Automated Cross Layer Feature Selection for Effective Intrusion Detection in Networked Systems

Azeem Aqil*, Ahmed Fathy Atya*, Srikanth V. Krishnamurthy*,
Paul Yu†, Ananthram Swami†, Jeff Rowe+, Karl Levitt+,
Alexander Poylisher‡, Constantin Serban‡ and Ritu Chadha‡
*UC Riverside, †U.S. Army Research Laboratory +UC Davis ‡Applied Communication Sciences
{*aaqil001, afath001, krish*}*@cs.ucr.edu*, {*paul.l.yu.civ, ananthram.swami.civ*}*@mail.mil* {*rowe, levitt* }*@cs.ucdavis.edu*,
{*apoylisher, cserban, rchadha*}*@appcomsci.com*

***Abstract*—Traditionally, anomaly detection mechanisms have relied on the inspection of certain manually (by domain experts) chosen features in order to determine if a networked system is under attack or not. Unfortunately, the approach, while somewhat effective in flagging known attacks, yields either low true positive rates or high false positive rates when the attacks are mutated slightly or in the presence of zero day attacks. One can traditionally gather a lot of data at different layers (packet contents, application logs, OS behaviors, etc.) as evidence that could be used for intrusion detection. However, it is not easy to determine which of these evidence vectors or features are useful in facilitating highly accurate intrusion detection. In this paper, we undertake an in-depth experimental study to determine whether appropriately trained search algorithms can help us find the right set of features for detecting a class of attacks (e.g., denial of service). The output of such algorithms yields a set of features that should potentially improve detection accuracy. Towards this we monitor 365 features across system layers and compare the detection performance of 3 popular feature selection algorithms to reduce the state space of the feature set for two classes of attacks. We find that the approach can yield significantly improved detection accuracy in comparison to statically chosen single features, sub or super sets of features of what the algorithms yield.**

I. Introduction

There has been a recent increase in both the frequency and impact of cyber threats [1]. With network based attacks expected to rise [2], it is critical that highly efficient evidence collection and intrusion detection techniques be designed and deployed. Anomaly detection and signature based detection are the two most popular detection approaches (e.g., [3] [4]). The effectiveness of these approaches however, is tightly dependent on the underlying features (feature set of evidence) that are chosen.

There are multiple distinct sources of data that one can use to collect evidence. Features can be selected from the network, operating system, hardware or network layers. However, selecting the optimal subset of features to enable highly accurate intrusion detection is not easy. The quality of the eventual feature set is dependent both on the actual features, and the number of features. A set that is too small lacks the information to correctly reason about mutated or unknown attacks, while a set that is too large contains frivolous features that introduce noise and increase misclassification. Features are typically selected by studying the behavior of known attacks. This further complicates the problem for unknown or unseen attacks. Most detection approaches use features that have been carefully selected by domain experts. Such approaches, by definition, require precise knowledge about network threat semantics while also being prone to human judgment errors. Modern detection approaches also generally only use features from the network layer. Recent work has demonstrated the utility in considering features across different layers [5].

In this paper, we seek to design a *unified* systematic framework to collect meaningful cross-layer features that are applicable to multiple classes of network attacks. Our framework must automate and sequentialize the process of feature selection and thereby enable the effective handling of high-volume data for highly accurate intrusion detection. Futher, we want our evidence collection to be general, in that the approach does not require deep knowledge about network behavior.

A high level depiction of our framework is shown in Figure 1. First, the system consists of an offline phase wherein it is trained with attack and normal behaviors. A large volume of evidence is collected offline from multiple layers and for each case (different attacks, normal), an appropriate feature selection algorithm is then used to downselect the number of features. During runtime, only these downselected set of features are actively monitored. These are then fed to an inference engine which then provides an assessment of whether or not the networked system is under attack.

Challenges: There are a number of challenges that we need to address while building our framework. First, while some features are readily available, the networked system must be instrumented to collect other forms of evidence that are not exposed (e.g., create hooks in the OS). Second, one has to choose the right feature selection algorithm to achieve the right trade-off between accuracy and complexity. Towards understanding this trade-off we compare and contrast three algorithms, namely Linear Forward Selection (LFS), Sequential Backward Selection (SBS) and Simple Genetic Algorithm (SGA) on a large feature set that is collected. While one can expect the genetic algorithm to yield the highest accuracy it also is complex and takes a long time to run even on powerful servers; the others could potentially yield lower accuracy but run faster and on desktop computers. We seek to understand the tradeoffs between accuracy, complexity, and computation time for these three classes of algorithms.

To evaluate the effectiveness of automated cross-layer feature selection, we use the features selected with two detection (inference) engines viz., Dempster-Shafer Theory based inference and K-Means classification. The former outputs measures of likelihood (referred to as belief and

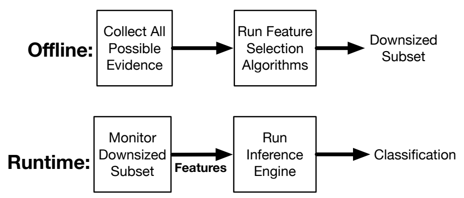


Fig. 1: Overview of our approach

plausibility) with respect to normal and different attack behaviors and is thus able to better differentiate between attacks and properly classify mutated attacks. The latter is arguably the most popular classification approach and allows us to reason about the presence or absence of an attack, in more traditional terms. We point out here that the automated feature selection process is independent of the inference engine (i.e., any inference engine can be used to classify behaviors).

