SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2012; **00**:1–15
DOI: 10.1002/sec

RESEARCH ARTICLE

An Exemplar-based learning approach for Detection and Classification of Malicious Network streams in Honeynets

Fahim H. Abbasi[1]*, Richard Harris[1], Stephen Marsland[1], and Giovanni Moretti[1]

[1]School of Engineering and Advanced Technology (SEAT), Massey University, Private Bag 11 222, Palmerston North, New Zealand
{F.Abbasi, R.Harris, S.R.Marsland, G.Moretti}@massey.ac.nz

ABSTRACT

Variants of malware and exploits are emerging globally at an ever-increasing rate. There is a need to automate their detection by observing their footprints over network streams, but signature-based intrusion detection systems alone cannot cope with the dynamic nature of modern security threats. In this paper we approach intrusion detection as a classification problem and describe a system using exemplar-based learning to correctly classify known classes of malware and to detect, learn and classify unknown malicious streams into classes. Copyright © 2012 John Wiley & Sons, Ltd.

KEYWORDS

Intrusion Detection System (IDS), Malware, Normalized Compression Distance (NCD), Spamsum, Exemplar-based Learning

*__Correspondence__

E-mail: f.abbasi@massey.ac.nz

Received . . .

1. INTRODUCTION

Our IT infrastructure and resources are threatened on a daily basis by worms, viruses, botnets and even state-sponsored cyber militants. The attacker's motivation can range from pranks, to personal gain, to lucrative business models and cyber-espionage. The goals, however, are the same; to gain unprivileged access to private data, information and computing resources. Symantec, a major anti-virus company reported blocking over 5.5 billion malware attacks in 2011, with 403 million of these being new variants of malware[1]. Intrusion detection systems (IDS) are an important part of the fight again these attacks.

Current intrusion detection methods tend to be static, rule-based systems such as firewalls and anti-virus software. They typically match against the signatures of known threats in packet content and apply rules to drop, forward, log, or accept such packets. The signatures are identified and analysed by techicians within a company and added to the database used by the system [2]. This process is labour intensive and requires a detailed analysis of the malware or exploit. A good signature should be specific enough to narrow down the exploit and avoid false positives and yet be generic enough to catch variants of the same exploit.

Another approach to intrusion detection has been to learn or otherwise identify models of normal or expected network traffic and raise an alarm when significant deviations from this behaviour are identified. These are known as anomaly-based IDS [3].

Before either signatures or a model of 'normal' traffic can be created, one challenge is to discover the tools and tactics that are being used by hackers. To this end, apparently benign computer systems designed to keep detailed logs of system activity are widely deployed today by security researchers. These systems, known as 'honeypots', are designed to record a hacker's activities to gain an insight into the methods used. The logs typically would include intruder keystrokes, processes, and system-wide and network data.

As has already been mentioned, the specification of a sufficiently specific signature is not easy. However, the matching of signatures is also a problem, since relatively small variations in the packet data can make the overall appearance look significantly different, which means that it is not identified. This is a common approach in malware [4]. Any attempt to make the signature matching too generic leads to false positives, which greatly undermine the confidence in the IDS, and can result in it being ignored or disabled.

The most common methods of introducing variation into packet data are data scrambling or transposition, where the order of symbols forming the pattern changes, but the inherent symbols remain the same. These

Prepared using secauth.cls [Version: 2010/06/28 v2.00]

syntactically distinct, but symbolically similar, strings have similar entropy measures and so can be identified in this way using information theoretic measures such as the Kolmogorov complexity [5]. However, this does not work if new symbols are added, or the old ones are re-encoded, although other string metrics can be useful here.

In this paper we describe an IDS that uses string matching and compression on honeypot network data and an exemplar-based learning system to detect variants of known attacks while they being transferred over a network. The similarity between known malicious stream samples and the incoming traffic is calculated based on entropy measures of the strings. Synergies are thus quantified to yield a similarity score with a certain level of confidence, and this is used to classify examples of known attacks, and to add new classes of attack where there is no known exemplar, so that the system can engage in lifelong learning and continue to extend itself. We demonstrate our approach on data from real honeypots.

2. ORGANIZATION

The remainder of this paper is organized as follows: ...

3. RELATED WORK

Generally IDS can be categorized under 2 main categories, which are:

1. System or Host-based IDS (HIDS)
2. Network-based IDS (NIDS)

In order to solve the intrusion detection challenges, researchers in the past have focused on many techniques, which can be categorized under the following main categories, which are:

1. Misuse-based intrusion detection
2. Anomaly-based intsudion detection
3. Hybrid detection
4. Scan detection
5. Profiling Modules

Another interesting way to categorize Intrusion detection techniques, is by determining the study perspective or the algorithm or approach as:

1. Header-based detection
2. Payload/content-based detection
3. Hybrid approach
4. System calls based
5. TCP/IP data based
6. Binary/executable analysis based
7. Network flow aggregation based

Within machine learning they can be categorized as:

1. Statistical method based detection
2. Classification based detection
3. Clustering based detection
4. Information theoretic based detection

Misuse detection is used to identify unique patterns of unauthorized activity, and use these patterns to predict and detect subsequent similar attempts. Lee et al. [6] applied association rules in audit data and network traffic for misuse detection. Mukkamala et al. [7] applied SVM on host and network logs to identify attacks and misuse patterns causing computer security breaches. Kruegel and Toth [8] designed an algorithm to generate a decision tree for fast detection of malicious events from network data. Chebrolu et al. [9] applied the CART-tree algorithm on the KDD cup 1999 intrusion detection dataset and eliminated features that did not contribute to the ranking, which increased overall accuracy of the system. They also investigated feature selection and classification algorithm involving bayesian networks on host data [9]

Anomaly detection systems raise alarms on detecting significant deviations from normal patters or models. Liu et al [10] used artificial neural network (ANN) techniques to analyze sequences of system calls for anomaly detection with good accuracy. Chen et al. [11] compared SVM and ANN on 1999 DARPA evaluation set and BSM audit data and found that SVM outperformed ANN. Liao et al. [12] used KNN to classify program behaviours as normal or intrusive based on their system calls. For their dataset they observed a high accuracy with very low false positive rate. Wang et al. [13] applied HMM to host audit data to detect anomaly data quickly at a lower mismatch rate. However the training phase requires multiple passes. Soule et al. [14] used kalman filters to recognize traffic patterns using a network-wide view. Portnoy et al. [15] applied clustering anomaly detection methods on both DARPA and KDD cup 1999 datasets and obtained good results in terms of classifier accuracy for known or labelled data, whereas, clustering unlabelled data resulted in lower detection. Zhang and Zulkernine [16] applied the random forest algorithm to the KDD cup and DARPA datasets. They observed comparatively better results than other unsupervised techniques, however the classifiers performance got degraded over minority attacks.

Information-theoretic approaches for intrusion detection have recently been explored by researchers. Lakhina et al. [17] demonstrated the detection of a wide range of anomalous network flows by examining their entropy measures. Feinstein et al. [18] identified DDoS attacks by examining the entropy of packet header fields. Wehner [19] created a fast method for guessing the family of an observed worm without disassembly, by comparing the compressibility amongst binaries to determine their similarity. For network traffic [19] used average compression ratios of good or expected traffic and flagged anomalies on deviations from it. There has been a significant amount of work on network traffic characterization but little has been done in compression-based clustering and classification for traffic. This can be because compression is an expensive

Security Comm. Networks 2012; **00**:1–15 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

operation in terms of CPU time. Kulkarni and Bush [20] attempted a similar approach to monitor network traffic without using compression. Work carried out by Evans and Barnett [21] used compression to compare the complexity of legal FTP traffic with illegal traffic, but at a header level only, whereas we use entire TCP sessions. Kulkarni, Evans and Barnett [22] performed denial of service measures using compression ratios based on Kolmogorov complexity by estimating the entropy of 1's contained in the packet, which has its limitations. Such techniques generally prove to be quite efficient in determining worm and worm-like activity. Eiland and Liebrock [23] detected network scans from normal traffic using Kolmogorov complexity by estimating the inverse compression ratio of scan and normal traffic. This technique fails when applied to partition or classify various types of attack traffic, as we have in our dataset.

