Worth the Wait? Time Window Feature Optimization for Attack Classification

Casey Wilson
Department of Cyber
and Computer Sciences
The Citadel
Charleston, SC, USA
Email: cwilso20@citadel.edu

Xenia Mountrouidou
Department of Computer Science
College of Charleston
Charleston, SC, USA
Email: mountrouidoux@cofc.edu

Anna Little
Department of Computational Mathematics
Michigan State University
Lansing, MI, USA
Email: littl119@msu.edu

***Abstract*—Time as a variable for generating features has been widely overlooked in Intrusion Detection System (IDS) research. Computer and network attacks are time series, where time is an important factor that may affect feature generation, and as a result, classification. Nevertheless, there has been little exploration on how to calibrate time for IDSs and attack classification techniques. In this paper we explore time windows as a technique for generating more effective and descriptive features for attack classification. We suggest a framework for feature generation and selection that uses Recursive Feature Elimination (RFE) and time window exploration. Our initial results when applying this framework indicate that there is up to 47% improvement of F1 scores in attack classification when attack features are generated over a variety of time windows, compared to a single, global time window. We find that features calculated over longer lengths of time may be more useful for detecting attacks than over shorter lengths of time. Our methods seem to be most effective at detecting DDoS attacks, particularly those that occur over medium or long durations of time.**

***Index Terms*—Feature Selection; Attack Classification; Intrusion Detection;**

I. Introduction

Cyber attack classification is a widely known cyber defense problem [1]. It involves using network and host data to automate attack recognition, with the ultimate goal of deploying a targeted defense mechanism. As networks and hosts become more diverse, with Internet of Things (IoT) devices changing the landscape for computer networks, the problem of attack classification is also diversifying. IoT worms such as the Mirai malware [2] are undetected by traditional monitoring systems and can disrupt network functionality and have a large financial impact to business operations [3]. Persistent attacks, such as the Office of Personnel Management (OPM) Advanced Persistent Threat (APT) [4] require a change in thinking about the time frame of observations and data gathering. Thus, time is an important factor to consider when tackling the problem of attack classification.

Intrusion Detection Systems (IDSs) offer a widely accepted solution to the attack detection and classification problem [5]. Whether an IDS monitors the host (HIDS) or the network (NIDS), the goal is to raise an alert specifying the type of attack that is observed. A signature-based IDS [6] relies on a database of known attacks and their characteristics, while a behavior based IDS [7] relies on known behaviors. To classify attacks accurately, IDSs often rely on specific, well-known attack characteristics, i.e., features.

As attacks and network infrastructures increase their variability, the number of available features also increases, with hundreds of features that are good candidates for attack classification. It is generally desirable to restrict to a small set of informative features; this reduces the computational time of Machine Learning (ML) algorithms used for attack classification [8], results in more interpretable models, and avoids overfitting and retaining redundant features [9]. However, the success of the ML algorithm is highly dependent on the feature selection process, i.e., the combination of features used in the classification algorithm [10]. Trying every possible combination of features is not computationally tractable [11], and feature selection is often done with a greedy algorithm which iteratively adds or eliminates features [10]. Developing strategies to find useful feature combinations remains an ongoing research problem.

Even though analyzing data over multiple time windows is considered critical in other domains, this method is often overlooked in the cybersecurity domain, where the focus is to find the best combination of features over a global time window [12], [13]. For example, in signal processing [14], multi-scale features are created by analyzing a signal over various time windows; in agent based systems, features computed on micro time scales can be used to predict behavior on macro time scales [15], [16]; in recurrent neural networks [17], [18], short and long term history are integrated to define the most relevant output. In contrast, this is a less common consideration for attack classification.

A feature calculated over a shorter time window generally provides a smaller traffic sample size, making confident classification of anomalous behavior more challenging than it might be over a larger window with a larger sample size. Contrarily, an entire attack could also take place over a short period of time, rendering a large time window feature calculated from mostly irrelevant traffic data expendable. This suggests that

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

the optimal time window for feature calculation varies greatly depending on attack type.

Another consideration is that time window calculations slow down the speed of real-time classification; if a feature is calculated over a ten minute time-window, this feature's value cannot be determined until the entire time window has passed. This means a classifier could not use this feature until ten minutes after some of the relevant network traffic arrived. In addition, the size of the attack may affect the optimal time window for feature calculation. In this paper, we explore these widely unanswered questions.

We propose a solution to these problems based on feature extraction and selection, where features are defined from a constantly sliding time window. We vary the size of the time window, thus creating multiple versions of the same feature, where each version uses the same formula for calculation over a different length of time. Each version of a feature corresponds to a new unique feature defined by the length of time that it was calculated over. This allows us to employ standard feature selection techniques to select for the optimal time windows of certain features. We focus on the following guiding questions:

- What is the most appropriate combination of features and time windows to classify a specific type of attack?
- How does a time window affect the correctness of attack classification?
- Should we use a global time window or feature specific time windows when calculating features?

To our knowledge, the questions above are widely unanswered in the related literature.

Our contributions are listed below:

1) A recursive feature elimination algorithm to select a feature vector with varying time windows for a specific type of attack,
2) Initial results that demonstrate how time windows affect attack signature calculations and, as a result, classification accuracy.

It is important to note that the intention of this work is not to create a standalone IDS; some attacks do not have obvious time-based signatures and are ill-suited to classification based purely on time series features. Rather, the insights gained in this paper are intended to be used alongside traditional IDS techniques for an added layer of security against attacks that are well-defined by their time-based signatures. This research illustrates that the specific time windows chosen to be used with an IDS are important, and when IDSs use time-based features, time window optimization should be treated with the same significance as standard feature set optimization. The insights gained are valuable, for example, in the design of deep learning architectures in cybersecurity [19]. The performance of neural networks is highly dependent on selecting an appropriate architecture [20], and knowledge of the natural time scales associated with specific features and attacks enables the selection of a strategic architecture.

This paper is organized as follows. In Section II we present a novel framework for feature creation and selection used in attack classification. In Section III we present initial results on how feature selection and time window calculations affect classification accuracy. We discuss the results and limitations of our framework in Section IV. Related work on feature selection and attack classification is listed in Section V. We discuss our future work and conclusions in Section VI.

