3.2 Shell based Identification

In a study conducted by F. Khosmood, P. Nico, and J. Woolery, researchers found that users are strongly identifiable based solely on their shell usage [22]. Using only the bash history, they were able to identify users from a large sample with almost

90% accuracy using natural language processing and machine learning. In particular, they found that the most useful features for identification were the unique tuples of 3 words, and that the most useful algorithm was the multinomial logistic regression classifier.

Chapter 3

METHODS

This study seeks to prove the viability of an attack. Towards that end, we setup up our recruitment and data collection methods to imitate how such an attack would occur in the wild. Once data has been collected, our preprocessing system separates and tags the data for each participant and performs the feature extraction. Finally, we feed the tagged data into the machine learning classifiers for analysis and compute the final accuracy and power score pairs.

3.1 Attack Model

The attack we imagine follows the following flow:

1. Attacker obtains a position between the victim and their server.
2. Attacker filters SSH connections based on IP to limit the scope of the attack.
3. The attacker acquires a sample of the victim's typing patterns.
4. Attacker uses machine learning to identify victim.

The attacker's system will look like Figure 3.1.

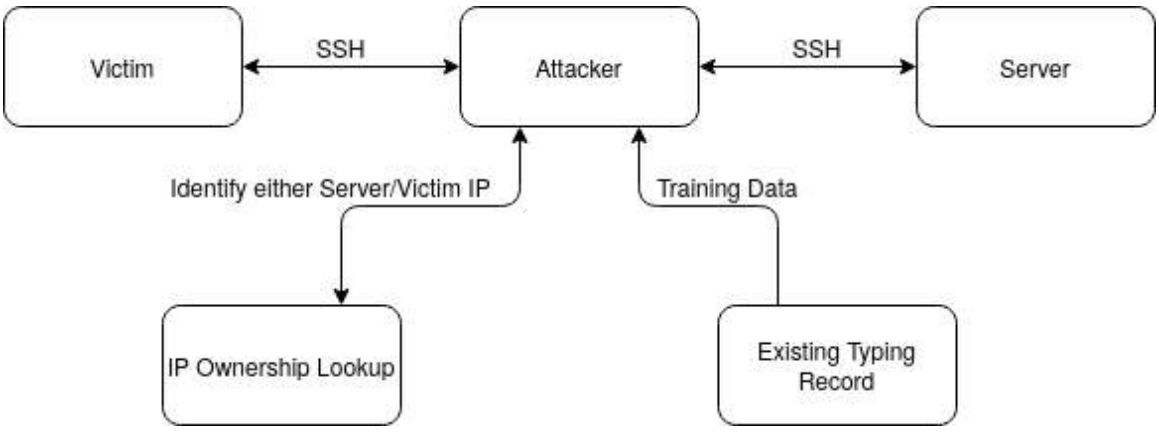


Figure 3.1: Model of the Attack

Step 3 in this attack can occur at any point before step 4. We will address how any attacker may achieve each of the first three steps here and we analyze the viability of step 4 in the results section.

3.1.1 Obtaining MitM Status

Obtaining a privileged position between the victim and their server has challenges but is possible. An attacker with state level resources could easily obtain such a position through the Autonomous Systems (AS) which already sit in such a position [7]. An attacker with fewer resources could position themselves close to the victim and advertise a false route to the server. This technique is known as a route attraction attack, in which an attacker advertises a phoney route to a target resulting in the network routers sending all the data to the target to the attacker [2, 24]. And because the connection is not disturbed in any way, the attack can be difficult to identifying the bad actor. In either case, attackers have known methods for achieving this position.

3.1.2 Filtering IPs

IP addresses are generally not considered sensitive information by virtue of being publicly registered and advertised as a core part of routing packets [11, 34]. The IP addresses of companies and organizations can be found by an easy search [4].

3.1.3 Obtaining Typing Record

Finally obtaining a recording of the victim's typing can be the most difficult of the three but still possible. Users type into web sites freely and without concern for the data they generate. Popular web sites like Facebook already record activity information such as this [16]. If the attacker does not have access to such data already, they could make a web site with the purpose of extracting typing information from a victim [19]. Certain patterns such as a victim's typing speed, propensity to copy and paste, and activity patterns such as how frequently they switch between tasks could all be measured by a user's normal web usage. At the same time, we do not have any guarantee that patterns identified from one dataset will translate well into another.

Additionally if this attack is viable, it suggests that more autonomous methods of identification may be possible. In this study we use supervised learning, but future studies might find success with semi-supervised or unsupervised learning algorithms which would negate the need for a separate dataset.

3.2 Recruitment

In this paper, we assess the viability of an attacker identifying an victim by the Secure Shell traffic. To accurately test the attack, we created a test environment

which reflects an attack in the wild as closely as is possible with our resources. In particular, we focus on these factors:

- Different people typing
- Different types of tasks
- Real network latency/jitter

To meet the first point, we make sure to recruit at least 25 distinct participants. The second point is addressed in Section 3.2.1 and the last point is addressed in Section 3.3.

Because this study involves the monitoring of participants and the aggregation of their data, we made a concerted effort to protect the privacy of participants To this end, all procedures involving data collection were reviewed and approved by Cal Poly's Institutional Review Board prior to the collecting of any data.

3.2.1 Participants

Participants were recruited from the student body of California Polytechnic State University (Cal Poly). Each student is required to either be enrolled in or have taken CPE 357, the Systems Programming class which introduces C, Unix, and shell [33]. This requirement was necessary to ensure a baseline level of competency with a Unix shell. Because it was not strictly relevant to the study and to preserve the privacy of participants, no other demographic data was collected. We could have asked questions about what instructors students had and how frequently they used shell terminals, but this would have added an extra survey and we wanted to minimize the overhead of participating to attract participants.

Students performed classwork activities from these courses:

- Systems Programming
- Introduction to Operating Systems
- Introduction to Security
- Programming Languages
- Dynamic Web Development

Because we did not collect demographic information of individual participants, we do not have a distribution of how many participants came from each course, but the number of volunteers was not even across all the courses. Additionally, some participants may have offered data from multiple courses. Finally, two different professors taught Systems Programming and two professors taught Introduction to Operating Systems during the course of this study. Each of these professors had different sets of assignments and many of these samples were collected at different times during the course. Therefore, even participants in the same course were often performing different tasks.

The use of students from a single university rather than directly from industry introduces a potential for bias. The students may or may not have distinct typing patterns from those with more experience or working in an industry environment. Additionally, the Cal Poly curriculum may train students to use SSH in ways that are more or less similar to students from other schools.

These participants were incentivized to participate with entry into raffle for 2 Amazon gift cards worth $50 each [14].

3.2.1.1 Consent

All participants were asked to sign a consent form, approved by the Institutional Review Board, explaining the details of the study before participating. This form can be found in Appendix B and C. Additionally, participants received a description of the study and how their data would be used before consenting. As a final measure, at the end of the Secure Shell session the testing environment would provide a log of what was recorded and confirmation that the participant wants to submit this data. If the participant does not accept at the end, no additional data will be sent and any existing data will be discarded.

3.2.1.2 Anonymization

Once data is collected, the log of the Secure Shell session is tagged with an sha256 hash of the participants id. This allowed for the participant to be uniquely identifiable without preserving any of their personal information in the tag. Because we needed to be able to link multiple data collection sessions with the same participant together, we were unable to randomly salt the hash.

3.3 Collection Environment

As stated before, realistic representation of a wild attacker was a priority for this study. In particular we wanted to include the real life inconsistencies of a connection traveling over the internet, including the network latency and jitter. This determined how we designed our collection environment to capture as much of this as possible.

A diagram of the data collection environment can be seen in Figure 3.2.

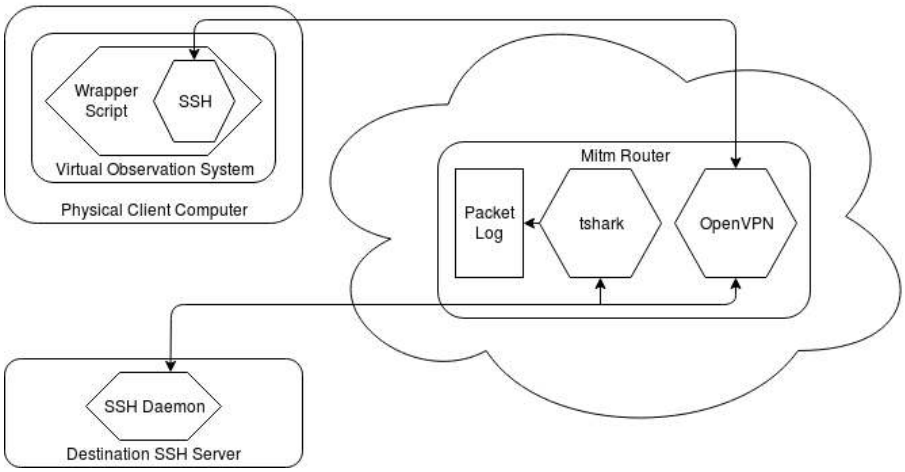


Figure 3.2: Layout of Data Collection Environment

The Physical Client Computer represents the physical computer the participant uses during the study. This computer used either a 64bit CentOS or 64bit Ubuntu 16.04 as an operating system. The reason it may be either of these two options is because the lab the participant worked on depended on which room was most convenient to them for data collection. The physical computer ran a 32bit Ubuntu virtual machine running in VMware Workstation version 15 [21, 29, 18]. This virtual machine was the Virtual Observation System (VOS). Next a virtual machine hosted on Amazon Web Services (AWS) represents our MitM router [15]. The MitM records all traffic going to TCP port 22, the port dedicated to Secure Shell by IANA, and saves it to a packet log using tshark for later analysis [36, 20]. We configured the virtual machine to route through the MitM using OpenVPN, [17]. To minimize the overhead of OpenVPN and the impact it would have on the network traffic, we disabled all encryption and hashing OpenVPN provides. The goal of this study is not to asses the security of OpenVPN and by disabling these features we minimize the overhead of the connection. Because the MitM records all outgoing TCP connections to port 22, it will record all the routed traffic from the VOS to any SSH server. We could have hosted the MitM on the same network or had the VOS record the packets directly,

but then the packets would not have travelled over the internet before being recorded and the network noise would not be included. By hosting the MitM in AWS, we force the connection to travel over the internet before and after the recording point.

On the VOS, there exists a wrapper script which encapsulates the SSH client. The wrapper script passes all its data directly through to SSH with exceptions at the beginning and end of the connection and creates a log of the input forwarded to SSH. At the beginning, it will ask the user to enter which dataset they are participating in, either guided or free. At the end the VOS filters the log, remove everything except the start of the connection and the keys sent to the server. This filtering removes useless and unwanted data such as passwords that may have been entered as part of logging in to the remote server. The VOS then presents the filtered log of recorded data for the participant to review and verify contains no sensitive information. Once the user has finished reviewing the log, the script prompts them to confirm their willingness to have their data included in the study. If they agree, the filtered log of the inputs to SSH are tagged with the sha256 hash of their id and response to the dataset prompt and sent to the MitM for collection by a researcher.

A special thanks to Tedd Akdag for his assitance deploying the VOS and maintaining the AWS MitM.

It should be noted that on occasion the network device in the VOS would blink out of and back into existence. Because of the complexity of the university network and lack of superuser access, we were unable to find the root cause of the problem. This blinking disrupts the VPN connection to the MitM by resetting the routing rules in the VOS. We fixed this problem by running a script which detects the broken connection and promptly repairs it. These blips may have some impact on the timing of the packets. When such a blip occurs, the packets may be delayed for up to one second. Because these network blips happen on average less than once per recording

session and all the features are averages of a sample, the blips will be amortized out. We therefore do not believe they will significantly compromise the validity of the data.

3.4 Datasets

Participants were asked to participate in two rounds of data collection which we call the guided and free dataset. The former provides a common baseline of participants performing similar actions. The later provides a more realistic look at how participants work in their daily assignments. Participants were not required to participate in both.

3.4.1 Guided Dataset

The goal of this dataset is to establish a common baseline of known typing patterns which could be used for analysis and potential extensions to this study.

The guided dataset involved follow a series of prompts as detailed in Appendix A.

The prompts were divided into these sections:

1. Loose English
2. Strict English
3. Loose shell
4. Strict shell

3.4.1.1 Loose English

In this section, participants were asked to respond to a series of questions is an English sentence. This allowed for a baseline of how a person would type under normal circumstances.

3.4.1.2 Strict English

Because the Loose English data does not guarantee that participants will try a variety of keys and to establish a comparable set of recordings, we then asked participants to transcribe a passage from the book A Tale of Two Cities by Charles Dickens [8]. This section was used in conjunction with the Loose English section because of concerns that the transcribing text would result in participants typing differently than they would if they used their own words. This section was presented after the Loose English to avoid priming the participants to be cognisant of their typing and word choice.

3.4.1.3 Loose shell

Participants were then asked to execute a number of instructions into a shell prompt. Each of the tasks provided were chosen to test potential differences in how a user might interface with a shell prompt such as whether they use a text editor or file redirection or whether they prefer to change their working directory or use paths.

3.4.1.4 Strict shell

Finally, subjects were asked to follow a series of provided shell commands. This section was chosen for similar reasons as the Strict English, to test different character combinations and establish a common baseline among participants. Similarly, this section was presented after the Loose shell section.

3.4.2 Free Dataset

The second dataset, named the Free Dataset, is intended to accurately reflect how a user interacts the system. During this portion, rather than providing a prompt, we instructed participants to perform their normal classwork activities as they would if they were not monitored. This went on for at least an hour per participant, although participants were welcome to work longer if they so desired. This allowed us to capture the normal patterns of typing and observe differences in work patterns between participants.

3.5 Preprocessing

After data is collected, the records consist of a series of keylogs tagged with a hash and dataset (one per ssh connection), and a stream of all the packets observed going to or from port 22. Before we can make use of this data, we need to transform it into a set of tagged log of packets. This process takes place in 4 stages:

1. Split each TCP flow into a separate log
2. Match each keylog to a TCP flow log
3. Extract features from each TCP flow
4. Filter the data down and remove unusable data.

3.5.1 Flow Splitting

Each SSH session forms a single TCP flow between the client and the server. When the SSH client begins, the operating system randomly assigns it a port and it connects to the server port 22 through that port. The Secure Shell client will continue to use this connection until the program terminates. This means that there is a one-to-one correlation with each keylog of a Secure Shell session and a TCP flow in the logs. A C program using libpcap reads the packet log and builds a table of packet logs [30]. When a TCP SYN packet is discovered, it creates a new packet log and associates the source ip, destination ip, source port, and destination port to the it. For each of the succeeding packet which matches the those four identifiers, the splitter adds it to the packet log. When the TCP closing handshake of FIN, FIN-ACK, ACK is observed, it closes the connection and writes the new packet log. This splits a single packet log into multiple, each of the new logs containing exactly one TCP flow.

3.5.2 Data Matching

After the flows have been separated into files, they need to be tagged with the participant hash and the type of data contained. Both of these tags can be found in a corresponding key log file. For each key log we have in our records, we find a flow file with a matching timestamp for the start of connection with up to 15 seconds of padding to allow for network latency. Additionally the number of keys logged is compared to the number of PUSH packets captured to ensure that two participants connecting at the same time are not confused. If the number of packets observed is different from the number of keys pressed, the files are not matched. If no match can be found, the log file marked as unmatched. This happened for several unrelated reasons, such as one day when the network latency was unusually high one day. This

causes a noticeable delay between when the application started and the MitM began observing traffic. This pushed the difference between the two times outside the boundaries of what the matching script would normally consider valid.

All of the matches are dumped to a text file for a researcher to review and verify is accurate. Additionally, the researcher may correct errors such as connections that should have matched but were disqualified from the automatic system.

Because we only match packet logs to key logs, any packet log for which no corresponding key log is found is ignored. Additionally, since the key logs contain all of the tagging information, any participant who opts out after recording data will not have any of their tags available to the researchers and their packets will be ignored.

3.5.3 Feature Extraction

Next the feature extractor pulls information of interest out of the packet logs. In particular, we focused on the observed inter-packet arrival time (IAT), the number and size of pastes of data copied from outside the SSH session, and the time spent in different levels of activity.

First, each packet log is divided into samples of fixed number of packets. We call the number of packets in each sample the Sample Size. Each sample has it's features calculated separately. In the case that a packet log contains a number of packets not divisible by the sample size, the remainder are disregarded.

Since each keystroke produces a single packet, the IAT of incoming packets measures the frequency with which a person presses a key.

Additionally, this can indicate the participants typing speed. Unfortunately, people rarely type at full speed continuously from the moment a shell session opens to when

it closes. To account for this, we created 4 categories of activity level: low, mid, high, and none. At any point in time, a participant is determined to be in one of the four activity levels based on the IAT of a lookback period before the packet. If the IAT of the lookback period is above the mid activity threshold, it is considered low activity. If the IAT of the lookback period is below the mid activity threshold but above the high activity threshold, it is considered mid activity. If the IAT of the lookback period is below the high activity threshold, it is considered high activity. If the IAT could not be calculated because there were no packets in the lookback threshold, that period is retroactively considered low-activity time. Any time which was not classified into one of the existing three activity states because the IAT was longer than the lookback period is considered to have no activity.

A graph of the activity states from two participants can be seen in Figure 3.3 and 3.4. These graphs were created with a lower activity threshold of 1 P/s and high activity threshold of 2.75 P/s and a looback of 3 seconds.

This allows us to single out the IAT of certain activity states as features of interest. This also has the additional benefit of allowing us to measure how frequently and for how long a participant spends time in each of these states.

The sample size, activity thresholds, and lookback time are chosen by hypertuning.

Finally, although each keystroke results in a single packet not all keystrokes result in a single character. If the user pastes data from their clipboard into the Secure Shell session, they will produce many characters which will all be encrypted together and sent to the server as one packet so long as the user does not exceed the maximum packet size. This means that the size of the packet can reveal information about user behavior.

Since all packets are padded to 8-byte blocks, we have limited granularity to identify and classify pastes. Additionally and data which did not use the client system's clipboard, such as yanking and pasting in vi, would not show as a paste because the data did not cross from the client system to the server.

Table 3.1 lists all of the preprocessor parameters.

Parameter Name	Unit	Definition
Sample Size	Packets	Number of packets per sample
Lookback Period	Seconds	How long to look back to determine current activity state
Low-Activity Threshold	Packets/s	Maximum number of packets in the lookback period to be considered 'Low Activity'
High-Activity Threshold	Packets/s	Minimum number of packets in the lookback period to be considered 'High Activity'
Small-Paste Threshold	Blocks	Minimum number of blocks beyond the standard packet size to classify a packet as a paste
Large-Paste Threshold	Blocks	Minimum number of blocks beyond the standard packet size to classify a packet as a large paste

Table 3.1: List of Preprocessing Parameters

We created the features to count the total number of pastes and the average size of the packets. Additionally we split packets into a large and small paste category and calculated these features again for each of these two subdivisions.

A list of all the features which are extracted and their definitions are described in Table 3.2.

Feature Name	Definition
AVERAGE_IAT	Average time between packet observations
DEAD_TIME	Time which is not spent in either high, mid, or low activity states
TOTAL_TIME	Total time spent
TOTAL_PASTES	Number of pastes
AVG_PASTE_SIZE	Average size of a paste
SMALL_PASTES	Number of pastes in the small category
AVG_SMALL_PASTE_SIZE	Average size of a paste in the small category
AVG_LARGE_PASTE_SIZE	Average size of a paste in the large category
LARGE_PASTES	Number of pastes in the large category
HIGHAVERAGE_IAT	Average time between packets observations in high activity mode
HIGH.BURST_SIZE	Number of packets observed in a single high activity state.
HIGH.BURST_COUNT	Number of times a user enters a high activity state
HIGH.TIME_SPENT	Number of seconds spent in a high activity state
HIGH.TOTAL_PACKETS	Number of packets observed in a high activity state
MID.AVERAGE_IAT	Average time between packets observations in mid activity mode
MID.BURST_SIZE	Number of packets observed in a single mid activity state.
MID.BURST_COUNT	Number of times a user enters a mid activity state
MID.TIME_SPENT	Number of seconds spent in a mid activity state
MID.TOTAL_PACKETS	Number of packets observed in a mid activity state
LOW.AVERAGE_IAT	Average time between packets observations in low activity mode
LOW.BURST_SIZE	Number of packets observed in a single low activity state.
LOW.BURST_COUNT	Number of times a user enters a low activity state
LOW.TIME_SPENT	Number of seconds spent in a low activity state
LOW.TOTAL_PACKETS	Number of packets observed in a low activity state

Table 3.2: List of Features

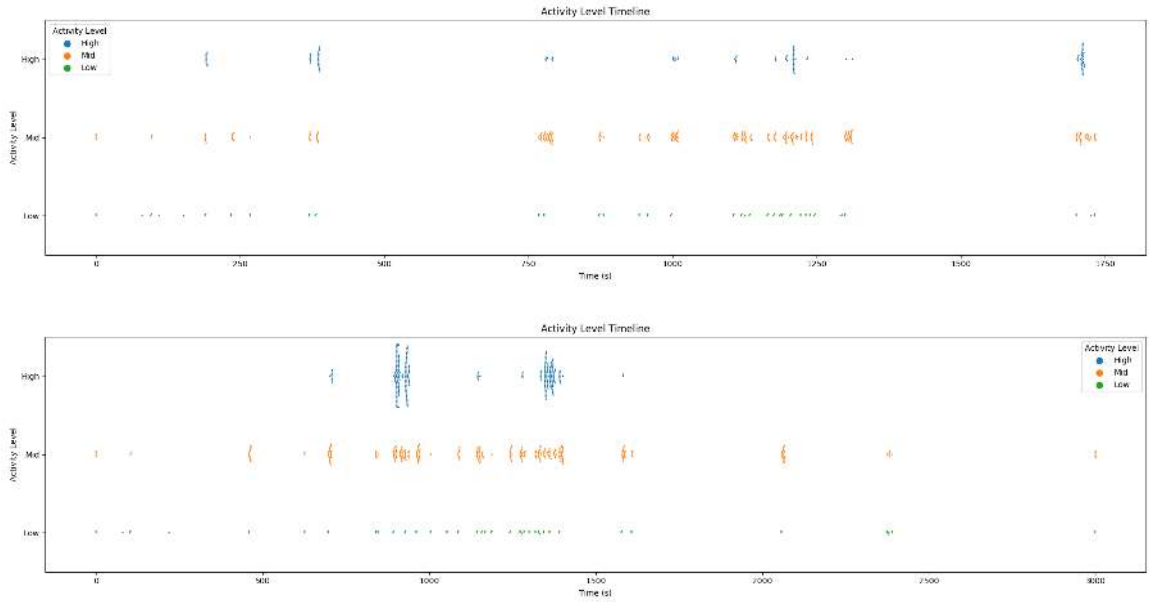


Figure 3.3: Activity Levels for Participants 893

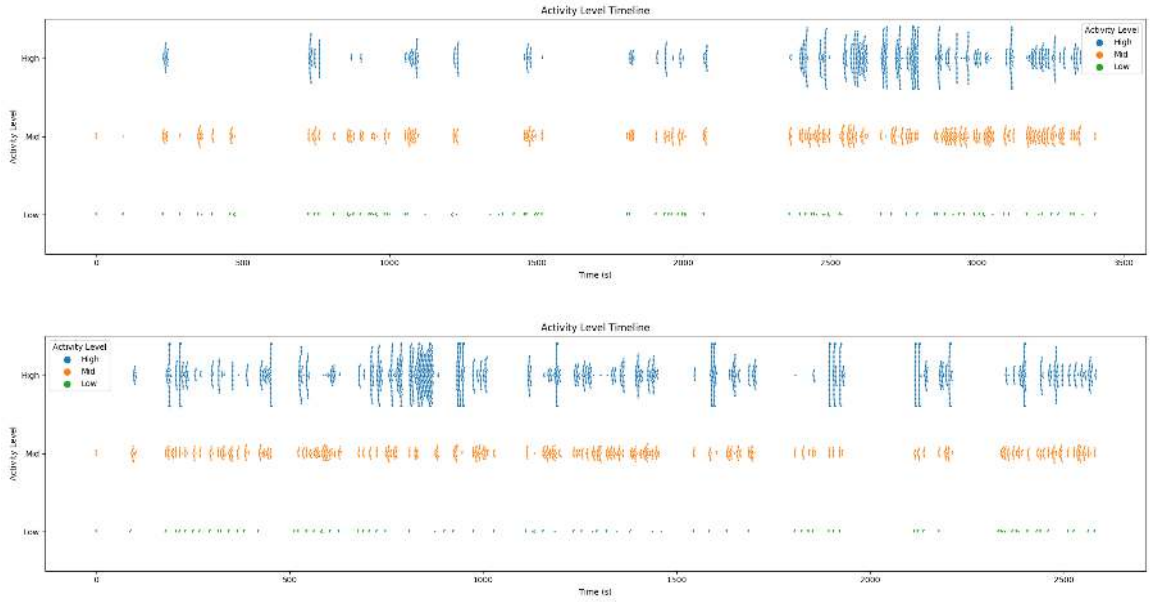


Figure 3.4: Activity Levels for Participants DC9

3.5.4 Filtering

Finally, not all participants provided equal amounts of data. Some participants only typed a handful of commands into the terminal and some generous participants provided dozens of hours of data.

Before performing feature extraction, a data set is specified as either guided, free, or both. If a packet flow does not have a matching tag, it is removed from the dataset before feature extraction.

To prevent the algorithms from developing a bias towards users who provided more data, each users sample set is truncated to the lowest of all the participants.

To prevent the users who provided insufficient data to train from limiting the sample sets, we set a minimum number of samples required for a participant to be included in the training and testing.

And ultimately, because the researcher may unwittingly choose a set of preprocessing configurations which restricts the qualifying participants to an unreasonably small set, the number of participants remaining in the sample is checked against a minimum threshold. If the number of participants is below the threshold, the extractor throws an error and not sample set file is generated.

