Early Detection of Cyber Security Threats using Structured Behavior Modeling

Xiaohua Yan, Carnegie Mellon University
Joy Ying Zhang, Carnegie Mellon University

The rapid evolution of network intrusions has rendered traditional Intrusion Detection Systems (IDS) insufficient for cyber attacks such as the Advanced Persistent Threats (APT), which are sophisticated and enduring network intrusion campaigns comprising multiple imperceptible steps of malicious cyber activities. Dealing with such elaborated network intrusions calls for novel and more proactive defense methodologies. In this paper, we propose an effective early intrusion detection system called SID based on structured modeling of cyber attack behavior, which aims to discover the underlying high-level behavioral patterns within network traffic that are likely to be early signs of cyber attacks. Our system is capable of detecting cyber attacks in their early phases so that courses of defense actions can be made in advance of real compromises. Our experiments on the KDD99 dataset show that using high-level behavioral patterns for predicting cyber attacks achieves much better performance than that of using low-level analysis of network traffic, which is a standard approach in current IDS. F1 scores of more than 0.9 are reported for early detection of network attacks in the KDD99 dataset within windows of certain sizes.

1. INTRODUCTION

The detection of network threats and attacks has been one of the most challenging and persistent problems in the field of computer security for years. Network intrusion detection systems (NIDS) attempt to discover unauthorized access to certain network resources by analyzing network traffic data for signs of malicious activities which could undermine the normal operation of a network. And they have been proved effective in defending against traditional cyber threats such as malwares, Denial of Service attacks (DoS), buffer overflow attacks and so forth. The techniques utilized by most NIDS fall into two categories: anomaly detection and misuse detection.

Anomaly detection works by modeling the intended behavior of users and applications with normal operation traffic, interpreting deviations from this “normal” behavior as an anomaly. The main advantage of anomaly detection techniques lies in that they are able to detect previously unknown attacks (also known as zero-day attacks). Nevertheless, in actual systems the process of building and training “normal” profiles of the network traffic can be time-consuming and difficult for highly dynamic environments. Furthermore, since any violation against the “normal” patterns is identified no matter whether it is part of the real threat or not, a high false-positive rate is a prevalent problem for these NIDS.

On the other hand, misuse detection systems use signatures that describe already-known attacks and match them against the audit data stream, looking for evidence of known network threats [Kemmerer and Vigna 2002]. Compared with anomaly detection systems, they usually produce fewer false positives at the expense of being incapable of discovering new attacks such as zero-day attacks [Bilge and Dumitras 2012] which exploits previously unknown vulnerability of the network.

With the rapid and enormous development of computer networks, however, new and more advanced types of attacks emerge to ask for novel and more sophisticated defense mechanisms. For example, a new class of threats known as Advanced Persistent Threat (APT), represents well-resourced and trained adversaries that conduct enduring intrusion campaigns targeting highly confidential information [Hutchins et al. 2011]. Usually starting with zero-day exploits, APT actors follow multiple carefully planned stages to achieve their objectives. For example, an APT whose goal is to exfiltrate sensitive data may consist the following steps:
(1) attacker sends email with attached Word document to several people
(2) a victim opens the Word document and enables macros
(3) macros exfiltrate all files in any directory that has a “Recent File” in Word via FTP

Despite the many successes achieved by conventional anomaly or misuse detection systems, they are insufficient for novel types of network intrusions like the APT because of the following reasons.

(1) Both of the defense mechanisms are based on the analysis of low-level (either packet-level or flow-level) network traffic while overlooking the latent structural information hidden in the raw traffic data. To be specific, most of the current solutions to intrusion detection rely on low-level features of network traffic such as IP addresses and ports of individual connections and consider each cyber attack as a single step instead of multiple carefully planned steps that characterize APTs.

(2) Both intrusion detection techniques are reactive countermeasures rather than proactive prevention mechanisms assuming that the compromise is the result of a fixable flaw. Nevertheless, the persistence characteristic of APT-like network intrusions implies that only through complete analysis of early phases of the attacks can actions be taken at those phases to mitigate future intrusions.

In this work, we present a Structured Intrusion Detection (S-ID) system which not only overcomes the above drawbacks of conventional NIDS, but is also capable of detecting possible cyber threats in their early phases so that courses of defense actions can be made in advance of real compromises. We address the problem of early detection of cyber threats based on the high-level structured information captured in the time series of network traffic using the Helix model proposed by Peng et al. [Peng et al. 2011], which was originally introduced as a Natural Language Processing (NLP) approach used for behavior recognition in mobile sensing problems. In the field of NLP, extensive efforts have been made to enable computers to understand important linguistic concepts such as syntax and semantics, which is difficult because the recovery of the grammatical structure in natural languages is stymied by uncertainty and ambiguity. Considering the analogy between network traffic and natural languages as stated in Table I, however, it is clearly promising to apply language approaches to describing patterns indicative of malicious network activities with the assumption that there also exist “grammars” underlying the network traffic which are less ambiguous. And the discovery of misuse and anomalous patterns can be well treated as the problem of learning syntactic structures and semantic fragments of the “network language”.