4. MACHINE LEARNING FOR INTRUSION DETECTION

Intrusion detection technologies are reactive in nature. They analyse user, system and network activity and from them, attempt to recognize malicious patterns, whereupon alarms can be generated or appropriate actions taken. We treat intrusion detection as a classification problem, where we can classify network and system events as a set of known classes, or as an unknown class that needs to be added to the database. Note that in our work we do not consider benign network traffic. This is one of the interesting features of a honeypot. Since it is a network server with no actual functionality, and usually low security, anybody using it (except when its logs are being retrieved) must be a hacker who is using a resource that they are not officially allowed to access, and so should be identified by the intrusion detection system. For our project we use honeypots in our existing honeynet setup [24]

Where possible, such classification systems are supervised, so that a training set of pre-labelled data points are used, and the inputs strings and similarity metrics are used to learn a model that separates out the different classes. Given this labelled data, there are a huge variety of classification methods available in the literature. However, many of them are ruled out by the fact that we want the IDS to be able to extend itself during use. Our learning process is in two stages: during initial setup, a set of labelled data can be used to identify known categories of attack, but later, when the IDS is in use, we want it to identify inputs that do not match any currently-known class (novelty detection) and automatically learn about these new classes of attack, which will be unlabelled.

Determining similarity between packet/stream profiles

To detect an intrusion, a misuse-based intrusion detection system or IDS relies on signatures it has in its signature database to compare with incoming network streams and packets. These signatures are composed of distinct syntactic features observed from past attacks. These features are in the form of patterns or sub-strings. Scrambling this information within packets can effectively evade misuse-based IDS, however is it possible to work around this shortcoming.

In the case of scrambled information or transposition, the order of symbols forming the pattern changes, but the inherent symbols remain the same. We believe that these syntactically distinct but symbolically similar strings can be measured effectively for similarity due to their similar corresponding information or symbols. Thus scrambling the information has little or no effect on the entropy of the strings. However, if new symbols are added or replaced, the entropy may change drastically. Information theory provides several techniques to measure the similarity between two strings. One way to do this is by noting that each string contains information and the amount of information stored in a string is quantifiable and can be articulated by its Entropy or Kolmogorov Complexity [5]. From mathematics, we may use string metrics to determine the approximate similarity or dissimilarity between strings.

Our Hypothesis

Applying the same theme to network streams, we can now perform similarity measures between known attacks or malicious streams and features extracted from incoming network streams. The advantage over plain misuse-based IDS is that instead of matching small patterns within the streams or packets we shall now match information patterns within them. This process can be automated and intelligence can be added to the IDS by using machine learning techniques to detect known and unknown or novel malicious network streams

5. SIMILARITY METRICS

Edit distance is a class of distance functions, which map a pair of strings x and y to a real number $\mathbb{R}$. Smaller values indicates greater similarity and a larger values represents increasing dissimilarity between x and y. Levenshtein distance and Hamming distance are examples of such class of distance functions. [25]. Other similarity metrics include Normalized Compression Distance (NCD)[5] and an approximated edit distance metric like Spamsum [26]

At the heart of the IDS is a measure of the difference between two strings, since the attacks are transferred across the network using normal data streams. There are a wide variety of string metrics, and in previous work [27] we have compared a variety of them, from standard techniques like Levenshtein distance and

Hamming distance, to information theoretic measures. In that work we found that Normalized Compression Distance (NCD) [5] quantifies similarities between various network profiles in our dataset well. A related measure is Spamsum [26].

Figure 1 shows an ROC analysis of the similarity metrics studied. It can be seen that Spamsum and NCD outperformed other similarity metrics, and so we focus only on these two in this paper. These results were computed using threshold-based nearest neighbour (T-NN) [27] as the classification algorithm.

In our previous work [27] we concluded that Threshold based nearest neighbor or T-NN served as the best clustering algorithm for our dataset. For a dataset having N data points we want to find the nearest or closest point in the metric space that is below the threshold value defined as a parameter at the beginning of the algorithm. T-NN was used as the best-fit clustering algorithm for our dataset.

In fact, Spamsum and NCD are highly correlated, with a Pearson product-moment correlation coefficient of $r = 0.96$ and coefficient of determination $R^2 = 0.93$. However, these scores do not tell the whole story, and we identified places where only one of the two metrics identified similarities, and we therefore decided to use them both by basing the similarity decision by taking the minimum score of the two as the criteria to determine similarity.

In the next section we describe how we use this similarity measure in order to classify input strings.

6. DATASETS

To evaluate the effectiveness of an IDS we require datasets or traffic or attack generation tools. In the absence of standard reference IDS datasets, it is difficult to evaluate and compare different methods of detecting intrusions, and to fine-tune a system for optimal results.

A good dataset should contain a greater coverage of the various types of network traffic including labelled attacks, while also being representative of the properties and nature of network traffic. Unfortunately, no such standard datasets exist. While various research groups have maintained their own datasets for evaluation, they are not very reliable and very few datasets are publicly available, One exception is the MIT DARPA 1998 and 1999 dataset, also known as the IDEVAL corpus. Though dated, the dataset comprises of categorized or labelled traces of benign as well as malicious network activity. J. McHugh [28] gave a critical analysis and determined that results for synthetic data are not practical to real data [28]. Hence the IDEVAL corpus is not analogous to the properties of real network traffic. Many datasets used by researchers focus on statistical features extracted from network traffic and mainly headers to identify anomalies instead of actual network dumps. Features such as source and destination IP and Ports, packet size, header size, bytes sent and received, flags, protocol type fields and connection rate etc. These features may serve to identify some types of anomalous traffic, but cannot serve as a general rule for sophisticated attacks. We are more interested in the packet/stream payloads, as they contain representative digital fingerprints of exploits.

Werner created a manually analysed and labelled dataset for Nebula [29] and was kind enough to share this dataset for our IDS evaluation. The dataset comprises a set of 6631 unique attacks that he collected from two honeypot instances during the last quarter of 2007. This dataset contains 28 main labelled attack categories and a total of 55 labels including all subcategories. The labels for two subcategories a and b of a main category c are expressed by c.a and c.b. The malicious streams range from buffer-overflows, to exploits, to shellcode to worm and worm variants. The benign but suspicious streams include traffic like HTTP GET,POST and OPTIONS requests, FTP and TFTP download requests, Oracle and Mysql database connection requests.

We also created our own dataset using a Dionaea honeypot. Dionaea is a low interaction honeypot that has proven to be quite useful for collecting binary malware samples, binary network streams, shellcode and extensive attack detail. We manually collected, labelled and classified 6600+ samples using Dionaea bi-streams, and labelled the dataset using Tillmann's methodology.

7. EXEMPLAR LEARNING FOR INTRUSION DETECTION

Inductive learning is the process of learning by examples. A machine learner is trained to induce a general rule, compare with other rules and perform classification of the observation from a set of observed instances. Exemplar learning involves choosing exemplars or instances from a set of objects based on their resemblance to a certain category or group by the learner. The identification of an object as an exemplar or instance and the rules to determine matching objects group membership are the two key measurements that the exemplar theory depends on.