Our work in perspective: There have not been many attempts at addressing the problem we seek to solve. Attempts at applying feature selection algorithms to detect network based threats ([6], [7], [8] [9]) have considered only relatively small feature sets (less than 50) which are often chosen manually themselves. They are also specific to one particular kind of network attack. We consider feature sets that are much larger (> 300), and could reveal features that are better suited for specific attacks while covering large classes of attacks. We also seek to automate the entire feature selection process with little or no need for domain expertise about network threats. Virtually all attempts at detecting network threats, along with any attempts at feature selection, only consider network layer features. This is a problem because as new threats emerge, experts are forced to come up with increasingly novel ways to use network based features. We argue that contemporary approaches fail to capitalize on information that is present in features captured at other layers (OS, hardware, application). Further, prior approaches do not address the inherent uncertainty and noise that any selected features are bound to have. We point out that our work seeks to determine ”what features to consider” while making an inference with regards to whether or not the networked system is under attack. It is agnostic to how those features are used in a detection engine. In our evaluations we show the effectiveness of our features with two different types of inference engines.

Contributions: To the best of our knowledge, this work is the first attempt at a unified approach (from collecting the initial set of features to downselecting to the eventual subset) at extensive cross layer feature selection for intrusion detection. Specifically, we make the following contributions:

- We design a framework that sequentializes and automates the process of cross layer feature collection and downselection and eventual use of the downselected features for intrusion detection.
- Our framework incorporates a novel, evidence collection module that captures a relevant set of initial features by placing numerous monitors that are spread across the hardware, network, OS and application layers. We demonstrate this via a set of monitors that collect extensive cross-layer features for 3 DoS attacks, 2 SQL injection attacks, and normal behavior.
- Via extensive experiments, we test the feature selection capabilities of three popular search algorithms (LFS, SBS, SGA) and provide an analysis of their respective accuracy versus cost trade-offs. We find that LFS and SGA have similar results, in terms of detection accuracy, for DoS attacks. This is very surprising since SGA performs a much more exhaustive search; LFS finishes in a fraction of the time it takes SGA to complete. SGA produces slightly better results (detection accuracy) for SQL injection and SBS selection lags behind the other two for both kinds of attacks. Given the significantly higher runtime associated with SGA, one has to carefully assess if the slightly increased accuracy (< 10 %) warrants the significant increase in complexity
- We build a complete system and show that feature selection algorithms lead to good detection accuracy not only in detecting known attacks but also a previously unseen attack that we construct. Our system decouples the feature selection process from the inference engine of an intrusion detection system, and is thus readily deployable.
- We show that selecting too few or too many features can actually hurt detection accuracy.

System Layer	Source of Evidence
Hardware	Hardware Counters, Perf Events
Network	Raw Stream Data, Packet Headers, Socket Statistics
Operating System	/proc Filesystem
Application	Log Files

TABLE I: Source of Evidence

Scope: Our evidence collection approach is generic and we expect it to be applicable to other network based threats where features manifest at different layers. For tractability, we only test two different kinds of attacks. While our initial set of features is large (365 features), it is by no means an exhaustive set. However, other sources of evidence, when available, can be added to the initial set and our approach would still be applicable.

II. Evidence Collection

The first step in our framework is to collect a very large set of of features that is potentially relevant for intrusion detection. Evidence collection is generally a hard problem. Modern systems are so complex that they present a seemingly limitless supply of features. This is one of the primary reasons why traditional anomaly detection systems rely so heavily on features selected by domain experts. However, not only is it hard to do this for all types of attacks, this may result in some key evidence features being left out from consideration. Our approach is to place monitors or sensors at multiple system layers. This is because the effects of attacks on networked systems are not felt in isolation at a single layer; their effects manifest themselves at different system layers. Some system based diagnostics are readily available (e.g., application logs), while for others we place hooks (e.g., in the OS) to collect the evidence. The rest of this section describes our monitoring of four system layers.

Hardware Layer: Typical Unix based systems export hardware statistics via hardware counters. These counters are intended for diagnostics purposes since they are indicators of the overall system health. They are extremely optimized with little sampling overhead. The values in these counters can be easily accessed via system API calls. We use all available hardware counters (on our test system) as features that can be sampled as a function of time. Examples of such features include CPU operating frequency, CPU utilization, memory consumption, cache hits, cache misses, core temperature, etc. The actual set of hardware level features that can be used is ultimately a function of the system under consideration.

OS Layer: The OS layer is the source of a wealth of information, not all of which, unlike hardware counters, can be easily accessible. However, the OS can be instrumented to provide information. We use the linux */proc* file system as our primary source of OS level features in this work. The */proc* is a virtual file system that exports information about OS state in the form of parsable text files (the files usually take the form *Variable: Value*). It is important to realize however, that the */proc* file system contains thousands of files (each process has its own set of files cataloging behaviors) the active monitoring of which, together can typically overwhelm a single server. For now, we only consider files in the top level directory and even this results in a large volume of evidence; in heavy duty custom systems, the sub-directories can be parsed also. We developed a file parser that periodically pings all top level files and records the value of each variable. We use a sampling frequency of one lookup per second so as to not overwhelm the server. Examples of features that can be obtained using this methodology include the number of system calls, the kernel load, the number of filesystem lookups and the number of system interrupts.

Application Layer: Primary sources of application layer information are log files. To detect DoS attacks we look at all log files produced by the Apache web server and to detect SQL injection attacks we look at all available MySQL and Wordpress logs. The methodology is similar to the one we described above but instead of */proc* files we parse application layer log files. We have developed a log file parser that checks log files periodically and records the value of each variable that is present. Examples of features that are extracted from log files include web server accesses, the number of requests that are being served, number of requests queued, database lookups and SQL errors.

Network Layer: At the network layer, examine the raw packet stream plus raw network statistics exported by the OS. We only consider packet header information as features. Packet analysis allows us to collect information such as the number of packets plus the kind of packet (TCP SYNs, TCP ACKs, etc). We primarily use the *netstat* tool which gives various network layer diagnostic information that we use as features. Examples include SYN packets, Sockets in ESTABLISHED state, sockets in CLOSED state and network bandwidth.