<table>
<thead>
<tr>
<th>Table I: Analogy between natural languages and network traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
</tr>
<tr>
<td>Symbolic representation</td>
</tr>
<tr>
<td>Reusable patterns</td>
</tr>
<tr>
<td>High-level meaning derived from structures</td>
</tr>
</tbody>
</table>

The remainder of this paper is organized as follows. We describe the problem definition and the language-based model in Section 2. We provide more implementation
Early Detection of Cyber Security Threats using Structured Behavior Modeling

...details of the SID system in Section 3. In Section 4 we show that the proposed system is effective in early detection of cyber attacks based on our experiment results on the KDD99 dataset. An in-depth discussion is carried out in Section 5 regarding the experiment results. Finally, in Section 6 we conclude the paper and point out some future work which will be interesting to work on.

2. STRUCTURED MODELING OF NETWORK TRAFFIC BEHAVIOR

Following the illustration of the framework of SID in Figure 1, we first define the mathematical notations as follows that are used for describing the proposed approach.

Given the time series of the network traffic \( S = \{S_1, ..., S_N\} \) of length \( N \). Each record \( S_i \) is represented by \( d \)-dimensional sensor readings defined by \( S_i = \{O_{i1}, ..., O_{id}\} \), we define the Trace Sequence of the captured network traffic as \( T = \{T_1, ..., T_N\} \) where \( T_i = f_M(S_i) \) with \( f_M \) denoting the mapping function that converts the \( d \)-dimensional sensor readings of \( S_i \) into a single trace symbol \( T_i \) of the network traffic. In other words, the trace sequence \( T \) is a 1-dimensional representation of the raw network traffic which makes further computation more efficient due to the reduced dimensionality. In addition, we define the multi-level Network Activity Text as \( A = \{A^1, ..., A^L\} \) of \( L \) levels, where \( A^l = \{a^l_1, ..., a^l_{|A^l|}\} \) in which each \( a^l_i \) represents the \( i \)-th activity in level \( l \). Note that \( A^1 \) here denotes the lowest-level activities (i.e. \( A_1 = T \)) while \( A^L \) denotes the highest. And subject to \( A \), the Network Activity Grammar \( G \) is defined as \( G = \{V, R\} \) where \( V \) denotes the multi-level Activity Vocabulary of \( L \) levels and \( R \) denotes the Relations specifying the grammar structure. To be specific, \( V = \{V^1, ..., V^L\} \) where \( V^l \) stands for the activity vocabulary at the \( l \)-th semantic level, such that \( a^l_i \in \bigcup_{i=1}^{L} V^l \). Accordingly, we define that any directed relationship \((v_{i1}^l \rightarrow v_{i2}^l) \in R \) iff. \( l_1 > l_2 \) and \( v_{i1}^l \) is a super-activity or a generalized activity of \( v_{i2}^l \).

![Diagram](image)

Fig. 1: Framework of the Structured Intrusion Detection system. The raw network traffic is first preprocessed to be represented by network sequences. And structured modeling of network behavior is performed on these trace sequences.

With the notations defined above, the problem of early detection of cyber threats based on structured modeling of network behavior captured in the data center is defined as

— inducing the grammar \( G = \{V, R\} \) that best describes the network traffic \( S \) and finding the multi-level early indicators belonging to \( V \) based on \( G \) that have high probabilities to trigger either malicious or normal network activities;

— giving early predictions of network behavior when attack indicators are identified as new network traffic is captured.

In the following sections, we describe the language-based approaches to this problem consisting of three main steps: (1) Network Traffic Conversion (2) Grammar Induction, (3) Structured Trigger Discovery and (4) Early Detection, given the sensor readings $S$ of the low-level network traffic.

2.1. Network Traffic Conversion using DBSCAN

At the beginning of the proposed method is the preprocessing of the low-level network traffic captured by multiple monitoring sensors such as tcpdump. The example below shows the information of a packet captured by tcpdump.

```
22:00:26.201715 arp who-has 192.168.1.2 tell 192.168.1.1
```

Such network monitoring logs produced from various sensors serve as the input of the proposed intrusion detection system. Based on monitoring logs, we collect statistics and define relevant features, either continuous or discrete, that are characteristic of each network trace in an attempt to capture the “shallow semantics” of the network traffic.

The features for each network trace are not limited to the basics of network transactions such as IP addresses, port numbers and protocol types. For example, Stolfo et al. [Lee et al. 2000] defined high-level features of network connections that help in distinguishing normal connections from attacks, including duration of connections, number of “compromised” conditions, percentage of connections that have “REJ” errors, etc. While there are various ways of defining features that characterize the network traffic for different purposes, it should be noted that feature selection is beyond the scope of this paper, though it is possible that performance improvement of the system can be achieved by selecting more relevant features and eliminate the irrelevant ones for our specific task.

Following the data preprocessing step, the network traffic is converted into a trace sequence using clustering algorithms such that similar network traces are represented by the same symbol in the trace sequence. Specifically, each transaction record of the network traffic is converted into a single symbol analogous to a word in natural languages, resulting in a trace sequence $T = \{T_1, ..., T_N\}$ representing the network traffic comprised of $N$ transactions where $T_i = f_M(S_i)$ with $f_M$ being the transformation from raw network traffic to trace symbols based on clustering results.

Among the plenty of clustering algorithms available for our task, we use the DBSCAN algorithm proposed by [Ester et al. 1996] because of its capability of finding arbitrarily shaped clusters. The basic idea of DBSCAN clustering algorithm is that every point in a certain cluster should have a minimum number of $MinPts$ points in its neighborhood of radius $\epsilon$. Figure 2 illustrates an example clustering result obtained by DBSCAN from a simulated 3-D dataset, where points of different colors indicate different clusters.