In machine learning terms, concept learning comes under the category of supervised learning. A class labelled dataset is provided to the algorithm to train on. The algorithm or learner uses exemplars from this labelled training data set to create generalizations to classify other objects. Such algorithms are often referred to as instance learning algorithms or lazy learning algorithms.

We have chosen to use exemplar learning, since we have a labelled dataset of previously known attacks available, and since at a later stage we expect that novel attacks, which will be detected automatically by our system, will be analysed by humans (or eventually other software systems) in order to identify suitable responses to them. Exemplar learning is also known as instance learning or lazy learning.

Security Comm. Networks 2012; **00**:1–15 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

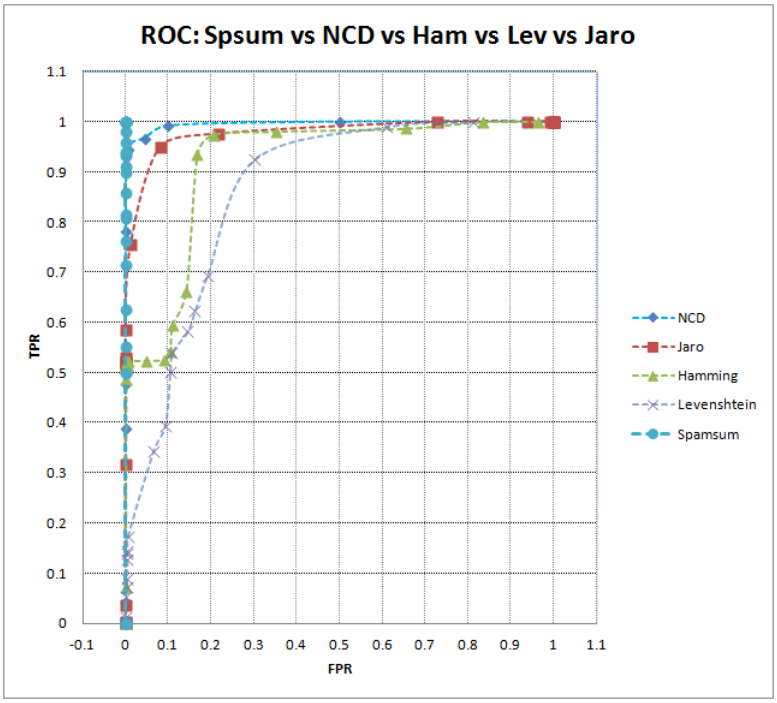

Figure 1. An ROC curve comparing Hamming, Levenshtein, Jaro, NCD and Spamsum similarity measures clustered using T-NN [27] . It can be seen that Hamming and Levenshtein distance measures perform poorly, with many false positives at the start and getting worse towards the end. Jaro distance metric is much better, with true positive results until at the end a few false positives results start to appear. NCD and Spamsum outperform the other similarity metrics.

Given the data and the similarity metric, there are two problems for the exemplar learner:

1. To identify suitable exemplars or instances from the training set (TR).
2. To learn an appropriate similarity threshold within which the data points belong to the same class as the exemplar class.

Once this is done, the acquired model consists of a set of (exemplar data point, threshold) tuples. A schematic of this is shown in figure 2 and figure 3.

Choosing the exemplars appropriately is important: a poorly selected exemplar can result in a large number of false positives, or a large number of exemplars being required. This is particularly important when the training dataset could be unbalanced or skewed, so that there are a very small number of examples of some classes.

8. METHODOLOGY

8.1. Model Creation using exemplar-based learning

In machine learning, the performance of a learning classifier depends directly on the effectiveness of its mathematical or statistical model. In other words, how well the given dataset or algorithm output fits to the prescribed model. Model selection, demands selecting the best or most appropriate model from candidate models, to solve a given machine learning problem.

We have an imbalanced labelled dataset of malicious network streams that we want to classify. For classification or supervised learning problems the classifier is trained on classified examples and is expected to classify unseen samples. It is assumed that every instance (input sample) belongs to one, and only one, class. The model M would include multiple instances of exemplar ϵ and threshold τ tuples. Our model will be able to determine the best exemplar(s) per category, and suggest the optimal threshold for that exemplar, for classification. This exemplar-based learning classifier model would work as follows:

Which instance to select from the training set (TR)?

Choosing an exemplar for each category is a difficult task. A poorly selected exemplar may result in a single, noisy exemplar, which in turn may result in a large number of false positives. This may be due to the imbalanced/skewed dataset or the nature of the samples in it. It is required to prune the noisy exemplars (Edition 10) and update the model with the appropriate exemplars and threshold values per class (Condensation 10).