Table I summarizes the sources of the features we collect.The methodology outlined above allows us to collect a set of 365 features. This list is by no means exhaustive. For example, one potential source of evidence for SQL injection might be obtained by looking at the SQL queries themselves. This requires deep packet inspection, which we do not consider due to its computational overhead. Other evidence sources, however, can be added to our framework as desired.

In essence, the relevant features are selected by launching normal behavior and attacks. A large set of features is collected, and passed on to the feature selection algorithms. The algorithms each output a subset of features that they think are optimal. During runtime, these can then be sampled with higher frequencies to facilitate highly accurate detection.

III. Feature Selection

Our evidence collection module yields a large number of features (in our prototype we collect 365 features in total). The features are collected to classify each attack and normal behaviors. Feature Selection (or choosing the best subset of features from a large set) involves two components. The first is an objective function and the second is a search algorithm. The objective function evaluates candidate subsets and returns a quantification of their "usefulness". This is then used by the search algorithms to select new candidate subsets. We want to select features that have high correlation with attacks and low cross-correlation across features (the latter identifies redundant and therefore unnecessary features.) We consider three search algorithms, viz., Linear Forward Selection (LFS), Sequential Backward Selection (SBS) and a Simple Genetic Algorithm (SGA).

Objective Function: We use the correlation based subset evaluator detailed in [10] as our objective function. This evaluation function has been shown to have good performance and it also ties in nicely with our search goal. It evaluates the usefulness of a subset by computing the correlation between the features and the classification classes (attacks and normal behavior.) It also tries to minimize the redundancy between features. Two features that are highly correlated (regardless of the labeled classes) are considered redundant and only one is selected. In simple terms, subsets of features that have low cross correlation and high correlation with the labeled classes (attacks and normal behavior) are preferred.

Linear Forward Selection: Linear Forward Selection [11] is an optimization of the popular search algorithm, Sequential Forward Selection [12] (SFS). In its simplest form, SFS starts with an empty set and sequentially adds features (i) such that at each step, $F(Y, i)$ is maximized, where Y is the set of previously selected features and F is a function that calculates the usefulness of a subset of fearures. SFS is essentially a simple hill-climb search; it evaluates all possible single feature expansions of the current set. The feature that results in the highest score is added permanently and the algorithm terminates when no single feature expansion improves the current score. The problem with SFS is that the number of subset evaluations that must be performed grow quadratically with the number of features. This is not a big problem when the search space is small but becomes a concern for large sets. LFS improves on SFS by limiting the number of features that must be considered at each step to 1. This means that LFS has an upper bound on the running time given by $\frac{N(N+1)}{2}$ where N is the total number of features under consideration.

A forward selection algorithm however, cannot remove a priori added features that become redundant with the addition of new features. However, forward selection algorithms in general are known to perform well when the optimal subset of features is small. In addition, LFS has low computational overhead.

Sequential Backward Selection: Sequential Backward Selection [12] is the logical reverse of SFS. It starts with the full set of features and sequentially eliminates the feature i that "least" reduces the total value of the set under consideration (as measured by the objective function).

The main weakness of SBS is that it cannot re-examine the usefulness of a feature after it has been eliminated. SBS works best (in terms of the usefulness of its output) when the optimal feature subset is large because SBS, due to the fact that it starts with the full set, spends most of its time evaluating larger subsets.

Genetic Algorithm: Genetic Algorithms [13] are a class of search algorithm that try to emulate the process of Darwinian natural selection. A more detailed description can be found in [13]. All genetic algorithms can be thought of as 5 stage processes. Initial population selection, fitness function application, selection, crossover and mutation. Genetic algorithms have been shown to produce very good results in a variety of domains [13]. However, they are also known to be computationally expensive.

A genetic algorithm begins with an initial population that consists of random subsets of solutions (which correspond to features in our context). The fitness function (synonymous with previously discussed objective function) evaluates the fitness of each member of this population. The genetic algorithm ranks each subset according to its fitness. Then, some of the fittest subsets are chosen to 'reproduce' to create a new generation of subsets. This process continues until a terminating condition is met. The question then is how this reproduction takes place. The reproduction of two pairs involves randomly selecting crossover points in the two pairs and combining them. As an example, consider two feature subsets that have been chosen to reproduce. The crossover process involves splitting both subsets at random points and combining the split of one subset with that of the other subset to come up with two new subsets. Each set of features is then subjected to a random mutation where elements in the set can randomly change (this probability is usually very low). The steps from selection to mutation are then repeated until a terminating condition is met or the algorithm is stopped. We use the Simple Genetic Algorithm [14] in our experiments.

The feature selection algorithms are given as input, the initial labeled set of features and they output subsets that they think are optimal. In Section VI, we provide a detailed comparison of how the three algorithms described here perform with an evidence feature set for detecting DoS and SQL injection attacks.

IV. Inference Engines

In this section we will describe the two inference engines we consider for detection viz, Dempster-Shafer Theory of evidence, and K-Means classification. In terms of traditional detection engines, DST is closest to supervised learning. However, it provides some advantages discussed later. K-means, on the other hand, is an unsupervised learning algorithm for classification. Our expectation is that the optimal set of features should be usable with any inference engine to accurately detect the presence or absence of attacks.

A. Dempster-Shafer Theory of Evidence

The Dempster-Shafer theory of evidence (DST) [15] is a theory for combining evidence and reasoning about uncertainty. We use it in our framework to reason about the quality of the feature subsets that are output by the feature selection algorithms in terms of detection accuracy. Every hypothesis is assigned a belief ranging from 0 to 1 where 0 means that there is no evidence to support a hypothesis and 1 means absolute certainty with regards to the hypothesis. DST is fundamentally different from Bayesian reasoning because belief in a hypotheiss and its negation need not sum to 1; in fact, both values can be 0 (meaning that there is no evidence either for or against the hypothesis).