2.2. Grammar Induction using Helix

Given the trace sequence $T$ converted from the raw network traffic captured in the data center, we used Helix [Peng et al. 2011] to induce the context-free grammar (CFG) $G = \{V, R\}$ that best describes the hierarchical activity structure embedded in $S$. The process of grammar induction with Helix consists of two steps described as follows.

2.2.1. Super-activity discovery. To discover the super-activities of low-level network activities, we first assume that activity pairs occurring jointly within a window may be components of a super-activity. For a certain pair of activities $[v^i_l, v^j_l]$, we accumulate the
marginal occurrence frequency of $v^t_i$ and $v^t_j$, as well as their joint occurrence frequency along the trace sequence. Based on these marginal and joint frequencies, collocation is computed to determine whether they constitute a super-activity. For our problem in particular, collocation between two activities suggests that their co-occurrence stems from inherent dependency rather than pure randomness. The statistical collocation significance can be measured by constructing the contingency table for each activity pair $[v^t_i, v^t_j]$ and calculating the $\chi^2$ statistics [Cochran 1952], which is further compared to a threshold parameter (e.g. $\chi^2_{0.05}(1) = 3.841$) to decide whether to merge $[v^t_i, v^t_j]$ to form a super-activity.

2.2.2. Vocabulary generalization. The collocation discovery works when there are repeating sequences of sub-activities. However, such repetitions are unlikely to be exact for network activities due to the loss of information when performing clustering. In order to better detect higher-level activities in such condition, we need to generalize similar activities before constructing the contingency tables. As suggested in [Peng et al. 2011], the similarity between two activities at the same level can be measured in terms of content similarity $\phi_c$ and context similarity $\phi_x$, which are further aggregated into one single similarity measure $\phi$ by their arithmetic mean. This aggregate $\phi$ is then used in a complete-link clustering algorithm where a link exists between two activities only if their similarity $\phi$ is larger than a threshold $\phi_T$. As a result, all activities grouped in the same cluster are highly similar to each other. Algorithm 1 summarizes the Helix algorithm. And in Figure 3 we illustrate an example grammar tree diagram resulted from Helix algorithm, where the nodes represent the words of activity vocabulary at different levels and the directed arrows indicate relationship between the nodes (i.e. the parent node is either a super-activity or a generalized activity of its children nodes).

Note that as new grammar rules are induced (line 6 in Algorithm 1), there could be multiple ways of generating the new activity text $A'$ from $A'$ due to the ambiguity of parsing $A'$ using the new grammar. This problem can be solved finding the most probable sequence according to the “weight” (Chi-squared statistics in our case) of each rule using Dynamic Programming algorithms (e.g., Viterbi algorithm).
Fig. 3: Example grammar tree induced by Helix. Nodes represent words of activity vocabularies and directed arrows indicate the relationship between nodes (i.e., the parent node is either a super-activity or a generalized activity of its children nodes).

**ALGORITHM 1:** Grammar induction on network trace sequence

**Input:**
- $S$, network trace sequence obtained from DBSCAN clustering
- $\alpha_T$, merging threshold parameter
- $\delta_T$, generalization threshold parameter

**Output:**
- $G = \{V, R\}$, discovered hierarchical grammar
- $A$, hierarchical network activities labeled using $V$

1. $(A^1, V^1) = \text{initialize}(S)$
2. $(A', V') = (A^1, V^1)$
3. $l = 1$
4. while True do
5.   $l = l + 1$
6.   $(A^l, V^l) = \text{discover super-activities from } (A', V')$
7.   break if $|V^l| == 0$
8.   for $v_i \in V^l, v_j \in V'$ do
9.     add edges $(v_i, v_j)$ into $R$ for all collocations
10. end
11. $(A', V') = \text{generalize vocabulary from } (A^l, V^l)$
12. end
13. $V = \{V^1, ..., V^{l-1}\}$
14. return $G = \{V, R\}, A$

2.3. Structured Trigger Discovery

Once the grammar of the network traffic is induced, the grammar rules can be used to parse new network trace sequences into structural representations. In particular, we are able to obtain all the valid parse subtree structures, which are referred as constituents in NLP, at different levels along the trace sequence using parsing algorithms. One common approach is the Cocke-Younger-Kasami (CYK) chart parsing algorithm [Cocke and Schwartz 1970; Younger 1967; Kasami 1965] which uses dynamic programming – partially hypothesized results are stored in a reusable structure called a chart. Let the induced context-free grammar be a four-element tuple $G = (N, \Sigma, P, R_S)$ where $N, \Sigma, P, R_S$ stand for the set of non-terminal symbols (higher-level activity vocabularies in our case), the set of terminal symbols (lowest-level activity vocabulary in our case), the set of grammar production rules and the distinguished start symbol, respectively. The problem of finding all the valid constituents of different levels on the trace sequence can be rephrased as checking whether any subsequence of symbols $w$ belongs to $L(G)$ where $L(G)$ is defined as all those sequences that can be derived in
a finite number of steps from the start symbol belonging to \( R_S \), which in our case is equivalent to \( \mathcal{N} \) such that all the symbols in high-level activity vocabularies can be valid start symbols of a derivation for grammar \( G \). We tailor the CKY parsing algorithm for our need in Algorithm 2 for the task of finding all valid constituents in the trace sequence.