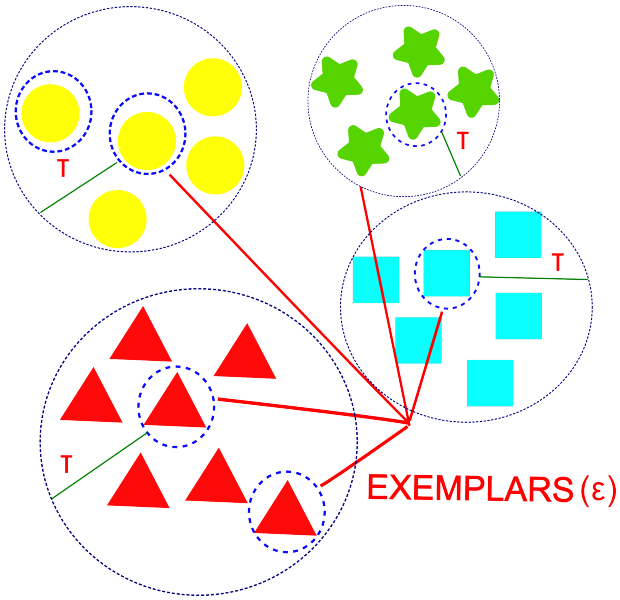

Figure 2. For categories a, b, c, d, each with a set of samples such as $a_1 \ldots a_n$, a model can be defined as a list of tuples, where each tuple comprises of an exemplar a_ϵ selected from the samples and a corresponding distance threshold value a_τ. If there are significant varieties within one class, there may be several exemplars from the same class.

```
2-4c6784225bb509cd6d33e2ed852d22d1.raw,0.771429
3-87937439c24a34f5eaef3422a3424d54.raw,0.2
4.2-0c46eace7006b062883311d87a6ddcac.raw,0.26
5.2-1b1f137c7c020fb02936967ac3f90af4.raw,0.540541
5.3-718a9b08a38c979146cece8ff674ecd2.raw,0.2
6.2-3ca4908cf4fc08e5fc0e7a07e92f6df8.raw,0.838912
6.1.2-eff838940f72ba3f05c25b6e82196862.raw,0.18
7-27bfdaaa288c4f28018cb2f4ed871a27.raw,0.769397
8-2868ff07e1bc760488962955a850a7be.raw,0.349057
8-6bf6cb9da054e0d5b22b0f4c5f54f59a.raw,0.2
11-3a28452fc0b7572a345514aa66a5e628.raw,0.601504
12-b9fe2c1354670d81c61a3d222cfa587f.raw,0.705479
13.3-b3350817df8b4325a32beb9fbe3c6dee.raw,0.542355
14.6-5c8c0cf8755b027234eb50717723113e.raw,0.835874
14.1.2-ae4652b546a10c092c22036bbc0db641.raw,0.121834
15-ca25c55fd256d92b48b81cb36f1a6db7.raw,0.2
16-bd63683b8461833784a25c0c5103318f.raw,0.734375
18-d7535adefffd3895cd099a6a63a3ba76.raw,0.569412
18-9d484817327e5204821c4e6f1443325f.raw,0.2
19-abbb2c6849cb68b0da4601eb181ad5b0.raw,0.2
20-9fe5317a305e0281b9565a65af03da0e.raw,0.60355
21.2-850614dd2159b299ebb070fe238f0faa.raw,0.2
22.3.2-7ea4921beff80e610ebf67cff4add312.raw,0.243243
23-22d6e30971b1ed67ee14bcf0711b3027.raw,0.2
24.2-a641b4d1a37d9a53b81c53f44de5c011.raw,0.2
24.1-735cfdb0fdf65297e870874fd639144c.raw,0.2
25-e3173170378f294fecea500ee36f978c.raw,0.2
26-52957384c31989b60220038ac726544d.raw,0.2
27.1-ab179e4c4ac589ed3c62709ac7a303db.raw,0.636364
28-7b2695c036cb9224db54f73e5326f135.raw,0.2
29-7e1342bba87976788 0aaf65d9e6859c9.raw,0.370370358229
29.1-257cc578a26560b7ae9ec7888f54a1ad.raw,0.2
29-7d153ea9f08017d5b539621d88792597.raw,0.37037
2-0914a08278451121 41cf3563170b8c0.raw,0.2
27.1-c59ebdb083ea1dafecca6210944d6038.raw,0.2
32-be655f217dc7778dfbff549f9dc7dbda.raw,0.2
14.5-392b618c7ab897ccfe8cbdad9a1531db.raw,0.304192692041
30-794e7a0aac0e8d2ee0830e898a2046a4.raw,0.2
30-e71f0dc30175fd909045ecd88d17f737,0.356643348932
30-3dad61ba915388c827ea5185d4211196,0.213793098927
30-b80e7f6d94e32c117130c3456d153467,0.22260273993
30-51fdcd4d1abfab41f2976908c6008c35,0.214532867074
30-9e6689fcd308b57815ebb2b603299406,0.2
30-2f5f00e3cef4e32b63c8415f104b97ac,0.21875
30-4dc78ae1278cc652a650b07064f5929e,0.383561640978
30.1-1baf4cd2b3ace50e07a6684eb71b44b4,0.566433548927
30.2-cdc1ff8fc8a345943665ab8a07e82fc2,0.675990700722
```

Figure 3. The model written out to a csv file. Each row can be read as: The labelled exemplar ϵ , and the threshold τ. There can be several exemplars from the same class, indicating rich variations within the class samples or candidate sub-classes. Otherwise if all samples in the class are quite similar, then they can be represented by a single exemplar

Algorithm 1 shows our exemplar selection algorithm. As a first step an $n \times n$ similarity matrix of samples in the training set (TR) is calculated. From this matrix, an exemplar ϵ for a category is chosen by comparing all samples in a particular class to find the one with the lowest overall distance (i.e. greatest similarity). This is done by taking the minimum of the sum of all corresponding similarity scores of the data points for each sample. This sample then becomes our exemplar label.

The threshold value is then calculated by taking the max of the corresponding similarity scores of this sample with others in the same category. The optimal threshold

Security Comm. Networks 2012; **00**:1–15
DOI: 10.1002/sec
Prepared using secauth.cls

value is determined by testing for false-positives on each data point below the initial threshold, until a value with no false-positives is established. Note that this assumes that each data point belongs to precisely one class, which is true for the datasets that we have analysed. The algorithm iterates over the remaining points for that category, until all the points are accounted for by the model. Finally a sanity check is performed on the model to ensure that that the threshold τ satisfies the condition $0.1 < \tau < 0.8$. Here a very tight threshold i.e. less than or equal to 0.1 will encompass very few or data points that are exactly the same as the exemplar. This inhibits the ability of the system to be able to detect variations in the form of similar samples and not just the same samples. A very high threshold i.e. greater than 0.8 would cause false-positives due to negative class classification.

8.2. Lifelong Learning: Detecting and Adding Novel Input Classes

Our previous work on exemplar-based learning for intrusion detection, incorporated creating a model (comprising of exemplars and thresholds) from a known labelled training dataset and then trying to classify a large test/target dataset with this model. This work was very much a two-class classification problem, where all the samples in the set were classified as one of the classes identified in the model.

There were, however, a few unknown samples which did not get classified/identified by any of the classes in the model and were scattered across between the boundaries of known classes in multidimensional space. Identification and classification of these unknown or novel data samples became a motivation to study and solve the problem of novelty detection for our domain. In the literature, the terms One-class classification, novelty detection, outlier detection and concept learning refer to different applications of the same problem.

Using the algorithm given in the previous section, we are able to identify a set of exemplars and their corresponding distance thresholds. New input data can now be classified into one of the existing classes by simply comparing it to each exemplar in order and checking if it is within the appropriate threshold distance. However, we also want to deal with novel exemplars that are introduced and do not match any of the current classes in the model. All such samples are added to a set of outliers. A pictorial manifestation of this is shown in Figure 4.

Algorithm 2 describes a methodology to:

1. Classify the outliers into new classes
2. Select exemplars and thresholds for these new classes
3. Add these new exemplars and thresholds to the model to enhance classification.

Any data point not recognised as belonging to any class is added to a list of outliers. The algorithm first calculates the distance between the exemplars ϵ_{model} from the model and all the points in the list of outliers. This results in a $n \times m$ distance matrix of similarity scores. An initial outlier exemplar $\epsilon_{outlier}$ is chosen as the data point p in the list of outliers with the lowest score to all the other outlier data points and the current exemplars in the model ϵ_{model}. The initial threshold $\tau_{outlier}$ is set to the maximum score observed from the corresponding scores for p. Next, the exemplar $\epsilon_{outlier}$ and threshold $\tau_{outlier}$ are tested for false positives.

For the novelty detection algorithm, the false positive observations would be the exemplar ϵ_{model} data points taken from the model. If any exemplar ϵ_{model} from the input model has a similarity below the initial threshold $\tau_{outlier}$ of p, it is considered to be a false positive. While the true positive values would be all data points in the list of outliers, under the test threshold. In the case where false positives exist, the threshold is adjusted to the maximum true positive value that is less than the minimum cost false positive observation. A sanity check is performed to ensure that the threshold $\tau_{outlier}$ value satisfies the condition $0.1 < \tau < 0.8$.

This new exemplar p or $\epsilon_{outlier}$ and its adjusted threshold $\tau_{outlier'}$ is then appended to a list of exemplars in the model and this exemplar and the true positive data points that it was able to classify are removed from the list of outliers. The process is repeated until there are no more items remaining in the list of outliers.

9. VALIDATION

9.1. Validation of the Exemplar Selection Algorithm

In order to validate Algorithm 1 we conducted an experiment in which we randomly selected samples from the training set (TR) to create a model and compare the results with the model provided by this algorithm. A key question here would be that: how do we determine a threshold value for the exemplar? Do we need to run a sensitivity test for that? Such a test will involve sliding the threshold in small increments to determine the threshold value that will encompass maximum classification. This will be time consuming and far less efficient, as randomly selecting exemplars can never guarantee good classification by the classifier, as the classes may contain sub-classes, which might not get classified at all by the randomly chosen exemplar.