II. Feature Selection Framework

The proposed framework of feature selection and creation that consists of three stages is shown in Figure 1. Below we explain the input and output of each stage as well as its operation:

- *Generate Feature Tables*: A generic network packet capture dataset with labeled packets that has duration D $secs$ is used as an input in this stage. Then N pre-calculated feature matrices of size $M \times D$ are generated, where N is the number of features and M is the number of different time windows that are explored. The number of rows in those matrices is equal to the times that we can calculate the feature based on $Stride = 1$ sec, thus the rows are equal to $\frac{D}{Stride}$. We define the variable $Stride$ as the number of seconds forward that the sliding time window moves for each iteration of feature calculation. The vector of M time windows is an input variable that can be defined by the business goals, i.e., performance vs. accuracy. For example, a business that is interested in detecting long duration persistent threats may choose time windows that span to several days. Note that all these arguments can change to accommodate different datasets and time window requirements.
- *RFE Stage 1: Feature Selection*: RFE Stage 1 seeks to find the optimal combination of features for Decision Tree (DT) using the F1 score as the ranking metric and all M features calculated during a *single* time window. The F1 score is defined as the harmonic mean of precision and recall $F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$, where $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives. The output of RFE Stage 1 is a $maxFeatures_1$-feature subset with the highest F1 score using RFE, where $maxFeatures_1$ is a predefined constant. That is because when M different time windows are used to generate $M \times maxFeatures_1$ features for RFE Stage 2, the cardinalities of the feature subsets of RFE Stage 2 will be the same for all attack types, barring the removal of any "dirty" feature vectors, such as those with a high number of undefined values.
- *RFE Stage 2: Window Optimization*: The top $maxFeatures_1$ features from RFE Stage 1 are calculated over M distinct time windows, creating a starting feature set with $M \times maxFeatures_1$ unique features. RFE is repeated on this new feature set, again using DT as the ML algorithm and F1 as the ranking score. The feature subset with the highest F1 score

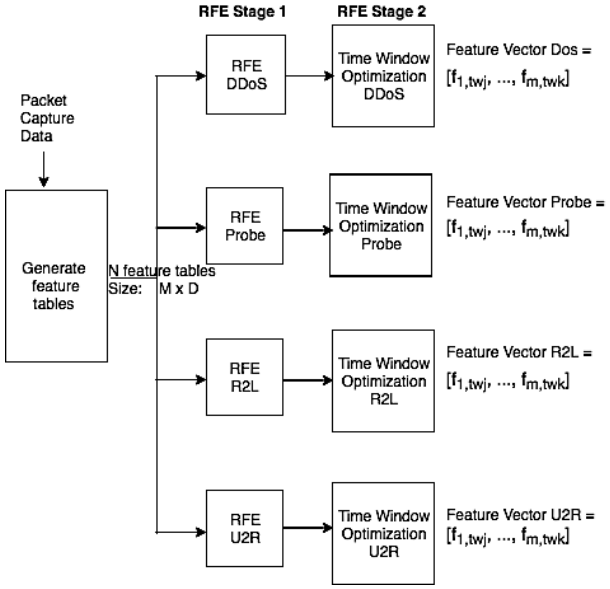


Fig. 1: Feature Selection Algorithm.

and cardinality of $maxFeatures_2$ or less is chosen as the optimal time window feature subset, where $maxFeatures_2$ is another predefined constant. This optimal time window feature subset is the output of the feature extraction and selection process, and will be used as the feature set to evaluate the fitted model's classification success on the test part of the dataset.

In the remainder of this section, we first present the preprocessing for the generation of time-based features. Secondly, we discuss the implementation details of RFE Stages 1 and 2 and give the RFE Algorithm pseudocode for our framework.

A. Feature Generation

The first stage of our feature selection process consists of generating features based on different time window durations. We chose the set of features based on simplicity, speed of calculation, prior research, and domain expertise. We present these features and dataset details in Section III. Note that any generic set of features, discrete or continuous, can be used with our framework.

We consider every time window generated for a feature as a distinct feature itself. Thus f_{i,tw_j} is defined as feature i calculated during time window tw_j, where $j \in \{1, 2, 4, 8, 16, 32, 60\}$ in our implementation of the framework. For example, for SYNCount, there would be seven distinct features, $(SYNCount_1, SYNCount_2, ..., SYNCount_{60})$, where the subscript signifies the length of the time window, ending at the current second, used to calculate the feature.

$Stride$ is the number of seconds the sliding window moves forward for each feature calculation. Since we calculate the features at each second, this corresponds to a stride with a constant size of $Stride = 1\ sec$. This implies that each individual second of the duration of the dataset is a sample.

The general complexity of generating features depends on the number of time windows and set of features. The upper bound for the formula is $O(N \times M \times \frac{D}{Stride})$, where N is the total number of distinct features, M is the number of time windows, and since we use a $Stride = 1\ sec$, we calculate each feature at D points in time.

We note that both the size of the time window and the stride affect the quality of information provided by a feature. In this paper, we do not attempt to optimize stride. Instead we focus on the time window size and how its effects may vary depending on different attack lengths and categories.

B. Recursive Feature Elimination Algorithm

Algorithm 1 Recursive Feature Elimination Algorithm
Input: FeatureVector (full set), GroundTruth, numFolds, maxFeatures
Output: ReducedFeatureVector (reduced feature set associated with highest $F1$ score)

```
1:  procedure RFE(FeatureVector, GroundTruth, numFolds)
2:      if cardinality[FeatureVector] <= maxFeatures then
3:          % return FeatureVector corresponding to best F1 score
4:          % across iterations
5:          return ReducedFeatureVector = FeatureVector
6:      end if
7:      F1Scores = []  %initialize empty list
8:      for missingFeature in FeatureVector do
9:          tempVector = FeatureVector - missingFeature
10:         predictions = []  %predicted values from cross-validation
11:         for training, test in length(numFolds) do
12:             dt = DecisionTree.fit(tempVector, GroundTruth)
13:             foldPredictions = dt.predict(tempVector)
14:             predictions.append(foldPredictions)
15:         end for
16:         % Get F1 score for current feature set:
17:         currentF1 = CalcF1(predictions, GroundTruth)
18:         F1Scores.append([currentF1, missingFeature])
19:     end for
20:     F1Scores.sortAscending
21:     FeatureVector.remove(worstFeature)
22:     % Store feature set associated with highest score:
23:     writeToFile(bestF1, FeatureVector)
24:     % recursion
25:     RFE(FeatureVector, GroundTruth, numFolds)
26: end procedure
```