**ALGORITHM 2**: Structured trigger discovery with modified CYK parsing

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>: ( S ), trace sequence of ( N ) symbols ( S_1, ..., S_N )</td>
</tr>
<tr>
<td>: the set of ( M ) non-terminal symbols ( R_1, ..., R_M )</td>
</tr>
<tr>
<td>: ( \text{maxSpanLen} ), the maximum length of parsing span to be considered</td>
</tr>
</tbody>
</table>

| Output | the locations of all valid constituents with corresponding derivation start symbols |

1. Let \( P \) be an \( N \times N \times M \) array of booleans. Initialize all elements of \( P \) to False.
2. **for** \( (i = 1; i \leq n; i++) \) **do**
   
   3. **for** each unit production \( R_m \rightarrow S_i \) **do**
   
   4. \( P[i, 1, m] = \text{True} \)
   
   **end**
   
   5. **end**
   
   6. // \( i \) iterates over the length of the span
   
   7. **for** \( (i = 2; i \leq \text{maxSpanLen}; i++) \) **do**
   
   8. // \( j \) iterates over the start of the spans
   
   9. **for** \( (j = 1; j \leq n - i + 1; j++) \) **do**
   
   10. // \( k \) iterates over the partition of the span
   
   11. **for** \( (k = 1; k < i; k++) \) **do**
   
   12. for each generalization rule \( R_A \rightarrow R_B \) **do**
   
   13. if \( P[j, k, B] \) **then**
   
   14. \( P[j, k, A] = \text{True} \)
   
   15. **end**
   
   16. **end**
   
   17. for each production rule \( R_A \rightarrow R_B R_C \) **do**
   
   18. if \( P[j, k, B] \) and \( P[j + k, i - k, C] \) **then**
   
   19. \( P[j, k, A] = \text{True} \)
   
   20. **end**
   
   21. **end**
   
   22. **end**
   
   23. **end**
   
   24. **end**
   
   25. **if** any of \( P[j, i, x] \) is true \( (x \) is iterated over the set of starting symbols \( R \)) **then**
   
   26. **return** \( P[j, i, x] \)
   
   27. **end**


Once the grammar \( G \) is induced and all the \( n \) valid constituents of level \( l \) \( C^l = \{C^l_1, ..., C^l_N\} \) along the training sequence are obtained, the training trace sequence can be represented by a set of \( \{C^l_i, W^l_i\} \) pairs for level \( l \) where \( W^l_i \) indicates the sequence of network activities that correspond to \( C^l_i \) within a window of fixed length occurring after \( C^l_i \), as illustrated in Figure 4. It should also be noted that multiple valid parses are possible for the same span of symbols on the trace sequence due to the ambiguity of the grammar. In our case, we can avoid the problem of ambiguity by always choosing the same starting symbol representing the derivation of the same span of symbols on the trace sequence.

For the purpose of identifying constituent triggers that predict possible incoming attack activities, we again construct the contingency table and calculate the \( \chi^2 \)
statistics to measure the correlation between each constituent and network behavior. By walking through the training trace sequence from left to right, we accumulate the joint frequencies of all \((C_i, A_i)\) pairs for level \(l\), where \(A_i \in W^l_i\) denotes the label of the trace symbol which can be either binary (e.g. “normal” or “attack”) or multi-categorical (e.g. “normal”, “dos”, “probe”, etc.). For example, the constituent-window pair \((8000, \{normal, normal, attack\})\) accounts for two \((8000, normal)\)s and one \((8000, attack)\). Such counts are recorded in a Joint Frequency Table (upper-left of Figure 5). The Marginal Frequency Table (upper-right of Figure 5) is also constructed similarly. For example, the pair \((8000, \{normal, normal, attack\})\) accounts for one \((8000, -)\), one \((- , attack)\) and two \((- , normal)\)s. The contingency table (lower-left of Figure 5) is then constructed according to the joint and marginal frequency tables. The \(\chi^2\) statistics of each \((C_i^l, A_i^l)\) pair is computed based upon the contingency table (lower-right of Figure 5), which is stored for further use of the final model that makes prediction of incoming network behavior given the testing trace sequence.

\[
\begin{align*}
\chi^2 &= \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \\
&= 30.945
\end{align*}
\]

Fig. 5: Example calculation of \(\chi^2\) statistics based on joint frequency tables, marginal frequency tables and contingency tables.

### 2.4. Early Detection of Cyber Threats

The early detection of cyber threats on the testing trace sequence works by examining the correlation between each constituent trigger identified on the trace sequence with possible attack types. To be specific, by walking through the testing sequence (i.e., reading in a new trace symbol every time) we are able to find all the valid constituent triggers of different levels for each network activity window. It is worth noting that each activity window on the trace sequence may correspond to multiple constituent triggers of different levels. In this case, we employed a backoff method to choose the constituent trigger up to level \(L\) as the basis of prediction for the activity window. That
is, if for a certain prediction window $W$ on the testing sequence there is no constituent trigger of certain level $L$ discovered by our parsing algorithm, we back off to triggers of lower levels until we find one for the prediction window to ensure that information of highest possible level is used to base our predictions. Clearly, the worst case would be that we could only find level-1 trace symbols corresponding to $W$, which is guaranteed.