As an initial, 'straw-man' test we created a training set comprising of 300 samples, representing 30 known classes and a test set comprising of 1200 samples from the same 30 classes. We used a set of exemplars chosen at random from the training set, up to five for each class, and assigned each of them a fixed threshold value of 0.5. We did this for 10 different random choices of exemplar sets, and saw an overall accuracy of around 70% on average. We observed that a large part of the problem with this method is that 0.5 is not a fair threshold to choose, and so we repeated

Algorithm 1 Calculate $model = train$(labelled distance matrix, list of classes)

Require: list_of_class , similarity_matrix
Ensure: list_of_outliers

for each class C in list_of_class **do**
    find the datapoint $p \in C$ that has minimum distance to all points in C
    set initial threshold τ to max (distance from p to all elements of C)
    if any exemplar is within threshold τ distance of p **then**
        set threshold $\tau =$ first point below min(distance from p to exemplars)
        check that $0.1 < \tau < 0.8$, otherwise set to bottom/top of range
    end if
    $add(p, \tau)$ to model
    remove all points within distance τ of p
    if any points in C remain **then**
        leave C in list_of_classes
    else
        remove C from list_of_classes
    end if
end for

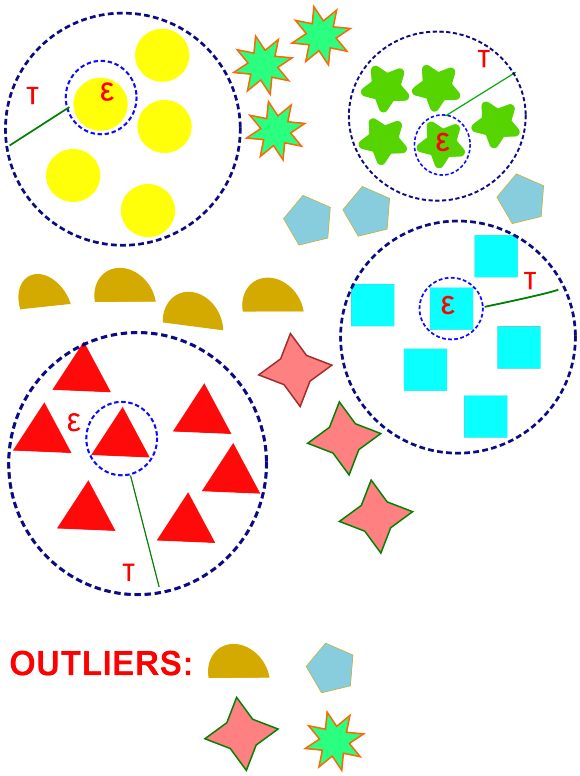

Figure 4. Novelty detection

the experiment, but this time determined the threshold by starting from 0.3 and increasing it in small increments of 0.1 to determine the maximum classification. This is time-consuming, but makes a fair test. We performed 10 iterations of this experiment and found an average accuracy of 88%. In contrast, using our algorithm to select exemplars and thresholds led to 95% accuracy, and used only 49 exemplars instead of the 120 used by the random method.

We found that the experiments based on randomly selecting exemplars led to fluctuating results. The highest accuracy results obtained via randomly picking exemplars led to an overall accuracy of 92%, compared to 95% using our algorithm. This is a significant increase in the

Security Comm. Networks 2012; **00**:1–15 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

Algorithm 2 Calculate $update\_model = add\_Exemplar(list\_of\_outliers, similarity\_score\_matrix, model)$

Require: list_of_class , similarity_matrix, list_of_outliers
Ensure:

while outliers in list_of_outliers **do**
    find the data point p that has minimum distance to all points in similarity_score_matrix
    set initial threshold τ to max (distance from p to all outliers)
    if any exemplar is within threshold τ distance of p **then**
        set threshold $\tau =$ first point below min(distance from p to exemplars)
        check that $0.1 < \tau < 0.8$, otherwise set to bottom/top of range
    end if
    remove all outliers within distance τ of p
    $add(p, \tau)$ to model
end while

Method	Exemplars	% Accuracy
TNN-Fixed	120	70
TNN-Adaptive	120	88
TNN using Algorithm 1	49	95

Table I. Comparison of methods random exemplars vs selected exemplars

Method	% Accuracy	Outliers
Algorithm 1	70	1300
Algorithm 2	97	0

Table II. Learning with Algorithm 2

classifier's performance and provides an automated and robust method of exemplar selection for model creation. The relatively high average percentage (88%) in terms of accuracy for the random exemplar-selection methodology is due to the fact that we were picking five exemplars per class and optimising the choice of threshold for each, which is considerably more computationally expensive than using our algorithm. It also points to the fact that similarity within a category of network streams exists because of the protocol design, which reduces complexity and enhances the similarity for classification. The results are summarized in table I

Compared to T-NN with adaptive thresholding, our algorithm requires fewer exemplars with more definitive thresholds. On the 300 sample training set, our algorithm proposed only 49 exemplars compared to 150 exemplars. Hence with one-third the size of exemplars, and with a more accurate or definitive threshold in the model we get at least a 10% increase in terms of accuracy.

9.2. Validation of the Novelty Detection Algorithm

In order to validate Algorithm 2, we used Algorithm 1 to create a model from a small training set (TR) comprising of 50 samples representing five classes, and then deleted the exemplars of two classes from it, as shown in figure 5. We then used the first step of Algorithm 1 to test on a dataset that included both known and unknown classes. This resulted in 80% correct classification in terms of accuracy. The unknown samples were appended to the outlier list as shown in figure 6.

Following this, the outlier list was used with Algorithm 2 to create new exemplars for these unknown classes, as illustrated in 7. A full description of this experiment and its results follows:

1. The model was trained with five classes, leading to 9 exemplars (2 for class 1, 3 for class 2, 1 for class 3, 2 for class 4, 1 for class 5).
2. Both exemplars for classes 1 and 4 were removed from the model.
3. After classification, the system found that all elements of class 1 and 4 were outliers, and none of classes 1, 2, 3, 5 were outliers.
4. The novelty detection algorithm was applied and three exemplars of class 1 and two of class 4 were added to the model.
5. Thus the classes that were manually removed earlier were automatically detected and added to the model, and no others.

Finally, we tested the combination of the two algorithms on a larger set of 6600 labelled samples with 50 classes. We started with a small training set (TR), of around 300 samples representing 30 classes. Algorithm 1 was used to learn a model of this dataset, and created 49 exemplars representing 30 classes. The model was then tested on an independent test set of around 6000 samples with all 50 classes represented. This resulted in an accuracy of approx. 70%. The data points identified as outliers were then learnt about using Algorithm 2 and their exemplars appended to the model. A second test set (of the last 300

data points) was applied, giving classification results of 97% on this second test set, with all 50 classes represented as summarized in table II

These results are very pleasing, and show that this simple method of lifelong learning based on our exemplar learning has distinct promise for the IDS application.

The question that motivated our first validation test was that: can the novelty detection algorithm detect known classes, which were removed from the model for testing?

The first test conducted for validating the algorithm involved creating a model from a small training set, removing exemplars from random classes from this model and then performing classification. This resulted in a partial or no classification for classes that were removed from the model and a large list of outliers. These outliers were then run through the novelty detection algorithm to yield results. The results were quite satisfactory as the algorithm correctly identified the missing classes and proposed respective exemplars for their classification.

Our second validation test was motivated by the question: How much does learning new and novel classes increases the overall accuracy of the classifier?

For this test, we created a model from a carefully created training set of approx 300 samples extracted from a large dataset of 6600 samples. The model was used to classify the test set, comprising of 6600 samples. After classification, the classifier gave an overall accuracy of 90%. A list outliers was identified and the novelty detection algorithm was then applied to them to identify novel classes and exemplars. These new exemplars were then appended to the model for re-classification. The new classification resulted in a an overall accuracy of 98%.

10. COMPARATIVE RESULTS

Using a small labelled test set comprising of 1200 samples and a training set of 300 samples, we performed a comparative analysis of the exemplar learning technique with both of our prior techniques i.e. (a) Classification using a fixed threshold and (b) Classification at the optimal threshold obtained by adaptive thresholding. Both of these techniques provide a good approximation, but do not give a precise measurement in terms of: (a) the optimum number of exemplars that are required per class and (b) the optimum similarity threshold for those exemplars in that class. Next, these results were compared with well-known machine learning algorithms like nearest neighbour and k-nearest neighbour, which do not require threshold constraints, and finally concluded with a comparison using our proposed exemplar-learning and novelty detection algorithm.