Let $\Theta = \{\theta_1, \theta_2, ..\theta_n\}$ be the set of possible conclusions to be drawn, then the θ_is are mutually exclusive and Θ is exhaustive. The goal of DST in our context is to predict the state of the system as a function of time. More precisely, let the set of known system states, for a particular class of attacks (e.g., DoS or SQL injection), be $SS = \{sa_1, sa_2, sa_3 ... sa_n, sn\}$ where the state $sa_i, 1 \leq i \leq n$ represents the state of the system when it is under attack a_i and state sn represents the state of system when it is operating normally. DST assumes that the current state of the system at time t , CS(t), is unknown but must be one of the values from the *Frame of Discernment*, SS. Note that each state in SS is observable.

CS(t) is determined by providing and combining *Basic Belief Assignments (BBAs)*. If SS is the frame of discernment then a function $m : 2^{SS} \rightarrow [0,1]$ is called basic belief assignment if the following conditions hold:

$$m(\emptyset) = 0, \qquad \sum\nolimits_{A \subset 2^{SS}} m(A) = 1.$$

The term m(A) is called A's BBA and is a measure of the ***belief*** that is committed to exactly A. In DST, the notions of *belief* and *plausibility* are used to reason about the certainty (or lack of) in the system being in a particular state. The belief function, Bel, is a mapping $Bel : 2^{SS} \rightarrow [0,1]$ and is given by:

$$Bel(A) = \sum_{B \subset A} m(B).$$

The Plausibility Pl is a mapping $Pl : 2^{SS} \rightarrow [0,1]$ and is given by:

$$Pl(A) = \sum_{B \cap A \neq 0} m(B).$$

As should be evident, the belief and plausibility with DST serve as upper and lower bounds on the degree of certainty of the system being in a particular state. The rest of this section will describe how we use system wide features (or observables) and DST to compute belief and plausibility about system states.

Observables: An observable in our context is simply anything that can be sampled or monitored to produce a time series. Relevant examples of observables are the number of total bytes received over a network interface and CPU consumption expressed as a percentage. We use the terms observables and features interchangeably. The observables might be important indicators of the presence or the absence of an attack. For example, one might expect the total number of bytes received over a network interface to be very high when the system is under a flooding based DoS attack. More specifically, the set $O = \{o_1, o_2, o_3...o_n\}$ is the set of observables (or features) used to detect attacks. A better O will result in higher detection rates.

An observable, depending on its current value, provides a belief over SS that provides a hypothesis on the CS. As an example, consider the number of SYN packets received in a given time window, denoted by A. As is well established, a large number of TCP SYN packets in a small time frame is highly indicative of a SYN flood attack[16]. When the value of A crosses a predefined threshold, it provides a belief in a SYN flood attack denoted by $B_A(SYN) = 0.8$ (say). The remainder of A's belief is always allocated to the frame of discernment specified as $B_A(SS) = 0.2$.

Combining Evidence from different sources: A core feature of DST is combining beliefs from independent sources of evidence. In our particular context, this can be thought of as observing two different features (or observables), say A and B. When A and B exceed their predefined thresholds they provide beliefs in attack x given by $B_A(x)$ and $B_B(x)$ respectively. Combining different features results in a unified belief over a particular state in SS. Because we have evidence from multiple sources, our case is not as simple as using a single fusion operator (fusion operators are designed to work on exclusively dependent or exclusively independent beliefs). Thus, we use the averaging and the cumulative fusion operators (details in [17] and [18], respectively). The averaging operator is intended to be used with dependent beliefs and the cumulative operator is intended to be used with independent beliefs. We first use the averaging operator to combine subsets of dependent beliefs and then use the cumulative operator to combine the resulting independent beliefs. (dependent beliefs can be thought of as those belonging to the same sensor).

The Averaging function is given by:

$$\begin{aligned} AVG(x) &= \frac{B_A(x)B_B(SS) + B_B(x)B_A(SS)}{B_A(ss) + B_B(SS)} \\ AVG(SS) &= \frac{2B_A(SS)B_B(SS)}{B_A(SS) + B_B(SS)} \end{aligned} \tag{1}$$

The cumulative function is given by:

$$\begin{aligned} CUM(x) &= \frac{B_A(x)B_B(SS) + B_B(x)B_A(SS)}{B_A(SS) + B_B(SS) - B_A(SS)B_B(SS)} \\ CUM(SS) &= \frac{2B_A(SS)B_B(SS)}{B_A(SS) + B_B(SS) - B_A(SS)B_B(SS)} \end{aligned} \tag{2}$$

Belief and Plausibility: The application of the Dempster-Shafer framework results in belief and plausibility values for each attack. These allow us to reason about the usefulness of a particular subset of features in terms of providing high detection accuracy.

B. Clustering Algorithm

Clustering algorithms are one of the most popular classes of machine learning (ML) algorithms that are used for anomaly detection [19] [3]. Generally speaking, clustering is a technique for finding patterns in unlabeled data. We use the K-Means clustering algorithm because it is one of the simplest and relatively efficient of clustering algorithms which has been successfully used for anomaly detection.

It works by grouping similar objects into K disjoint clusters. We will only provide a high level overview of how the algorithm works here. Details can be found in [20] which demonstrates how to apply the algorithm for intrusion detection. Fundamentally however, the algorithm is built around the notion of a centroid.The centroid of a cluster is a point in the feature space that can be thought of as the most representative point for that cluster. Once the centroid for each of the K clusters is known, the algorithm simply compares each new instance to each of the K centroids to determine which one it is closest to. The algorithm has two phases, an offline clustering (training) phase, and an online classification phase.

Clustering: During clustering the goal is to train the system. Put another way, the goal is to determine the optimal centroids. The algorithm is fed a set of instances (which in our case are defined by the output of the feature selection algorithms). Clustering then is a five step process.