3.6 Machine Learning

The samples had their features extracted and tagged, they were fed into our machine learning algorithms. In this study we looked at four in particular: k-nearest neighbors, random forest, support vector machine, and a perceptron network.

Each of these models randomly shuffle the data before training and testing. Additionally, all results of each run of any of the ML algorithms are computed using the average of a 5-fold validation.

All experiments were run using python version 3.6.5. For the K-NN, RF, and SVM classifiers, scikit-learn version 0.18 was used. For the Percreptron network Keras version 2.1.3 and Tenserflow version 1.5.0 were used.

All of these models were developed with the assistance of Ethan Goldfarb.

3.6.1 Hypertuning

Each Machine Learning algorithm has several hyper-parameters which determine how the algorithm runs. Which configuration is best depends on the type of data being classified.

For each combination of classifier and pre-processing parameters, some features do not provide any useful information. These features can prevent the algorithms from training properly without sufficient data. Since we have a very limited data set acquired from volunteers, we must prune the features which do not add significant benefit to classification.

Because some algorithms train slowly and the possible combinations of features is so great, testing every combination is not possible. For RF and SVM, we instead use principal component analysis to create a new set of input features which retains 98% of the information of the original feature set. For K-NN, pruning the feature set is achievable, so we find the optimal feature set and compare its results to the PCA feature set. Because NNs are resistant to a large feature set, we do not do any pruning or PCA [40].

The optimal set of pre-processing parameters can vary based on the algorithm and chosen features, so these will be hypertuned as well. Since most of these are scalar values, a set of reasonable values are chosen to start for each parameter. Once hypertuning converges on an value or set of values which out-performs the others, a second round of hypertuning determines the optimal values with better granularity around the previous optimal values.

And finally, each algorithm has its own unique parameters listed in Table 3.3.

Algorithm	Parameter	Function
K-Nearest Neighbors	K	Number of neighbor who get a vote [23]
	Weight Fn	How each neighbors vote is weighted [23]
Random Forest	Criterion	What measure to use to determine the best split for a node [37]
	Features/Node	Number of features to consider in a decision nodes [37]
	Samples/Leaf	Number of samples required to make a leaf node [37]
Support Vector Machine	Kernel Fn	Function to translate from input space to feature space [23, 37]
	C	Regulation parameter [37]
Perceptron Network	Activation Fn	Determines whether or not the inputs are enough to fire
	Hidden Layers	Number of hidden perceptron layers
	Epochs	Number of training rounds before testing

Table 3.3: Hypertuning Parameters for each Algorithm

Because a the number of combinations of hyperparameters is very large, some configurations are likely achieve above average performance by random chance even with 5-fold validation. This can create misleading results where certain configurations seem to perform better than they reliably can. To account for this, each configuration is run 7 times and the median accuracy is used as the expected accuracy of this configuration for the purposes of hypertuning.

To efficiently test the large combinations of hyper-parameters, hypertuning jobs were distributed to a set of 672 computers by a system of Secure Shell connection. Each slave computer, once the connection from the master to the slave is established the masters sends a job. The slave computes the job, stores the result, and asks for a new job off the master queue. The master servers new jobs on a first-come-first-serve basis. Each slave saves the top 100 computation results to conserve space by not storing sub-optimal results. Because a slave is only assigned a task when it asks for one, failed connections to slaves will never receive a job. This system has no failure recovery, so any data lost to an unexpected failure is not tracked. Because jobs are assigned first-come-first-serve, the jobs assigned to each computer have no distinct pattern and each slave has a representative sample of the jobs. The loss of a subset of the data will therefore not distort the distribution of the results.

In a typical hypertuning run, approximately 115 computers failed to authenticate with the SSH server for unknown reasons, 30 computers do not find the host, and around 5-30 computers fail unexpectedly after having begun to collect data depending on how long the run takes. This means that each run only 1% to 6% of runs are lost because of this.

After all this, the top results are aggregated and run 50 times and averaged to get the most accurate assessment of each configuration's performance for comparison. In all of our results, this allowed us to create a 95% confidence interval with less than $\pm 1\%$ margin for error.

3.6.2 Analysis

To calculate the accuracy of a run, we run the ML algorithms 100 times and average the accuracy to create a final accuracy score. In all of our results, this allowed us to

create a 95% confidence interval with less than $\pm 0.5\%$ margin for error. Additionally, we calculate the p-value of each accuracy using a one-sided t-test against the null hypothesis the the system is guessing randomly. Finally, we compute the F_1 score to determine the predictive power of the system.

Chapter 4

RESULTS

In this chapter, we review the collected data and the results from the hypertuning. Each classifier was hypertuned in a number of rounds, each narrowing down the parameter range until an optimal value is found. Finally, we look at how these algorithms perform under different conditions and compare their accuracy and power.

4.1 Dataset

Because each participant was allowed to participate in either or both datasets and some participants volunteered more data than others, we have an uneven distribution of data. The total distribution can be seen in Table 4.1. The summary statistics for each dataset in the table exclude any participant who did not submit data for that dataset. With the guided dataset, we see the an approximately normal distribution with an average of 2509 packets and a mean of 2463 packets and a range of 2000 packets per user. The free dataset had a much wider range, with some participants providing very little data (approximately 500 packets) while so provided over 9000 packets. On average, we collected approximately 50% more data from the free dataset than the guided dataset, with a heavy right skew.

Id Hash	Guided	Free	Id Hash	Guided	Free
01b	0	3127	749	0	830
192	0	8041	893	0	1042
1aa	0	1573	949	1856	3860
26d	0	5832	ab2	0	3181
2b7	0	4506	aff	2716	5377
2e2	2354	0	b14	2149	2686
31f	1549	0	bcf	0	987
3c7	3771	0	bf0	0	442
4ee	2205	0	c1b	2463	2682
4f8	2977	0	c71	0	701
599	0	9124	c7f	2701	1197
5a6	0	3804	cb5	2606	1928
5b4	0	1583	d77	0	4498
67e	2251	7673	dc9	0	6610
69f	0	5502	e77	2826	0
6df	0	7880	e93	2166	0
min	1549	442	average	2509.33	3716.77
max	3771	9124	median	2463	3154

Table 4.1: Amount of data per participant per dataset

4.2 Hypertuning

For the initial round of hypertuning, a range of values was chosen for each of the preprocesing parameters and ML algorithms. These ranges can be seen in Table 4.2. If our initial hypertuning shows the performance trends upward beyond the boundaries of the chosen ranges, we expand the range in the next round. Activity thresholds were selected according to the average typing speed on a computer (approximately 3.4P/s). Going significantly above this threshold would not be useful as users would not trigger it. Increments of 0.5 P/s were chosen as a compromise between granularity and having too many hypertuning parameters. Further hypertuning rounds can increase granularity on promising ranges of values.

Lookbacks were chosen to potentially include the entire range of values for high-activity threshold.

Paste-size thresholds were chosen based on the average size of packets observed in the data.

The number of users was restricted to 20 to keep the comparisons fair. Any parameter combinations which did not allow for 20 users were discarded.

For the first round of hypertuning for the K-NN classifier, the preprocessor and feature-set were hypertuned together and the other features were delayed for later rounds.

Because the other algorithms did not have time to hypertune the featureset, their hyperparameters were tuned with the preprocessor. For example, 100 runs of the Random Forest classifier takes approximately 46 times longer than 100 runs of the Nearest Neighbor classifier similar configurations. The K-NN classifier took 1.5 months to run, which implies a similar round of hypertuning would require 5.5 years.

Parameter	Values	Unit
Preprocessing		
Dataset	Free, Guided, Both	N/A
Sample Size	50, 100, 150, 200, 300	Packets
L.-Activity Threshold	0.5, 1, 1.5, 2	Packets/Second
H.-Activity Threshold	1.5, 2, 2.5, 3, 3.5, 4	Packets/Second
Lookback	1, 2, 3, 4, 5	Seconds
S.-Paste Threshold	1, 2, 3, 4, 5	Blocks
L.-Paste Threshold	2, 3, 4, 5, 6	Blocks
K-Nearest Neighbors		
k	1..20	N/A
Weight	Uniform, Distance	N/A
Featureset	$\rho(\{\text{features}\})$	N/A
K-Nearest Neighbors PCA		
k	1, 3, 5, 7, 9, 11	N/A
Weight	Uniform, Distance	N/A
PCA Kernel	linear, poly, rbf, sigmoid, cosine	N/A
Random Forest		
Criterion	Gini, Entropy	N/A
Estimator	50, 100, 150, 200	RDTs
Min-Samples/Leaf	1, 5, 10	Samples
Max-Features/Node	1, 3, None	Features
PCA Kernel	linear, poly, rbf, sigmoid, cosine	N/A
Support Vector Machine		
C	1/8, 1/4, 1/2, 1, 2, 4, 8	N/A
SVM Kernel	Linear, Poly, RBF, Sigmoid, Precomputed	N/A
PCA Kernel	linear, poly, rbf, sigmoid, cosine	N/A
Perceptron		

Table 4.2: Range of values for hyper parameters in first round of hyper-tuning

4.2.1 K-Nearest Neighbors

We begin with the K-NN classifier because it is fast and has shown to be success in prior studies with similar goals [22]. This allows for more robust testing and may provide insight into the expected performance of the other classifiers.

4.2.1.1 Round 1

After hypertuning the K-Nearest Neighbors for around 6 weeks, 492 computers returned results giving 49,200 of the top performing configurations. Of these, the top 10 across all systems are shown in Table 4.3.

Accuracy	Sample Size	Low-Act T.	High-Act T.	Lookback	Small-Paste T.	Large-Paste T.	Features
0.2409	150	1.0	2.00	2.00	1	2	AVG_PASTE_SIZE, AVG_SMALL_PASTE_SIZE, HIGH.AVERAGE_IAT, LOW.BURST_COUNT, MID.BURST_COUNT
0.2405	150	0.5	2.50	2.00	1	5	HIGH.TIME_SPENT, LARGE_PASTES, MID.AVERAGE_IAT, MID.BURST_COUNT, TOTAL_PASTES
0.2397	150	1.5	3.50	2.00	1	2	HIGH.TIME_SPENT, LARGE_PASTES, LOW.BURST_COUNT, MID.BURST_COUNT, SMALL_PASTES
0.2395	150	0.5	2.50	2.00	1	2	HIGH.TIME_SPENT, LARGE_PASTES, MID.AVERAGE_IAT, MID.BURST_COUNT, TOTAL_PASTES
0.2373	150	1.0	2.50	2.00	3	6	AVERAGE_IAT, HIGH.TIME_SPENT, LARGE_PASTES, MID.BURST_COUNT, MID.TIME_SPENT
0.2373	150	1.0	2.50	2.00	1	3	AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, MID.AVERAGE_IAT, MID.TIME_SPENT
0.2371	150	1.0	4.00	2.00	1	6	HIGH.BURST_COUNT, HIGH.TIME_SPENT, LOW.BURST_COUNT, MID.BURST_COUNT, TOTAL_PASTES
0.2366	150	1.5	3.50	2.00	1	3	AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, LOW.BURST_COUNT, MID.AVERAGE_IAT, MID.BURST_COUNT
0.2365	150	2.0	3.50	3.00	1	3	AVG_SMALL_PASTE_SIZE, HIGH.AVERAGE_IAT, MID.BURST_COUNT, SMALL_PASTES, TOTAL_PASTES
0.2363	150	1.5	3.50	2.00	1	4	HIGH.TIME_SPENT, LOW.BURST_COUNT, MID.AVERAGE_IAT, MID.BURST_COUNT, SMALL_PASTES

Table 4.3: Top 10 Parameter sets for K-NN, Round 1 of hypertuning. Features are defined in Table 3.2

These results suggests a sample size of 150 packets performs well with the configuration ranged we have tested. Additionally, lower activity thresholds (refer to Table 3.1 for definitions) seem to be most effective when at or below 1packet/second and high activity threshold at around 3P/s. The Lookback seems to be best somewhere between 2s and 3s. The small paste size seems to be best at 1 block. Large paste does not seem to converge from this data.

It's interesting to note that from the features, the large pastes seem to be significantly less useful than the small paste. The average small paste size and time spent in a high activity state seem to be the most useful features, combined with support from other features from low and mid activity states. Since the high-activity threshold is set just under or above the average typing speed in our top-performing results (discussed in more detail later), the time spent in high activity feature effectively measures when a student would be actively typing. We suspect this feature contrasts participants who spend more time writing code, an activity which largely involves typing, with participants who spend more time debugging, and activity which involves little typing and more examination or other similar activities. Since the large paste statistics are not performing well, it's possible that something is causing a large packets that are not useful for identification (perhaps an overhead type of packet which was mistakenly identified as a keystroke). In this case, the small paste category would serve as a proxy for pastes as a whole. Also of note, only the combined datasets made it into the top results. We look into this more in 4.4.

A graph of the total occurrences of each feature count in the top 49,200 results can be see in Figure 4.1. Interestingly this graph confirms the high activity time is the occurred most often in the top results, but average small paste size did not have the same presence. If we assume that each feature has equal predictive power, we would expect each feature to be equally represented in the top results. This over-

representation of the this feature suggests it adds significantly more predictive power than the others, causing the featuresets which included it to on average perform better than those that didn't. This implies that some of the features might have factors that combine with other features in non-linear ways.

Also interesting is that after high-activity time, the success of each feature trails off until it plateaus around average mid-activity burst size. For round 2 of hypertuning, any feature at or below the level of AVG_PASTE_SIZE is removed from the feature set because they did not show with the exception of MID.AVG_BURST_SIZE and MID.AVERAGE_IAT because of their frequent presence in the top performing configurations.

This excludes DEAD_TIME, TOTAL_TIME, AVG_LARGE_PASTE_SIZE,LOW.AVERAGE_IAT, LOW.TOTAL_PACKETS, LOW.AVG_BURST_SIZE, HIGH.AVG_BURST_SIZE, HIGH.TOTAL_PACKETS, LOW.TIME_SPENT, MID.AVG_BURST_SIZE, MID.AVERAGE_IAT, and AVG_PASTE_SIZE.

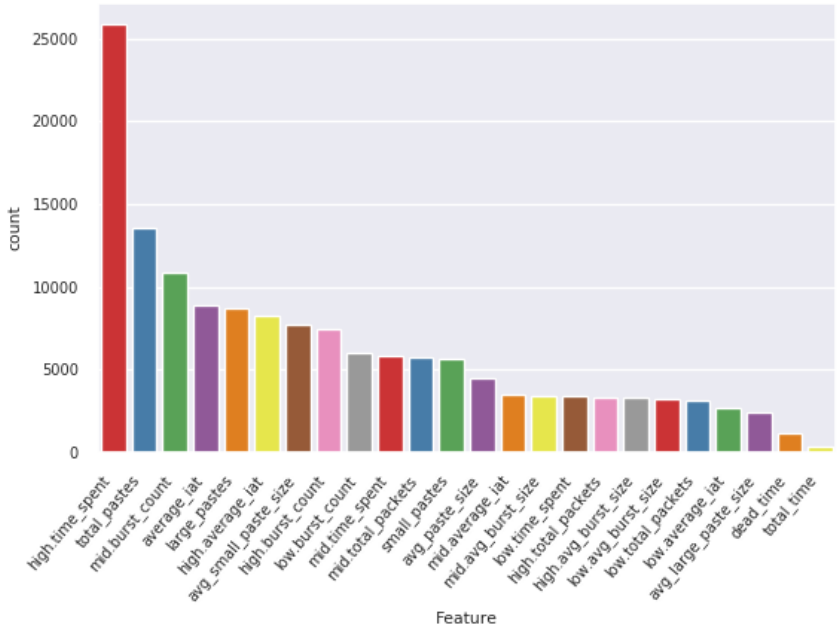


Figure 4.1: Occurrences of each feature in the top performing feature sets in K-NN hypertuning, Round 1

Figure 4.2 shows the relationship between each pair of features. Unfortunately, because high-activity time spent dominates so significantly, this does not provide significant insight.

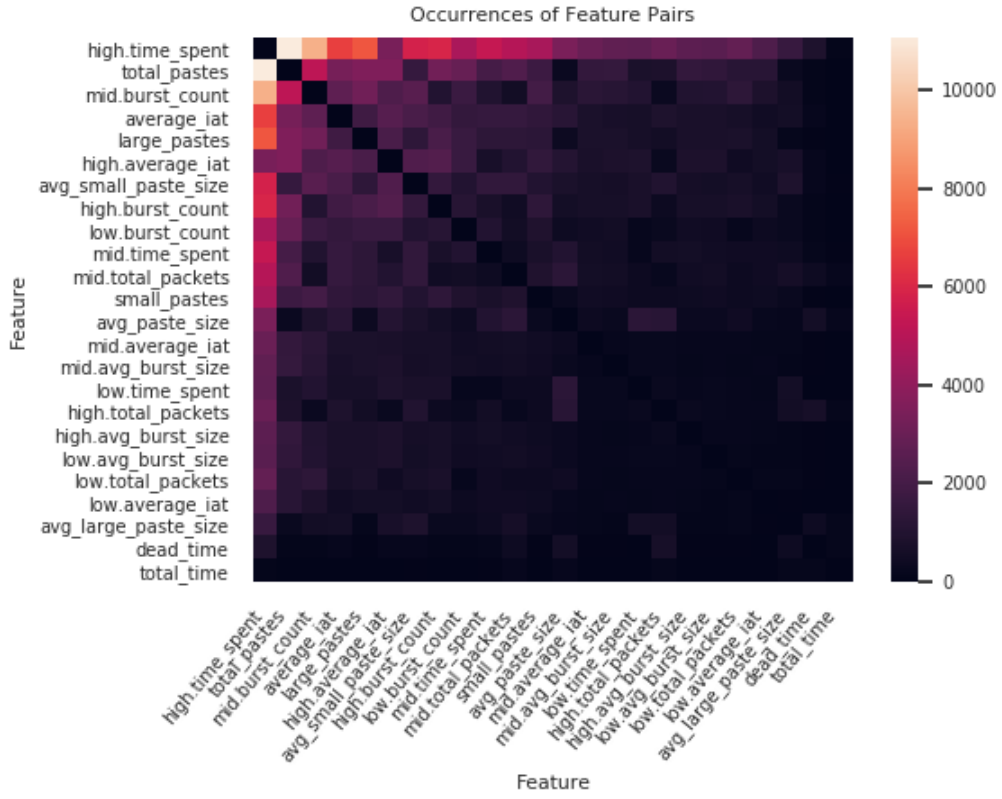


Figure 4.2: Occurance of each feature pair in the top performing feature sets in K-NN hypertuning, Round 1

The Figure 4.3 shows the number of occurrences of each sample size in the top results. This supports the earlier results from the top-10 configurations.

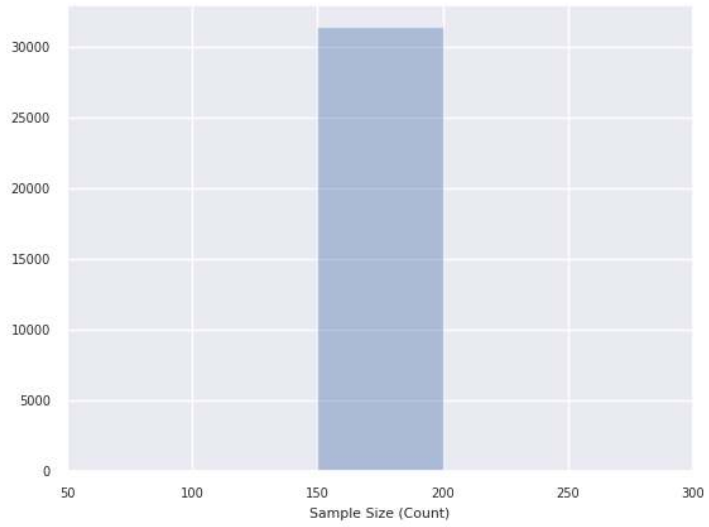


Figure 4.3: Occurrences of each sample size value in the top performing feature sets in K-NN hypertuning, Round 1

The number of occurrences of each low-activity threshold are graphed in Figure 4.4. Judging by this graph, the optimal value would likely be below or at 0.5P/s.

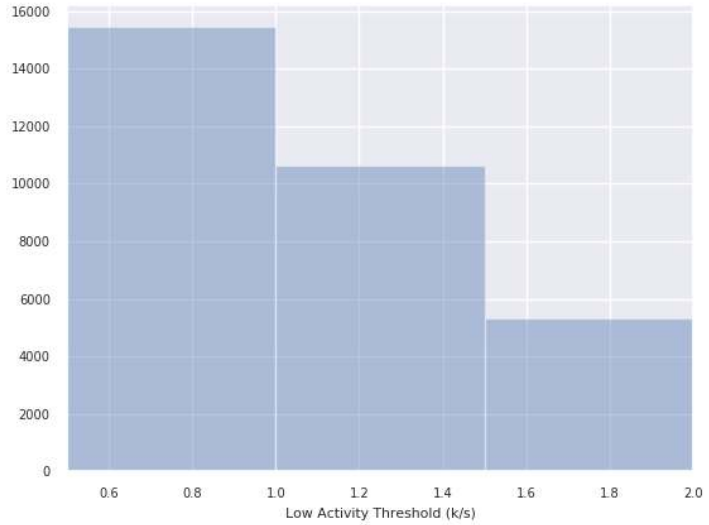


Figure 4.4: Occurrences of each low-activity threshold value in the top performing feature sets in K-NN hypertuning, Round 1

Figure 4.5 shows the occurrences of each high-activity threshold. While 2.5P/s seems to be the optimal value, the graph continues to trend upward past the range we examined. In the next test, we will add more resolution to the 2.5-3P/s range and test values higher than 4.5.

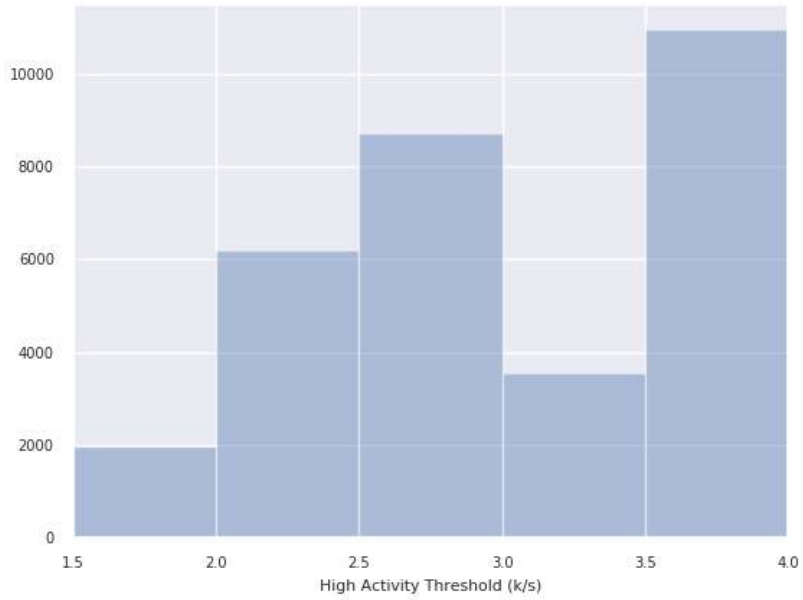


Figure 4.5: Occurrences of each high activity theshold value in the top performing feature sets in K-NN hypertuning, Round 1

Figure 4.6 shows the occurrences of each lookback range. It seems the optimal value for this field lies around 2s. We test with more resolution in round 2.

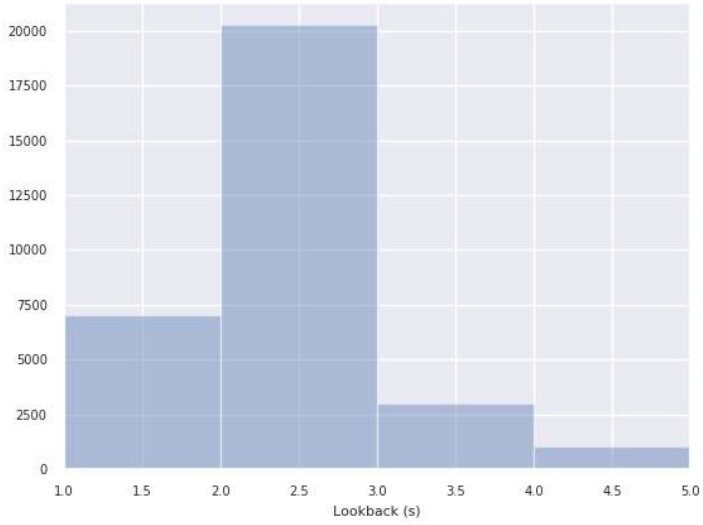


Figure 4.6: Occurrences of each lookback value in the top performing feature sets in K-NN hypertuning, Round 1

Figure 4.7 shows the occurrences of each small-paste threshold. Since the value must be a natural number, 1 block is the confirmed optimal value.

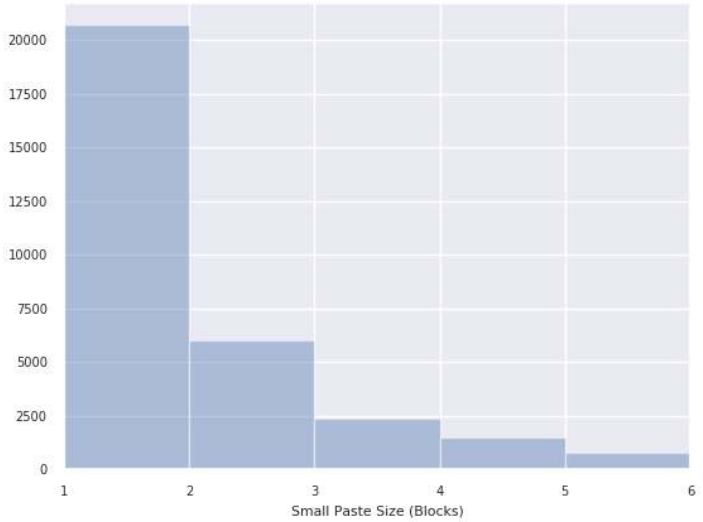


Figure 4.7: Occurrences of each small paste threshold value in the top performing feature sets in K-NN hypertuning, Round 1

Figure 4.8 shows the occurrences of each large-paste threshold. Since the graph still trails upward past the tested range, we continue extending this range in round 2.