For comparison we performed a two-stage experiment to answer the following questions:

How to select the exemplars or instances required per class for classification?

We designed an experiment such that step-wise increments of 10 exemplars ($\epsilon = \epsilon + 10$), are picked randomly from the training set starting from $\epsilon = 10$ till $\epsilon = 90$ exemplars. Before this we sampled on a smaller scale by randomly selecting 1 upto 9 exemplars from the training set (TR) in increments of 1. In each iteration, a set of exemplars were produced that were then subjected to the second stage. The second stage answers.

How to select the optimal threshold required per instance per class?

To answer this question we used an adaptive threshold method that requires small step-wise threshold increments of $\tau = \tau + 0.25$ starting from 0.0 till 1.0 creating a confusion matrix for each class individually and determining the optimum threshold τ for the whole test set.

Comparisons

For comparison of the classification techniques, we divided the methodologies into three main categories:

1. Methodologies that involve threshold for classification or threshold-based e.g. fixed and adaptive thresholds.
2. Methodologies that involve just instances for classification or instance-based learning e.g. nearest neighbour and k-nearest neighbour
3. A hybrid methodology that uses both instance-based learning guided by the threshold for classification e.g. threshold-based exemplar-learning techniques

Accuracy results for (1) fixed threshold and (2) optimized thresholds (3) nearest neighbour and (4) k-nearest neighbour at k = 3 and 5 were plotted using an XY Scatter plot for each exemplar set. Additionally, results from our proposed exemplar-learning and novelty detection algorithm were added for comparison.

The results from this experiment are shown in Figure 8 , it can be seen that as the number of exemplars increases, it increases the accuracy since the chances of getting a better classification increases with it, due to better coverage. This however does not guarantee the best or optimal exemplar selection for each class, as the exemplars are picked randomly from the training set. The performance of the (a) optimized threshold technique is better than (b) the fixed threshold, as enumerating through every possible result to yield the best one, on average, gives approximately 8% better results. Results from nearest neighbour and the k-nearest neighbour algorithm have been added to the results for comparison. Here it can be seen that the nearest neighbour algorithm performs very well as the number of exemplars increases and is found to be better than (a) and (b). For the k-nearest neighbour algorithm, we are not

Security Comm. Networks 2012; **00**:1–15 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

```
model: ['2-4c6784225bb509cd6d33e2ed852d22d1.raw', 0.77142859]
model: ['3-87937439c24a34f5eaef3422a3424d54.raw', 0.039999999]
model: ['4.2-0c46eace7006b062883311d87a6ddcac.raw', 0.25999999]
model: ['5.2-1b1f137c7c020fb02936967ac3f90af4.raw', 0.54054052]
model: ['5.3-718a9b08a38c979146cece8ff674ecd2.raw', 0.2]
model: ['6.2-3ca4908cf4fc08e5fc0e7a07e92f6df8.raw', 0.83891213]
model: ['6.1.2-eff838940f72ba3f05c25b6e82196862.raw', 0.18000001]
model: ['7-27bfdaaa288c4f28018cb2f4ed871a27.raw', 0.76939654]
model: ['8-2868ff07e1bc760488962955a850a7be.raw', 0.3490566]
model: ['8-6bf6cb9da054e0d5b22b0f4c5f54f59a.raw', 0.2]
model: ['11-3a28452fc0b7572a345514aa66a5e628.raw', 0.60150373]
model: ['12-b9fe2c1354670d81c61a3d222cfa587f.raw', 0.70547944]
model: ['13.3-b3350817df8b4325a32beb9fbe3c6dee.raw', 0.54235536]
model: ['14.6-5c8c0cf8755b027234eb50717723113e.raw', 0.83587444]
model: ['14.1.2-ae4652b546a10c092c22036bbc0db641.raw', 0.12183353]
model: ['15-ca25c55fd256d92b48b81cb36f1a6db7.raw', 0.0]
model: ['16-bd63683b8461833784a25c0c5103318f.raw', 0.734375]
model: ['18-d7535adefffd3895cd099a6a63a3ba76.raw', 0.56941175]
model: ['18-9d484817327e5204821c4e6f1443325f.raw', 0.2]
model: ['19-abbb2c6849cb68b0da4601eb181ad5b0.raw', 0.0]
model: ['20-9fe5317a305e0281b9565a65af03da0e.raw', 0.60355031]
model: ['21.2-850614dd2159b299ebb070fe238f0faa.raw', 0.080402009]
model: ['22.3.2-7ea4921beff80e610ebf67cff4add312.raw', 0.24324325]
model: ['23-22d6e30971b1ed67ee14bcf0711b3027.raw', 0.0]
model: ['24.2-a641b4d1a37d9a53b81c53f44de5c011.raw', 0.2]
model: ['24.1-735cfdb0fdf65297e870874fd639144c.raw', 0.2]
model: ['25-e3173170378f294fecea500ee36f978c.raw', 0.0]
model: ['26-52957384c31989b60220038ac726544d.raw', 0.0]
model: ['27.1-ab179e4c4ac589ed3c62709ac7a303db.raw', 0.63636363]
model: ['28-7b2695c036cb9224db54f73e5326f135.raw', 0.0]
model: ['29-7e1342bba879767880aaf65d9e6859c9.raw', 0.37037035822868347]
model: ['29.1-257cc578a26560b7ae9ec7888f54a1ad.raw', 0.2]
model: ['29-7d153ea9f08017d5b539621d88792597.raw', 0.37037036]
```

Figure 5. Original model, with highlighted samples that will be removed from the model for testing the novelty detection algorithm

```
'8-ddc13f738bfde488e90a7ef5a71a6db1.raw'
'8-0b24deaba05f352d96b58df0ffac475a.raw'
'2-0914a08287845112141cf3563170b8c0.raw'
'8-0b7d99c126630c338d6dc51fc387d0b8.raw'
'8-2a83e7f5b96d7e36e1512fef88bc0de6.raw'
'8-4ec0c06a0e1c517206431c8b8e173fd5.raw'
'2-1ab33f0980ffdaf71a80fa1e9b34a8da.raw'
'2-4c6784225bb509cd6d33e2ed852d22d1.raw'
'8-5d1055fe2e8bc8697474fc261336b476.raw'
'8-2868ff07e1bc760488962955a850a7be.raw'
'21.2-bc2db559c6ba343d8b904d101a30f564.raw'
'2-1d58be103d9de9739318d1428d6aaaad.raw'
'8-6ebb313ee65a5915aadd971df455ce02.raw'
'14.5-7f7e973fe844fa162f3025843cf4782e.raw'
'8-6bf6cb9da054e0d5b22b0f4c5f54f59a.raw'
'2-9fc3f20b92dc5b2a2d39e4f3c39465eb.raw'
'8-6c5b9065372dddec27eb4fb49fe160d8.raw'
'14.5-392b618c7ab897ccfe8cbdad9a1531db.raw'
'27.1-c59ebdb083ea1dafecca6210944d6038.raw'
'2-8b95ea1626adbeb7a0320dbbcc1a5ca9.raw'
'2-3198a859272ea31824e4ed89a323e25d.raw'
'8-2f949a63fccdb4458a081aa33172b0a0.raw'
'2-3175eb2416c74a21578880850279a9b9.raw'
'14.5-c0e600b27a7de6714e7d75c66967151f.raw'
```