1) Initiate the number of clusters, K to some user defined value. In our case, K is the number of elements in SS, which corresponds to the number of attacks and normal behavior.
2) Initiate the K cluster centroids. This is typically done by arbitrarily choosing K data points from the set of training data.
3) Iterate over all training objects and compute the distance of each object to the centroids. Assign each object to the cluster with the nearest centroid.
4) recalculate centroids (ensuring that a previously chosen point is not chosen again)
5) Repeat step 3 until assignment of objects is static. i.e, between two different iteration with different centroids, the assignment of objects to clusters remains the same

For the distance measure, we use the Euclidean distance. Its effectiveness was demonstrated in[20]. At the end of the clustering phase we get K centroids that will be used as future references.

Classification: During classification (done online) each new data point is simply compared against all the previously computed K centroids and assigned to the one it is closest to. As in the case of our DST approach„ we perform this entire process seperately for DoS and SQL injection.

V. Experimental Setup

Testbed: We perform our experiments on the The Cyber Virtual Assured Network (CyberVAN) testbed [21]. The CyberVAN is a state of the art cyber security testbed that was designed to support experimentation in a virtual cyberspace. CyberVAN models hosts as full fledged virtual machines and models the underlying network using discrete event network simulation. The testbed can be used to model a realistic cyber network environment with high fidelity. Virtual machines that act as end hosts are time synchronized with the discrete event network simulator. This enables CyberVAN to slow down VM time if the simulated network cannot keep up with real time such as in the case of high volume traffic.

Network Topology: Our network topology consists of one server machine, client machines from 50 different subnets with IP addresses widely distributed across the public Internet, and one attacker machine. The server has *CentOS* server version 6.6 and *Apache* version 2.2. The server hosts 20 different websites, each with a complex navigational structure. The client machines are all running *Ubuntu* version 14.04. Each client machine has a synthetic user that interacts with the server using a FireFox web browser. The clients run in one of two different modes, web-only or database-only.

The normal traffic (which keeps flowing regardless of whether or not an attack is under way) is generated based on the patterns of real users on the Internet as discussed in [22]. The synthetic users are emulated to use actual applications. For web-only applications the users interact with static web content using a Firefox web browser and for database-only, they interact with a Wordpress blog. The eventual traffic contains realistic short and long term HTTP and TCP connections.

Methodology: We consider two classes of attacks to validate our framework, viz.. Denial of Service (DoS) and SQL injection. Since we expect these two classes of attacks to manifest different (and often disjoint) kinds of symptoms, we experiment with only one class of attack at a time. Specifically, we first collect features explicitly for DoS attacks via the data collection approach outlined above. We test out three different TCP based DoS attacks. SYN Floods [16], Sockstress [23] and Slowloris [24].

For SQL injection we test two different attacks. The first attack exploits a Wordpress vulnerability wherein a specially crafted SQL query results in the entire database being returned as the response. The second attacks exploits another vulnerability wherein a specially crafted string opens up a shell and gives the attacker root access. SYN flooding is one of the oldest forms of DoS attacks and operates by flooding a victim with TCP SYN packets. Sockstress, is more complicated in that it actually completes the TCP handshake in an attempt to exhaust all sockets. Slowloris is an HTTP attack that opens numerous HTTP connections within a time and then periodically sends keep alive messages to hold them.

To test our approach against a previously unobserved attack, we build such an attack. Specifically, we employ the methodology described in [5] to intelligently combine a SYN Flood and Slowloris attack. Each component of the attack, by itself, does no discernible damage and thus, cannot be detected by traditional detection approaches; but together the components consume ports and thus, constitute a powerful but stealthy DoS attack.

We monitor a total of 365 features for each state in SS. Each state is monitored for a minute after which we are left with a time series of each feature. This data is then labeled according to the state it belongs to and is then fed into our feature selection engine. Figure 1 depicts how the entire process works to produce a set of features. Once those features are selected they are fed into

	DoS	**SQL Injection**
LFS	Context Switches, Sys calls, Free Memory, HTTP Connections	Established Connections, DB Errors, Bytes Sent
SGE	Sys Interrupts, Sys Calls, Used Memory, HTTP Connections	DB Errors, Bytes Sent, DB lookups
SBS	Page Faults, CPU1 Utilization, Established TCP connections, Swap Space	CPU1 Utilization, Bytes Sent, Page Faults

TABLE II: A subset of features selected by each algorithm

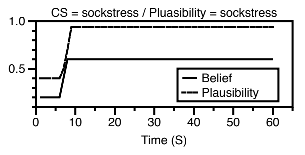
each inference engine seperately. For DST, we examine the generated time series and assign thresholds which, when triggered, output a pre-assigned belief in an attack. Taken as such these observables are just binary belief functions. For example, one obvious feature for detecting SYN floods is the number of SYN packets received. We assign equal beliefs to all features regardless of perceived importance. A better assignment of these beliefs can yield better detection accuracy but is out of the scope of this paper. For K-Means, we train the system as described above setting the value of K to the cardinality of SS (the number of clusters is equal to the number of attacks plus one for normal behavior)

During runtime, the set of features that were output by the different algorithms (via offline training) are monitored. The attacks that were previously described are launched in real time. For DST, the observations (of the features monitored) are input into equations (1) and (2), and we obtain measures of the belief and the plausibility for each candidate scenario (different attacks and normal behavior). For K-means we simply monitor whether each attack (or normal behavior) is correctly classified.