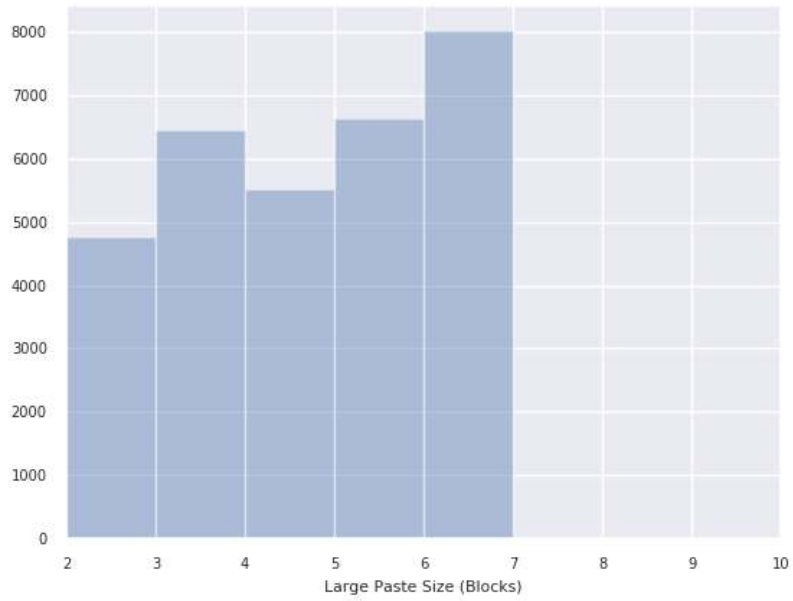


Figure 4.8: Occurrences of each large paste threshold value in the top performing feature sets in K-NN hypertuning, Round 1

4.2.1.2 Round 2

Based on the results from the previous round of hypertuning, we developed a new set of parameters to test, listed in Table 4.4.

Parameter	Values	Unit
Low-Activity Threshold	0.5, 1, 1.5	Packets/Second
High-Activity Threshold	2.5, 2.75, 4, 5, 6	Packets/Second
Lookback	1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75	Seconds
Large-Paste Threshold	6, 7, 8, 9	Blocks
Featureset	$\rho(\{\text{features}\}\backslash\{$DEAD_TIME, TOTAL_TIME, AVG_LARGE_PASTE_SIZE,LOW.AVERAGE_IAT, LOW.TOTAL_PACKETS, LOW.AVG_BURST_SIZE, HIGH.AVG_BURST_SIZE, HIGH.TOTAL_PACKETS, LOW.TIME_SPENT, MID.AVG_BURST_SIZE, MID.AVERAGE_IAT, AVG_PASTE_SIZE$\})$	N/A

Table 4.4: Range of values for hyper parameters in second round of hypertuning

This round of hypertuning ran in 1.5 hours on 400 computers, producing 40,000 top configurations. The top 10 results are listed in Table 4.5. From this table we can see a marginal improvement as well as some values converging.

Accuracy	Sample Size	Low-Act T.	High-Act T.	Lookback	Small-Paste T.	Large-Paste T.
0.2604	150	1.0	2.50	1.75	1	7
0.2593	150	1.0	2.50	1.75	1	7
0.2591	150	1.0	2.75	1.50	1	7
0.2578	150	1.0	2.50	1.75	1	9
0.2577	150	1.5	2.50	1.75	1	8
0.2575	150	1.0	2.75	1.50	1	7
0.2574	150	1.5	2.50	1.75	1	7
0.2567	150	1.5	2.75	1.75	1	7
0.2566	150	1.0	2.50	1.75	1	9
0.2564	150	1.5	2.50	1.75	1	9

Features
AVG_SMALL_PASTE_SIZE, HIGH.BURST_COUNT, HIGH.TIME_SPENT, LOW.BURST_COUNT, MID.TOTAL_PACKETS
AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, LOW.BURST_COUNT, MID.TIME_SPENT, MID.TOTAL_PACKETS
AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, LOW.BURST_COUNT, MID.TIME_SPENT, MID.TOTAL_PACKETS
AVERAGE_IAT, AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, MID.TIME_SPENT, SMALL_PASTES
AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, MID.AVG_BURST_SIZE, MID.BURST_COUNT, SMALL_PASTES
AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, LOW.BURST_COUNT, MID.AVERAGE_IAT, MID.TIME_SPENT
AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, MID.AVERAGE_IAT, MID.BURST_COUNT, MID.TIME_SPENT
AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, MID.AVERAGE_IAT, MID.BURST_COUNT, MID.TIME_SPENT
AVERAGE_IAT, AVG_SMALL_PASTE_SIZE, HIGH.AVERAGE_IAT, HIGH.TIME_SPENT, MID.TIME_SPENT, SMALL_PASTES
AVG_SMALL_PASTE_SIZE, HIGH.TIME_SPENT, MID.AVERAGE_IAT, MID.BURST_COUNT, SMALL_PASTES

Table 4.5: Top 10 Parameter sets for K-NN, Round 2 of hypertuning.

Figure 4.9 shows the occurrences of each feature in the top results. HIGH.TIME_SPENT lost some of its significant lead, and a few features shifted order. Some of this may be attributable to HIGH.TIME_SPENT and AVG_SMALL_PASTE_SIZE doing enough on there own to include results from featuresets composed of otherwise ineffective features. With the ineffective features removed, the graph becomes more balanced.

Also of note, not all of the top performing features were well representing in the top results. For example, AVERAGE_IAT only occurred twice in the top 10 configurations despite its dominance in the larger dataset. This implies some degree of redundancy between features.

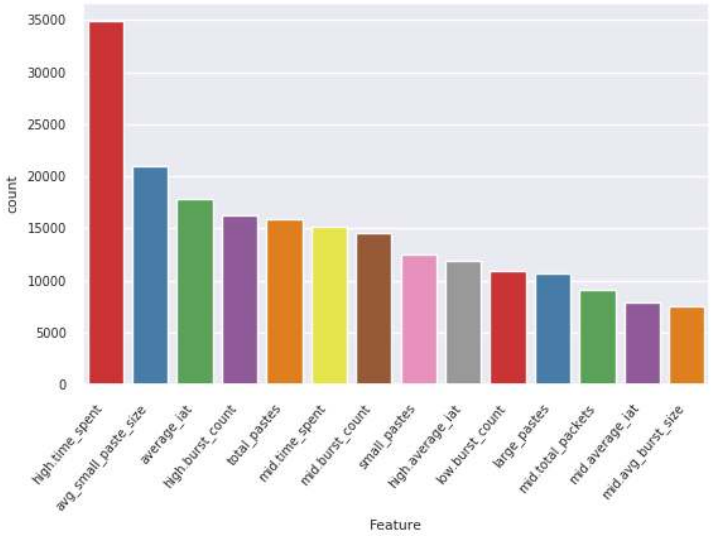


Figure 4.9: Occurrences of each feature in the top performing feature sets in K-NN hypertuning, Round 2

Figure 4.10 shows the relationship between each pair of features. It does not seem like any two features combine in an unpredictable way.

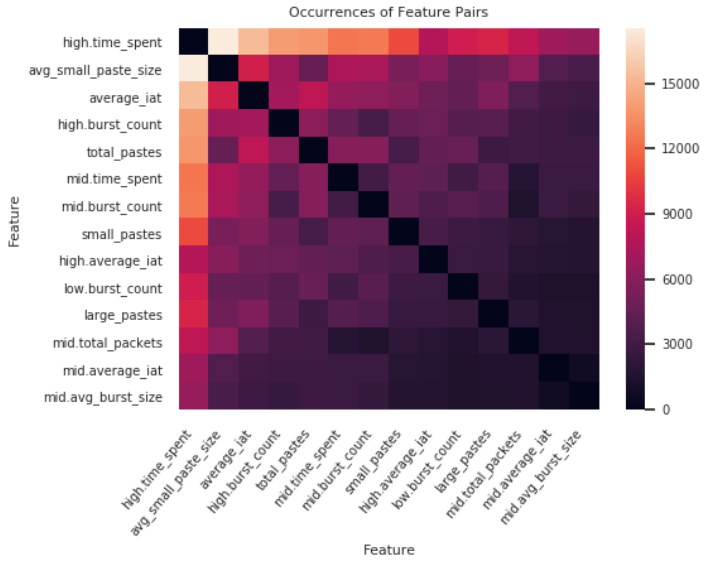


Figure 4.10: Occurrence of each feature pair in the top performing feature sets in K-NN hypertuning, Round 2

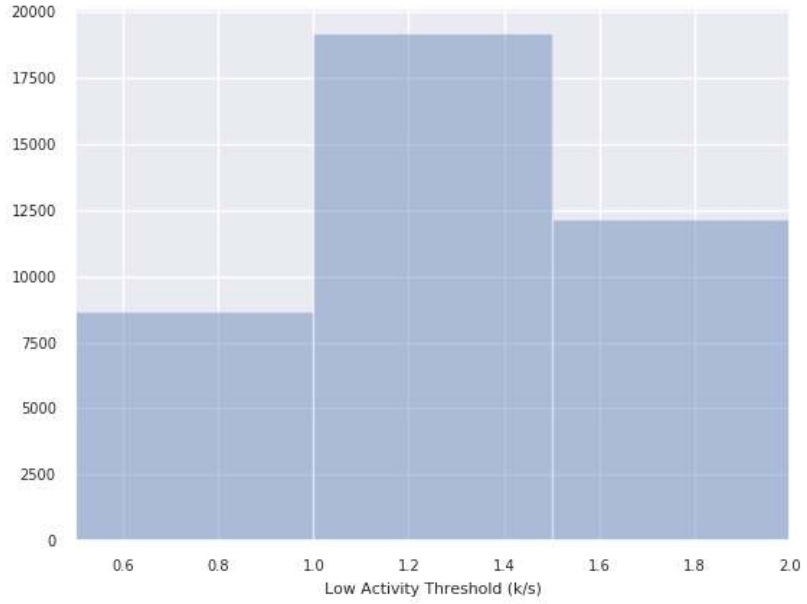


Figure 4.11: Occurrences of each low-activity threshold value in the top performing feature sets in K-NN hypertuning, Round 2

The number of occurrences of each low-activity threshold are graphed in Figure 4.11. Judging by this graph and the results from the top-10 configurations, 1P/s seems to be the best option.

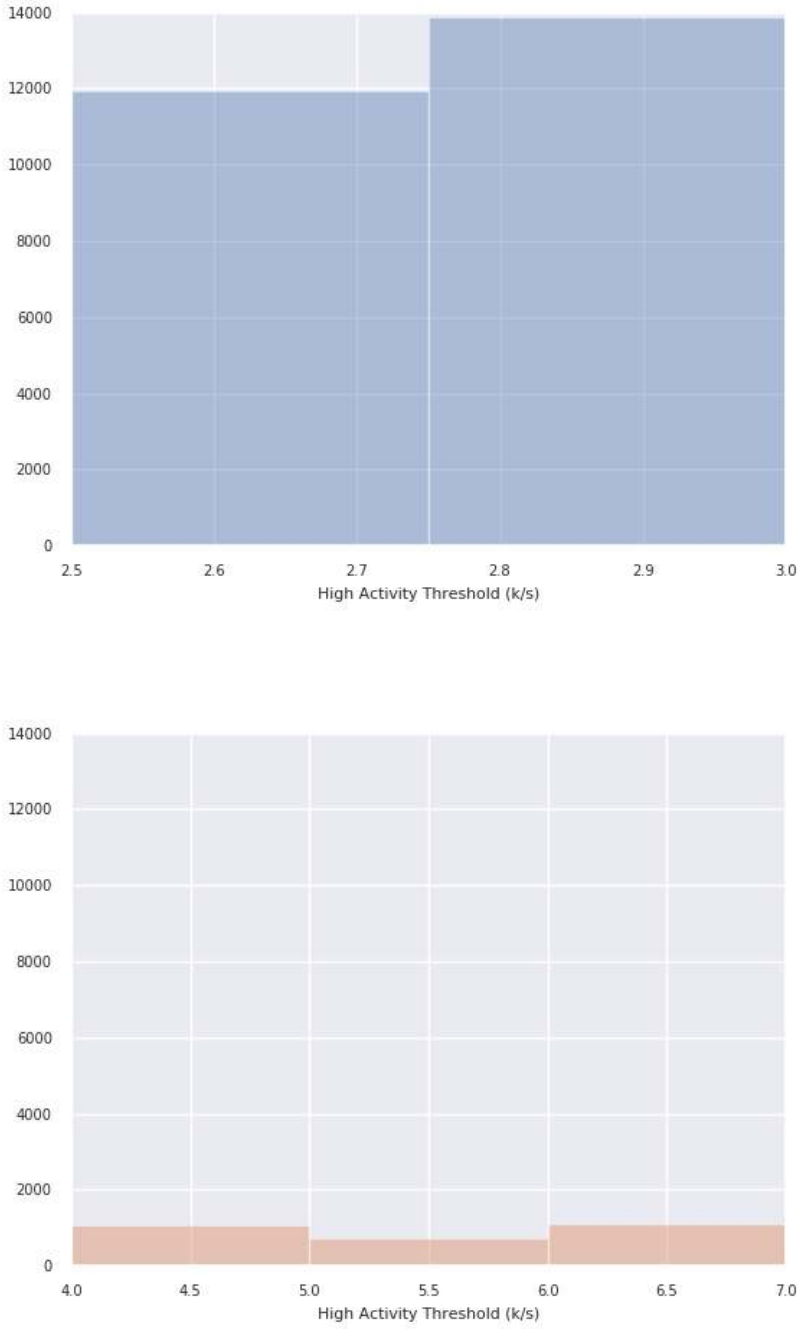


Figure 4.12: Occurrences of each high-activity threshold value in the top performing feature sets in K-NN hypertuning, Round 2

The number of occurrences of each low-activity threshold are graphed in Figure 4.12. From here we can see the spike in values from 3-4P/s has disappeared and 2.75P/s dominates. This seems interesting that there was such a strong effect in the last round that is non-existent here. We speculate that some of HIGH.AVG_BURST_SIZE, HIGH.TOTAL_PACKETS, LOW.TIME_SPENT, MID.AVG_BURST_SIZE, or MID.AVERAGE_IAT combined with a high threshold for high activity gave interesting results. When these features were removed from the featureset, the power of the higher threshold disappeared. To examine this, we listed the features which were most prevalent in Round 1 of hypertuning when the high-activity threshold was set to 4.0, see in Figure 4.13. Unfortunately, none of the suspect features occurred very prominently. MID.AVG_BURST_SIZE is the first to appear in the list, but even it shows little more power than half the other features. It's possible some combination of variables helped to produce this result, but further analysis would require time beyond what was available.

While 2.75s and 2.5s did comparably in this graph with 2.75s having a slight lead, this was contradicted by the top-10 configurations which show a similar bias towards 2.5s. It also seems that 2.5s high activity thresholds often paired with 1.75s lookback periods in the top results, while 2.75s thresholds paired with 1.5s lookback periods. Since both lowering the lookback period and increasing the activity thresholds have a similar effect of stabilizing the activity state, it seems one of these two pairs is the best chosen together. Since Figure 4.15 shows that a lookback period of 1.75s out-performs 1.5s more than a high activity threshold out performs a high activity threshold od 2.75s a threshold of 2.5s, this suggests the lookback and high activity threshold pair of 2.75 and 1.5 is the best option This is also supported by the top-10 configurations, of which this pairing composed 70%.

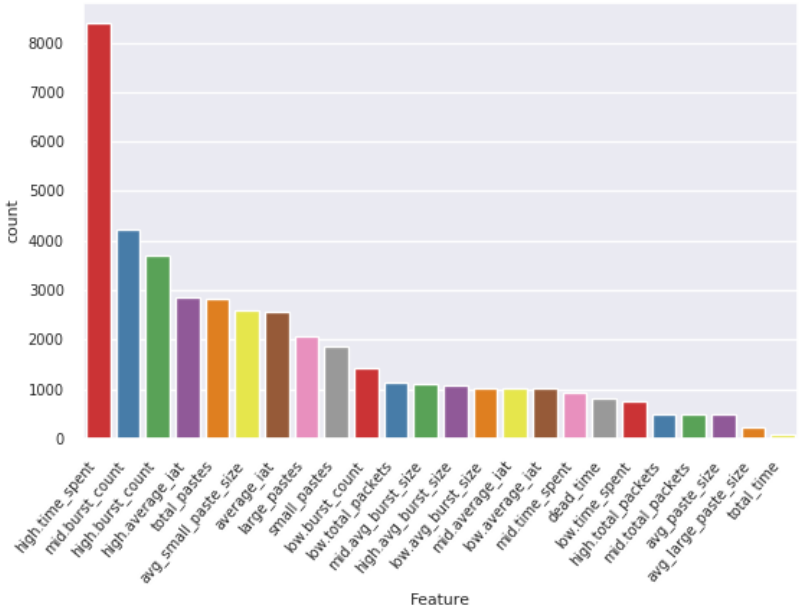


Figure 4.13: Occurrences of each feature in the top performing feature sets in K-NN hypertuning, Round 1 when High-Activity threshold is 4P/s

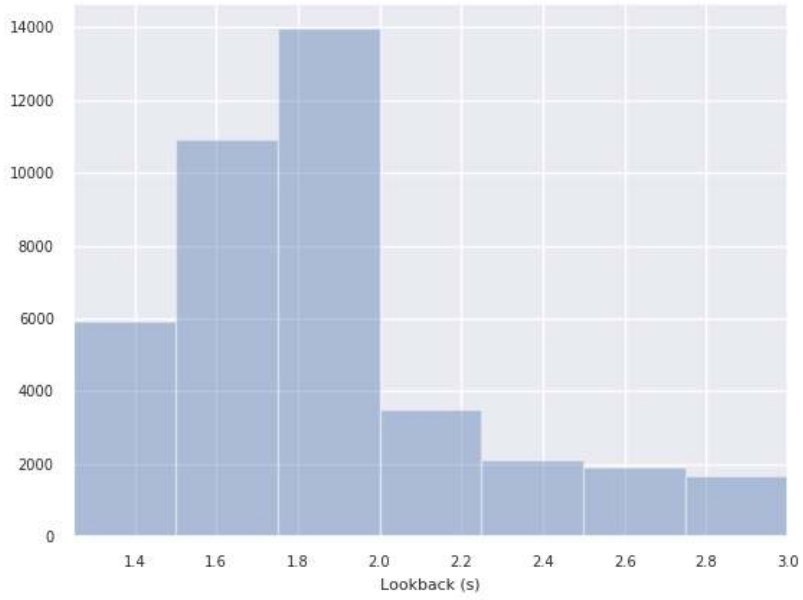


Figure 4.14: Occurrences of each lookback value in the top performing feature sets in K-NN hypertuning, Round 2

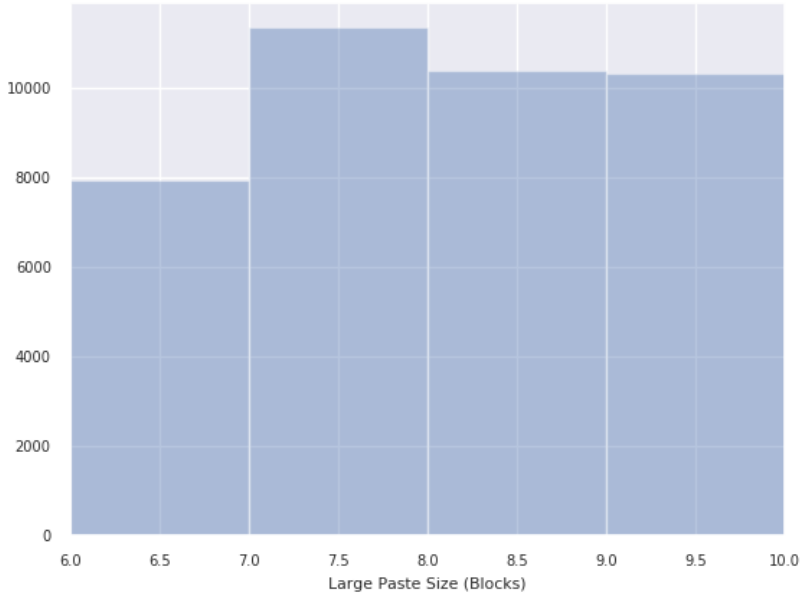


Figure 4.15: Occurrences of each large-paste threshold value in the top performing feature sets in K-NN hypertuning, Round 2

The number of occurrences of each large-paste threshold are graphed in Figure 4.15. Here we see that all the values performed roughly equally, with 7 blocks having a slight lead.

Since all of the top performing preprocessing parameters match that of the top performing configuration, this will be the one we use. This does exclude some of the higher performing features, but their lack of representation in the top performing configuration implies that the information is superfluous to the other features already includes. The final configuration for the preprocessor and features for the K-NN algorithm is show on Table 4.6.

Parameter	Value	Unit
Dataset	Both	N/A
Sample Size	150	Packets
Low-Activity Threshold	1	Packets/Second
High-Activity Threshold	2.5	Packets/Second
Lookback	1.75	Seconds
Small-Paste Threshold	1	Blocks
Large-Paste Threshold	7	Blocks
Featureset	{AVG_SMALL_PASTE_SIZE, HIGH.BURST_COUNT, HIGH.TIME_SPENT, LOW.BURST_COUNT, MID.TOTAL_PACKETS}	N/A

Table 4.6: Values for hyper parameters determined by the second round of hypertuning the K-NN

4.2.1.3 Final Round

With the preprocessing parameters thoroughly tested and a configuration decided, we more to the K-NN specific parameters, namely K and the weight function.

We test each combination of the values in Table 4.7 by running them 100-times and graphing the mean with a 95% confidence interval.

Parameter	Values	Unit
Weight	Distance, Uniform	N/A
K	$[1, 25]$	N/A

Table 4.7: Range of values for hyper parameters For the last round of K-NN

Figure 4.16 shows all the combinations with their results. From this graph, we can see that uniform and distance functions perform equally up until $K = 6$. Afterwards the uniform function drops while distance plateaus. This implies that after 6, the additional nodes do not provide value and that the distance function maintains accuracy by weighting those further nodes lower. Therefore, it does not matter which function

is used so long as $K = 6$, the optimal point for both functions. Since uniform weights is easier to compute, we use this for the rest of the paper.

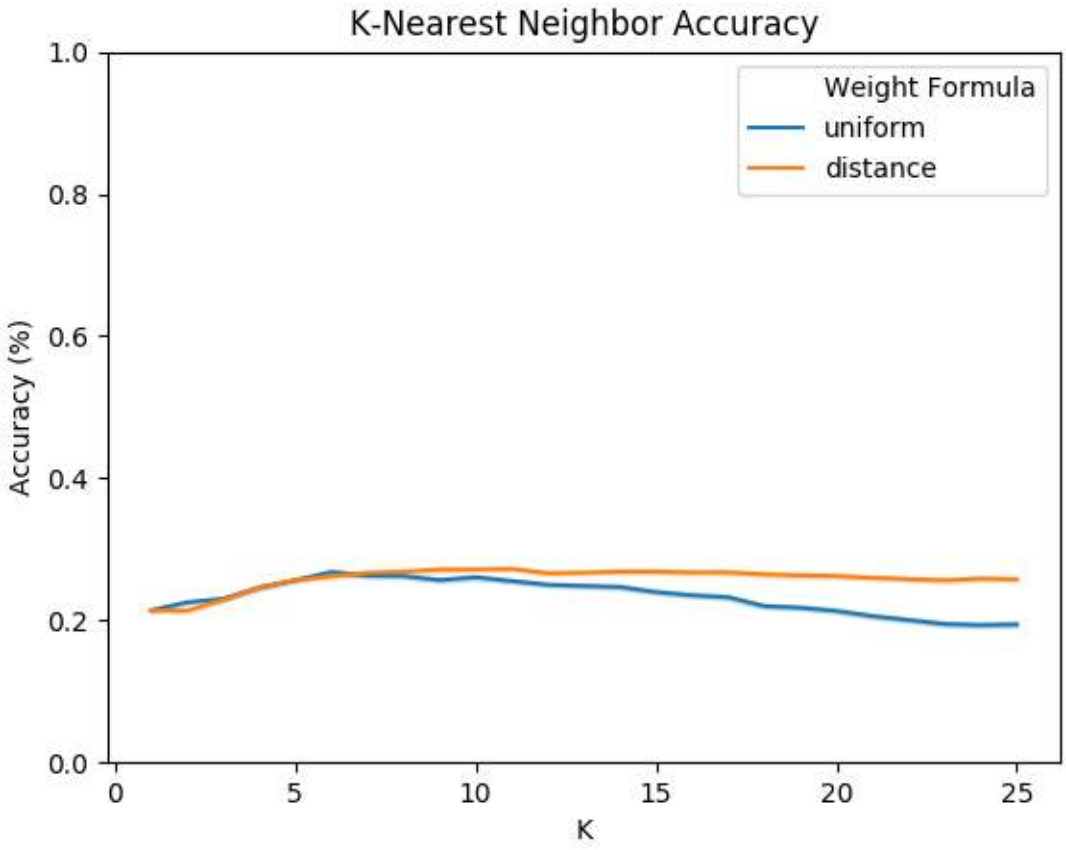


Figure 4.16: Graph of all possible K and weight functions with the optimal preprocessing parameters.