Figure 6. List of outliers found by the novelty detection algorithm

```
8-2f949a63fccdb4458a081aa33172b0a0.raw,0.794117629528
2-1ab33f0980ffdaf71a80fa1e9b34a8da.raw,0.78571414948
21.2-bc2db559c6ba343d8b904d101a30f564.raw,-
0.0145728647709
27.1-c59ebdb083ea1dafecca6210944d6038.raw,0.559090936184
8-6bf6cb9da054e0d5b22b0f4c5f54f59a.raw,0.951672852039
14.5-392b618c7ab897ccfe8cbdad9a1531db.raw,0.304192692041
```

Figure 7. New Exemplar ϵ_{novel} and Threshold τ_{novel} pairs proposed by the algorithm. The highlighted samples/exemplars are the samples from the same class that we initially removed from the original model

certain about the value of k, so we use 2 iterations using k = 3 and 5 respectively. It can be observed here and also in table **??** , that as the size of k increases, the accuracy decreases as it is believed to add more false positives

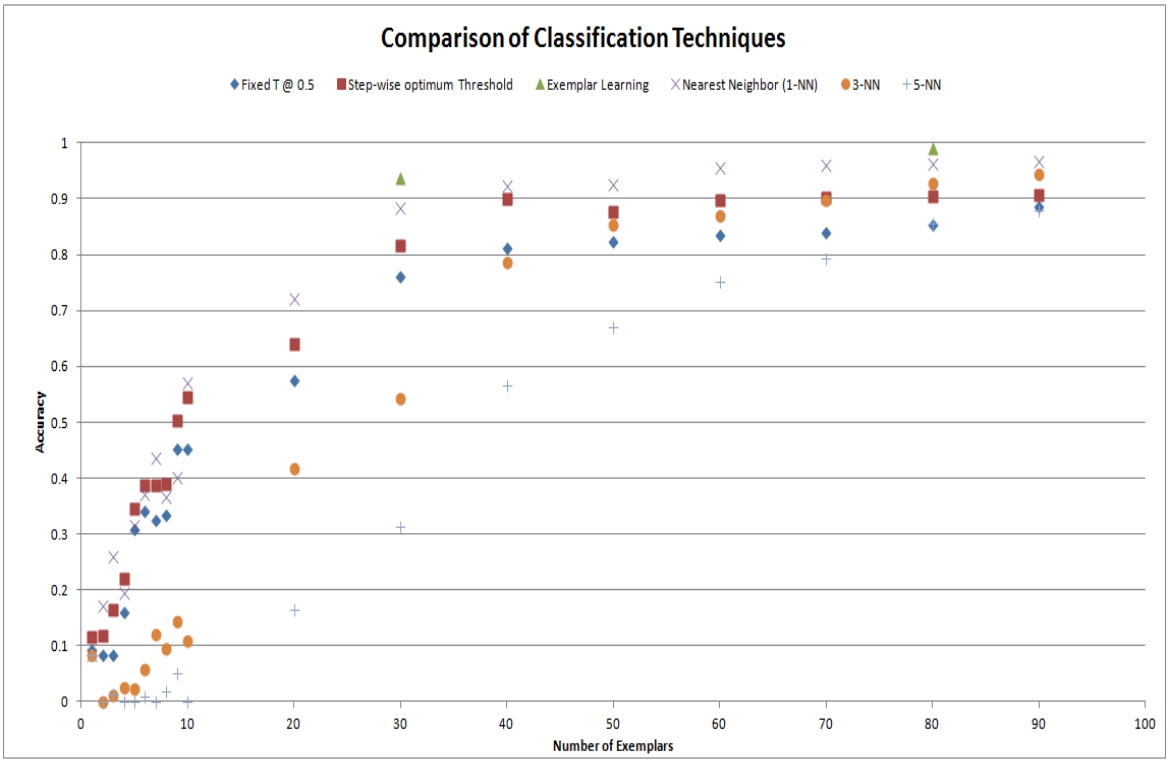

Figure 8. A comparison of classification methodologies showing accuracy results for: (1) Fixed threshold (2) Optimum threshold using step-wise thresholding (3) Nearest Neighbour (4) k-Nearest Neighbour and (5) for exemplar learning. Results are calculate at varying number of exemplars for techniques 1-4. It can be seen here that exemplar learning results using T-NN outweigh results of other techniques.

to the results, which eventually have a greater vote and thus impact gravely on the accuracy of the classifier. Finally, results of exemplar learning (Algorithm 1) have been added to the graph. It can be seen in Figure 8 that using Algorithm 1 results in 30 exemplars with a percentile accuracy of approximately 90%, with some outliers. Next we apply Algorithm 2 on the outliers which results in approximately 80 exemplars with an overall 98% accuracy result.

Comparison with instance selection Algorithm, ENN

Instance selection algorithms can be classified into 3 main groups namely, edition algorithms, condensation algorithms and hybrid algorithms [30]

Edition algorithms edit out noise instances as well as close border class. e.g ENN [30],

Condensation algorithms aim to condense the selection set from the training set, while retaining points close to the class boundary. e.g. CNN, RNN, SNN. [30]

Hybrid algorithms aim to increase classification accuracy by using a combination of approaches from edition and condensation algorithms. e.g. instance-based learning family of algorithms IB3 [30]. Our proposed instance selection algorithm is a hybrid algorithm of incremental type as it performs both edition and condensation at a per class level to select the the most appropriate instances for creating and extending the model.

Difference between ENN and our proposed algorithm

ENN [30] is an edition algorithm which uses a decremental search. ENN starts by considering the entire selection set S as the training set i.e. $S = TR$ and then removes all such points/selections which have a negative class neighbour majority discovered by $k - nn$.

Our proposed instance selection algorithm is a hybrid instance selection algorithms of incremental search type, which starts with an empty selection set i.e. $S = []$. After that it creates an $n \times n$ similarity matrix of all points in the training set (TR), then for each class it finds the point with the shortest distance to all the other points in that same class/category. This point is chosen to be the initial instance or the exemplar for that class, and is added to the list of selections. Next an optimal threshold is calculated in a way to avoid any possible false positives i.e. points/samples from other classes This threshold is also added to the selection list in correspondence to its exemplar or instance. The process is iterated for the remainder points/prototypes in the class, till we have

Security Comm. Networks 2012; **00**:1–15 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

Exemplars	Accuracy @ Fixed Threshold	Accuracy using adaptive threshold	Accuracy using Nearest Neighbour	Accuracy using k-NN, k=3	Accuracy at k=5
10	0.452157337	0.544964664	0.571024735	0.109010601	0
20	0.576236749	0.640812721	0.721024735	0.417137809	0.1642226150
30	0.759805654	0.817226148	0.884628975	0.5422261480	0.3131625440
40	0.811627081	0.900676605	0.923233216	0.78524735	0.565282686
50	0.822666334	0.877459058	0.925088339	0.853533569	0.66934629
60	0.835433643	0.898047689	0.955778444	0.869257951	0.752385159
70	0.83998132	0.903421871	0.959452297	0.898144876	0.792667845
80	0.854127104	0.904411873	0.963250883	0.926855124	0.856713781
90	0.8854324	0.907568	0.967844523	0.944169611	0.878091873

Table III. Comparison of algorithms using incremental number of exemplars

enough instances/exemplars and thresholds to classify the training set with maximum accuracy and minimum exemplars. Compared to ENN this method can guarantee a comparatively small number of exemplars/instances that are required for classification, as shown in table **??**.