Suppose that a valid constituent trigger $C$ is identified on the testing sequence, with $W$ being the corresponding activity window. We first look for all possible types of network activities that can be triggered by $C$, which we denote as $A$, by looking up the stored mappings derived in the trigger discovery step of the training phase. We then compare the $\chi^2$ statistics between each $(C, A_i)$ pair with fixed thresholds to make our predictions, where $A_i \in A$. Here we assume that $A$ contains both negative label (i.e., “normal”) and positive labels (i.e., “attack”, “dos”, etc.) since if $C$ is only possible to trigger normal activities, the prediction can be easily determined as negative and vice versa.

Finally, we are able to give prediction results based on Algorithm 3, where $normalThr$ and $attackThr$ stand for two threshold values of $\chi^2$ statistics. Namely, if the $\chi^2$ statistics of the constituent-activity pair $(C, A_i)$ is larger the respective threshold, the prediction is determined to be consistent with $A_i$.

**ALGORITHM 3:** Early prediction of network behavior on the testing trace sequence

```
Input : $S_{test}$, the test sequence of length $N$ indexed by $[0, 1, ..., N − 1]$
Input : chart, the chart derived from CYK parsing on the testing sequence
Input : $chiSqrMapping$, the mappings from each constituent trigger to its $\chi^2$ statistics at each vocabulary level
Input : $windowSize$, the size of prediction windows
Input : $maxLevel$, the maximum level of constituent triggers used for prediction
Output: prediction label for each valid trigger along the test sequence
1 startIndex = 0
2 while $startIndex \leq N-windowSize$ do
3     triggerFound = False
4     predictionWindow = $S_{test}[startIndex:startIndex+windowSize]$  
5     possibleTriggers = find all valid constituent triggers for $predictionWindow$ by looking up $chart$
6     targetTrigger = get the constituent trigger up to level $maxLevel$ from $possibleTriggers$
7     $A = get all possible network activity types that can be triggered by targetTrigger$
8     if $normal \in A$ then
9         if $chiSqrMapping[(targetTrigger, normal)] > normalThr$ then
10            prediction = $normal$
11       else
12           prediction = $attack$
13       end
14     else
15         maxChiSqr = $max(getChiSqrValues(chiSqrMapping, A))$
16         if $maxChiSqr > attackThr$ then
17             prediction = $attack$
18         else
19             prediction = $normal$
20       end
21     end
22     startIndex = startIndex + 1
23 end
```
3. EXPERIMENTS WITH THE KDD99 DATASET

In this section, we evaluate the performance of the SID system for early detection of network attacks on the well-known KDD99 dataset [Haines et al. 2000], which is a version of the DARPA 1998 dataset and all the network traffic including the entire payload of each packet is recorded in tcpdump format for evaluation. It consists of approximately 4,900,000 data instances, each of which is a vector of extracted feature values from a connection record comprising a sequence of TCP packets to and from some IP addresses, starting and ending at some well defined times. Each connection was labeled as either normal or as exactly one specific kind of attack including DoS, R2L, U2R and probing.

Evaluating the proposed intrusion detection system with DARPA datasets may not be representative of the performance with more recent attacks or with other attacks against different types of machines or other network infrastructure since this dataset was made publicly available over ten years back, which have caused a lot of criticisms against this IDS evaluation dataset. However, the analysis of Thomas et al. [Thomas et al. 2008] shows that the inability of the IDS far outweighs the limitations of the dataset. Considering the huge size of the dataset, we performed all of our experiments based on the 10 percent portion of the dataset which consists of 494,021 instances in total.

Despite the justification of the use of the KDD99 dataset for IDS evaluation, [Tavallaee et al. 2009] pointed out that the most important deficiency of this dataset is the huge number of redundant records. They found that about 78% and 75% of the records are duplicated in the train and test set respectively. This large amount of redundant records in the train set would cause learning algorithms to be biased by the more frequent records, and thus prevent it from learning unfrequent records which are usually more harmful to networks such as U2R attacks (i.e., unauthorized access to local superuser privileges). Therefore, in the preprocessing step of our experiment, we first collapse the original dataset such that it does not include consecutive redundant records, resulting in 281,644 data instances.

3.1. DBSCAN clustering

As stated in Section 2, each connection in the raw network traffic is first represented as a feature vector characterizing the behavior of the connection. Regarding the KDD99 dataset, [Lee et al. 2000] defined 41 higher-level features that help in distinguishing normal connections from attacks. The feature set can be divided into 3 categories.

- Basic features of individual TCP connections. These features give a basic characterization of each single connection, which can be derived by packet filtering and reassembling engine, including duration (length of the connection), protocol type, flag (normal or error status of the connection) and so forth.

- Traffic based features: the traffic based features examine the patterns and relations among connection records within a certain time window. For example, the “same host” features examine only the connections in the past two seconds that have the same destination host as the current connection, and calculate statistics related to protocol behavior, service, etc.

- Content features. Domain knowledge is used to add features that look for suspicious behavior in the data portions, such as the number of failed login attempts.

Some features of this dataset take on both continuous and discrete values, which makes it difficult to define appropriate similarity (or distance) measures for clustering algorithms. In our experiment, we propose to use the well-known Gower’s Similarity Coefficient [Gower 1971] $S_{ij}$ to measure the similarity between instances of the KDD99
dataset, which compares two data instances $x_i$ and $x_j$ and is defined as

$$S_{ij} = \frac{\sum_k^n w_{ijk} S_{ijk}}{\sum_k^n w_{ijk}}$$  (1)

where $S_{ijk}$ denotes the contribution provided by the $k$-th variable and $w_{ijk}$ is 1 or 0 depending on whether the comparison is valid for the $k$-th variable. For continuous variables (e.g. duration of the connection, number of file creations, etc.), we have

$$S_{ijk} = 1 - \frac{|x_{ik} - x_{jk}|}{r_k}$$  (2)

where $r_k$ is the range of values for the $k$-th variable. And for discrete variables such as protocol type and service, the value of $S_{ijk}$ is 1 if $x_{ik} = x_{jk}$ or 0 otherwise. And $w_{ijk} = 1$ if both instances have observed values for the $k$-th feature.