4.2.2 K-Nearest Neighbors PCA

Next, we hypertune the K-NN algorithm with PCA for comparison. Because PCA reduces the feature space, we can afford to hypertune the preprocessing parameters with the k and weight functions.

4.2.2.1 Round 1

Hypertuning the K-Nearest Neighbors with PCA took a about 2 hours, with 649 computers returning results giving 64,900 of the top performing configurations. Of these, the top 10 across all systems are shown in Table 4.8.

Accuracy	Sample S.	Low-Act	High-Act	Lookback	Small-Paste	Large-Paste	PCA Kernel	K	Weight Fn
= 0.2636	200	0.50	3.50	1.00	1	3	linear	11	distance
0.2636	200	0.50	3.50	1.00	1	2	linear	9	distance
0.2591	200	0.50	4.00	1.00	1	3	linear	9	uniform
0.2591	200	0.50	4.00	1.00	1	3	linear	9	distance
0.2591	200	0.50	4.00	1.00	1	3	linear	11	distance
0.2591	200	0.50	4.00	1.00	1	2	linear	11	distance
0.2591	200	0.50	3.50	1.00	1	4	linear	11	uniform
0.2591	200	0.50	3.50	1.00	1	3	linear	9	uniform
0.2591	200	0.50	3.50	1.00	1	3	linear	11	uniform
0.2591	200	0.50	3.50	1.00	1	2	cosine	9	uniform

Table 4.8: Top 10 Parameter sets for K-NN with PCA, Round 1 of hypertuning. Features are defined in Table 3.2

These results suggest a sample size of 200 packets performs well with the configuration ranged we have tested. Additionally, lower activity thresholds seem to be most effective when at or below 0.5 packet/second, high activity threshold at around 3.5-4P/s, lookback at 1 second, and small-paste at 1 block. Large paste does not seem to converge from these results.

The linear kernel seems to have perform better than the other kernels according to these results, and a high k-value seems to perform better than low. The weight function also shows a slight bias towards distance over uniform based on these top-10 results.

The Figure 4.17 shows the number of occurrences of each sample size in the all of the top results. Contrary to what we saw in the top-10 results, 150 packets/sample occur most often despite 200 dominating the top results. For round two, we test both of these values again.

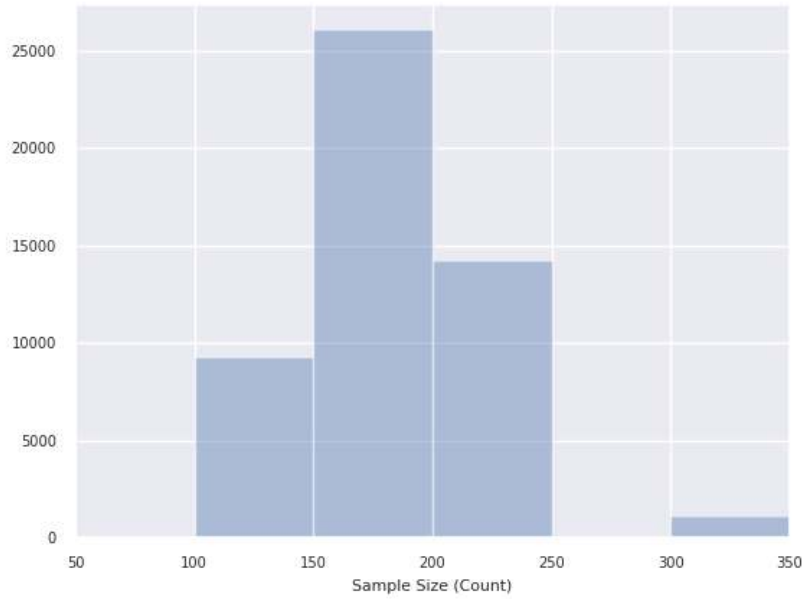


Figure 4.17: Occurrences of each sample size value in the top performing configurations in K-NN hypertuning with PCA, Round 1

The number of occurrences of each low-activity threshold are graphed in Figure 4.18. This confirms the results from the top-1o results. From this we can guess that the optimal low-activity threshold is at most 0.5 seconds, but may be lower. In the next round, we test 0.25, 0.5, and 0.75P/s.

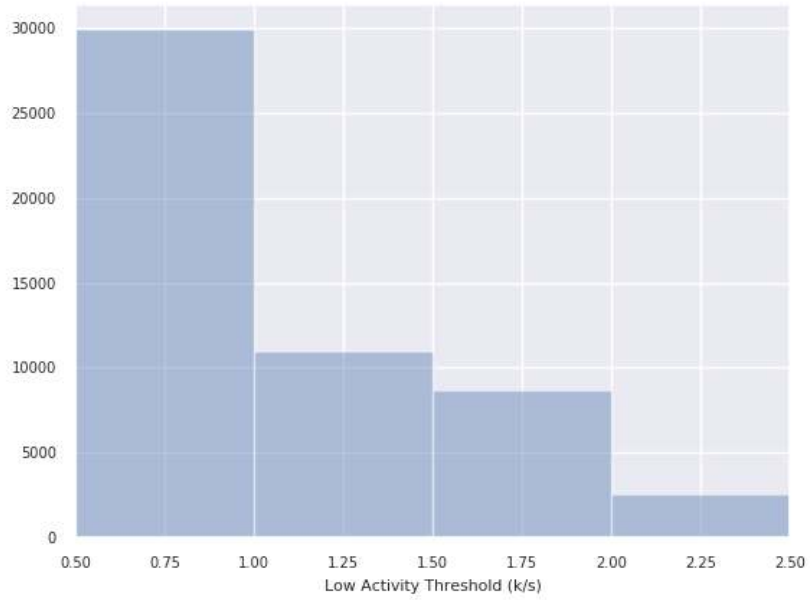


Figure 4.18: Occurrences of each low-activity threshold value in the top performing configurations in K-NN hypertuning with PCA, Round 1

Figure 4.19 shows the occurrences of each high-activity threshold. Similar to the sample size, we see 2.5P/s occurs most frequently here despite 4.0P/s obtaining higher results. In the next round of hypertuning, we test both these values again, as well as the intermediate values.

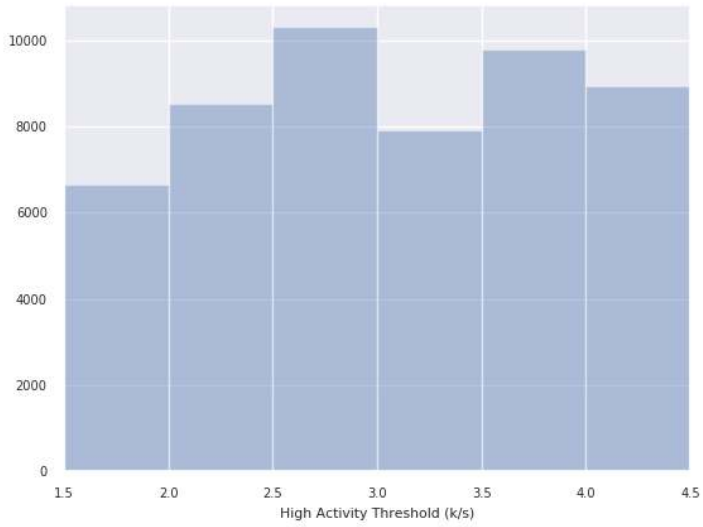


Figure 4.19: Occurrences of each high activity threshold value in the top performing configurations in K-NN hypertuning with PCA, Round 1

Figure 4.20 shows the occurrences of each lookback range. Again, despite 1s performing the best of top results, 2s occured most frequently. We test both of these values as well as 1.25s, 1.5s, and 1.75s in the next round.

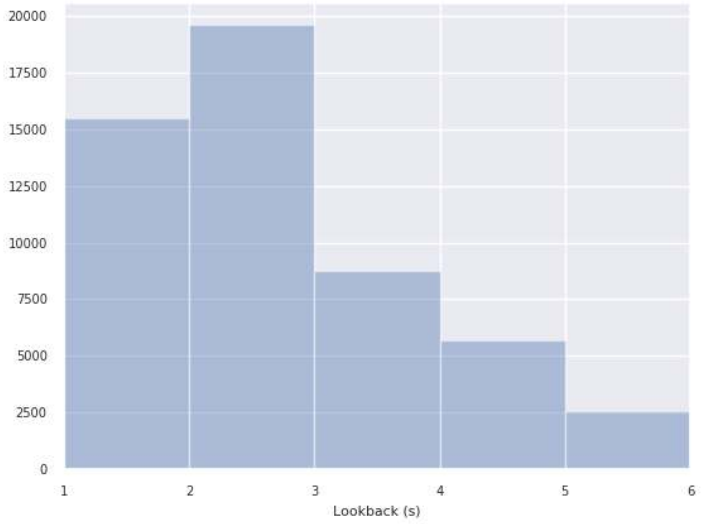


Figure 4.20: Occurrences of each lookback value in the top performing configurations in K-NN hypertuning with PCA, Round 1

Figure 4.21 shows the occurrences of each small-paste threshold. Since the value must be a natural number and both the top-10 results and the occurrences graph agree, 1 block is used for K-NN with PCA from here on in this paper.

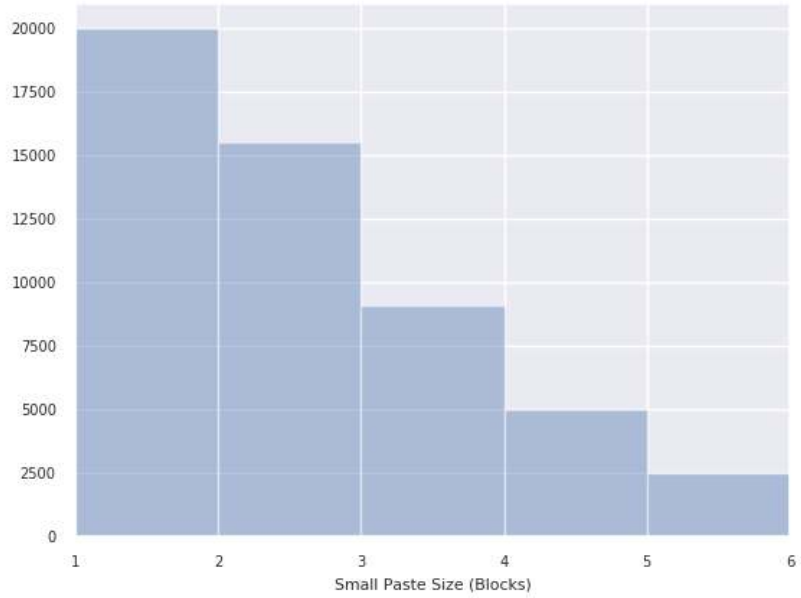


Figure 4.21: Occurrences of each small paste threshold value in the top performing configurations in K-NN hypertuning with PCA, Round 1

Figure 4.22 shows the occurrences of each large-paste threshold. While the top-10 results did not converge, the occurrences graph shows a clear favoritism towards 6 blocks. Since this graph shows an upward trend continuing past the range of values we tested, we expand the range for the next round.

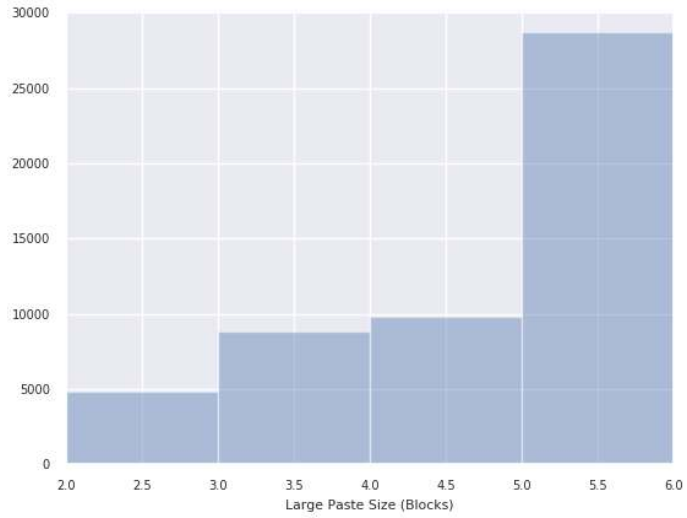


Figure 4.22: Occurrences of each large paste threshold value in the top performing configurations in K-NN hypertuning with PCA, Round 1

Figure 4.23 shows the occurrences of each kernel function in the top results. We can see linear and cosine both performing significantly stronger than the other functions. For the next round, we keep both of these values.

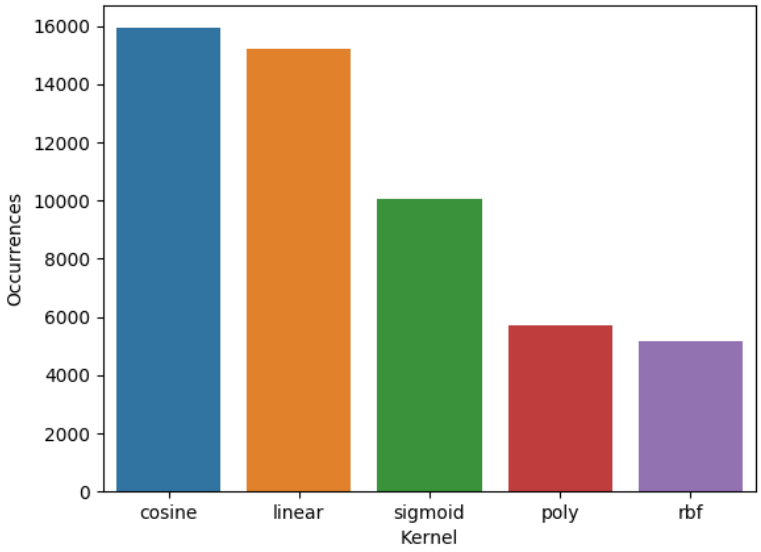


Figure 4.23: Occurrences of each kernel function in the top configurations in K-NN hypertuning with PCA, Round 1

Figure 4.24 shows the occurrences of each weight function in the top results. Distance seems to perform better than a uniform weight function in this graph, but this may be because the distance function is compensating for high K values by de-weighting the distant nodes. For the next round, we test both of these again to be sure.

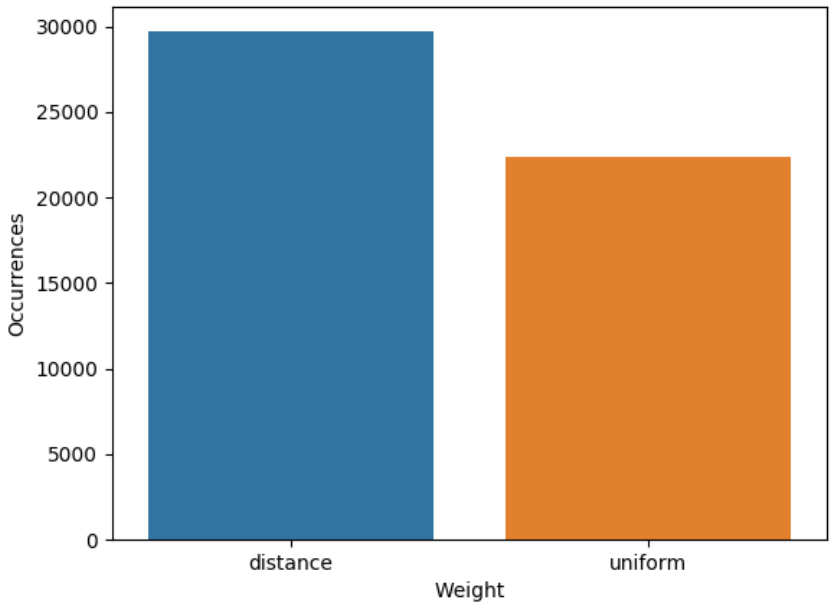


Figure 4.24: Occurrences of each weight function in the top configurations in K-NN hypertuning with PCA, Round 1

Figure 4.25 shows the occurrences of each weight function in the top results. We see the K values tending upward, but also beginning to level out around the upper end of our tested range. For the next round, we test 8, 9, 10, 11, and 12 to both increase our granularity and ensure the tend continues to level out.

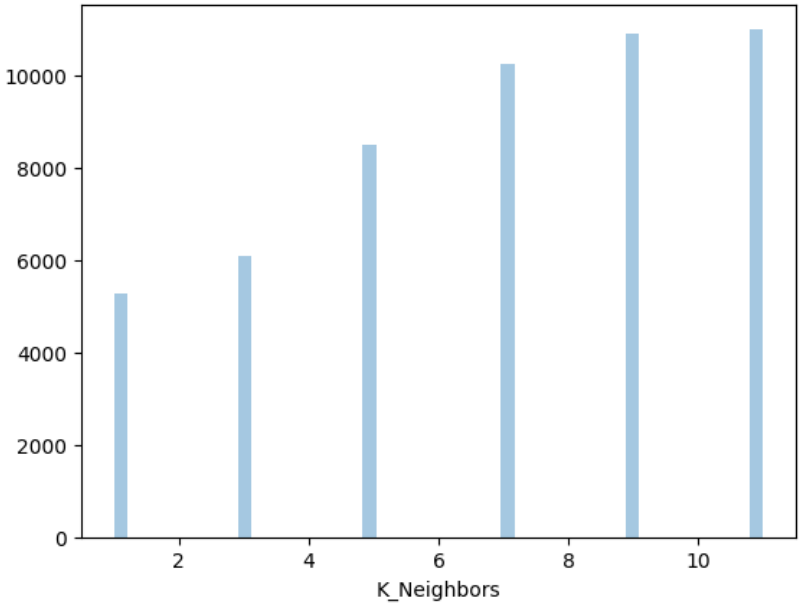


Figure 4.25: Occurrences of each K value in the top configurations in K-NN hypertuning with PCA, Round 1

4.2.2.2 Round 2

Based on the results from the previous round of hypertuning, we developed a new set of parameters to test, listed in Table 4.9.

Parameter	Values	Unit
Sample Size	150, 200	Packets
Low-Activity Threshold	0.25, 0.5, 0.75	Packets/Second
High-Activity Threshold	2.5, 3, 3.5, 4	Packets/Second
Lookback	1, 1.25, 1.5, 1.75, 2	Seconds
Large-Paste Threshold	6, 7, 8, 9	Blocks
Kernel Fn.	Linear, Cosine	N/A
Weight Fn.	Distance, Uniform	N/A
K	8, 9, 10, 11, 12	N/A

Table 4.9: Range of values for hyper parameters in second round of hypertuning K-NN with PCA

This round of hypertuning ran in 0.5 hours on 527 computers, returning 9574 results. Because almost all of the configurations were run and stored, all parameters were equally represented in the top results. Therefore, we will only analyze the top-10 results, listed in Table 4.10.

Sample Size of 200 packets dominates the top results, as does a lookback of 1.75s. The Low-activity threshold seems to need to be at its peak somewhere between 0.25P/s and 0.5P/s. Similarly, the high activity threshold seems to be at its peak somewhere between 3.5P/s and 4P/s. Since there does not seem to be a significant gain from further narrowing this range, we will continue to use 0.25P/s for the low threshold and 3.5P/s for the high threshold. Similarly, the large paste size threshold seems to work well as 7 or 8 blocks. We use 8 blocks for the remainder of this study.

Cosine dominates the top-10 results, so we use this for the PCA kernel for K-NN for the remainder of the study, and the same goes for the uniform weight function. The K values have many values represented, but 8 neighbors seems to be the most common and occurs in the configuration which produced the best result. We therefore continue with K as 8.

Accuracy	Sample S.	Low-Act	High-Act	Lookback	Small-Paste	Large-Paste	PCA Kernel	K	Weight Fn
0.2705	200	0.25	3.50	1.75	1	8	cosine	8	uniform
0.2682	200	0.50	4.00	1.75	1	8	cosine	8	uniform
0.2682	200	0.25	3.50	1.75	1	8	cosine	10	uniform
0.2659	200	0.50	4.00	1.75	1	7	cosine	8	uniform
0.2636	200	0.50	4.00	1.00	1	8	linear	11	uniform
0.2636	200	0.50	3.50	1.75	1	8	cosine	10	uniform
0.2636	200	0.50	3.50	1.75	1	7	cosine	9	uniform
0.2636	200	0.50	3.50	1.75	1	7	cosine	8	uniform
0.2636	200	0.25	4.00	1.75	1	8	cosine	8	uniform
0.2636	200	0.25	4.00	1.75	1	7	cosine	10	uniform

Table 4.10: Top 10 Parameter sets for K-NN with PCA, Round 2 of hypertuning.

4.2.3 Random Forest

Because the RF classifier has a significantly more CPU intensive training period than K-NN, we use PCA to reduce the feature space rather than directly manipulating features. We use the top 5 PCA features because this maintains 98% of the variance and helps the RF converge more quickly.

4.2.3.1 Round 1

Round 1 of hypertuning completed in approximately 24 hours with 523 computers returning 52,300 results.

Table 4.11 shows the results from the first round of hypertuning. The results are slightly better than the first round of K-NN, but we can attribute that to the PCA reducing the feature space. Most of the parameters seem to have converged nicely.

Accuracy	Sample S.	Low-Act	High-Act	Lookback	Small-Paste	Large-Paste
0.2600	150	0.5	2.50	1.00	2	6
0.2533	150	0.5	4.00	1.00	2	6
0.2533	150	0.5	3.50	1.00	2	6
0.2533	150	0.5	3.00	1.00	2	6
0.2533	150	0.5	3.00	1.00	2	5
0.2533	150	0.5	3.00	1.00	2	3
0.2533	150	0.5	2.50	1.00	2	6
0.2533	150	0.5	2.50	1.00	2	5
0.2533	150	0.5	2.50	1.00	2	5
0.2533	150	0.5	2.50	1.00	2	5

Estimators	Max Features	Min Samples/Leaf	Criterion	PCA Kernel
200	None	5	Entropy	Cosine
150	3	5	Entropy	Linear
150	3	5	Entropy	Linear
200	3	5	Gini	Cosine
200	3	5	Gini	Cosine
200	None	1	Gini	Cosine
200	3	5	Entropy	Cosine
200	3	1	Entropy	Cosine
150	None	5	Gini	Cosine
100	None	5	Entropy	Cosine

Table 4.11: Top 10 Parameter sets for RF, Round 1 of hypertuning.

The graphs of sample sizes, low activity thresholds both show a strong bias for 150 samples, 0.5P/s respectively as shown in Figure 4.26 and Figure 4.27. For the next

round of hypertuning, we will increase the resolution on the low-activity threshold to include 0.25P/s and 0.75P/s.

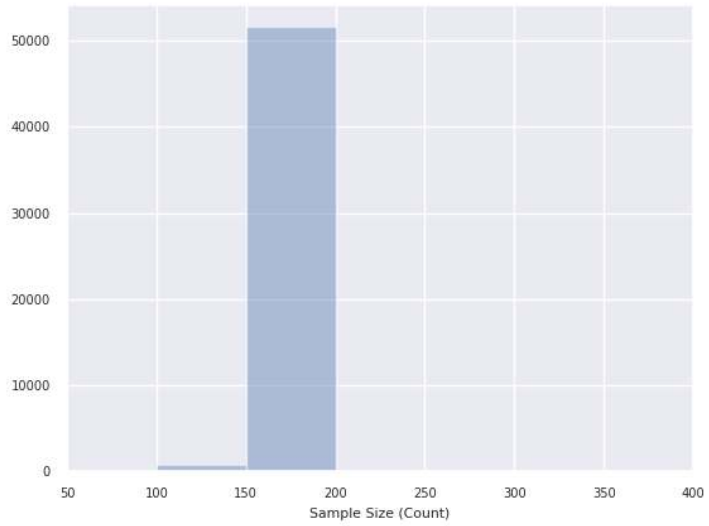


Figure 4.26: Occurrences of each sample size in the top performing feature sets in RF hypertuning, Round 1

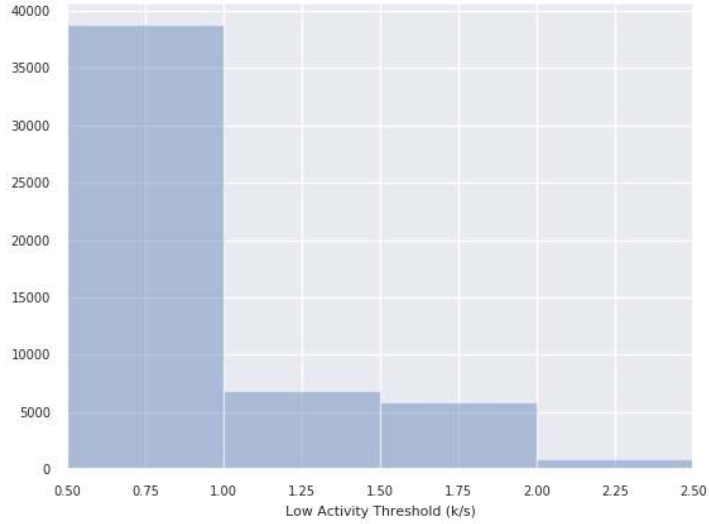


Figure 4.27: Occurrences of each low activity threshold in the top performing feature sets in RF hypertuning, Round 1

Figure 4.28 show the high-activity threshold does not converge as strongly as the previous values. All the values above 2.5P/s seem to perform approximately equally well ad this is reflected in the top-10 results. From this, we can conclude that the RF is adaptable to any of these thresholds, but seems to have a slight preference for 2.5P/s. Hence we will use this value in future tests.