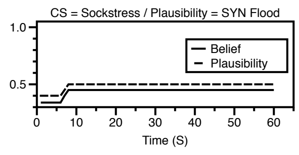
VI. Experimental Evaluations

In this section we discuss the results from our experiments. For each algorithm, we are interested in evaluating how well the features that it selects are able to differentiate between the different kinds of attacks and normal behavior. Table II lists a small subset of the features (due to space constraints) that were selected by each algorithm. The selected features, due to the nature of modern systems, will be highly dependent on the execution environment. The table highlights the fact that LFS and SGE tend to select similar features while SBS does not. We will first present results using the DST inference engine, and later using the K-Means inference engine.

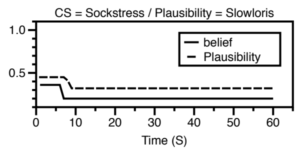
Detection using DST: Figures 2, 3, and 4 show how well the features chosen by LFS, SBS, and SGA, respectively, perform when a Sockstress attack is initiated. The figures shows how the belief and the plausibility vary over the time period of the attack. The attack is initiated at time 6 seconds. Figures 2a, 3a and 4a show how both the plausibility and the belief increase, indicating a strong conviction in the present state (sockstress). The other figures show that there are low values of belief and plausibility with respect to the other possible scenarios (e.g., normal behavior) indicating that DST has very little conviction that the current state corresponds to one of these. Figures 2d, 3d and 4d show how the belief and the plausibility in the current state being "normal" plummets once the attack is launched. The slight increase in belief and plausibility exhibited in Figures 2b, 3b, and 4b is



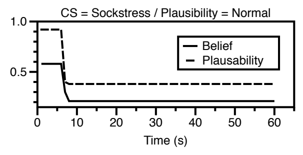
(a) sockstress



(b) Syn Flood

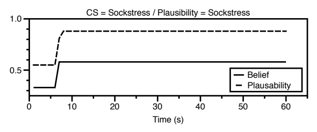


(c) Slowloris

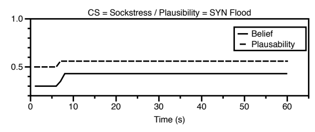


(d) Normal Traffic

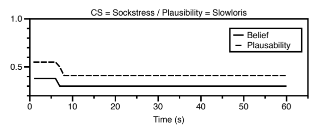
Fig. 2: Belief and plausibility during a Sockstress attack with features selected by LFS



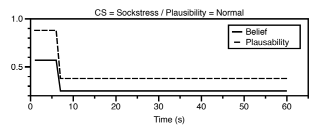
(a) sockstress



(b) Syn Flood

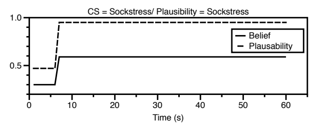


(c) Slowloris

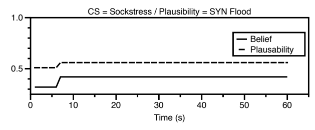


(d) Normal Traffic

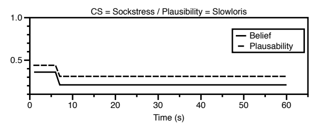
Fig. 3: Belief and plausibility during a Sockstress Attack with features selected by SBS



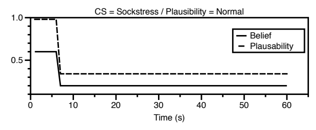
(a) sockstress



(b) Syn Flood

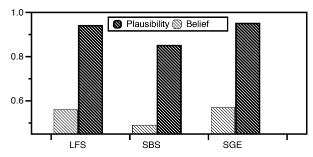


(c) Slowloris

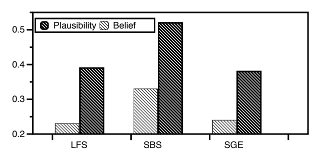


(d) Normal Traffic

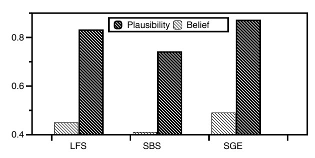
Fig. 4: Belief and plausibility during a Sockstress Attack with features selected by SGA



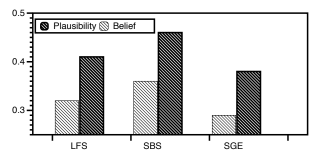
(a) DoS: Current State is the correct one



(b) DoS: Current State is the wrong one



(c) SQL Injection: Current State is the correct one



(d) SQL Injection: Current State is the wrong one

Fig. 5: Average Belief and plausibility for DoS and SQL Injection

due to the fact that SYN Flood and sockstress are similar kinds of attacks. For example, both involve large volumes of traffic and thus, sometimes trigger the same sensors. The interesting observation from these figures is that the detection accuracy (belief and plausibility) with LFS is comparable to that of SBS (they are within 2% of each other); however, SBS lags visibly. The belief and the plausibility results with all the algorithms, for each state, exhibit the same behaviors observed in Fig 2. For the current "true" state these metrics increase and generally decrease otherwise. We omit the results for the SYN flood attack due to space limitations but they exhibit similar patterns of results.

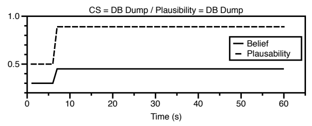
In Figures 5a and 5b, we summarize the plausibility and belief results for DoS with the different algorithms. Specifically, we show the average value of these metrics observed (over all cases) with regards to the true or "correct" state (e.g., the belief and plausibility with Sockstress when it is actually in effect) and the "wrong" state (e.g., the belief and plausibility with Sockstress when Slowloris is in effect). We see that the values of these metrics are much higher with the correct case (low values are exhibited for wrong cases).

SQL injection results for the database dump attack, are detailed in Figures 6, 7 and 8. They exhibit behaviors similar to what was observed with DoS. However, across all the three algorithms, the difference is that the belief and the plausibility results are not as high (compared to DoS) in the "correct" state, and not as low with the wrong "state". We believe this is because a lot of good features for SQL injection are in fact embedded in SQL queries which we do not consider in this work (this is left for the future).