Based on the similarity measure described above, we use the first 90 percent of the whole dataset as training set, on which DBSCAN is performed to produce the training trace sequence. The parameters for DBSCAN algorithm is set as $\epsilon = 0.02$ and $minPts = 4$, resulting in a total of 203 clusters including noise and therefore we are able to represent each data instance in the training dataset with their cluster IDs. Namely, we obtain an activity vocabulary of size 203 for the lowest-level network traffic. Notice that the parameter values of DBSCAN are set empirically and tuning the parameters is possible to increase the performance of the system, which is beyond the scope of this paper.

The labels of the training instances (i.e., either normal or attack) can be used to evaluate the performance of DBSCAN clustering. One simple approach is by classifying the elements within each cluster as either normal or attack according to the majority vote – all the instances of each cluster are classified as the same label that dominates the cluster. Following this classification approach, the performance of DBSCAN clustering is measured in terms of precision, recall and F1 score, as displayed in Table II.

<table>
<thead>
<tr>
<th># total instances</th>
<th># true normal(negative) instances</th>
<th># true attack(positive) instances</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>253479</td>
<td>86580</td>
<td>166899</td>
<td>0.9980</td>
<td>0.9753</td>
<td>0.9865</td>
</tr>
</tbody>
</table>

As can be observed from Table II, DBSCAN achieves considerably good performance distinguishing between normal and attack instances, which also justifies the further use of the resultant trace sequence as the input for grammar induction.

3.2. Language modeling of the trace sequence

The trace sequence derived from DBSCAN clustering can be thought of as a long sentence in analogy to natural languages. Before we move to the step of inducing the underlying grammar of the sequence, it is instructive to examine whether there are actual repeating structures in the sequence. One simple but powerful approach is to model the sequence using a statistical n-gram language model, which assigns a probability $P(w_1, ..., w_m)$ of a sentence of $m$ words by means of a probability distribution.

The basic assumption of an n-gram model is that the probability of observing the $i$-th word $w_i$ can be approximated based on the context history of the preceding $n - 1$ words
rather than all the \(i - 1\) words in the history. Specifically, the probability \(P(w_1, ..., w_m)\) of observing the sequence \([w_1, ..., w_m]\) can be approximated as

\[
P(w_1, ..., w_m) = \prod_{i=1}^{m} P(w_i|w_1, ..., w_{i-1}) \simeq \prod_{i=1}^{m} P(w_i|w_{i-(n-1)}, ..., w_{i-1})
\]

(3)

where the conditional probability can be modeled using Maximum Likelihood Estimation (MLE):

\[
P(w_i|w_{i-(n-1)}, ..., w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, ..., w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, ..., w_{i-1})}
\]

(4)

When \(n = 1\), the \(n\)-gram model (also named unigram model in this case) simply assumes that the occurrence of each word all depends on its own. For natural languages, the \(n\)-gram model usually characterizes a language better with bigger values of \(n\) such as \(n = 2\) (a.k.a. bigram model) and \(n = 3\) (a.k.a. trigram model) because of the syntactic patterns of languages. In an effort to prove the existence of structural patterns in the network traffic, we model the network trace sequence with \(n\)-gram language models of different orders. Instead of modeling the whole training sequence, however, we collapse all the consecutively identical symbols in both the training and test sequences so that the model is not biased towards the large quantities of sequentially repeated trace symbols in the sequence and we can better model the transitions of network behavior. The collapsing results in a training sequence of length 19230 and a testing sequence of length 2914.

The goodness of a language model can be measured by perplexity on both the training and test data. For the training sequence, the perplexity is computed by

\[
\text{perplexity}_{\text{train}} = 2^H(p) = 2^{-\sum_{w \in S_{\text{train}}} p(w) \log_2 p(w)}
\]

(5)

where \(p\) is a discrete probability distribution estimated from the training data. And the perplexity of the proposed model \(p\) for the test sequence is defined as

\[
\text{perplexity}_{\text{test}} = 2^H(p) = 2^{-\sum_{w \in S_{\text{test}}} \frac{1}{N} \log_2 p(w)}
\]

(6)

where \(N\) here refers to the length of the test sequence. Better models of the trace sequences will tend to assign higher probabilities to the test events and therefore have lower perplexity. Also note that it is possible that some symbols in the test sequence were not observed in the training sequence, in which case we choose to use the Witten Bell discounting [Witten and Bell 1991] scheme to approximate the probabilities of unseen events. Finally, in Table III we compare the perplexity of language models of different orders on both the training and test sequences.