Differences with ENN can be highlighted as follows:

1. Starts with $S = []$
2. Does not use kNN
3. Uses a similarity matrix
4. Computation and instance selection is done on a per class basis
5. Selects fewer exemplars

Experiment

We conducted an experiment to empirically compare the performance of ENN with our proposed model selection algorithm:

1. Created a training set (TR) comprising of 334 samples, selected from the dataset using the criteria of 10 or less available instances per class.
2. Implement ENN on this training set TR to select instances/exemplars.
3. Implement our proposed Model Selection Algorithm on the training set TR to select instances/exemplars.
4. Compared the results.

Results

1. Out of a total of 334 samples, ENN was able to reduce the TR by 4.2%, resulting in a total of 320 instances or a selection set comprising of approx 95% of the TR.
2. The exemplar selection algorithm selected 36 instances/exemplars out of the 334, thus reducing the TR to be approximately 89% with a selection set comprising of only about 11% of the TR. With this model, the classifier was able to get an overall accuracy of approx 90% on a reasonable test set.

Algorithm	Exemplars/Instances Selected	% reduction in TR
ENN	320	4.2
Algorithm 1	36	88.75

Table IV. Comparison with ENN

11. CONCLUSIONS AND FUTURE WORK

We have demonstrated the use of machine learning algorithms, especially exemplar-based learning to deterministically and automatically create a model comprising of exemplars and threshold pairs. We have also demonstrated the effectiveness of novelty detection to detect novel samples and classify them to increase the overall accuracy of the classifier.

In the future we would like to apply this methodology to classify system-wide events and behavior profiles of binary malware and investigate other problems where this methodology might work.

ACKNOWLEDGEMENTS

We would like to thank Tilmann Werner for contributing his stream dataset for the evaluation of our IDS. Fahim ul Huda Abbasi is supported by the Pakistani Higher Education Commission (HEC).

REFERENCES

1. Symantec. Symantec internet security threat report 2011 trends 2011. URL http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf.
2. Phoha V. *Internet security dictionary*, vol. 1. Springer-Verlag New York Inc, 2002.
3. Scarfone K, Mell P. Guide to intrusion detection and prevention systems (idps). *NIST Special Publication* 2007; **800**(2007):94.
4. Ptacek T. Insertion, evasion, and denial of service: Eluding network intrusion detection. *Technical Report*, DTIC Document 1998.
5. Cilibrasi R, Vitányi P. Clustering by compression. *Information Theory, IEEE Transactions on* 2005; **51**(4):1523–1545.
6. Lee W, Stolfo S, Mok K. A data mining framework for building intrusion detection models. *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, 1999; 120 –132, doi:10.1109/SECPRI.1999.766909.
7. Sung A, Mukkamala S. Identifying important features for intrusion detection using support vector machines and neural networks. *Applications and the Internet, 2003. Proceedings. 2003 Symposium on*, 2003; 209 – 216, doi:10.1109/SAINT.2003.1183050.
8. Kruegel C, Toth T. Using decision trees to improve signature-based intrusion detection. *Recent Advances in Intrusion Detection*, *Lecture Notes in Computer Science*, vol. 2820, Vigna G, Kruegel C, Jonsson E (eds.). Springer Berlin Heidelberg, 2003; 173–191, doi:10.1007/978-3-540-45248-5_10. URL http://dx.doi.org/10.1007/978-3-540-45248-5_10.
9. Chebrolu S, Abraham A, Thomas JP. Feature deduction and ensemble design of intrusion detection systems. *Computers & Security* 2005; **24**(4):295 – 307, doi:10.1016/j.cose.2004.09.008. URL http://www.sciencedirect.com/science/article/pii/S016740480400238X.
10. Liu Z, Florez G, Bridges S. A comparison of input representations in neural networks: a case study in intrusion detection. *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, vol. 2, 2002; 1708 –1713, doi:10.1109/IJCNN.2002.1007775.
11. Chen WH, Hsu SH, Shen HP. Application of svm and ann for intrusion detection. *Comput. Oper. Res.* Oct 2005; **32**(10):2617–2634, doi:10.1016/j.cor.2004.03.019. URL http://dx.doi.org/10.1016/j.cor.2004.03.019.
12. Liao Y, Vemuri V. Use of k-nearest neighbor classifier for intrusion detection. *Computers & Security* 2002; **21**(5):439 – 448, doi:10.1016/S0167-4048(02)00514-X. URL http://www.sciencedirect.com/science/article/pii/S016740480200514X.
13. Wang W, Guan X, Zhang X, Yang L. Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data. *Computers & Security* 2006; **25**(7):539 – 550, doi:10.1016/j.cose.2006.05.005. URL http://www.sciencedirect.com/science/article/pii/S0167404806000903.
14. Soule A, Salamatian K, Taft N. Combining filtering and statistical methods for anomaly detection. *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, IMC '05, USENIX Association: Berkeley, CA, USA, 2005; 31–31. URL http://dl.acm.org/citation.cfm?id=1251086.1251117.
15. Portnoy L, Eskin E, Stolfo S. Intrusion detection with unlabeled data using clustering. *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, 2001; 5–8.
16. Zhang J, Zulkernine M, Haque A. Random-forests-based network intrusion detection systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* sept 2008; **38**(5):649 –659, doi:10.1109/TSMCC.2008.923876.
17. Lakhina A, Crovella M, Diot C. Mining anomalies using traffic feature distributions. *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, ACM: New York, NY, USA, 2005; 217–228, doi:10.1145/1080091.1080118. URL http://doi.acm.org/10.1145/1080091.1080118.
18. Feinstein L, Schnackenberg D, Balupari R, Kindred D. Statistical approaches to ddos attack detection and response. *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, vol. 1, 2003; 303 – 314 vol.1, doi:10.1109/DISCEX.2003.1194894.
19. Wehner S. Analyzing worms and network traffic using compression. *Journal of Computer Security* 2007; **15**(3):303–320.
20. Kulkarni A, Bush S. Active network management and kolmogorov complexity. *inproceedings of IEEE OpenArch*, 2001; 27–28.
21. Evans S, Barnett B. Network security through conservation of complexity. *MILCOM 2002.*, vol. 2,

IEEE, 2002; 1133–1138.
22. Kulkarni A, Bush S. Detecting distributed denial-of-service attacks using kolmogorov complexity metrics. *Journal of Network and Systems Management* 2006; **14**(1):69–80.
23. Eiland E, Liebrock L. An application of information theory to intrusion detection. *Information Assurance, 2006. IWIA 2006. Fourth IEEE International Workshop on*, 2006; 16 pp. 134, doi:10.1109/IWIA.2006.3.
24. Abbasi F, Harris R. Experiences with a generation iii virtual honeynet. *Telecommunication Networks and Applications Conference (ATNAC), 2009 Australasian*, IEEE; 1–6.
25. Cohen W, Ravikumar P, Fienberg S. A comparison of string distance metrics for name-matching tasks. *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, Citeseer, 2003; 73–78.
26. Tridgell A. Efficient algorithms for sorting and synchronization. *Doktorarbeit, Australian National University* 1999; .
27. Abbasi F, Harris R, Morretti G, Haider A, Anwar N. Classification of malicious network streams in honeynets - accepted for publication. *Global Telecommunications Conference, 2012. GLOBECOM '12. IEEE*, vol. 12, 2012; 6 pp. 35, doi:10.1109/GLOCOM.2012.1578391.
28. McHugh J. 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.* Nov 2000; **3**(4):262–294, doi:10.1145/382912.382923. URL http://doi.acm.org/10.1145/382912.382923.
29. Werner T, Fuchs C, Gerhards-Padilla E, Martini P. Nebula-generating syntactical network intrusion signatures. *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, IEEE, 2009; 31–38.
30. Garcia S, Derrac J, Cano J, Herrera F. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* march 2012; **34**(3):417 –435, doi:10.1109/TPAMI.2011.142.