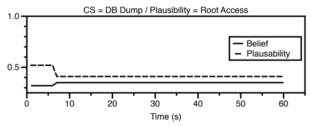
The results with SQL injection are summarized in Figures 5c and 5d. Again, we see that the performance with SGA and that with LFS are still close (always within 10 % of each other). However, here SGA gives an approximately 6 % performance improvement over LFS consistently, thus demonstrating that higher complexity could yield better accuracy. Both LFS and SGE outperform SBS again.

Detection using K-Means clustering:

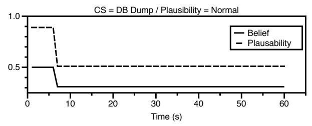
To evaluate the features when K-Means is used for inference we use accuracy (percentage of samples that are correctly classified) as our metric. Tables III and IV detail the results. For both SQL injection and DoS we see that the performance of LFS is comparable to SGE. As with DST we observe that we get better accuracy in detecting DoS attacks. This is due to the fact that a lot of good features for SQL injection are in the actual



(a) Database Dump

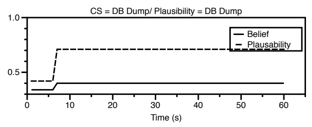


(b) Root Access

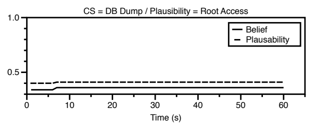


(c) Normal Traffic

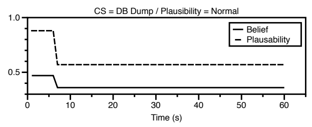
Fig. 6: Belief and plausibility during a Database Dump Attack with features selected by LFS



(a) Database Dump

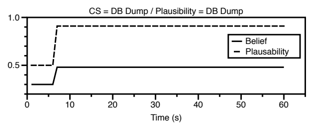


(b) Root Access

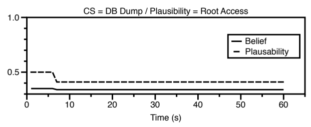


(c) Normal Traffic

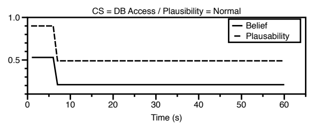
Fig. 7: Belief and plausibility during a Database Dump Attack with features selected by LBS



(a) Database Dump



(b) Root Access



(c) Normal Traffic

Fig. 8: Belief and plausibility during a Database Dump Attack with features selected by SGA

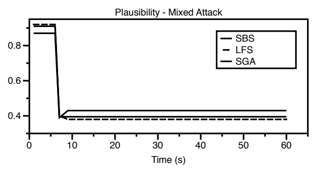


Fig. 9: Plausibility for a mixed attack

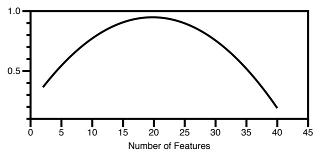


Fig. 10: Change in plausibility for DoS (DST)

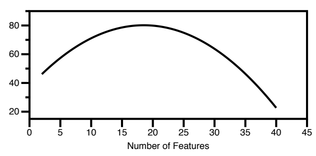


Fig. 11: Change in accuracy for DoS (K-means)

query accessible via DPI which we do not consider. We also observe that the performance of SBS lags similarly to the other two. However with K-Means, the discrepancy is much greater. This is greater testament to how completely SBS is outperformed. Some of the weaknesses inherent in SBS (for e.g. it cannot reexamine features that have been eliminated) are masked by DST because DST deals very well with noisy features.

Comparing algorithms: A comparison of performance across the different algorithms yields some interesting insights. First both LFS and SGE yield consistently better results than SBS. The fact that LFS outperforms SBS so consistently indicates that the ideal subset of features is small. SBS is known to underperform in such cases. LFS is also much quicker than SBS.

As mentioned previously, we expect SGA to perform well because of its resistance to getting caught in local maxima and minima. It also takes the most time to finish. However, in some cases we find that LFS does slightly better than SGA (e.g., see Figures 2a and 4a). The difference however, is always small (results are always within 6 % of each other) . These results suggest that local maxima and minima are unlikely and the use of LFS (much faster) can suffice for highly accurate detection.

Using more or less features than what is recommended by the algorithms: Using too few or too many features can hurt detection performance. To demonstrate this, we configured LFS to output multiple sized feature subsets. (i.e., the best 2, the best 4 the best 6 and so on). We then tested these feature sets with all the three DoS attacks considered and computed the average plausibility of predicting the correct state with DST and the average classification accuracy with K-means. The results are shown in Figures 10 and 11. We see that with DoS, the optimal number of features is somewhere between 15-20 for DST and between 11-16 for K-means. A higher set could lead to wrong conclusions; a smaller set could reduce detection accuracy.

Mutated attack: Finally, we are interested in evaluating how well the chosen features hold up to an unknown attack. To do this, we launch the mixed attack that was described previously in Section V. Figure 9 shows how the plausibility that the current state is normal drops in the presence of a mixed attack (signifying a high likelihood of an attack). LFS again outperforms the other two (SGA is still very close). K-means classifies the mixed attack as either one of the two mixes (depending on the dominant attack)

VII. Related Work

There has been a lot of work done on feature selection in the domain of anomaly detection. However, most prior efforts (unlike ours) are tied to specific classification approaches [6], [7], [8]) They are also typically only concerned

	SYN	Sock	Slowloris	Normal
LFS	86	88	83	85
SGE	86	87	85	83
SBS	75	72	70	77

TABLE III: Accuracy of k-means classifier under DoS attacks

	Root	DB Dump	Normal
LFS	74	77	75
SGE	75	78	74
SBS	62	65	66

TABLE IV: Accuracy of k-means classifier under SQL injection attack

LFS	SBS	SGA
2.9	10.8	65

TABLE V: Average completion time in minutes

with network layer features. In [25], the authors evaluate various selection techniques, including genetic algorithms in the context of intrusion detection. However they are only concerned with features that originate from the network layer. Their approach is limited because their initial feature set (on which they apply feature selection) is itself manually selected. In [26] the authors develop a decision tree based genetic algorithm. They use decision trees to guage the performance of their algorithm which is not immune to noise or uncertainty. In [27] the authors compare and contrast two well known feature selection algorithms. They employ the DARPA intrusion detection data set [28] which only contains network; cross layer features are not considered. [29], [30] and [8] are other examples which only consider network layer features for intrusion detection.