Our methodology for feature selection is based on two stages of Recursive Feature Elimination (RFE), a wrapper method of feature selection. Wrapper methods use the ML algorithm itself to find optimal feature subsets, treating the ML algorithm as a black box, where the features are selected based on the ML algorithm performance that relies on some

metric, such as accuracy or F1 score. With these methods, the optimal feature sets are those which have optimal scores of a performance metric [10]. Thus, a wrapper algorithm first uses a subset of features to train the model, measures the performance of the model, adds or removes features from the subset, and iteratively or recursively repeats this process.

Using RFE requires a supervised learning algorithm which can quantify the importance of each feature by ranking feature subsets based on evaluation metric scores. RFE is a recursive process of multiple rounds where at each round the least useful feature is removed from the current feature set. Within each round, for each distinct feature f_i in the starting feature subset $\{f_1, f_2, ..., f_N\}$, RFE fits a model feature set without feature f_i and evaluates this model with test data. The feature subset with the highest score is the feature set used as the input to the next round. After the least useful feature is eliminated, a new model is fit with the reduced feature set, and the usefulness of each remaining feature is reassessed using the same process. The outputs of each round are recorded until there are no features left or some other criteria is specified for the algorithm to halt, such as the $maxFeatures$. Algorithm 1 presents the pseudocode of the RFE procedure that we implemented [21]. Note that any machine learning algorithm may substitute the decision tree and any number of $maxFeatures$ can be selected based on efficiency versus accuracy criteria.

We have chosen to use two RFE stages after regarding runtime performance considerations. RFE requires approximately $O(\frac{1}{2} \times N^2)$ number of ML algorithm runs, where $N = SizeofFeatureSet \times NumberOfTimeWindows$. Since a single ML run can take substantial time, we sought to reduce the number of runs by performing two RFE runs, one to find an initial optimal $maxFeatures_1$-feature subset, and another to find the optimal time windows of the $maxFeatures_1$-feature subset, where $maxFeatures_1$ can be any constant.

To achieve this, we first run RFE on all of the features, computed with a single time window equal to the median time window size from set of considered time windows. The median was selected for initial feature filtering because both short and long term patterns should be visible at this resolution. After this initial round of feature selection, we choose the $maxFeatures_1$ optimal median-second time window features. We then add all other time windows corresponding to these optimal features to the feature set, resulting in a new feature set with cardinality $maxFeatures_1 \times NumberOfTimeWindows$. We again use RFE to select for the optimal feature set, and choose the feature subset which yields the highest F1 score while having a cardinality of less than $maxFeatures_2$ features, where $maxFeatures_2$ is another pre-selected constant. This final feature subset contains our optimized time-window features.

Using this method of chained RFE runs allows us to reduce the total number of ML algorithm runs to $O(\frac{1}{2} \times SizeOfFeatureSet^2) + O(\frac{1}{2} \times maxFeatures_1 \times NumberOfTimeWindows^2)$, which is approximately an order of magnitude smaller than the original runs that would have been necessary with a single RFE Stage. This entire process is repeated for each attack type, DOS, Probe, U2R, and R2L, shown in Figure 1.

Decision Tree (DT) was used as the supervised learning algorithm for RFE, due to its quick performance and better initial F1 test results compared to the other algorithms considered. Table II shows our justification for choosing DT and further discussion of this choice is in Section III. To avoid biasing our feature selection process, we first split the dataset into two parts, with both splits being used as the training set and the test set once. This way, the feature selection process could be performed on only the training set, and the selected features would not be biased on the test set. Since each split functions as both a training split and a test split in different runs, we end up with two feature sets and F1 scores for each attack type, one using the first split as the training and the second split as the test and vice versa. Using DDoS as an example, we refer to DDoS Split 1 as the DDoS where Split 1 is training and Split 2 is test, and DDoS Split 2 using Split 2 as training and Split 1 as test.

A further consideration is that our feature extraction process separates individual seconds into samples, and since a single attack occurs over multiple seconds, many contiguous second-samples contain the same attack. This too is an issue for inflating a classifier's scores, since a split down the middle of the dataset may split the same attack into both training and test sets. For this reason, for each attack type, we chose an index to split that did not separate an attack of that type into both splits, but tried to still retain close to half of the second-samples and half of the attacks within each split. This means that each attack type had a dataset that had slightly different splits. Our method for splitting is in essence 2-fold cross-validation, where the splits are manually determined.

Once we have split the dataset, we use RFE on the training split to select for features. Since only half of a dataset is used for feature selection, we employ standard K-fold cross-validation on the current training split, with $K = 5$ folds. Using 5-fold cross-validation, we divide our training split in 5 equal parts, and use 4 parts for model training and 1 part for testing. This process is repeated 5 times, each time with a different section of the split used for testing. Since every section of the training split is in the 5-fold testing section exactly once, we collect the scores from all 5 training sections and compare the predicted labels to the actual labels to gather an F1 score, which we use to rank our feature sets.

Even though with this method the same attack may be contained in different folds, we justify not splitting the folds manually for the feature selections process because the F1 scores we collect here are not used to describe the effectiveness of our model; they are simply a ranking metric for comparing feature subsets, and all of the feature subsets will have identical folds to the feature sets they are compared with.

III. Results

We present results of our experimentation with the RFE time window optimization framework using the KDD 1999 Dataset

[22]. First, we give a brief description of the dataset. Then, we demonstrate how our technique affects classification.

A. Dataset Description

We used the KDD 1999 dataset [22] packet captures. Although there has been controversy in using this dataset [23], it is a well labeled, complete set of packet and host data. In addition, there is a large amount of related work on feature selection for this specific dataset as we review in Section V. This offers an opportunity to compare other methods of feature selection to our work that introduces time windows as part of creating and selecting features.