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity on training data</th>
<th>Perplexity on test data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unigram</td>
<td>36.53</td>
<td>203.12</td>
</tr>
<tr>
<td>Bigram</td>
<td>6.07</td>
<td>8.80</td>
</tr>
<tr>
<td>Trigram</td>
<td>3.10</td>
<td>6.61</td>
</tr>
<tr>
<td>4-gram</td>
<td>2.80</td>
<td>6.76</td>
</tr>
<tr>
<td>5-gram</td>
<td>2.42</td>
<td>6.77</td>
</tr>
<tr>
<td>6-gram</td>
<td>2.27</td>
<td>6.83</td>
</tr>
<tr>
<td>7-gram</td>
<td>2.12</td>
<td>6.88</td>
</tr>
</tbody>
</table>
It is obvious from Table 3 that the perplexity of higher-order language models is significantly lower that of the unigram model for both training and test sequences, which implies the existence of dependency structures underlying the trace symbols of the sequence. In the next section we describe the experiments we performed on the dataset to discover the latent structural patterns of the network traffic sequence.

3.3. Early detection of cyber threats

Following the grammar induction procedure described in Section 2.2, we are able to obtain the underlying context-free grammar \( G \) which in total has 12,095 grammar rules resulted from the training network traffic. The effectiveness of the SiD system in early detection of intrusion activities is evaluated according to the prediction results of the network behavior on the testing trace sequence comprising 2,914 traces, using the decision tree algorithm described in Algorithm 3. The testing sequence is fairly balanced, which contains 1,507 positive (attack) samples and 1,407 negative (normal) samples. Binary predictions (whether attack activities would occur in the prediction window) are given for each constituent trigger identified on the testing trace sequence using modified CYK parsing. In Table IV we list the number of constituent triggers of different levels identified on testing trace sequence.

<table>
<thead>
<tr>
<th>Level</th>
<th>Level = 1</th>
<th>Level = 2</th>
<th>Level = 3</th>
<th>Level = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of constituents</td>
<td>2914</td>
<td>2665</td>
<td>1578</td>
<td>1035</td>
</tr>
</tbody>
</table>

Finally we are able to measure the performance of the early detection algorithm in terms of True Positive Rate (TPR), True Negative Rate (FPR) and \( F_1 \) measure, which are defined by

\[
TPR = \frac{TP}{TP + FN} \tag{7}
\]

\[
TNR = \frac{TN}{FP + TN} \tag{8}
\]

\[
F_1 = \frac{2TP}{2TP + FP + FN} \tag{9}
\]

By varying the values of \( \text{attackThr} \) and \( \text{normalThr} \) in the decision tree, we plot the Receiver Operating Characteristic (ROC) curves for constituent triggers up to different levels (i.e., \text{maxLevel} = 1, 2, 3, 4) in Figure 6 with different sizes of prediction windows (i.e. \text{window size} = 1, 3, 5, 7). Note that since there are two threshold values to be set based on grid search, the complete ROC curve is likely to have fluctuation, and we only preserve the monotonically increasing points on the final ROC curves to give a neat looking. In Table V we also list the best TPR, FPR that produce the highest \( F_1 \) score based on the best setting of threshold parameters corresponding to the best operating point on the ROC curve.

4. ANALYSIS AND DISCUSSION

As can be observed from Figure 6, the prediction performance of high-level (i.e., \text{level} = 2, 3, 4) constituent triggers far exceeds that of level-1 triggers on the per-prediction basis. Moreover, it is clear that the performance of the prediction algorithm degrades as the size of the prediction window increases, which is in accordance with our expectations since the dependency between constituent triggers and network
behavior becomes weaker as the prediction window grows. Despite the degradation of performance, however, we still obtained the best F-measures of 0.941 for level-2 triggers, 0.924 for level-3 triggers and 0.926 for level-4 triggers respectively when \textit{window size} = 7. Moreover, the best F1 scores achieved by high-level constituent triggers are approximately same for the same window size, with slight variation as observed in Table 6 while there are indeed consistent improvement in terms of F1 scores for certain window sizes (i.e. \textit{window size} = 2, 3, 7, 8, 9) as level grows higher. These

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
\textbf{Window size} & \multicolumn{3}{c|}{\textbf{maxLevel = 1}} & \multicolumn{3}{c|}{\textbf{maxLevel = 2}} & \multicolumn{3}{c|}{\textbf{maxLevel = 3}} & \multicolumn{3}{c|}{\textbf{maxLevel = 4}} \\
\hline
 & \textit{TPR} & \textit{TNR} & \textit{F} & \textit{TPR} & \textit{TNR} & \textit{F} & \textit{TPR} & \textit{TNR} & \textit{F} & \textit{TPR} & \textit{TNR} & \textit{F} \\
\hline
1 & 99.4 & 13.9 & 71.1 & 92.8 & 99.8 & 96.1 & 92.8 & 99.6 & 96.1 & 92.7 & 99.9 & 96.1 \\
2 & 99.3 & 12.8 & 71.1 & 90.8 & 100.0 & 95.2 & 91.0 & 100.0 & 95.3 & 91.2 & 99.9 & 95.4 \\
3 & 99.0 & 12.4 & 71.2 & 89.2 & 100.0 & 94.3 & 89.5 & 99.9 & 94.5 & 89.7 & 99.7 & 94.4 \\
4 & 99.8 & 12.3 & 71.4 & 88.2 & 98.5 & 93.6 & 87.9 & 99.9 & 93.6 & 88.0 & 99.9 & 93.6 \\
5 & 98.4 & 12.9 & 71.3 & 88.4 & 98.0 & 93.4 & 87.4 & 100.0 & 93.3 & 87.7 & 99.6 & 93.2 \\
6 & 98.1 & 11.8 & 71.6 & 87.8 & 99.0 & 93.1 & 87.2 & 99.8 & 93.1 & 87.6 & 99.4 & 93.1 \\
7 & 97.9 & 11.6 & 71.7 & 86.5 & 99.2 & 92.4 & 86.3 & 99.6 & 92.4 & 86.8 & 99.3 & 92.6 \\
8 & 97.6 & 11.3 & 71.7 & 85.2 & 99.3 & 91.7 & 85.5 & 99.3 & 91.9 & 85.9 & 99.4 & 92.1 \\
9 & 97.4 & 11.1 & 71.8 & 90.9 & 90.7 & 91.5 & 85.4 & 99.2 & 91.8 & 85.8 & 99.2 & 92.0 \\
\hline
\end{tabular}
\caption{Best performance measures in percentage acheived by triggers of different levels using backoff}
\end{table}
results suggest that our structured intrusion detection system is able to predict the occurrence of cyber attacks at considerably high precision and low false positive rate.