Other approaches such as [29] are concerned with extracting features so that classification accuracy is not hurt. This is fundamentally different from what we are trying to do because we are concerned with large data sets with potentially frivolous features.

VIII. Conclusion

In lieu of manually choosing features as done traditionally, we develop a framework that automates and sequentializes the process of feature selection for highly accurate intrusion detection. In building our framework, we design and implement a comprehensive evidence collection framework, and undertake an in-depth study to gain an understanding of which search algorithms to use for feature reduction. Our approach is agnostic to the engine that uses these features to perform inference. We demonstrate the efficacy of our framework with DoS and SQL injection attacks. We demonstrate that the features (which are automatically chosen) work very well in terms of providing high detection accuracy with respect to the true states (attack and normal scenarios) the networked system is in.

Acknowledgment: The effort described in this article was partially sponsored by the U.S. Army Research Laboratory Cyber Security Collaborative Research Alliance under Cooperative Agreement W911NF-13-2-0045. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation hereon.

References

[1] "Akamai releases q2 2015 state of the internet - security report," http://akamai.me/1qN434s.
[2] "Cyber attacks likely to increase," http://pewrsr.ch/1qN4agg.
[3] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Technical report Chalmers University of Technology, Goteborg, Sweden, Tech. Rep., 2000.
[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.
[5] A. Aqil, A. O. Atya, T. Jaeger, S. V. Krishnamurthy, K. Levitt, P. D. McDaniel, J. Rowe, and A. Swami, "Detection of stealthy tcp-based dos attacks," in *MILCOM*. IEEE, 2015.
[6] M. Kloft, U. Brefeld, P. Düessel, C. Gehl, and P. Laskov, "Automatic feature selection for anomaly detection," in *Proceedings of the 1st ACM Workshop on AISec*. ACM, 2008.
[7] W. Ng, R. Chang, and D. Yeung, "Dimensionality reduction for denial of service detection problems using rbfnn output sensitivity," in *Int'l Conf on Machine Learning and Cybernetics,*, 2003.
[8] A. H. Sung and S. Mukkamala, "The feature selection and intrusion detection problems," in *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making*. Springer, 2005, pp. 468–482.
[9] F. Iglesias and T. Zseby, "Analysis of network traffic features for anomaly detection," *Machine Learning*, 2015.
[10] M. A. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, The University of Waikato, 1999.
[11] M. Gütlein, E. Frank, M. Hall, and A. Karwath, "Large-scale attribute selection using wrappers," in *IEEE CIDM*, 2009.
[12] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques: concepts and techniques*. Elsevier, 2011.
[13] T. Bäck, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. OUP, 1996.
[14] D. Goldberg and J. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, pp. 95–99, 1988.
[15] G. Shafer *et al.*, *A mathematical theory of evidence*. Princeton University Press, 1976.
[16] W. Eddy, "TCP SYN flooding attacks and common mitigations," in *RFC 4987*, Aug 2007.
[17] A. Josang, J. Diaz, and M. Rifqi, "Cumulative and averaging fusion of beliefs," *Inf. Fusion*, vol. 11, no. 2, pp. 192–200, 2010.
[18] K. Sentz and S. Ferson, *Combination of evidence in Dempster-Shafer theory*. Sandia National Laboratories, 2002.
[19] M. Ektefa, S. Memar, F. Sidi, and L. S. Affendey, "Intrusion detection using data mining techniques," in *Information Retrieval & Knowledge Management,(CAMP), 2010*. IEEE.
[20] G. Münz, S. Li, and G. Carle, "Traffic anomaly detection using k-means clustering," in *GI/ITG Workshop MMBnet*, 2007.
[21] A. Poylisher, Y. M. Gottlieb, C. Serban, J. Lee, F. Sultan, R. Chadha, C. J. Chiang, K. Whittaker, J. Nguyen, and C. Scilla, "Building an operation support system for a fast reconfigurable network experimentation testbed," in *MILCOM*. IEEE, 2012.
[22] W. Dumouchel and M. Schonlau, "A comparison of test statistics for computer intrusion detection based on principal components regression of transition probabilities," in *30th Symposium on the Interface: Computing Science and Statistics*, 1998.
[23] "Sockstress tools & source code," http://bit.ly/1SgI9Qd.
[24] "Slowloris HTTP DoS," http://ha.ckers.org/slowloris/.
[25] C.-H. Tsang, S. Kwong, and H. Wang, "Genetic-fuzzy rule mining approach and evaluation of feature selection techniques for anomaly intrusion detection," *Pattern Recognition*, vol. 40, no. 9, pp. 2373–2391, 2007.
[26] 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*. ACM, 2005, pp. 136–141.
[27] S. Chebrolu, A. Abraham, and J. P. Thomas, "Feature deduction and ensemble design of intrusion detection systems," *Computers & Security*, vol. 24, no. 4, pp. 295–307, 2005.
[28] "DARPA intrusion detection evaluation," http://bit.ly/1NtBr50.
[29] S. Mukkamala and A. Sung, "Feature selection for intrusion detection with neural networks and support vector machines," *Transportation Research Record: Journal of the Transportation Research Board*, no. 1822, pp. 33–39, 2003.
[30] J. Platt *et al.*, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.