We specifically chose to use packet captures (pcap) since host configurations and technologies change more over time than pcap data, which has had the same format and is less architecture or technology dependent. Obviously, there has been an increase in network bandwidth and speeds. However, the features derived from pcap data, such as packet size, inter-arrival rate, and payload, have been unchanged.

Normal traffic, such as internet browsing and email exchange, is included in the data and it is interleaved with four types of attacks: Probe, DDoS, U2R, and R2L. The four types of attacks encompass the majority of modern adversarial techniques and are briefly described below:

- **Distributed Denial of Service (DDoS)**: An attack intended to deprive users from legitimate services. It is often distributed, i.e., executed by a set of devices (botnet) coordinated by Command Control (C2) infrastructure.
- **Probe or Scanning**: A passive attack intended to gather information about exposed, vulnerable services and hosts.
- **Remote to Local (R2L)**: An active attack intended to gain local user access when the adversary is remotely located.
- **User to Root (U2R)**: An active attack that escalates the privilege to root / administrator from simple user access. It usually precedes a U2R attack.

The preselected features that refer to packet capture files are presented in Table I. Initially, we considered a set of sixty data features from [24], that ranged from protocol and flag signatures, to packet size, inter-arrival rate, source and destination ports, and packet error rates. However, we did not include the whole set of features due to degrading processing time. RFE requires $O(N^2)$ number of ML algorithm runs, where N is affected by the size of the feature set and the number of time windows, i.e., $N = SizeOfFeatureSet \times NumberOfTimeWindows$. Thus, we limited the size of the feature set to a lower number of features to avoid large computation time. We have deployed a mix of statistically calculated features, such as mean and coefficient of variation, as well as discrete features, such as the count of a protocol, service, or flag. Note that these features are generic and not dataset-specific.

We have used the last two weeks of data, which amount to ten days of eight-hour work day packet network captures. The attack data comprises 61% of the total packet captures.

TABLE I: Randomly selected features calculated over n seconds. "#" indicates "number of"

Feature	Description
SYNCount	# packets with SYN flags
MeanNumberOfPackets	Mean number of packets per second
SYNBool	# previous n seconds containing at least 1 SYN flag
NumberOfPackets	Total number of packets
UniqProtcols	# unique protocols
UniqSrcIPs	# unique source IP addresses
UniqDestIPs	# unique destination IP addresses
Count DNS, TCP, ARP, ICMP, UDP, FTP, HTTP, RST, TELNET SSH	# packets using specific protocol or service (10 different features)
CCodeCount	# packets with *.c file extension in payload
EXECodeCount	# packets with *.exe file extension
	in payload
CorJSCount	# packets containing markers of c or javascript code
ThirdMNumberPackets	Third Moment number of packets per second
MeanPacketSize	Mean size of all packets
CVPacketSize	Coefficient of variation of size of all packets
ThirdMPacketSize	Third moment of size of all packets
HTTPorFTPandEXE	# packets using either HTTP or FTP, containing a *.exe file extension in payload
HTTPandMalformed	# malformed packets using HTTP,
FTPandC	# packets containing FTP and a *.c file extension in payload

B. Experimentation Results

We present runtime performance results of different machine learning algorithms in Table II. Even though our framework is not algorithm specific, Table II shows a good justification for the usage of Decision Trees regarding run time and preliminary F1 score. The F1 scores for DTs were the highest, although Support Vector Machine (SVM) and K-Nearest Neighbors (KNN) could also be used with proper parameter tuning. These results were gathered from DDoS split 1 using all of the 8-second time window features to fit the model, with DDoS split 2 as the test set. The three distinct sample lengths were chosen to see if relative performance changed with number of samples. The 71,348 samples is about half of the dataset, which is close to the amount of samples that a single split would consist of.

TABLE II: Performance in seconds of different machine learning algorithms for varying numbers of samples: 500, 5,000, and half of total dataset samples (71,348 samples). F1 score calculated for 71,348 samples.

ML Algorithm	500	5,000	71,348	F1
DT	0.05	0.08	1.63	0.18
K-NN	0.44	0.62	1.75	0.00
SVM	2.99	31.22	3410.53	0.06
NN	1.01	3.34	36.39	0.07

Our aim in this work is to demonstrate a proof of concept that using multiple time windows for feature creation has value

for classification. Thus, we attempt to show improvement when the element of time is a variable for attack feature creation and selection. To this end, the results presented below test the following hypotheses:

1) *A global time window is less effective compared to multiple time windows for feature creation:* In this case, our goal is to show results for feature selection and creation that demonstrate noticeable improvement for the classification of attacks.
2) *The more information, the better*: If a time window is large, i.e., there are more data for feature generation, the classification decision will give higher F1 scores. However, more features do not obviate more information, i.e., some features may have overlapping, redundant data, not needed for classification.
3) *The attack size affects the window size*: If an attack has long duration, long time windows improve its classification and equivalently for short-duration attacks, features that are generated with shorter time windows are more beneficial.

To demonstrate the first hypothesis, we have constructed a Table with the F1 scores calculated after the first stage of our algorithm, i.e., after RFE with only one specific time window duration of 8 secs, and the F1 scores after using RFE for time window optimization. Table III shows these F1 scores and the percentage difference before and after optimizing with more than one global time window size for feature creation. These scores are the average F1 scores from each of the two splits for each attack. Even though the F1 scores are not high, there is a noticeable improvement after optimizing and creating features with multiple time window granularities that varies from 11% - 47%.

TABLE III: F1 scores before and after time window optimization. RFE Stage 1: RFE with only one specific time window duration, i.e., 8 secs. RFE Stage 2: F1 scores after using RFE for time window optimization.