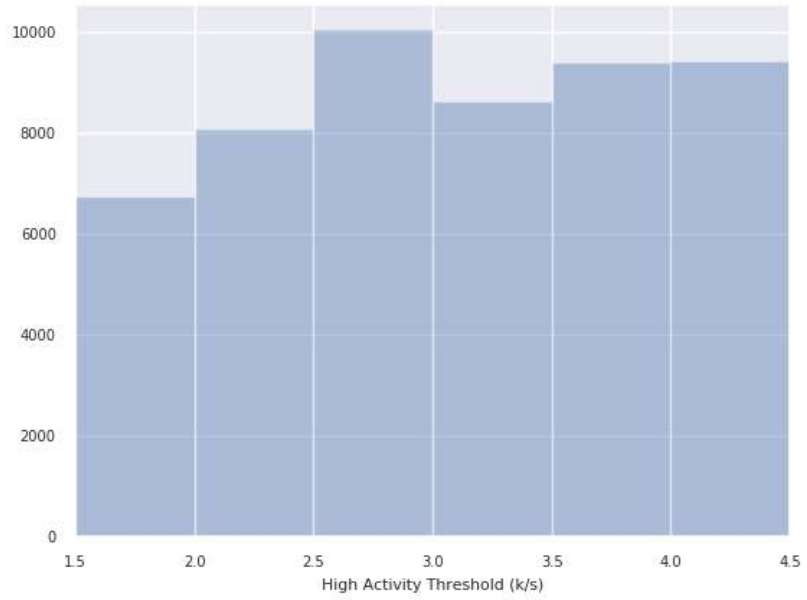


Figure 4.28: Occurrences of each high activity threshold in the top performing feature sets in RF hypertuning, Round 1

Figure 4.29 show that there are two values for the lookback time which performed comparably. For the next round, we will increase the resolution around the values to include 0.5s, 1s, 1.5s, and 2s.

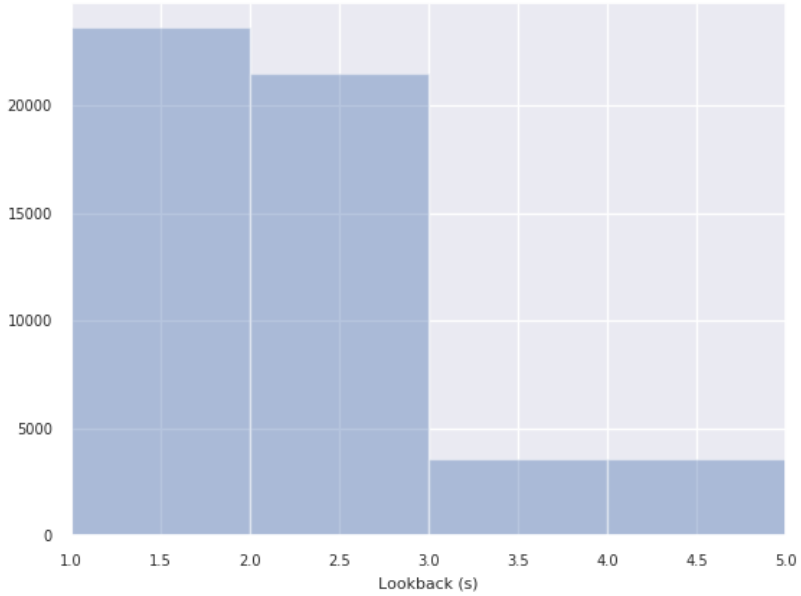


Figure 4.29: Occurrences of each lookback period in the top performing feature sets in RF hypertuning, Round 1

Figures 4.30 and 4.31 show the distribution of paste thresholds in the top results. We can see that although 1 and 2 blocks perform similarly, two block small-paste threshold has a slight lead which is reflected in the top 10. The large paste threshold again has a significant preference for the maximum value. In the next round, we extend the range as we have with K-NN.

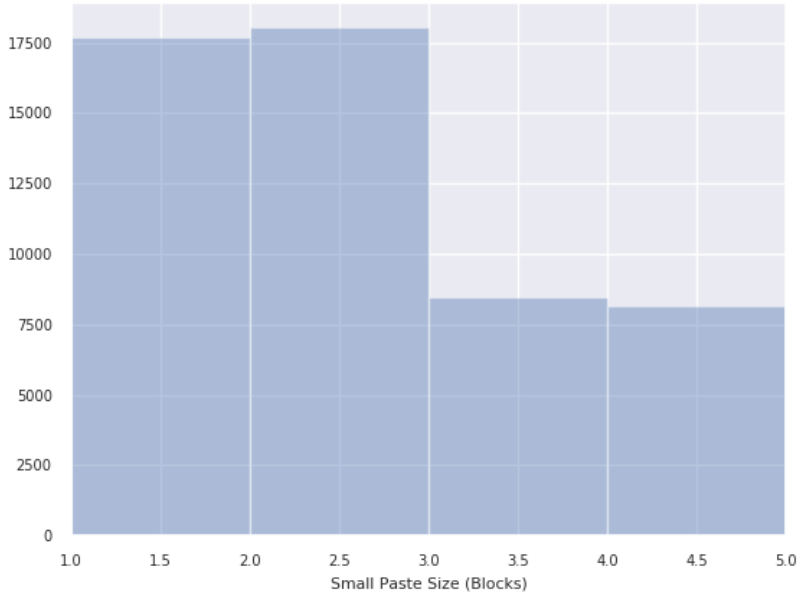


Figure 4.30: Occurrences of each small paste threshold in the top performing feature sets in RF hypertuning, Round 1

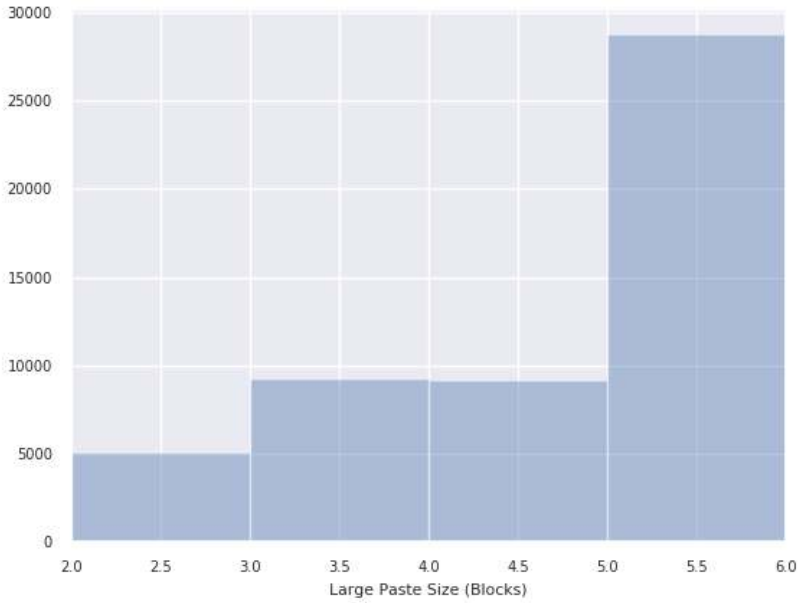


Figure 4.31: Occurrences of each large paste threshold in the top performing feature sets in RF hypertuning, Round 1

Not surprisingly, more estimators seems to improve performance, as shown in figure 4.32. We can also see the diminishing returns from ore estimators in this graph implying that more estimators won't add significant benefits. In light of that, we will use 200 estimators from here on.

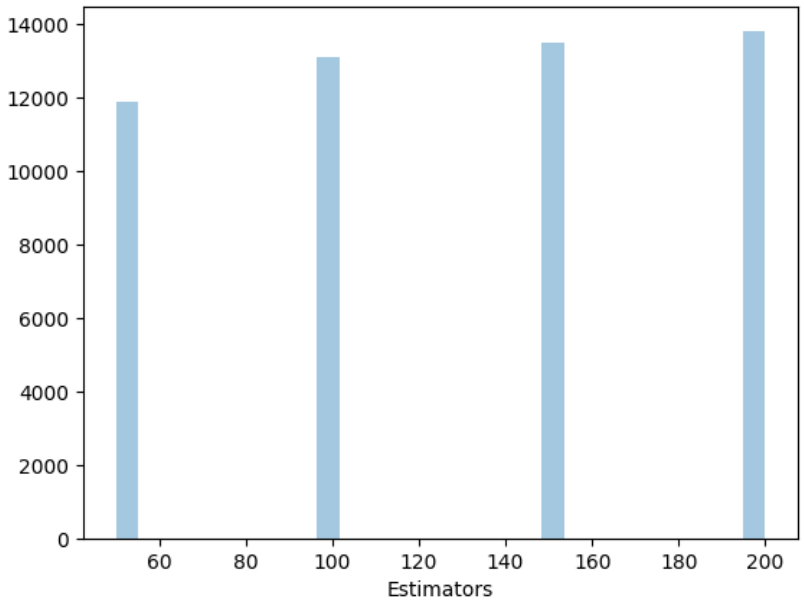


Figure 4.32: Occurrences of each estimator count in the top performing feature sets in RF hypertuning, Round 1

Figure 4.33 shows the occurrences of each limit to the number of features per node. We can see that 3 features and no limit are the top performers (recall that we used PCA to reduce the feature space to 5 features in total). For the next round, we will test 3, 4, and no limit.

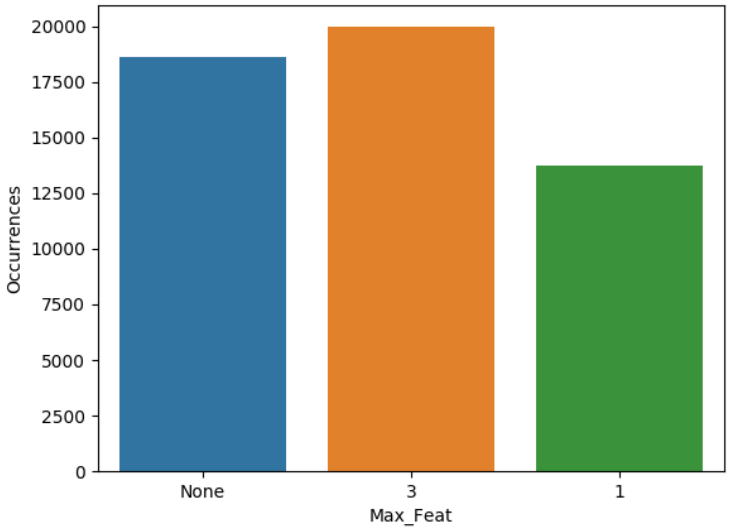


Figure 4.33: Occurrences of each maximum features/node in the top performing feature sets in RF hypertuning, Round 1

Figure 4.34 shows that 5 samples/leaf outperforms the others. For the next round, we increase the resolution on these values to include 3, 5, and 7 samples/leaf.

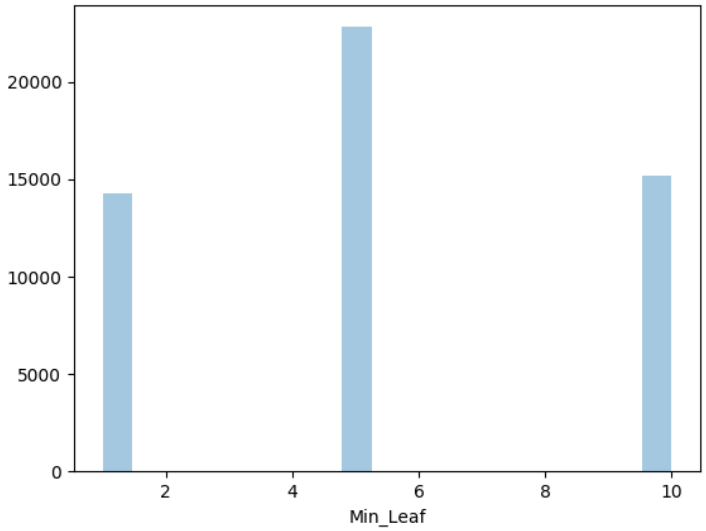


Figure 4.34: Occurrences of each minimum samples/leaf in the top performing feature sets in RF hypertuning, Round 1

Figure 4.35 shows that both the gini and entropy functions perform almost identically. From here on, we use the gini function.

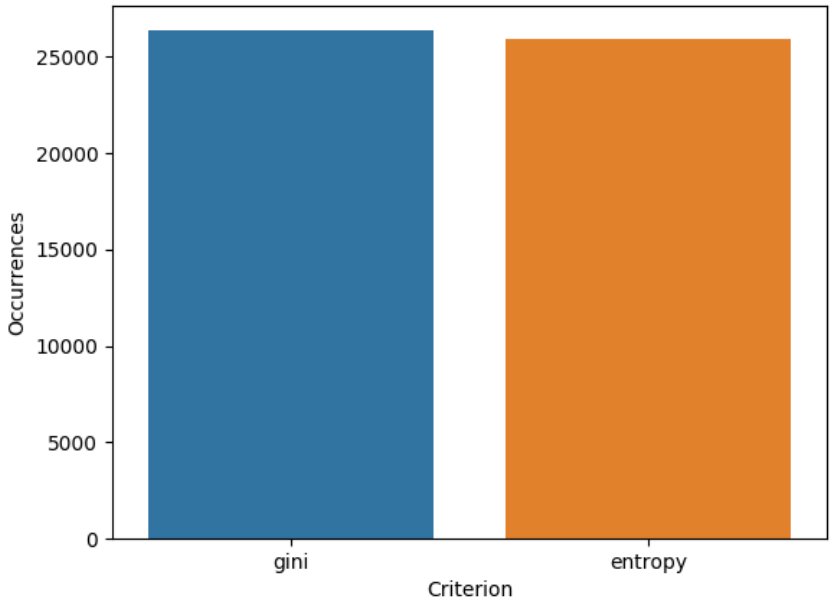


Figure 4.35: Occurrences of each split criterion in the top performing feature sets in RF hypertuning, Round 1

Figure 4.36 shows the number of occurrences of each PCA kernel function. Interestingly, there seems to be a clear hierarchy among all the kernels. Although linear outperforms all the other kernels overall, the cosine kernel dominates the top 10 results. For this reason, linear and cosine will both be used in round two.

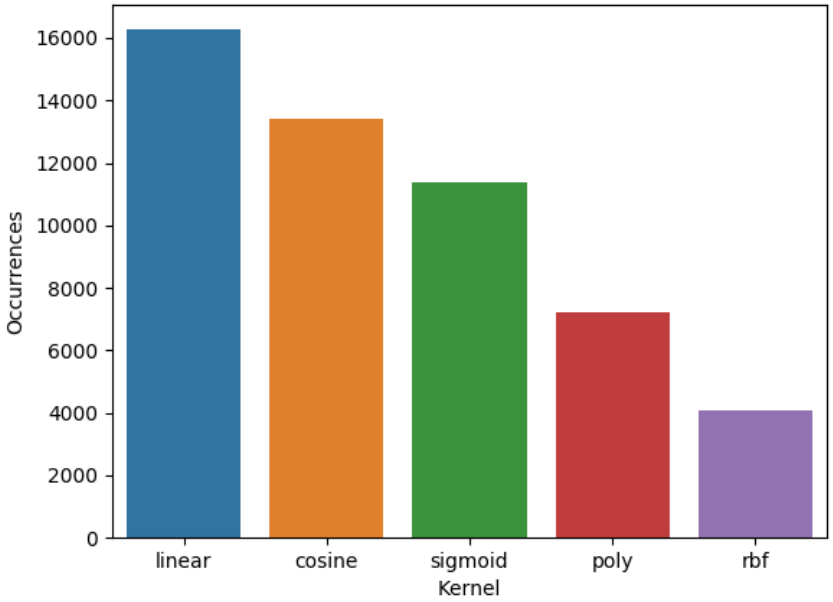


Figure 4.36: Occurrences of each PCA kernel in the top performing feature sets in RF hypertuning, Round 1

The final set of parameters for round two are listed in table 4.12.

Parameter	Values	Unit
Dataset	Both	N/A
Sample Size	150	Packets
Low-Activity Threshold	0.25 0.5, 0.75	Packets/Second
High-Activity Threshold	2.5	Packets/Second
Lookback	0.5, 1.0, 1.5, 2	Seconds
Small-Paste Threshold	2	Blocks
Large-Paste Threshold	6, 7, 8, 9	Blocks
Criterion	Gini	N/A
Estimator	200	RDTs
Min-Samples/Leaf	3, 5, 7	Samples
Max-Features/Node	3, 4, None	Features
PCA Kernel	linear, cosine	N/A

Table 4.12: Range of values for hyper parameters in second round of hypertuning for the Random Forest

4.2.3.2 Round 2

Round 2 completed in a few seconds with all 792 results stored. Because there were not enough runs to create a large distribution we only examine the top 10 results, seen in Table 4.13. While some of the values fail to converge on a best set of configurations, we can see from the accuracies all being within half of a percentage point from each other that the classifier will perform comparably with any of the configurations in these top 10.

Lookback, minimum samples/leaf, and PCA kernel all show a strong convergence to a single value. For the rest of the paper, we will use 1s, 3 samples, and cosine function respectively for Random Forest.

All three low-activity thresholds are represented in the top ten results suggesting that this may be too fine of a granularity to have significant impact on the RF classifier. We use 0.75P/s for the rest of this paper since it occurred the most in the top 10.

All of the large-paste thresholds are represented in roughly equal ratios, similar to what was seen with K-NN. This again suggests that the RF is robust to any of these values. We will use 7 blocks for the rest of this paper since it occurred the most.

Accuracy	Sample S.	Low-Act	High-Act	Lookback	Small-Paste	Large-Paste
0.2566	150	0.25	2.50	1.00	2	9
0.2563	150	0.75	2.50	1.00	2	7
0.2561	150	0.75	2.50	1.00	2	6
0.2554	150	0.50	2.50	1.00	2	9
0.2549	150	0.75	2.50	1.00	2	7
0.2547	150	0.75	2.50	1.00	2	8
0.2535	150	0.50	2.50	1.00	2	7
0.2533	150	0.25	2.50	1.00	2	6
0.2531	150	0.25	2.50	1.00	2	6
0.2530	150	0.75	2.50	1.00	2	7

Estimators	Max Features	Min Samples/Leaf	Criterion	PCA Kernel
200	3	3	gini	cosine
200	None	5	gini	cosine
200	3	5	gini	cosine
200	None	3	gini	cosine
200	4	3	gini	cosine
200	3	3	gini	cosine
200	4	3	gini	cosine
200	4	3	gini	cosine
200	3	3	gini	cosine
200	None	3	gini	cosine

Table 4.13: Top 10 Parameter sets for RF, Round 2 of hypertuning.

4.2.4 Support Vector Machine

As we did with RF, we use the top 5 PCA features while tuning the SVM.

4.2.4.1 Round 1

Round 1 of hypertuning completed in approximately 3 hours with 524 computers returning 52,200 results.

Table 4.14 shows the results from the first round of hypertuning. While the pre-processing parameters seem to converge strongly, the SVM Kernel and Regulation Parameter still have a wide variance.

Accuracy	Sample S.	Low-Act	High-Act	Lookback	Small-Paste	Large-Paste	SVM Kernel	Regulation Paramater	PCA Kernel
0.2700	150	0.50	4.00	1.00	1	2	rbf	4.0	cosine
0.2633	150	0.50	4.00	1.00	1	2	rbf	8.0	cosine
0.2633	150	0.50	3.50	1.00	1	2	rbf	8.0	cosine
0.2633	150	0.50	3.50	1.00	1	2	rbf	4.0	cosine
0.2633	150	0.50	3.50	1.00	1	2	poly	4.0	cosine
0.2617	150	0.50	4.00	1.00	1	2	poly	8.0	cosine
0.2600	150	0.50	3.50	1.00	2	3	rbf	8.0	cosine
0.2600	150	0.50	3.50	1.00	1	2	poly	8.0	cosine
0.2600	150	0.50	3.50	1.00	1	2	poly	2.0	cosine
0.2583	150	0.50	4.00	1.00	1	2	poly	4.0	cosine

Table 4.14: Top 10 Parameter sets for SVM, Round 1 of hypertuning.

Figures 4.37, 4.38, 4.39, and 4.40 all confirm the consistency of the results from the top 10 in the Sample Size, low-activity threshold, small-paste threshold, and PCA kernel respectively. For the next round, we increase the resolution on the low-activity threshold and lookback period. We test cosine and linear PCA kernels since they performed comparably.

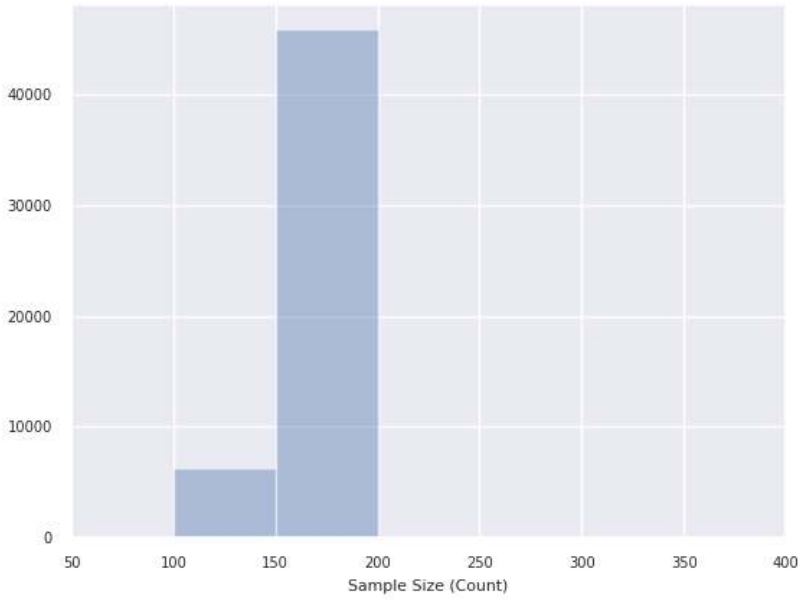


Figure 4.37: Occurrences of each sample size in the top performing feature sets in SVM hypertuning, Round 1

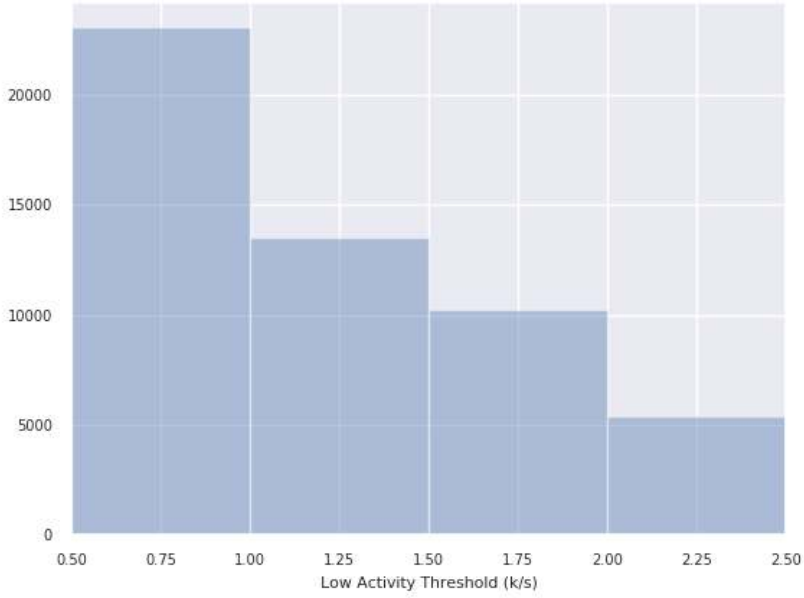


Figure 4.38: Occurrences of low activity thresholds in the top performing feature sets in SVM hypertuning, Round 1

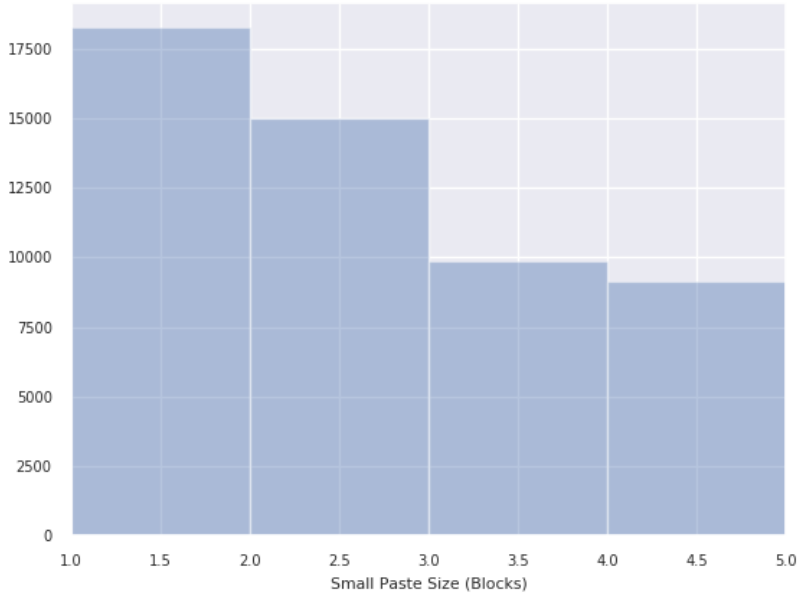


Figure 4.39: Occurrences of small paste thresholds in the top performing feature sets in SVM hypertuning, Round 1

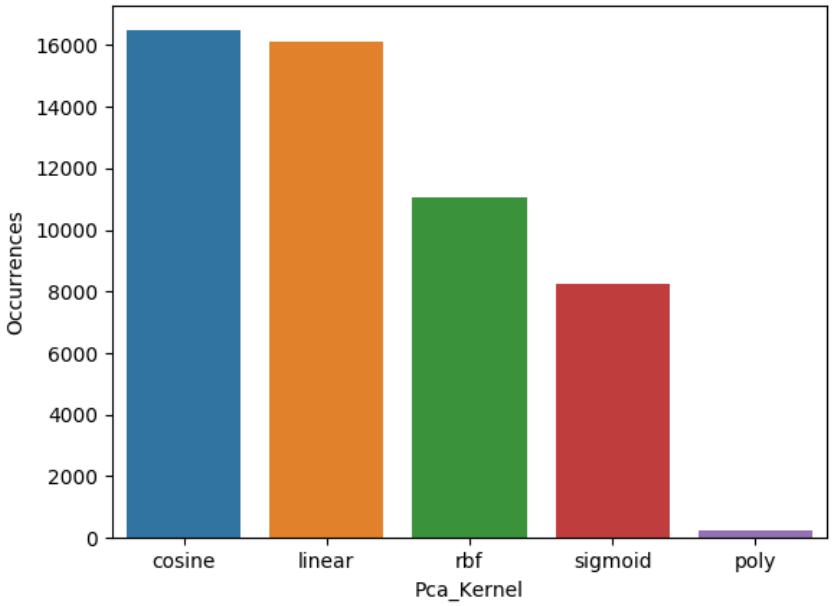


Figure 4.40: Occurrences of each PCA kernel in the top performing feature sets in SVM hypertuning, Round 1

Figure 4.41 shows a strong bias toward a longer lookback period, despite the top 10 results being composed of entirely 1s lookbacks. Ironically, 1s seems to be the least represented in the top results overall. Indicates that the lower lookback period performs bimodally depending on the other configurations, a situation that had not come up in the other results. To test this further, we will increase the maximum period and increase resolution around 1s.

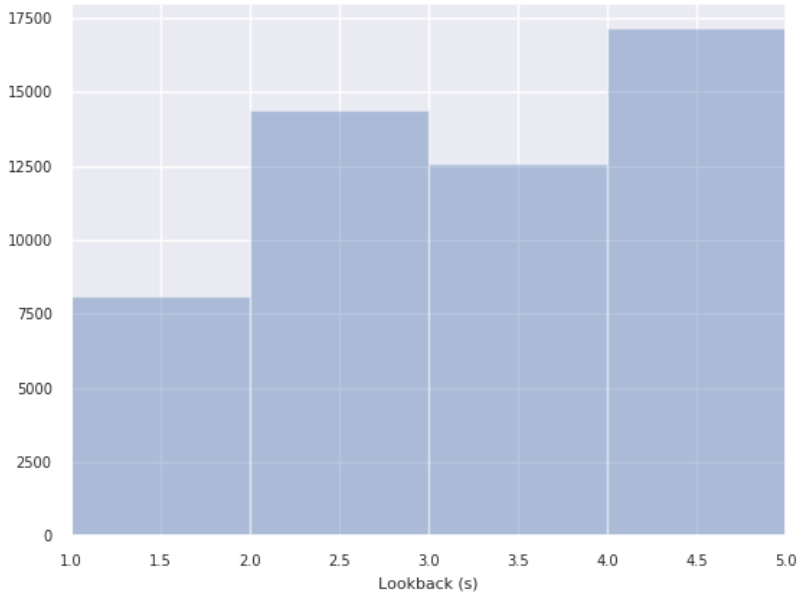


Figure 4.41: Occurrences of lookback periods in the top performing feature sets in SVM hypertuning, Round 1

Oddly, despite Figure 4.42 show a strong bias towards a high large-paste threshold, lower thresholds dominate the top 10 results. For the next round, we will extend the testing range and include a threshold of 2.

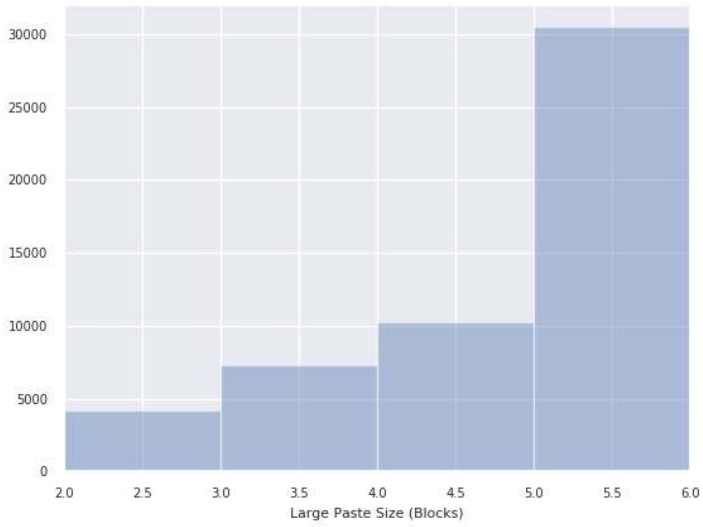


Figure 4.42: Occurrences of large paste thresholds in the top performing feature sets in SVM hypertuning, Round 1

Figure 4.43 shows that the high-activity threshold has a bi-modal distribution for this run. For the next round, we will test again with just the values 2.5P/s and 3.5P/s.

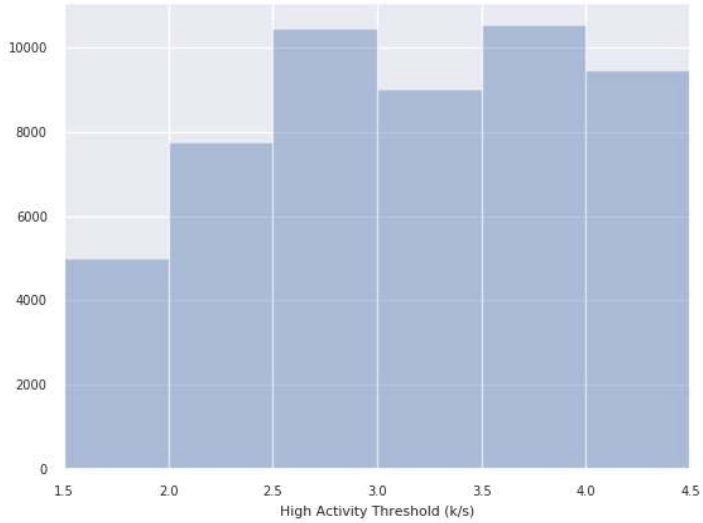


Figure 4.43: Occurrences of high activity thresholds in the top performing feature sets in SVM hypertuning, Round 1

Figure 4.44 shows the occurrences of each regulation constant in the top results. The graph makes clear that higher values perform much better but it seems to have diminishing returns. For the next round, we will use 4, 8, and 12.

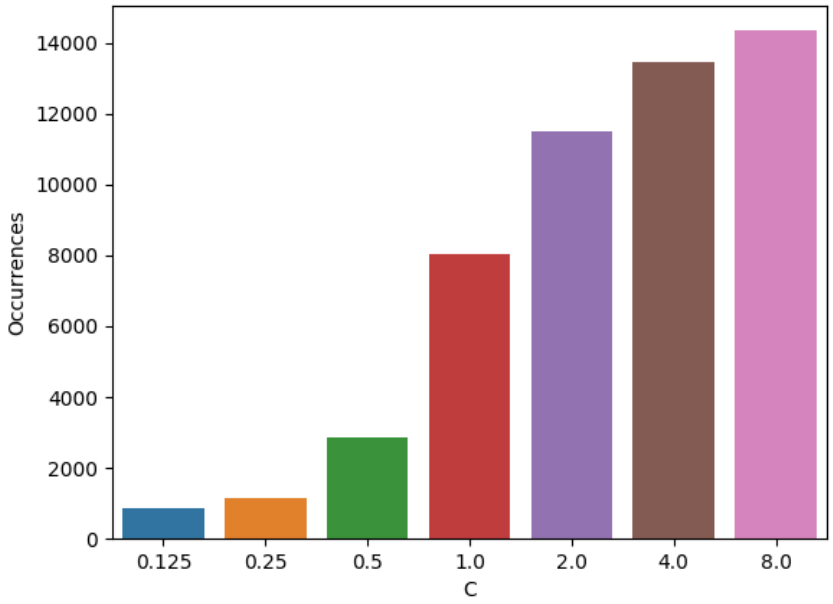


Figure 4.44: Occurrences of each regulation parameter in the top performing feature sets in SVM hypertuning, Round 1

Figure 4.45 shows the occurrences of each SVM kernel in the top results. RBF and Linear clearly perform better than poly and sigmoid, so we will keep the former and discard the later in the next round of hypertuning.

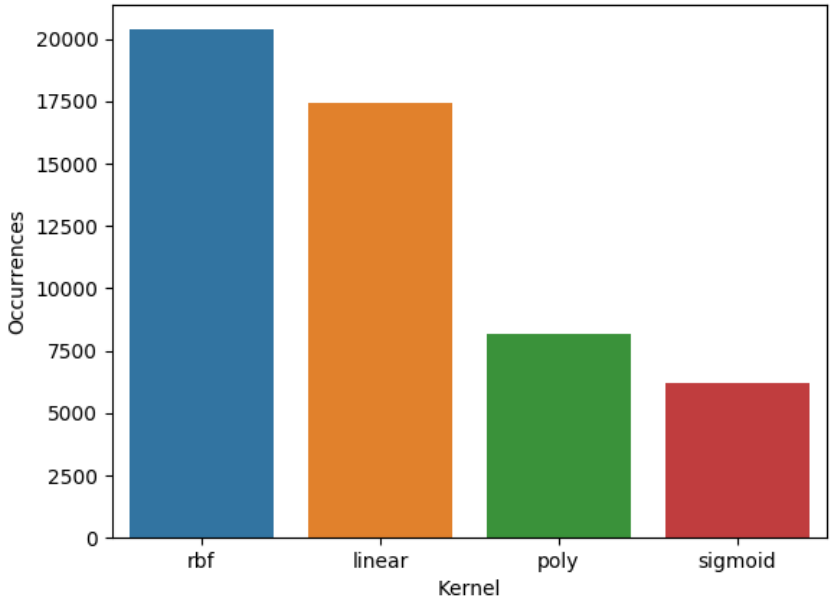


Figure 4.45: Occurrences of each kernel in the top performing feature sets in SVM hypertuning, Round 1

The final set of parameters for round two are listed in table 4.15.

Parameter	Values	Unit
Dataset	Both	N/A
Sample Size	150	Packets
Low-Activity Threshold	0.25 0.5, 0.75	Packets/Second
High-Activity Threshold	2.5, 3.5, 3.75, 4.0	Packets/Second
Lookback	0.5, 1.0, 1.5, 4, 4.5, 5, 5.5	Seconds
Small-Paste Threshold	1	Blocks
Large-Paste Threshold	2, 6, 7, 8, 9	Blocks
SVM Kernel	Linear, RBF	N/A
Regulation Constant	4, 6, 8, 10, 12	N/A
PCA Kernel	Linear, Cosine	N/A

Table 4.15: Range of values for hyper parameters in second round of hypertuning for the Support Vector Machine

4.2.4.2 Round 2

Round 2 completed in a few seconds with all 5040 results stored. Because there were not enough runs to create a large distribution we only examine the top 10 results, seen in Table 4.16. While some of the values fail to converge, we can see from the accuracies all being within half of a percentage point from each other suggesting that the classifier will perform comparably with any of the value combinations in the top 10.

The low-activity threshold seems to perform its best at 0.75P/s, the lookback period at 1s, and the large-paste threshold at 2 blocks. The high activity threshold seems to be robust to any of values tested, but has a slightly better preference with 3.75P/s.

The classifier performed clearly best with RBF as the SVM kernel, Cosine as the PCA kernel and a regulation parameter of 6.

Accuracy	Sample S.	Low-Act	High-Act	Lookback	Small-Paste	Large-Paste	SVM Kernel	Regulation Paramater	PCA Kernel
0.2700	150	0.75	3.75	1.00	1	2	rbf	6.0	cosine
0.2700	150	0.50	3.75	1.00	1	2	rbf	6.0	cosine
0.2700	150	0.25	3.75	1.00	1	2	rbf	6.0	cosine
0.2683	150	0.75	4.00	1.00	1	2	rbf	6.0	cosine
0.2683	150	0.50	3.75	1.00	1	2	rbf	8.0	cosine
0.2683	150	0.25	3.75	1.00	1	2	rbf	4.0	cosine
0.2667	150	0.75	4.00	1.00	1	2	rbf	4.0	cosine
0.2667	150	0.75	3.50	1.00	1	2	rbf	6.0	cosine
0.2667	150	0.75	2.50	1.25	1	2	rbf	4.0	cosine
0.2667	150	0.50	3.75	1.00	1	2	rbf	4.0	cosine

Table 4.16: Top 10 Parameter sets for SVM, Round 1 of hypertuning.

4.2.5 Perceptron Network

Due to issues hypertuning the Perceptron Network, we were not able to properly train the algorithm.

4.3 Accuracy

The final results for each classifier are listed in Table 4.17 These results were calculated using the combined dataset with the 25 participants who provided the most data.

Classifier	Accuracy	F_1-Score
K-NN Feature Sets	0.150286	0.140029
K-NN PCA	0.21136	0.181862
RF	0.195943	0.180032
SVM	0.213543	0.200641

Table 4.17: Accuracy and power of each classifier

All of the methods we developed performed comparably, with only slight differences in accuracy and power with the exception of K-NN without PCA.

Figures 4.46, 4.47, 4.48, 4.49 show the confusion matrices for each of the classifiers. Each of them shows a similar ability to classify some participants and a similar inability to classify others. For example, each classifier struggled to distinguish between participant 5A6 and 599.

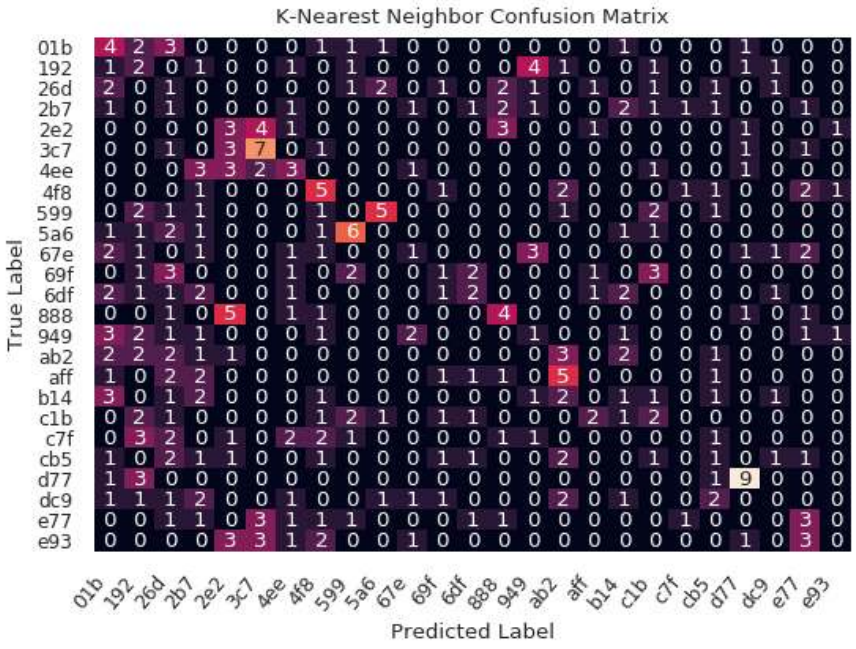


Figure 4.46: Confusion matrix for the K-NN classifier without PCA

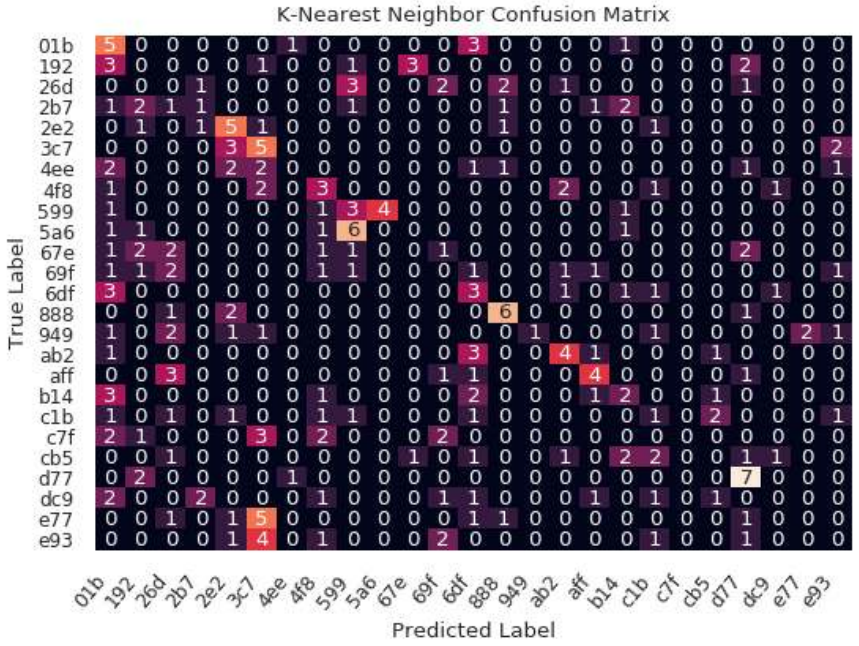


Figure 4.47: Confusion matrix for the K-NN classifier with PCA

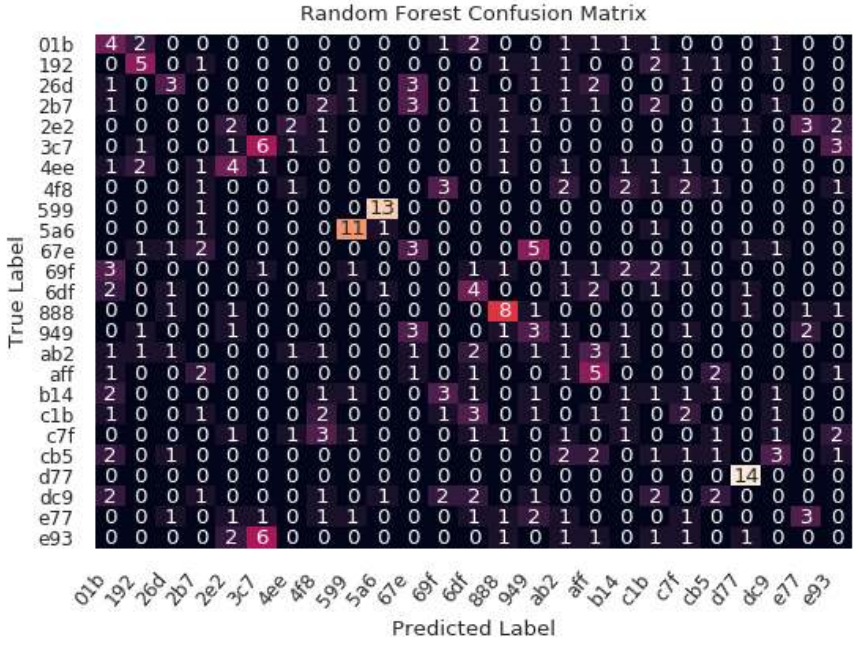


Figure 4.48: Confusion matrix for the RF classifier

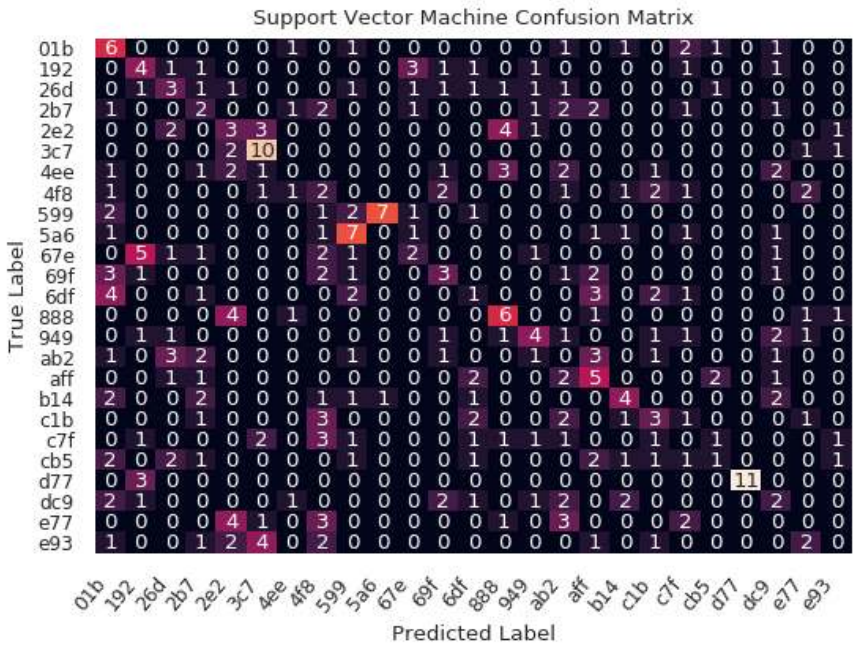


Figure 4.49: Confusion matrix for the SVM classifier

As mentioned before, one of the unique features of the K-NN classifier is that it can rank each of the possible outputs by votes. Figure 4.50 shows the accuracy of the the K-NN classifier without PCA when it is allowed to have up to N guesses per sample. We see the graph shows a strong upward tend up to 5 guesses, after which the graph levels out and eventually parallels the probability of guessing randomly. While this suggests that an attacker cannot obtain the user identity with a high degree of accuracy, they can reduce the search space by around 80% and still have a 50% chance of correctly identifying the user.

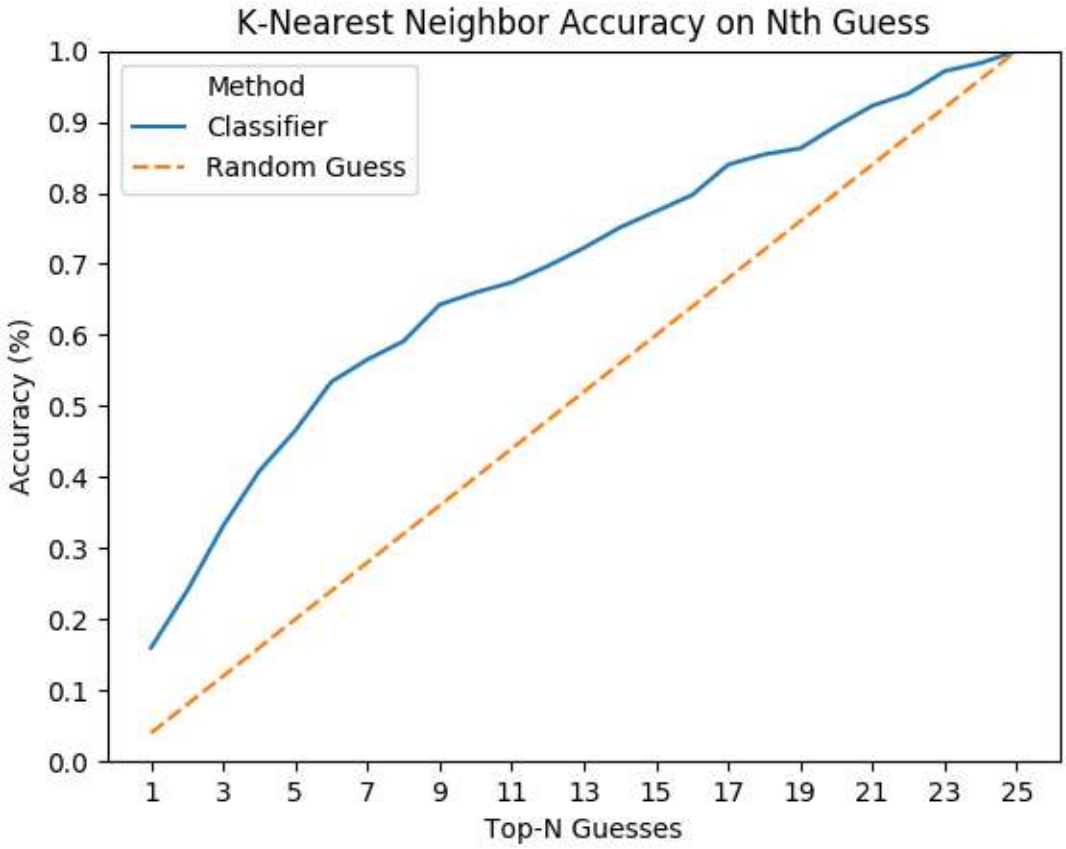


Figure 4.50: Accuracy of NN classifier without PCA by number of guesses

Figure 4.51 shows the accuracy of the the K-NN classifier with PCA when it is allowed to have up to N guesses per sample. Interestingly, it seems that while with PCA performs better with fewer guesses than without PCA, both curves seem to closely resemble each other and by 5 guesses the two have comparable accuracies. It's possible that this difference could be increased by hypertuning with multiple guesses, but this is outside the scope of our research.