Attack Type	RFE Stage 1	RFE Stage 2	% Diff.
DDDoS	0.166	0.243	46%
Probe	0.108	0.120	11%
U2R	0.192	0.229	20%
R2L	0.090	0.132	47%

Regarding the second hypothesis, Figure 2 demonstrates that features tend to have overlapping information, therefore more is not always better. In this Figure, we have shown the F1 scores of RFE Stage 2 iterations versus the number of features. Figure 2 parts a and b correspond to the two ways that we trained our model by using the first (Split 1) or second half (Split 2) of the dataset for training. Note that the F1 values in this figure are considerably higher than in Table III. This is justified because these F1 scores were used during the training process, which used the standard K-fold method of training and testing our model, which left some attacks in the training and testing sets, and also selected optimal features for the training set. For these reasons, it is not surprising that the F1 scores are higher for the feature selection process.

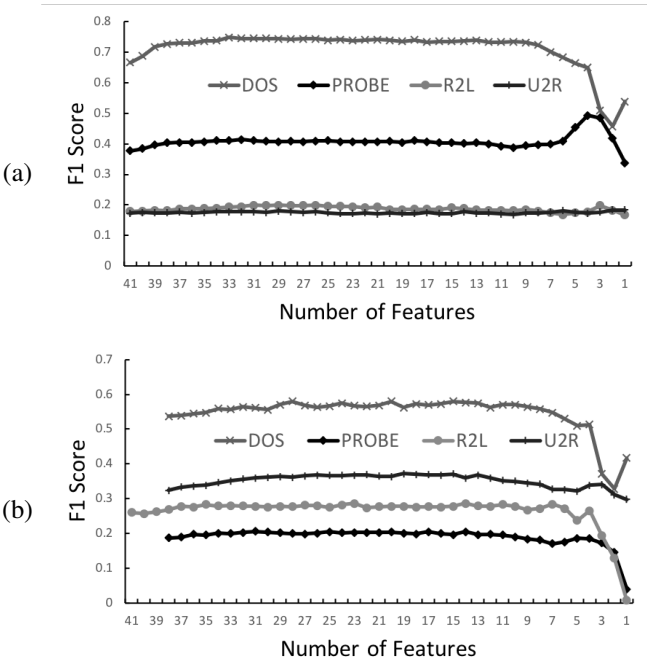
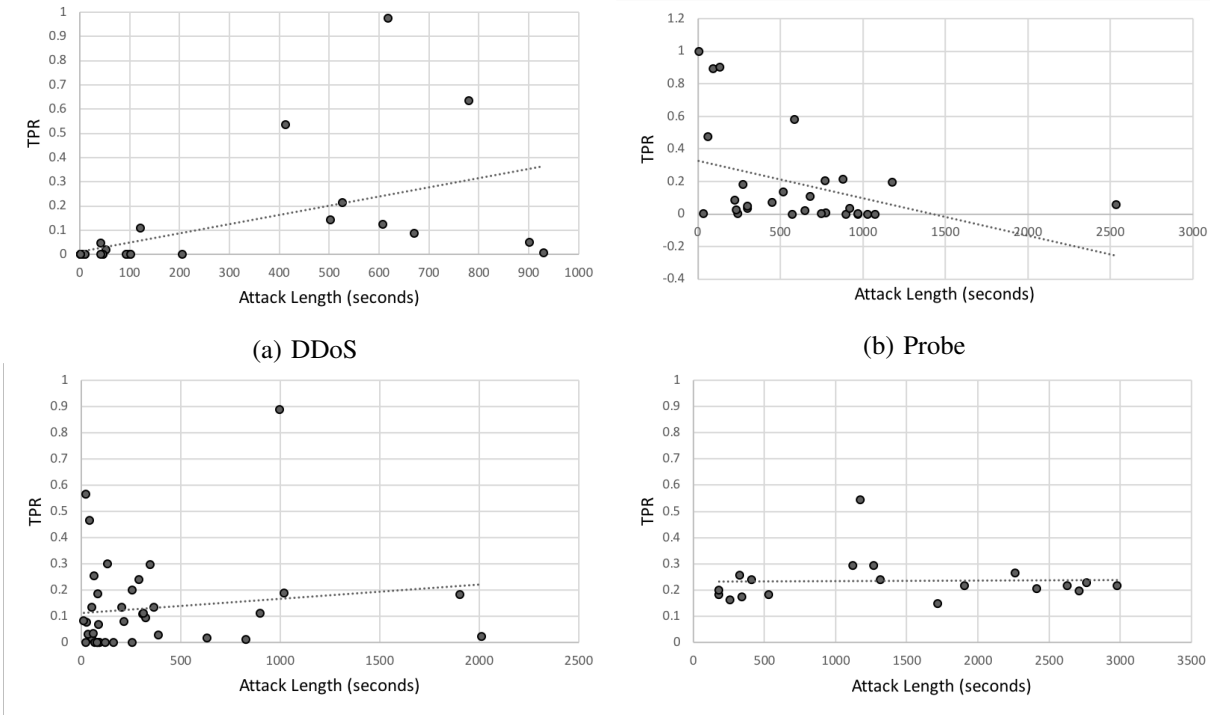


Fig. 2: F1 score vs number of features as was measure during the second RFE, starting with 41 time window features and reducing to the optimal windows. (a) F1 scores were calculated using the *first* half of the dataset as training/validation and the *second* half for testing, (b) F1 scores were calculated using the *second* half of the dataset as training/validation and the *first* half for testing.

For each attack type, we calculated the Pearson correlation between the attack length and the True Positive Rate (TPR) of the seconds within that attack. The correlation values for each attack type were: DDoS: 0.51, R2L: 0.14, U2R: 0.02, Probe: -0.41. Figure 3 supports our last hypothesis, i.e., there is correlation between the size of attack and size of time windows used for feature generation. This Figure shows the TPR for all four types of attacks vs the attack length. The TPR in this scenario is defined as the TPR within all of the seconds in an individual attack; this way, each attack has its own TPR, which is useful for analysis. DDoS and R2L have a noticeable increase in correlation of attack length with TPR. DDoS specifically has a correlation of 0.51, the highest of the four attack types. Notably, DDoS also has the highest final F1 score, and the second highest F1 percentage change when optimizing the time windows with the second RFE on the time window features. Both DDoS and R2L may expand to multiple sessions. For DDoS these sessions may try to overwhelm the server for a long period of time, and for R2L they may include brute force password attacks to gain access to a local machine. A unexpected result is that Probe attacks demonstrate a negative correlation between attack length and TPR. This may be justified by stealthy scans or short bursts of scans to avoid detection.

Figure 4 shows the False Positive Rates (FPR) for each



(a) DDoS

(b) Probe

(c) R2L

(d) U2R

Fig. 3: Pearson Correlation of Attack Length versus True Positive Rate

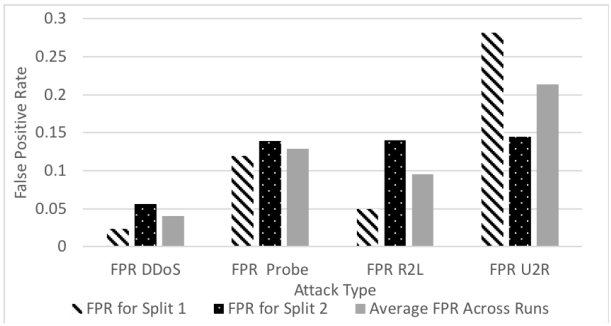


Fig. 4: False Positive Rates (FPR) By Attack Type

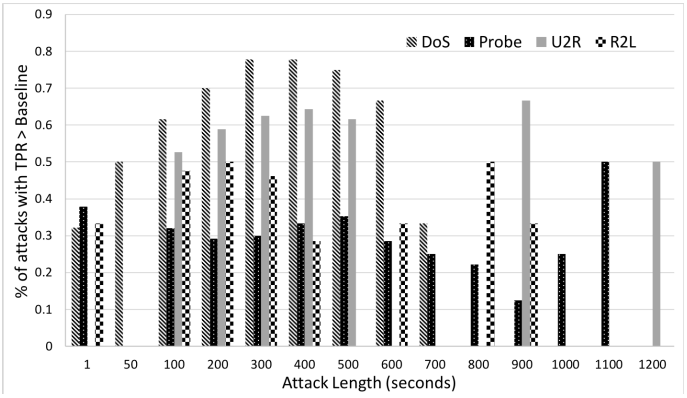


Fig. 5: Attack Length vs Classification as defined by percentage of attacks that have TPR higher than the baseline.

attack type across Split 1, Split 2, and the average across both runs. DDoS has an average FPR of 4.0%, less than half the amount of the next closest attack type R2L, which has a FPR of 9.5%. Probe has the third highest FPR of the attack types at 12.9%, and U2R has the highest FPR at 21.3%.

In Figure 5 we demonstrate the effect of attack length thresholds on ”detection rate”. Here we define a detected attack as one where the TPR within the attack is higher than the baseline rate of positive predictions on the test set. The graph shows a threshold of attack lengths in seconds on the x-axis. If an attack is the same size or longer than the a given value on the x-axis, it is considered in the percentage of detection rate. The y-axis shows the percentage of attacks that are detected, i.e., have TPR higher than the baseline. A higher value on the y-axis for a given attack length threshold implies that attacks of that length are more likely to be detected. DDoS, R2L, and Probe all have detection rates of 0 for the longest attacks, but otherwise have a generally positive trend, where as the length of attack increases, so does the detection rate. For example, the two longest DDoS attacks were not detected, ending with a negative trend, if these two attacks had been removed, there would have been a consistently positive trend. U2R does have a trend for attack detection rate to increase as attack size increases.

This is an intuitive result since time window-based features

seem to do better for attacks that take place over more time.

Table IV aims to give insights related to the feature selection and time windows that were chosen by our framework. Table IV shows the optimal features for both splits after the two stages of our framework feature selection and creation using multiple time windows. It can be seen that the number of optimal features for each attack in Split 1 correspond to the highest points on Figure 2a, and the number of optimal features for each attack in Split 2 correspond to the highest points on Figure 2b for that attack when there are less than ten features left. Note that several features related to the signature of a specific attack repeat with different granularities, such as *TCPCount* for U2R Split 2, where 1 second, 8 second, and 60 second windows were all selected. This suggests that a single feature calculated over different lengths of time can be useful in combination, as well as suggesting that different time windows should be considered as individual features in their own right. Many of the selected features calculated use known attack signatures for that attack type, such as SYNBool for DDoS, where many types of DDoS attacks, such as the SYN flood attack, where a high volume of packets containing a SYN flags are sent to a victim. Another example is ARPCount for Probe attacks, many of which may send several ARP requests looking at random IP addresses.

IV. Discussion

Even though our feature selection and generation framework is generic and can be used with any discrete and continuous feature set, it may not be ideal for fast paced changes in a real-time environment. The runtime of the algorithm is considerable and would preferably be completed off-line. Then an optimal feature set can be used as input to a real time IDS.

The F1 scores that were reported are low for attack classification, however, measuring the success of this classification task requires considering both the F1 score and the FPR. For this task, the F1 score alone is an insufficient measure of classification success because the F1 score is a function of precision and True Positive Rate (TPR). Since we are classifying seconds, and many seconds may compose an attack, a successful detection does not necessarily require a high TPR of all of the seconds of an attack. In a real scenario, an IDS would seek to classify attacks based on network flows, not seconds, so a given length of network traffic with a higher than average TPR may be concern enough to raise a flag, and a TPR of less than 1 could be effective at doing this. Considering the F1 score alongside the FPR provides a more robust idea of classification success.

When considering the two scores, clearly the most effective classification was for DDoS attacks, particularly those of substantial length. Not only does DDoS have the highest F1 Score, but it also has the lowest FPR by a large margin, with an FPR of about 4% on average. DDoS also has the strongest positive correlation between attack length and TPR (Figure 3), with a correlation of 0.51. DDoS attacks also had the second highest change in F1 between the initial feature selection round and the window optimization round (Table III), as well as a generally positive trend between the attack length threshold and detection rate (Figure 5). This is a result that matches intuition, since many DDoS attacks may consist of various methods of sending high amounts of consistent types of network traffic over long periods of time. Attacks of this type intuitively seem especially well-suited to using time-window features, and the results show that the optimal time window features are most effective for these types.