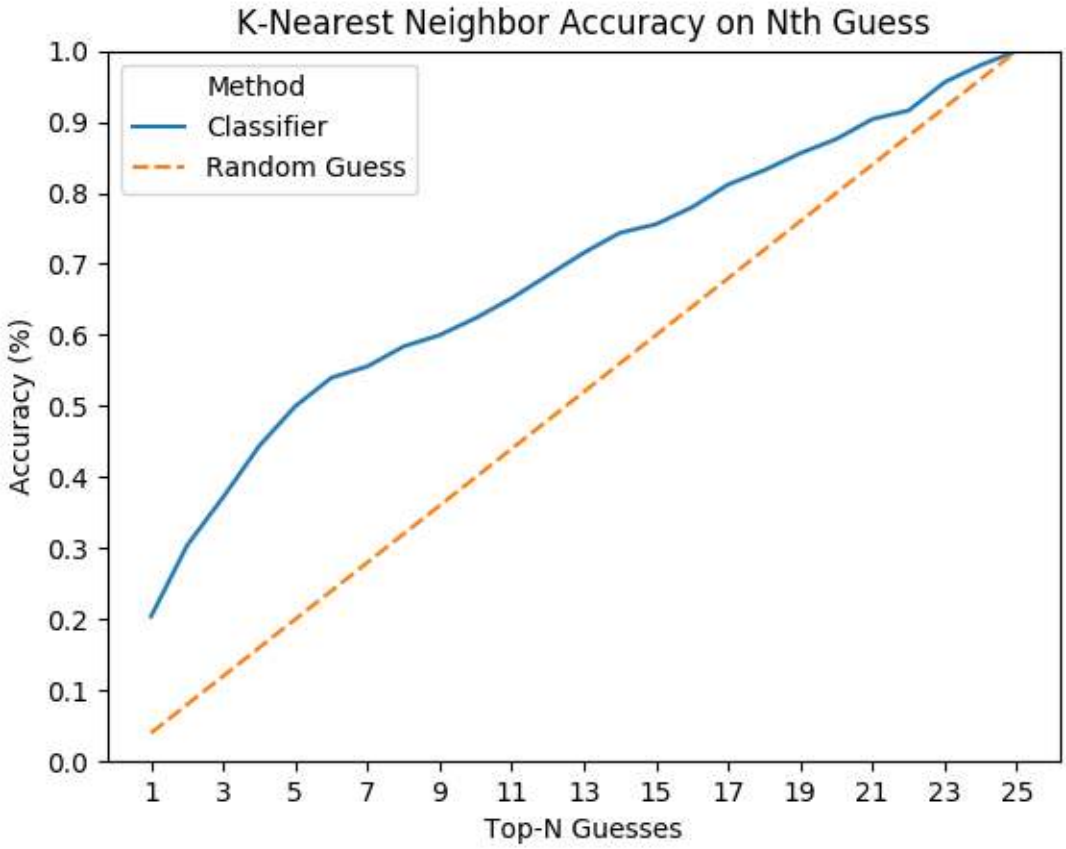


Figure 4.51: Accuracy of NN classifier with PCA by number of guesses

4.4 Guided versus Free

During research thus far we used a mix of our guided and free datasets. This was necessary given the limited amount of data we had to be able to have an adequate number of users to draw meaningful results. In this section, we discuss the impact of this mixing as well as the usefulness of each of the datasets for identification purposes.

Figure 4.52 shows the performance of each classifier trained on each dataset. These results were gathered with 10 different users and 10 samples per user for each classifier

which used 150 packet samples and 7 samples per user for K-NN PCA to keep an even comparison.

The results show a strong consistency across each of the classifiers. Using exclusively the free dataset provided the best results for any of the classifiers. Using exclusively the guided dataset provided the second best accuracy. And finally, the combined dataset performed the worst across the board, in come cases such as the NN classifier by a significant margin.

These results show that using data from one type of activity gives the best performance. Additionally, this indicates that our earlier results were using the worst dataset available, and had we had enough data in the free dataset, we could have achieved better results.

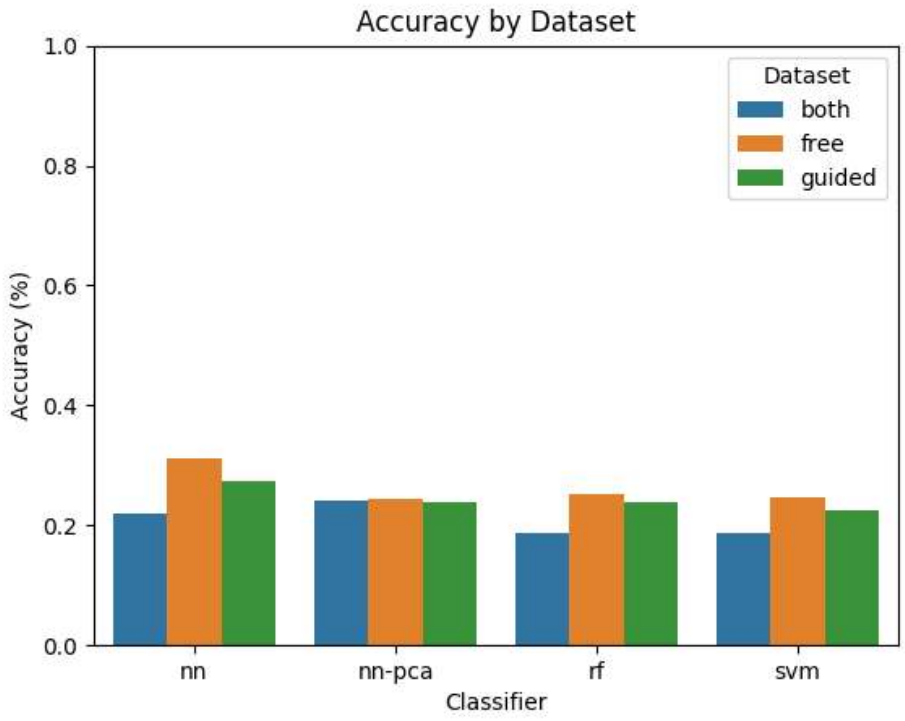


Figure 4.52: Accuracy by dataset

Interestingly, the K-NN classifier with PCA seems to have performed equally well with all three categories. Because it was the only dataset which used 200 packets per sample and thus had fewer samples per user, we suspect the number of samples may have played a factor. Table 4.18 compares the performance of each dataset with either 7 or 10 samples. We can see that the benefit from adding more data is much more powerful in the free dataset than the guided dataset. The combine dataset, unsurprisingly, seems to have the average of the improvements seen in the guided and free datasets.

# Samples	7	10
Both	0.2407	0.2613
Free	0.2431	0.2873
Guided	0.2392	0.2447

Table 4.18: Accuracy of each dataset with the K-NN classifier with PCA by number of samples

4.5 Scalability

Because we have such a wide number of users and variable amounts of data on each user, we have an opportunity to explore the scalability of the techniques discussed in this paper. In particular, we will discuss how the system accuracy scales as we introduce new users and how the accuracy scales as we introduce more data per user.

4.5.1 Number of Distinct Users

For testing the impact of the number of users, we fix our number of samples per user to 13 (or 10 in the case of K-NN with PCA) from the combined dataset and vary the number of users between 4 and 20. Because we wanted to maximize the upper bound for the number of users to test, we mix both guided and unguided datasets. Figure 4.53 shows the results from this testing.

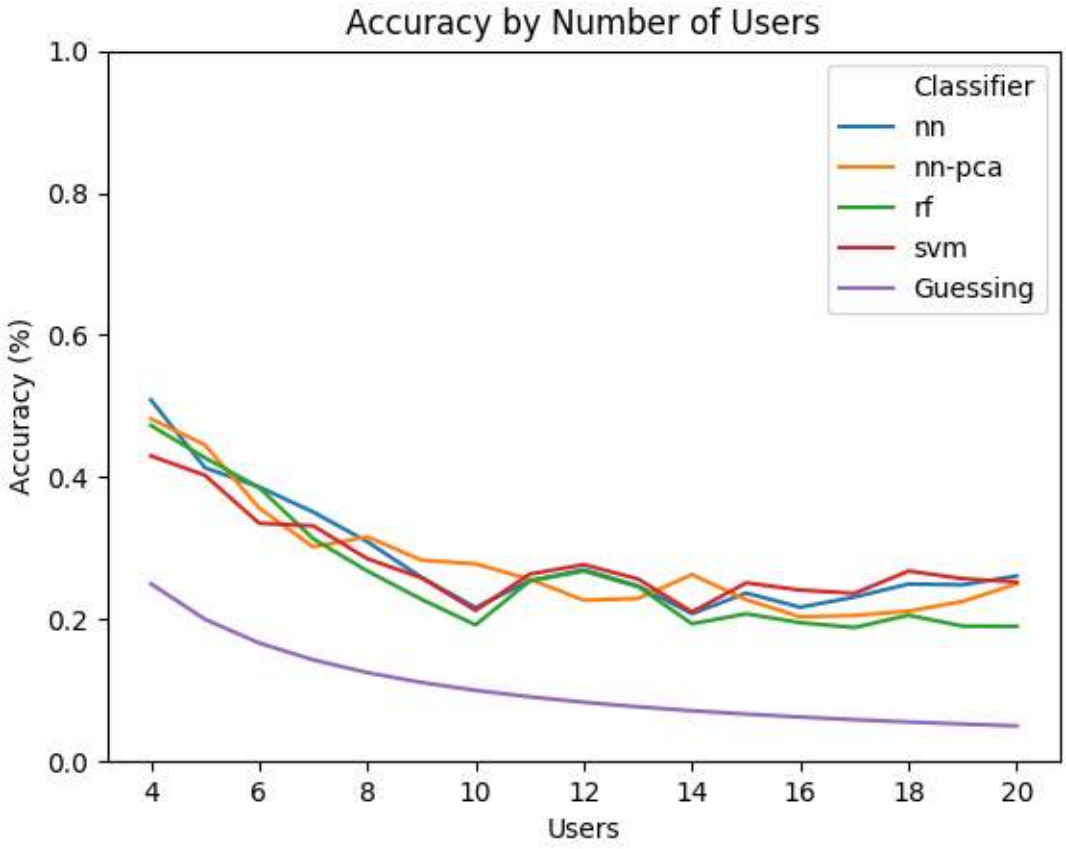


Figure 4.53: Accuracy by number of users

We plotted each of the classifiers separately, as well as included a line for randomly guessing. As expected, each of the classifiers mostly parallels the curve of the guessing line, but as random chance asymptotically approaches 0, the classifiers remain relatively consistent or actually increasing in accuracy in the case of NN and SVM. In particular it seems that the 10th and 14th users were particularly hard to idenity and dragged the averages down. As more users are introduced, the difficulty in identifying these users amortizes out.

The first 10 users still show a strong downward tend indicating that initially, number of potential users is the limiting factor. But afterwards, the graph quickly levels out which indicates that the patterns of individual users becomes the dominating

factors. We can conclude from this that homogeneity is the primary concern for this attack, rather than the potential search space. Additionally, while more data would be needed to confirm this the current findings indicate this solution should maintain its viability with a large number of users.

4.5.2 Amount of Data Per User

For this section, we decided to focus on the 10 users who provided the most data. Since each sample is 150 packets (for this test, we reduced K-NN PCA's sample size to match the others), we found that we could vary the amount of data from 10 samples/user to 24 samples/user. Additionally, we drew exclusively from the free dataset because we did not need a large number of users to test how this scales. We did not test below 10 samples because this would increase the likelihood that some folds would not have any samples for a user in either the training or testing set. Figure 4.54

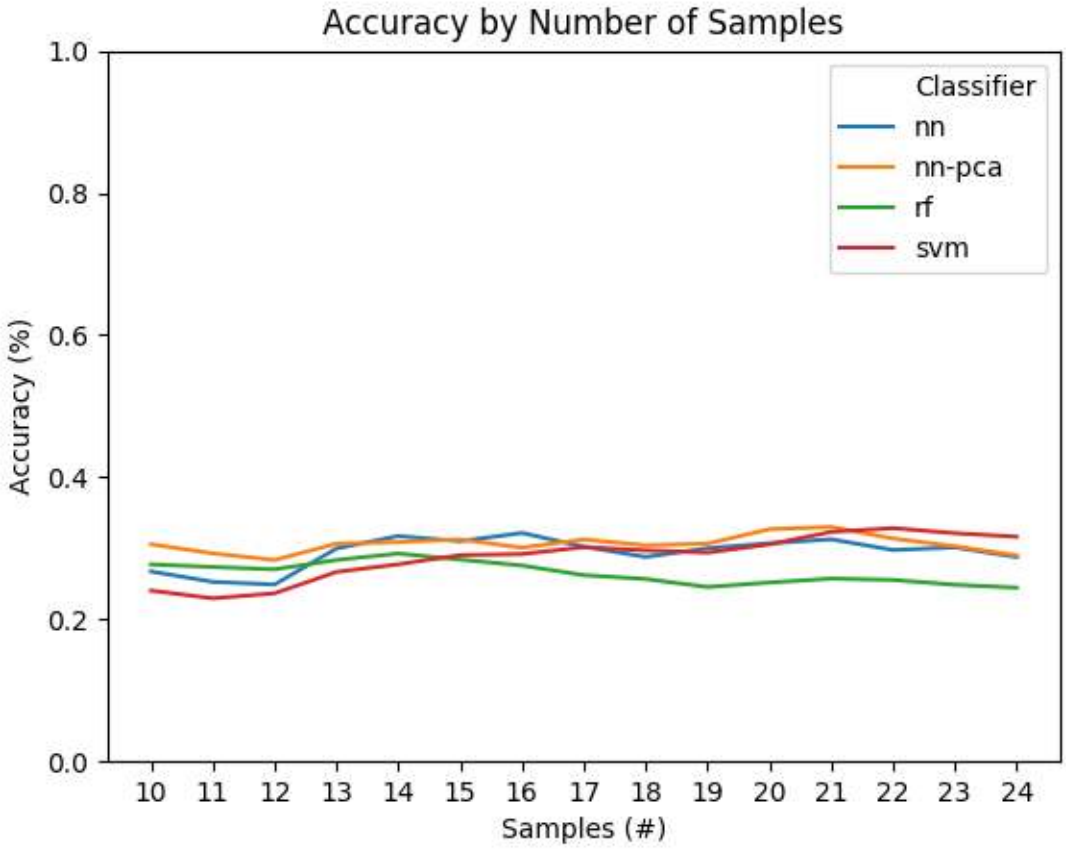


Figure 4.54: Accuracy by number of samples

The results show a modest improvement in overall accuracy. SVM in particular benefited the most from the additional data, while RF suffered. Why RF had worse performance despite more data is unclear. How far this slight upward tend extends is also unclear from the amount of data we have gathered. Because of this, future research would almost certainly obtain better results with a well-trained perceptron network.

Chapter 5

CONCLUSION

With our limited data set, we were able to prove this MitM attack is a viable threat to organizations who have Secure Shell connections traveling over an untrusted network.

While the accuracy was likely to identify an individual user, it demonstrated a reasonable threat that a particular user could be identified and a near certainty that one of many users could be identified reliably.

In particular, the K-NN method would allow an attacker to have over 50% accuracy within 20% of the search space. If an attacker could achieve their goals by identifying any one of a large number of possible victims, they could most likely achieve this.

We found that the K-Nearest Neighbors and Support Vector Machine classifiers were able to classify users with significant accuracy and that the Random Forest classifier could perform comparably. The amount of data available to an attacker makes a difference in their ability to identify the user but even small numbers of samples could provide meaningful results. The data also show that the attack can scale well with the number of users, and in our data set the classifiers were impacted more strongly by the diversity of the participants than the number.

Additionally, we find that it is significantly easier to identify a user based on a single type of activity rather than a mix.

Recall that all of the data collected in this study was performed with a live user at the keyboard and all network signals were measured at a route between the victim and server thus capturing real network noise. Given the conceit that the attacker

has placed themselves as a MitM and acquired an existing dataset, we believe this accurately represents the realistic complications of such an attack. While the ability of the attacker to inject themselves in the routing path is well proven, existing dataset about typing patterns are less accessible [24]. While some organizations could collect such a dataset, future works may be able to expand on this research to include unsupervised and semi-supervised learning to eliminate the necessity this conceit.

5.1 Limitations

Several key factors limited the ability to extrapolate these results.

One key factor was a lack of data. A tradeoff of our recruitment methods was to prioritize diversity of users over control of how much data was provided by being flexible in our minimum requirements. Although we were able to acquire recordings from many different users, if we wanted to use all of them, we had to limit the amount of data available per user. As shown in the previous section, we cannot know how much impact this had on our study without further research.

Additionally, this study drew its participants from Cal Poly Computer Science and Computer Engineering students. It is unclear whether or not the results from these student who all come from the same educational background sufficiently represent the diversity of a real work environment. Not only this, but students having less experience with shell terminals may have different habits and behaviors than a seasoned expert. Such experts would be the more valuable targets in an attempt to glean useful information.

5.2 Countermeasures

This attack is difficult to prevent because it relies exclusively on the metadata of the packets travelling over the network.

One potential solution is to introduce a random delay before sending the packet to obfuscate when a key is pressed. Unfortunately since all of the timing statistics were amortized, this would most likely in the best case make such an attack harder rather than fully prevent it. In the worst case, this may be completely ineffective. Additionally, any significant delay will likely impact the user's experience using Secure Shell.

On the other hand, the paste size statistics can be easily countered by padding the packets to a larger size. This does consume more bandwidth as each packet must contain extra overhead.

A compromise between these two solutions may be to buffer keystrokes and flush the buffer at a fixed time interval. This could significantly reduce the load on the network as at the time of writing each packet is padded to an 8-byte block despite most packets containing only one meaningful byte. This would reduce the resolution of an attacker's information as they would now only receive a range for number of keys pressed within a certain time span. Unfortunately, without testing this it is unclear if such restrictions would prevent an attacker from classifying users. Additionally depending on the length of the time interval, users may become impatient. A shorter time interval would minimize this problem but would also minimize the benefits.

The best countermeasure would be to only send packets when the server has a meaningful response rather than naively sending every keystroke. By default, the shell operates in canonical mode, only receiving data after new lines and other events

which have been configured to trigger a response such as signals [28]. Until the end of line is triggered or a new line entered, the input sits in a buffer. The current implementation of Secure Shell has the client operate in noncanonical mode and lets the remote terminal handle buffering. If the Secure Shell client was modified to operate in canonical mode matching the remote virtual terminal, then the buffered data would not be vulnerable to attacks. But this countermeasure would be a significant change to the current model. For example, this would require the remote server to detect changes to the terminal settings and notify the client so that it can mirror the new settings. Additionally, this model would not protect programs which naturally run in noncanonical mode such as emacs or vi. On the other hand, both of these programs currently have integrated Secure Shell support which minimizes network usage already. A Smart Secure Shell could take advantage of this integration to launch the editor locally, but this does not help for the wide variety of programs which operate in noncanonical mode and do not have integrated Secure Shell support. Finally, this too would leak a certain amount of information, as an attacker could still know how frequently the victim types a line and the length of the line to the nearest block. This research does not make clear whether or not such an attack would be still be viable with this type of integration.

5.3 Future Work

We do not believe we have achieved the best results we can with the data we have gathered. Our initial findings show that more data would likely help to improve model accuracy. Additionally, it would be beneficial to use data from more seasoned experts in industry.

Although we were not able to train the perceptron network, we suspect that it should be able to outperform the K-NN classifier when properly hypertuned. Because of this, future research would almost certainly obtain better results with a well-trained perceptron network.

In addition to this, there is an opening to research the potential countermeasures discussed in the prior section, to what extent they hinder an attacker's ability to identify a victim, and the impact to usability.

On the other hand, an attack like this could work and would be much more useful if the model could train on an untagged data set. Unsupervised or semi-supervised versions of this attack is another area which we did not have the time to examine in this study. Such models would more realistically represent an attacker's limited access to an existing pool of data.

Finally, we believe it may be possible that a convolutional neural network could learn to identify a user without preprocessing it. This would allow the classifier to identify features the developer did not anticipate. But such an approach would require significantly more data than was available to us for this research, and so we leave it to future works.

BIBLIOGRAPHY

[1] M. R. Albrecht, K. G. Paterson, and G. J. Watson. Plaintext recovery attacks against ssh. In *2009 30th IEEE Symposium on Security and Privacy*, pages 16–26. IEEE, May 2009. doi: 10.1109/SP.2009.5. URL `http://www.isg.rhul.ac.uk/~kp/SandPfinal.pdf`.

[2] M. Y. Arslan, K. Pelechrinis, I. Broustis, S. V. Krishnamurthy, P. Krishnamurthy, and P. Mohapatra. Detecting route attraction attacks in wireless networks. In *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 371–380, Oct 2011. doi: 10.1109/MASS.2011.44.

[3] Harikrishnan Bhanu. Timing side-channel attacks on ssh. Master's thesis, Clemson University, 2010. URL `https://tigerprints.clemson.edu/all_theses/781`.

[4] Comodo. Ip lookup. "Online", 2020. URL `https://www.ipaddress.com/ip-lookup`. Accessed: 2020-01-30.

[5] M. Conti, N. Dragoni, and V. Lesyk. A survey of man in the middle attacks. *IEEE Communications Surveys Tutorials*, 18(3):2027–2051, thirdquarter 2016. ISSN 1553-877X. doi: 10.1109/COMST.2016.2548426.

[6] Scott Crosby, Dan Wallach, and Rudolf Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur.*, 12, 01 2009. doi: 10.1145/1455526.1455530.

[7] Ronald Deibert and Rafal Rohozinski. Control and subversion in russian cyberspace. *Access controlled: The shaping of power, rights, and rule in cyberspace*, pages 15–34, 2010.

[8] Charles Dickens. *A Tale of Two Cities*. Chapman & Hall, 1859.

[9] B.S. Everitt. *Cambridge Dictionary of Statistics*, chapter Markov Chain, page 234. Cambridge University Press, 2002.

[10] OpenBSD Foundation. Openssh. "Online", 2020. URL `https://www.openbsd.org/openssh/users.html`.

[11] Elise Gerich. Iana ipv4 address space registry. "Online", December 2019. URL `https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml`. Accessed 2020-02-20.

[12] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In David E. Losada and Juan M. Fernández-Luna, editors, *Advances in Information Retrieval*, pages 345–359, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31865-1.

[13] Imperva. Man in the middle (mitm) attack. Accessed 2019-01-16, 2020. URL `https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/`.

[14] Amazon.com Inc. Amazon.com egift card. "Online", 2020. URL `https://www.amazon.com/dp/B07PCMWTSG/ref=s9_acsd_ri_bw_r2_GCRANK_1_t?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=merchandised-search-12&pf_rd_r=K8FG0P2K8MA5HN15AZSM&pf_rd_t=101&pf_rd_p=042f6344-e5c2-4d19-bc35-ccfad9f3308a&pf_rd_i=2238192011`. Accessed: 2020-01-30.

[15] Amazon.com Inc. Aws. "Online", 2020. URL `https://aws.amazon.com/`. Accessed: 2020-01-30.

[16] Facebook.com Inc. Terms of serivce, 2020. URL `https://www.facebook.com/legal/terms`. Accessed: 2020-01-30.

[17] OpenVPN Inc. Openvpn. "Online", 2020. URL https://openvpn.net/. Accessed: 2020-01-30.

[18] VMware Inc. Vmware. "Online", 2020. URL https://www.vmware.com/. Accessed: 2020-01-30.

[19] Tom N Jagatic, Nathaniel A Johnson, Markus Jakobsson, and Filippo Menczer. Social phishing. *Communications of the ACM*, 50(10):94–100, 2007.

[20] Markku Kojo Kumiko Ono Martin Stiemerling Lars Eggert Alexey Melnikov Wes Eddy Alexander Zimmermann Brian Trammell Jana Iyengar Allison Mankin Michael Tuexen Eddie Kohler Joe Touch; Eliot Lear, Allison Mankin and Yoshifumi Nishida. Service name and transport protocol port number registry. "Online", January 2020. URL https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. Accessed 2020-01-30.

[21] Carl Trieloff Free Seat Jim Perrin Johnny Hughes Mike McLean Ralph Angenendt Karanbir Singh, Karsten Wade and Tru Huynh. The centos project. "Online", 2020. URL https://www.centos.org/. Accessed: 2020-01-30.

[22] F. Khosmood, P. L. Nico, and J. Woolery. User identification through command history analysis. In *2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 1–7, 445 Hoes Lane, Piscataway, NJ 08854-4141 USA, Dec 2014. IEEE. doi: 10.1109/CICYBS.2014.7013363.

[23] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[24] Satoshi Kurosawa, Hidehisa Nakayama, Nei Kato, Abbas Jamalipour, and Yoshiaki Nemoto. Detecting blackhole attack on aodv-based mobile ad hoc networks by dynamic learning method. *IJ Network Security*, 5(3):338–346, 2007.

[25] Sam Leroux, Steven Bohez, Pieter-Jan Maenhaut, Nathan Meheus, Pieter Simoens, and Bart Dhoedt. Fingerprinting encrypted network traffic types using machine learning. *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018. doi: 10.1109/noms.2018.8406218.

[26] Jonathan Lewis. What it auditors need to know about secure shell. Slide 6, Accessed 2019-04-15, June 2014. URL `https://www.isaca.org/chapters3/Charlotte/Events/Documents/Event%20Presentations/06162014/ISACA%20presentation%20june%202014.pdf`.

[27] Krista Longi, Juho Leinonen, Henrik Nygren, Joni Salmi, Arto Klami, and Arto Vihavainen. Identification of programmers from typing patterns. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, pages 60–67, 2015. ISBN 978-1-4503-4020-5. URL `http://doi.acm.org/10.1145/2828959.2828960`.

[28] Sandra Loosemore, Richard M. Stallman, and Andrew Oram. The gnu c library reference manualthe gnu c library reference manual, 2020.

[29] Canonical Ltd. Ubuntu. "Online", 2020. URL `https://www.ubuntu.org/`. Accessed: 2020-01-30.

[30] Luis MartinGarcia. Tcpdump & libpcap. "Online", 2020. URL `https://www.tcpdump.org/`. Accessed: 2020-01-30.

[31] Donald Michie, David J Spiegelhalter, CC Taylor, et al. Machine learning. *Neural and Statistical Classification*, 13, 1994.

[32] J. V. Monaco. Feasibility of a keystroke timing attack on search engines with autocomplete. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 212–217, May 2019. doi: 10.1109/SPW.2019.00047.

[33] California Polytechnic State University San Luis Obispo. 2019-2020 catalog. "Online", 2020. URL `http://catalog.calpoly.edu/coursesaz/csc/`. Accessed: 2020-01-30.

[34] Information Sciences Institute University of Southern California. Internet protocol. Rfc, Defense Advanced Research Projects Agency, September 1981. URL `https://tools.ietf.org/html/rfc791`. Accessed 2019-04-15.

[35] Information Sciences Institute University of Southern California. Transmission control protocol. Rfc, Defense Advanced Research Projects Agency, September 1981. URL `https://tools.ietf.org/html/rfc793`. Accessed 2019-09-04.

[36] Wireshark Organization. Wireshark. "Online", 2020. URL `https://www.wireshark.org/`. Accessed: 2020-01-30.