Unexpectedly, our method of finding the optimal time window features seems to have been rather ineffective for detecting Probe type attacks with this dataset. Not only does Probe have the lowest final F1 score (Table III) but it has a negative correlation between attack length and TPR (Figure 3).

In general, longer time windows were selected as more useful than shorter time windows. This may be a simple result of the fact that many of the lengths of the attacks were significantly longer than our longest time window, 60 seconds. This selection of longer time windows is not too surprising of a result; they are less sensitive to short-term fluctuations of normal traffic that may limit the usefulness of some of the smaller time windows. Eventually time windows could get so large that they become insensitive to even the malicious fluctuations in traffic. Further investigation may use longer windows to find lengths of time that become too long to be effective. Since the longer time windows were generally chosen as the most effective already, starting with even longer time windows could positively affect classification as well. Further exploration of machine learning algorithms and feature selection methods will enhance the performance of classification. In addition, a more sophisticated set of initial features where domain knowledge is used, will result to better classification performance.

Our experimentation is limited to a dataset that is old and not considerably large. We did use only packet captures that have generic network characteristics; however a more current dataset can be used [25]. Experimentation with additional datasets is required to generalize our framework for time window feature creation. Furthermore, longer attacks, such as long duration botnets and APTs, will reveal additional information about the importance of calibrating feature calculations over time.

V. Related Work

The work related to our study is divided in two categories: feature selection techniques and attack classification. There are large bodies of work related to both categories. We have included the most relevant to our work in this section.

A. Feature selection for attack classification

The authors in [26] provide one of the earliest studies related to our work. They use information gain and decision trees to specify the relevance of a feature in attack classification. Flexible Neural Trees (FNTs) are deployed by Chen et. al. [27] to select features and classify attacks. The authors develop a genetic algorithm to select the optimal features. They reduce

TABLE IV: Optimal features selected by our framework. The format of features is: $< TypeOfFeature >< WindowSize >$.

Attack	Optimal Features Split 1	Optimal Features Split 2
DDoS	MeanPacketSize16, SYNBoolean60, TCPCount60, SYNBoolean32, DNSCount16, TCPCount16, DNSCount8, ICMPCount60, TCPCount2, SYNCount16	SYNBoolean60, ThirdMomentPacketSize60, TelnetCount60, TCPCount60, DNSCount8, TCPCount16, DNSCount32, ThirdMomentNumberOfPackets60, TCPCount8, TelnetCount4
Probe	ARPCount8, UDPCount60, ARPCount16, ARPCount60	DNSCount60 ThirdMomentNumberOfPackets32 MeanPacketSize60 MeanPacketSize32, RSTCount60, TelnetCount32, UniqSrcIps32, ThirdMomentNumberOfPackets60, UniqSrcIps60, MeanPacketSize16
R2L	CVPacketSize60, UniqSrcIps32, RSTCount6	DNSCount60, SSHCount60, SYNCount60, UniqSrcIps60, RSTCount60, RSTCount32, NumberOfPackets
U2R	ThirdMomentPacketSize60	SSHCount60, CorJavaScriptCount60, RSTCount60, ThirdMomentNumberOfPackets60, TCPCount60, CorJavaScriptCount32, ThirdMomentNumberOfPackets16, SSHCount32, TCPCount1, TCPCount8

the number features for the KDD dataset from 41 to 5, and their method classifies DoS and U2R attacks optimally.

Stein et. al. [28] use genetic algorithms to reduce the number of features, then deploy decision trees for classification and achieve better accuracy with less than the 41 prevalent features of the KDD 1999 dataset. Directed Bayesian Networks (BN) are used to find the minimal amount of features that maximizes performance as presented by Chebrolu et. al. [8]. The authors use Classification and Regression Trees based on binary recursive partitioning for feature deduction. Then they use an ensemble based intrusion detection system present results where they manage to reduce the feature dimensions with high performance. Mutual Information (MI) that employs entropy to eliminate overlapping features is studied by Amiri et. al. [29]. The authors use ROC curves and performance results to demonstrate the accuracy of the subset of features selected by three algorithms: Modified MI, Correlation, and Forward Feature Selection (FFS). Ambusaidi et. al. [30] use Mutual Information in a filter based algorithm for feature selection. The authors examine two additional datasets besides the KDD'99, i.e., NSL-KDD and Kyoto 2006. They create an IDS based on Least Squares SVM and their feature selection filter algorithm, that achieves better accuracy and performance compared to state-of-the-art methods.

Ganapathy et. al. [13] conduct an extensive survey on feature selection techniques and classification algorithms based on the KDD 1999 dataset. They review the following feature selection techniques: gradual feature removal (filter), mutual information techniques based on entropy, and Conditional Random Field (CRF), which use a layered approach of different features per attack group combined with domain knowledge and wrapper techniques. They then compare neural networks, linear programming, SVMs, and the layered approach for classification. They conclude with a new algorithm that uses information gain ratio for feature selection and multi-class SVM for classification and performs well regarding accuracy of classifying DoS and Probe attacks using the KDD'99 dataset.

In all the above related publications, the time window used to calculate the features is not discussed or used as a criterion to create new features. In addition, there is no study on how time duration of feature calculation affects the accuracy of attack classification.

B. Attack Classification

The most well known rule based IDSs such as Snort [6], Bro [31], and Suricata [32], use pre-set constant window sizes for attack signature calculations. Although these methods may partially address the question of how much history is relevant for an optimal global window, the current approaches fail to recognize the naturally varying time scales associated with diverse attacks and features.

Lee et. al. [33] present two algorithms that recognize anomalies and known attacks, i.e., the association rules algorithm and the frequent episodes algorithm. Their work is of the earliest in real-time detection of known and unknown attacks. Experimentation with SVMs and Artificial Neural Networks (ANNs) is presented by Sung et. al. [34]. The authors combine attack classification and feature ranking. First, they rank features based on performance metrics that they have defined, and then they use SVMs with ranked features and compare their results to ANNs. They conclude that SVMs outperform ANNs in accuracy and performance. Zhang et. al. [35] deploy single class of on-line SVMs in an effort to train and use them for classification of attack data in real-time. Their results are promising for improved classification accuracy without sacrificing performance. A comprehensive survey on intrusion detection using machine learning is presented by Tsai et. al. [36]. In this survey, well known algorithms such as k-Nearest Neighbor, SVMs, ANNs, Self-Organizing maps, Decision Trees, Naive Bayes, Genetic Algorithms, and Fuzzy Logic are reviewed as they have been used in intrusion detection.

All the above related literature demonstrate the importance of dimensionality reduction for intrusion detection and attack classification. As we have demonstrated in our study, it is important to consider time since it affects the information of a feature and the accuracy of classification.

VI. Conclusions

We have presented a framework to create and select features based on variable time window durations. Our feature selection techniques are used for attack classification for general types of attacks such as DDoS, Probe, R2L, and U2R. The novelty of our framework is the emphasis on time for feature generation, an element that has not been adequately explored in the area of attack classification and intrusion detection. The initial results

are promising, showing 11%-47% improvement in F1 scores when multiple time windows are used for feature generation.

This study raises several interesting questions for future research. First, can we optimize time windows for real-time intrusion detection? Another question is whether some specific features benefit more than others when shorter or longer time windows are used for their calculation. There is an additional variable, the window stride, that is also worth exploring. If there is low information overlap, i.e., larger window stride, how does this affect classification and our feature generation framework? Finally, the trade-off of frequent versus infrequent alerts needs to be explored for even larger and longer duration attacks, such as Advanced Persistent Threats.

VII. Acknowledgments

We would like to acknowledge the support provided by the US National Science Foundation under Award No. DUE-1700254.

References

[1] M. Uma and G. Padmavathi, "A survey on various cyber attacks and their classification," *International Journal of Network Security*, vol. 15, no. 5, pp. 390–396, 2013.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1093–1110.

[3] A. C. Haury, "10 Of The Most Costly Computer Viruses Of All Time," shorturl.at/xHRU0.

[4] B. I. Koerner, "Inside the cyberattack that shocked the us government," shorturl.at/nxFO0.

[5] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16 – 24, 2013.

[6] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, ser. LISA '99, 1999, pp. 229–238.

[7] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, no. 1-2, pp. 18–28, 2009.

[8] J. P. T. Srilatha Chebrolu, Ajith Abrahama, "Feature deduction and ensemble design of intrusion detection systems," *Computers & Security*, vol. 24, no. 4, pp. 295 – 307, 2005.

[9] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *Journal of machine learning research*, vol. 5, no. Oct, pp. 1205–1224, 2004.

[10] I. Guyon and A. Elisseeff, "An introduction of variable and feature selection," *J. Machine Learning Research Special Issue on Variable and Feature Selection*, vol. 3, pp. 1157 – 1182, 01 2003.

[11] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[12] A. KumarShrivas and A. Kumar Dewangan, "An Ensemble Model for Classification of Attacks with Feature Selection based on KDD99 and NSL-KDD Data Set," *International Journal of Computer Applications*, vol. 99, no. 15, pp. 8–13, Aug. 2014.

[13] S. Ganapathy, K. Kulothungan, S. Muthurajkumar, M. Vijayalakshmi, P. Yogesh, and A. Kannan, "Intelligent feature selection and classification techniques for intrusion detection in networks: a survey," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 1, p. 271, Nov 2013.

[14] S. Mallat, *A wavelet tour of signal processing*. Elsevier, 1999.

[15] C. W. Gear, J. M. Hyman, P. G. Kevrekidid, I. G. Kevrekidis, O. Runborg, and C. Theodoropoulos, "Equation-free, coarse-grained multiscale computation: Enabling mocroscopic simulators to perform system-level analysis," *Communications in Mathematical Sciences*, vol. 1, no. 4, pp. 715–762, 2003.

[16] I. G. Kevrekidis, C. W. Gear, and G. Hummer, "Equation-free: The computer-aided analysis of complex multiscale systems," *AIChE Journal*, vol. 50, no. 7, pp. 1346–1355, 2004.

[17] A. Graves, "Supervised sequence labelling," in *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, pp. 5–13.

[18] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[19] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35 365–35 381, 2018.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[21] C. Wilson, "Github repository (code and data)," https://github.com/CybersecurityXLab/ids_svm_slidingwindow.

[22] T. U. K. Archive. (1999) KDD cup 1999 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[23] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, Nov. 2000.

[24] J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer Networks*, vol. 34, pp. 579–595, 2000.

[25] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, Nov 2015, pp. 1–6.

[26] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting features for intrusion detection: A feature relevance analysis on kdd 99," in *PST*, 2005.

[27] Y. Chen, A. Abraham, and B. Yang, "Feature selection and classification using flexible neural tree," *Neurocomputing*, vol. 70, pp. 305–313, 12 2006.

[28] G. Stein, B. Chen, A. S. Wu, and K. A. Hua, "Decision tree classifier for network intrusion detection with ga-based feature selection," in *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 2*, ser. ACM-SE 43, 2005, pp. 136–141.

[29] F. Amiri, M. Rezaei Yousefi, C. Lucas, A. Shakery, and N. Yazdani, "Mutual information-based feature selection for intrusion detection systems," *J. Netw. Comput. Appl.*, vol. 34, no. 4, pp. 1184–1199, July 2011.

[30] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an intrusion detection system using a filter-based feature selection algorithm," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 2986–2998, Oct 2016.

[31] V. Paxson, "Bro: A system for detecting network intruders in real-time." San Antonio, Texas: Proceedings of the 7th USENIX Security Symposium.

[32] "Suricata Open Source IDS / IPS / NSM engine," https://suricata-ids.org/.

[33] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, ser. SSYM'98, 1998.

[34] A. H. Sung and S. Mukkamala, "Identifying important features for intrusion detection using support vector machines and neural networks," in *Proceedings of the 2003 Symposium on Applications and the Internet*, ser. SAINT '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 209–. [Online]. Available: http://dl.acm.org/citation.cfm?id=827273.829224

[35] Z. Zhang and H. Shen, "Application of online-training svms for real-time intrusion detection with different considerations," *Computer Communications*, vol. 28, no. 12, pp. 1428 – 1442, 2005.

[36] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994 – 12 000, 2009.