[37] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[38] M. Pleva, E. Kiktova, P. Viszlay, and P. Bours. Acoustical keystroke analysis for user identification and authentication. In *2016 26th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 386–389, 445 Hoes Lane, Piscataway, NJ 08854-4141 USA, April 2016. IEEE. doi: 10.1109/RADIOELEK.2016.7477360.

[39] Sebastian Raschka. Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack. *The Journal*

of Open Source Software, 3(24), April 2018. doi: 10.21105/joss.00638. URL http://joss.theoj.org/papers/10.21105/joss.00638.

[40] Dennis W Ruck, Steven K Rogers, and Matthew Kabrisky. Feature selection using a multilayer perceptron. *Journal of Neural Network Computing*, 2(2):40–48, 1990.

[41] Pascal Schmidt. A detailed introduction to cross-validation in machine learning. Accessed 2019-01-16, 10 2018. URL https://thatdatatho.com/2018/10/11/detailed-introduction-cross-validation-machine-learning/.

[42] K. N. Shiv Subramaniam, S. Raj Bharath, and S. Ravinder. Improved authentication mechanism using keystroke analysis. In *2007 International Conference on Information and Communication Technology*, pages 258–261, 445 Hoes Lane, Piscataway, NJ 08854-4141 USA, March 2007. IEEE. doi: 10.1109/ICICT.2007.375389.

[43] Dawn Xiaodong Song, David A Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium*, volume 2001, 2001. URL https://people.eecs.berkeley.edu/~daw/papers/ssh-use01.pdf.

[44] Ed. Tatu Ylonen, C. Lonvick. Ssh transport layer protocol. Rfc, SSH Communications Security Corp, Cisco Systems, Inc., January 2006. URL https://tools.ietf.org/html/rfc4253. Accessed 2019-04-15.

[45] Ed. Tatu Ylonen, C. Lonvick. The secure shell (ssh) connection protocol. Rfc, SSH Communications Security Corp, Cisco Systems, Inc., January 2006. URL https://tools.ietf.org/html/rfc4254. Accessed 2020-02-18.

[46] Ed. Tatu Ylonen, C. Lonvick. The secure shell (ssh) protocol architecture. Rfc, SSH Communications Security Corp, Cisco Systems, Inc., January 2006. URL https://tools.ietf.org/html/rfc4251. Accessed 2019-09-03.

[47] Lyle Ungar. Decision trees. Accessed 2020-01-16, 09 2019. URL https://alliance.seas.upenn.edu/~cis520/wiki/index.php?n=Lectures.DecisionTrees.

[48] Wikipedia contributors. P-value — Wikipedia, the free encyclopedia. "Online", 2014. URL https://en.wikipedia.org/wiki/P-value. [Online; accessed 2020-01-16].

[49] Tatu Ylonen. *ssh(1) Linux User's Manual*. BSD, openssh7.6p1 edition, September 2017. Accessed 2019-04-15.

[50] Gokmen Zararsiz, Ferhan Elmali, and Ahmet Ozturk. Bagging support vector machines for leukemia classification. *IJCSI International Journal of Computer Science Issues*, 9:355–358, 11 2012.

[51] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Trans. Inf. Syst. Secur.*, 13(1):3:1–3:26, November 2009. ISSN 1094-9224. doi: 10.1145/1609956.1609959. URL http://doi.acm.org/10.1145/1609956.1609959.

APPENDICES

Appendix A

INSTRUCTIONS FOR PARTICIPANTS (DATASET 1)

A.1 Protocol

1. Provide the informed consent form and describe the nature of the study.
2. Provide the Loose-Instructed English prompts
3. Provide the Strict-Instructed English prompts
4. Provide the Loose-Instructed Unix Shell prompts
5. Provide the Strict-Instructed Unix Shell prompts
6. Tag data with sha256 hash of Cal Poly username

A.2 Loose-Instructed English Language

Please answer the following prompts with a complete, English sentence.

1. Describe a T.V. show character.
2. What is the weather outside today?
3. Describe the inside of a restaurant.
4. Who is the current president of the United States?

5. Describe to an extra terrestrial how to use a microwave oven.

6. Describe the time without using a clock.

7. What is the name of a sport team and what sport do they play?

8. Choose an animal and give a brief description of it.

A.3 Strict-Instructed English Language

Write the following paragraph:

> It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way–in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only.

A.4 Loose-Instructed Shell

Perform the following tasks over an SSH session

1. Create a directory named 'foo'

2. Create an empty file in foo named 'bar'

3. Write a C program in foo which prints "Hello World" to the terminal
4. Compile the program to an executable named, "greet"
5. Run greet
6. Run greet and redirect the output to a file in foo
7. Check that the output from greet and foo/bar are different
8. Delete foo/bar
9. Print your current directory
10. Make a subdirectory of foo named bar/biz/boo
11. Print your current directory
12. Change your current directory to boo
13. Write "Hello World" to a file
14. Compare that file with the output of greet
15. Print the contents of the foo directory
16. Change your current directory to foo

A.5 Strict-Instructed Shell

Enter the following commands over an SSH session

1. `mkdir testsess1`
2. `whoami`

3. `touch testsess1/file.txt`
4. `echo Hello World > testsess1/file.txt`
5. `cp --verbose testsess1/file.txt testsess1/copy.txt`
6. `diff -q testsess1/file.txt testsess1/copy.txt`
7. `basename testsess1/copy.txt`
8. `egrep -i wo testsess1/copy.txt`
9. `cat > source.c`

```
int main() {
  int a = 7;
  int b = 2;
  return a - b;
}
```

10. `gcc -g source.c`
11. `./a.out`
12. `echo $?`
13. `mv a.out testprog`
14. `zip -j out.zip testsess1 source.c`
15. `which vim`
16. `wc --help`
17. `wc -c source.c`
18. `quit`

Appendix B

INVESTIGATING SSH INFORMATION LEAKS (DATASET 1)

This form asks for your agreement to participate in a research project for SecureShell. Your participation involves using the SecureShell program over a monitored network to type sentences and run commands from the attached document.

These tasks are expected to take approximately 30 minutes. If you stop early your data may not be used. There are no risks anticipated with your participation. Those in the cybersecurity, system administration and software development industries may benefit from your participation. If you are interested in participating, please review the following information:

The purpose of the study is to examine if the pattern of network traffic from SecureShell is unique to a user such that an outside party can identify them. Additionally, this study will examine if the timing of publicly visible network traffic can be used to identify keystrokes. Potential benefits associated with the study include improvements in secure internet communications.

If you agree to participate, you will be asked to complete the attached set of instructions. This research involves finding vulnerabilities in the SecureShell protocol and as such anything you type in the SecureShell application as part of this study will be recorded for validation purposes. You are advised to not type any passwords or otherwise sensitive information into a terminal being used as part of this study.

Please be aware that you are not required to participate in this research, refusal to participate will not involve any penalty or loss of benefits to which you are otherwise entitled, and you may discontinue your participation at any time. Your participation

or refusal will not contribute or impact your grade in any of your classes. The focus of this study is on the behavior of the SecureShell system and thus your responses to prompts and inputs to the terminal are only incidentally involved in the study. There are no risks anticipated with your participation in this study, as your data will be maintained confidentially.

As compensation for your participation in this study, you may opt-in to be automatically enrolled in a raffle for an Amazon gift card redeemable for $50 on Amazon.com. In accordance with California Law, an individuals may also enter the raffle by requesting a URL from an investigator and entering an email into the online prompt. The winner of the raffle will be contacted through the email address provided below or through the online interface. Upon completion of the study, each participant is expected to have a 1 in 30 chance of winning the raffle. The winner will have 7 days to accept or be removed from the pool of applicants.

This research is being conducted by Dr. Bruce DeBruhl and graduate student Thomas Flucke in the Department of Computer Science and Software Engineering at Cal Poly, San Luis Obispo. If you have questions regarding this study or would like to be informed of the results when the study is completed, please contact Dr. DeBruhl at `bdebruhl@calpoly.edu`. If you have concerns regarding the manner in which the study is conducted, you may contact Dr. Michael Black, Chair of the Cal Poly Institutional Review Board, at (805) 756-2894, `mblack@calpoly.edu`, or Ms. Debbie Hart, Compliance Officer, at (805) 756-1508, `dahart@calpoly.edu`.

If you agree to voluntarily participate in this research project as described, please indicate your agreement by performing one or more tasks from above when the researcher tells you. Please keep a copy of this form for your reference, and thank you for your participation in this research.

Appendix C

INVESTIGATING SSH INFORMATION LEAKS (DATASET 2)

This form asks for your agreement to participate in a research project for SecureShell. Your participation involves completing your class work using the SecureShell program over a monitored network from a virtual machine.

The experiment is not expected to hinder your ability to complete your assignments and will not require more than a negligible amount of time to participate beyond that you would normally take to do your class work. If you stop early your data may not be used. There are no risks anticipated with your participation. Those in the cybersecurity, system administration and software development industries may benefit from your participation. If you are interested in participating, please review the following information:

The purpose of the study is to examine if the pattern of network traffic from SecureShell is unique to a user such that an outside party can identify them. Additionally, this study will examine if the timing of publicly visible network traffic can be used to identify keystrokes. Potential benefits associated with the study include improvements in secure internet communications.

If you agree to participate, you will be asked to use the provided virtual machine to complete your classwork. This research involves finding vulnerabilities in the SecureShell protocol and as such anything you type in the SecureShell application as part of this study will be recorded for validation purposes. You are advised to not type any passwords or otherwise sensitive information into a terminal being used as part of this study.

Please be aware that you are not required to participate in this research, refusal to participate will not involve any penalty or loss of benefits to which you are otherwise entitled, and you may discontinue your participation at any time. Your participation or refusal will not contribute or impact your grade in any of your classes. The focus of this study is on the behavior of the SecureShell system and thus your inputs to the terminal are only incidentally involved in the study. There are no risks anticipated with your participation in this study, as your data will be maintained confidentially.

As compensation for your participation in this study, you may opt-in to be automatically enrolled in a raffle for an Amazon gift card redeemable for $50 on amazon.com. In accordance with California Law, an individuals may also enter the raffle by requesting a URL from an investigator and entering an email into the online prompt. The winner of the raffle will be contacted through the email address provided below or through the online interface. Upon completion of the study, each participant is expected to have a 1 in 30 chance of winning the raffle. The winner will have 7 days to accept or be removed from the pool of applicants.

This research is being conducted by Dr. Bruce DeBruhl and graduate student Thomas Flucke in the Department of Computer Science and Software Engineering at Cal Poly, San Luis Obispo. If you have questions regarding this study or would like to be informed of the results when the study is completed, please contact Dr. DeBruhl at `bdebruhl@calpoly.edu`. If you have concerns regarding the manner in which the study is conducted, you may contact Dr. Michael Black, Chair of the Cal Poly Institutional Review Board, at (805) 756-2894, `mblack@calpoly.edu`, or Ms. Debbie Hart, Compliance Officer, at (805) 756-1508, `dahart@calpoly.edu`.

If you agree to voluntarily participate in this research project as described, please indicate your agreement by performing one or more tasks from above when the re-

searcher tells you. Please keep a copy of this form for your reference, and thank you for your participation in this research.

Appendix D

VOS SSH WRAPPER SCRIPT

```
readonly SERVER="keylog@tf2.cscaws.com"
readonly PROG_NAME=$(basename $0)
readonly ORIG_PROG=$(which -a $PROG_NAME | tail -n-1)
readonly LOG_DIR="/var/log/thesis"
readonly KEY_FILE=~/.ssh/keylog_rsa
readonly DATE="$(date '+%Y%m%d%H%M%S')"
readonly OUT_FILE="$LOG_DIR/$PROG_NAME-log-$DATE.log"
for arg in $@; do
    uid="$(echo $arg | cut -d@ -f1 -s)"
    if [ "$uid" = "" ]; then
    break;
    fi
done
uid=$(echo "$uid" | sha256sum | cut -d' ' -f1)
$ORIG_PROG "$SERVER" "$uid" 2> /dev/null &

promptSubmit() {
  echo "Do you aprove this data to be submitted for
     this study? [y|n]"
  read REPLY

  if [ $REPLY = "y" ]; then
```

```
    scp -i "$KEY_FILE" "$OUT_FILE" "$SERVER":"$uid"
  elif [ $REPLY != "n" ]; then
    echo "Invalid response"
    promptSubmit
  fi
}

mkdir -p $LOG_DIR

echo "Reminder not to enter passwords into this SSH
   session."
strace -ttt -e trace=read -e read=0 -s200 -o $OUT_FILE
    \
        $ORIG_PROG $@

sed -i '2,${ /read([0-9], ".*", 16384)/!d }' "
   $OUT_FILE"

less "$OUT_FILE"
promptSubmit
```

Appendix E

VOS OPENVPN CONFIGURATION FILE

```
##############################################
# Sample client-side OpenVPN 2.0 config file #
# for connecting to multi-client server.     #
#                                            #
# This configuration can be used by multiple #
# clients, however each client should have   #
# its own cert and key files.                #
#                                            #
# On Windows, you might want to rename this  #
# file so it has a .ovpn extension           #
##############################################

# Specify that we are a client and that we
# will be pulling certain config file directives
# from the server.
client

# Use the same setting as you are using on
# the server.
# On most systems, the VPN will not function
# unless you partially or fully disable
# the firewall for the TUN/TAP interface.
;dev tap
```

```
dev tun

# Windows needs the TAP-Win32 adapter name
# from the Network Connections panel
# if you have more than one.  On XP SP2,
# you may need to disable the firewall
# for the TAP adapter.
;dev-node MyTap

# Are we connecting to a TCP or
# UDP server?  Use the same setting as
# on the server.
;proto tcp
proto udp

# The hostname/IP and port of the server.
# You can have multiple remote entries
# to load balance between the servers.
remote tf2.cscaws.com 1194
;remote my-server-2 1194

# Choose a random host from the remote
# list for load-balancing.  Otherwise
# try hosts in the order specified.
;remote-random

# Keep trying indefinitely to resolve the
```

```
# host name of the OpenVPN server.  Very useful
# on machines which are not permanently connected
# to the internet such as laptops.
resolv-retry infinite

# Most clients don't need to bind to
# a specific local port number.
nobind

# Downgrade privileges after initialization (non-Windows only)
user nobody
group nogroup

# Try to preserve some state across restarts.
persist-key
persist-tun

# If you are connecting through an
# HTTP proxy to reach the actual OpenVPN
# server, put the proxy server/IP and
# port number here.  See the man page
# if your proxy server requires
# authentication.
;http-proxy-retry # retry on connection failures
;http-proxy [proxy server] [proxy port #]

# Wireless networks often produce a lot
```

```
# of duplicate packets.  Set this flag
# to silence duplicate packet warnings.
;mute-replay-warnings

# SSL/TLS parms.
# See the server config file for more
# description.  It's best to use
# a separate .crt/.key file pair
# for each client.  A single ca
# file can be used for all clients.
ca ca.crt
cert client1.crt
key client1.key

script-security 1

# Verify server certificate by checking that the
# certicate has the correct key usage set.
# This is an important precaution to protect against
# a potential attack discussed here:
#  http://openvpn.net/howto.html#mitm
#
# To use this feature, you will need to generate
# your server certificates with the keyUsage set to
#   digitalSignature, keyEncipherment
# and the extendedKeyUsage to
#   serverAuth
```

```
# EasyRSA can do this for you.
remote-cert-tls server

# If a tls-auth key is used on the server
# then every client must also have the key.
;tls-auth ta.key 1

# Select a cryptographic cipher.
# If the cipher option is used on the server
# then you must also specify it here.
cipher none
auth none

# Enable compression on the VPN link.
# Don't enable this unless it is also
# enabled in the server config file.
;comp-lzo

# Set log file verbosity.
verb 3

# Silence repeating messages
;mute 20
```

Appendix F

MITM OPENVPN CONFIGURATION FILE

```
################################################
# Sample OpenVPN 2.0 config file for           #
# multi-client server.                         #
#                                              #
# This file is for the server side             #
# of a many-clients <-> one-server             #
# OpenVPN configuration.                       #
#                                              #
# OpenVPN also supports                        #
# single-machine <-> single-machine            #
# configurations (See the Examples page        #
# on the web site for more info).              #
#                                              #
# This config should work on Windows           #
# or Linux/BSD systems.  Remember on           #
# Windows to quote pathnames and use           #
# double backslashes, e.g.:                    #
# "C:\Program Files\OpenVPN\config\foo.key" #
#                                              #
# Comments are preceded with '#' or ';'        #
################################################

# Which local IP address should OpenVPN use
```

```
;local a.b.c.d

# Which TCP/UDP port should OpenVPN listen on?
# If you want to run multiple OpenVPN instances
# on the same machine, use a different port
# number for each one.  You will need to
# open up this port on your firewall.
port 1194

# TCP or UDP server?
;proto tcp
proto udp

# "dev tun" will create a routed IP tunnel,
# "dev tap" will create an ethernet tunnel.
# Use "dev tap0" if you are ethernet bridging
# and have precreated a tap0 virtual interface
# and bridged it with your ethernet interface.
# If you want to control access policies
# over the VPN, you must create firewall
# rules for the the TUN/TAP interface.
# On non-Windows systems, you can give
# an explicit unit number, such as tun0.
# On Windows, use "dev-node" for this.
# On most systems, the VPN will not function
# unless you partially or fully disable
# the firewall for the TUN/TAP interface.
```

```
;dev tap
dev tun

# Windows needs the TAP-Win32 adapter name
# from the Network Connections panel if you
# have more than one.  On XP SP2 or higher,
# you may need to selectively disable the
# Windows firewall for the TAP adapter.
# Non-Windows systems usually don't need this.
;dev-node MyTap

# SSL/TLS root certificate (ca), certificate
# (cert), and private key (key).  Each client
# and the server must have their own cert and
# key file.  The server and all clients will
# use the same ca file.
#
# See the "easy-rsa" directory for a series
# of scripts for generating RSA certificates
# and private keys.  Remember to use
# a unique Common Name for the server
# and each of the client certificates.
#
# Any X509 key management system can be used.
# OpenVPN can also use a PKCS #12 formatted key file
# (see "pkcs12" directive in man page).
ca ca.crt
```

```
cert ThesisVPN.crt
key ThesisVPN.key  # This file should be kept secret

# Diffie hellman parameters.
# Generate your own with:
#   openssl dhparam -out dh2048.pem 2048
dh dh2048.pem

# Network topology
# Should be subnet (addressing via IP)
# unless Windows clients v2.0.9 and lower have to
# be supported (then net30, i.e. a /30 per client)
# Defaults to net30 (not recommended)
topology subnet

# Configure server mode and supply a VPN subnet
# for OpenVPN to draw client addresses from.
# The server will take 10.8.0.1 for itself,
# the rest will be made available to clients.
# Each client will be able to reach the server
# on 10.8.0.1. Comment this line out if you are
# ethernet bridging. See the man page for more info.
server 10.8.0.0 255.255.255.0

# Maintain a record of client <-> virtual IP address
# associations in this file.  If OpenVPN goes down or
# is restarted, reconnecting clients can be assigned
```

```
# the same virtual IP address from the pool that was
# previously assigned.
ifconfig-pool-persist /var/log/openvpn/ipp.txt

# Configure server mode for ethernet bridging.
# You must first use your OS's bridging capability
# to bridge the TAP interface with the ethernet
# NIC interface.  Then you must manually set the
# IP/netmask on the bridge interface, here we
# assume 10.8.0.4/255.255.255.0.  Finally we
# must set aside an IP range in this subnet
# (start=10.8.0.50 end=10.8.0.100) to allocate
# to connecting clients.  Leave this line commented
# out unless you are ethernet bridging.
;server-bridge 10.8.0.4 255.255.255.0 10.8.0.50 10.8.0.100

# Configure server mode for ethernet bridging
# using a DHCP-proxy, where clients talk
# to the OpenVPN server-side DHCP server
# to receive their IP address allocation
# and DNS server addresses.  You must first use
# your OS's bridging capability to bridge the TAP
# interface with the ethernet NIC interface.
# Note: this mode only works on clients (such as
# Windows), where the client-side TAP adapter is
# bound to a DHCP client.
;server-bridge
```

```
# Push routes to the client to allow it
# to reach other private subnets behind
# the server.  Remember that these
# private subnets will also need
# to know to route the OpenVPN client
# address pool (10.8.0.0/255.255.255.0)
# back to the OpenVPN server.
;push "route 192.168.10.0 255.255.255.0"
;push "route 192.168.20.0 255.255.255.0"

# To assign specific IP addresses to specific
# clients or if a connecting client has a private
# subnet behind it that should also have VPN access,
# use the subdirectory "ccd" for client-specific
# configuration files (see man page for more info).

# EXAMPLE: Suppose the client
# having the certificate common name "Thelonious"
# also has a small subnet behind his connecting
# machine, such as 192.168.40.128/255.255.255.248.
# First, uncomment out these lines:
;client-config-dir ccd
;route 192.168.40.128 255.255.255.248
# Then create a file ccd/Thelonious with this line:
#   iroute 192.168.40.128 255.255.255.248
# This will allow Thelonious' private subnet to
```

```
# access the VPN.  This example will only work
# if you are routing, not bridging, i.e. you are
# using "dev tun" and "server" directives.

# EXAMPLE: Suppose you want to give
# Thelonious a fixed VPN IP address of 10.9.0.1.
# First uncomment out these lines:
;client-config-dir ccd
;route 10.9.0.0 255.255.255.252
# Then add this line to ccd/Thelonious:
#   ifconfig-push 10.9.0.1 10.9.0.2

# Suppose that you want to enable different
# firewall access policies for different groups
# of clients.  There are two methods:
# (1) Run multiple OpenVPN daemons, one for each
#     group, and firewall the TUN/TAP interface
#     for each group/daemon appropriately.
# (2) (Advanced) Create a script to dynamically
#     modify the firewall in response to access
#     from different clients.  See man
#     page for more info on learn-address script.
;learn-address ./script

# If enabled, this directive will configure
# all clients to redirect their default
# network gateway through the VPN, causing
```

```
# all IP traffic such as web browsing and
# and DNS lookups to go through the VPN
# (The OpenVPN server machine may need to NAT
# or bridge the TUN/TAP interface to the internet
# in order for this to work properly).
push "redirect-gateway def1 bypass-dhcp"

# Certain Windows-specific network settings
# can be pushed to clients, such as DNS
# or WINS server addresses.  CAVEAT:
# http://openvpn.net/faq.html#dhcpcaveats
# The addresses below refer to the public
# DNS servers provided by opendns.com.
;push "dhcp-option DNS 208.67.222.222"
;push "dhcp-option DNS 208.67.220.220"

# Uncomment this directive to allow different
# clients to be able to "see" each other.
# By default, clients will only see the server.
# To force clients to only see the server, you
# will also need to appropriately firewall the
# server's TUN/TAP interface.
;client-to-client

# Uncomment this directive if multiple clients
# might connect with the same certificate/key
# files or common names.  This is recommended
```

```
# only for testing purposes.  For production use,
# each client should have its own certificate/key
# pair.
#
# IF YOU HAVE NOT GENERATED INDIVIDUAL
# CERTIFICATE/KEY PAIRS FOR EACH CLIENT,
# EACH HAVING ITS OWN UNIQUE "COMMON NAME",
# UNCOMMENT THIS LINE OUT.
duplicate-cn

# The keepalive directive causes ping-like
# messages to be sent back and forth over
# the link so that each side knows when
# the other side has gone down.
# Ping every 10 seconds, assume that remote
# peer is down if no ping received during
# a 120 second time period.
keepalive 10 120

# For extra security beyond that provided
# by SSL/TLS, create an "HMAC firewall"
# to help block DoS attacks and UDP port flooding.
#
# Generate with:
#   openvpn --genkey --secret ta.key
#
# The server and each client must have
```

```
# a copy of this key.
# The second parameter should be '0'
# on the server and '1' on the clients.
;tls-auth ta.key 0 # This file is secret

# Select a cryptographic cipher.
# This config item must be copied to
# the client config file as well.
# Note that v2.4 client/server will automatically
# negotiate AES-256-GCM in TLS mode.
# See also the ncp-cipher option in the manpage
;cipher AES-256-CBC
# Disables all encryption/authentification
# We don't actually want to add extra protections,
# we just want to reroute the packets.
# Disabling encryption reduces overhead/variables.
auth none
cipher none

# Enable compression on the VPN link and push the
# option to the client (v2.4+ only, for earlier
# versions see below)
;compress lz4-v2
;push "compress lz4-v2"

# For compression compatible with older clients use comp-lzo
# If you enable it here, you must also
```

```
# enable it in the client config file.
;comp-lzo

# The maximum number of concurrently connected
# clients we want to allow.
;max-clients 100

# It's a good idea to reduce the OpenVPN
# daemon's privileges after initialization.
#
# You can uncomment this out on
# non-Windows systems.
user nobody
group nogroup

# The persist options will try to avoid
# accessing certain resources on restart
# that may no longer be accessible because
# of the privilege downgrade.
persist-key
persist-tun

# Output a short status file showing
# current connections, truncated
# and rewritten every minute.
status /var/log/openvpn/openvpn-status.log
```

```
# By default, log messages will go to the syslog (or
# on Windows, if running as a service, they will go to
# the "\Program Files\OpenVPN\log" directory).
# Use log or log-append to override this default.
# "log" will truncate the log file on OpenVPN startup,
# while "log-append" will append to it.  Use one
# or the other (but not both).
;log         /var/log/openvpn/openvpn.log
;log-append  /var/log/openvpn/openvpn.log

# Set the appropriate level of log
# file verbosity.
#
# 0 is silent, except for fatal errors
# 4 is reasonable for general usage
# 5 and 6 can help to debug connection problems
# 9 is extremely verbose
verb 3

# Silence repeating messages.  At most 20
# sequential messages of the same message
# category will be output to the log.
;mute 20

# Notify the client that when the server restarts so it
# can automatically reconnect.
explicit-exit-notify 1
```