It also should be noted that since the numbers of valid triggers discovered on the testing sequence are different for different levels (i.e., as level increases, the number of identified triggers drops), the prediction performance of constituent triggers of different levels might not be directly comparable according to Table V and Figure 6. To investigate this problem, we plot another ROC curve in Figure 7 without using backoff that separately measures the prediction performance of constituent triggers of different levels. Similar with the previous experiment, we recorded the best performance measures in Table VI.

One interesting observation from Figure 7 is that as the level of constituent triggers increases, the prediction power of the trigger does not necessarily improves. In fact, prediction performance of level-2 triggers is the best compared with that of level-4 triggers, which deviates from our expectation that prediction based on higher-level structures achieves better accuracy.

To explore this problem, we plot in Figure 8 the type-token curves [Youmans 1990] of constituent triggers of different levels identified on both the training and testing trace sequences. The type-token curves can be used to measure the diversity of the activity vocabularies of different levels. As can be observed from Figure 8, the type-token ratios of high level activity vocabularies are consistently higher than that of
Table VI: Best performance measures in percentage achieved by triggers of different levels without backoff

<table>
<thead>
<tr>
<th>Window size</th>
<th>Level = 1</th>
<th>Level = 2</th>
<th>Level = 3</th>
<th>Level = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TPR</td>
<td>TNR</td>
<td>F1</td>
<td>TPR</td>
</tr>
<tr>
<td>1</td>
<td>99.4</td>
<td>12.9</td>
<td>71.1</td>
<td>96.7</td>
</tr>
<tr>
<td>2</td>
<td>99.3</td>
<td>12.8</td>
<td>71.1</td>
<td>94.7</td>
</tr>
<tr>
<td>3</td>
<td>99.0</td>
<td>12.4</td>
<td>71.1</td>
<td>92.8</td>
</tr>
<tr>
<td>4</td>
<td>98.8</td>
<td>12.3</td>
<td>71.4</td>
<td>92.2</td>
</tr>
<tr>
<td>5</td>
<td>98.4</td>
<td>12.0</td>
<td>71.5</td>
<td>91.4</td>
</tr>
<tr>
<td>6</td>
<td>98.2</td>
<td>11.8</td>
<td>71.7</td>
<td>90.8</td>
</tr>
<tr>
<td>7</td>
<td>97.9</td>
<td>11.6</td>
<td>71.8</td>
<td>89.3</td>
</tr>
<tr>
<td>8</td>
<td>97.6</td>
<td>11.3</td>
<td>71.8</td>
<td>87.8</td>
</tr>
<tr>
<td>9</td>
<td>97.4</td>
<td>11.1</td>
<td>71.9</td>
<td>94.2</td>
</tr>
</tbody>
</table>

level-1 vocabulary on both training and testing sequences, which implies that new activity types occur more frequently as level increases, to some extent leading to the problem of sparsity in the training phase when certain high-level constituent triggers do not have sufficient counts to be reliable for the computation of the $\chi^2$ statistics. On the other hand, this problem could also result from the nature of the experiment dataset when higher-level (e.g., $level = 4$) structures are not as explicit as lower-level ones (e.g., $level = 2$).

Fig. 8: Type-token curves of training and testing trace sequences measuring the diversity of trigger types of different levels.

5. RELATED WORK

Network intrusion detection has been an extensively studied problem in the field of Computer Security. Traditional NIDSs incorporate anomaly or misuse detection techniques [Kemmerer and Vigna 2002], which in essence detect intrusions by analyzing network traffic for signs of malicious activities. Most of the traditional intrusion detection techniques are surveyed in [Debar et al. 1999] and [Sperotto et al. 2010].

In general, our approach follows the framework of intrusion detection systems proposed by Lee and Stolfo [Lee et al. 2000], the key idea of which is to first apply data mining programs to audit data to discover repeated patterns of network traffic based on extracted features and then generate the decision model for new data instances. In particular, clustering techniques, either supervised or unsupervised, are widely employed in intrusion detection systems, [Portnoy et al. 2001; Horng et al. 2011; Leung 2011;].
and Leckie 2005] being some relevant examples. [Portnoy et al. 2001] uses a variant of single-link clustering based on standard Euclidean distance to cluster the KDD99 dataset. Reference [Horng et al. 2011] reports an intrusion detection system that uses hierarchical clustering analysis to enhance the training time of Support Vector Machines for the classification on the KDD99 dataset. Authors of [Leung and Leckie 2005] propose a density-based and grid-based clustering algorithm suitable for unsupervised anomaly detection, which scales well to large number of data records of high dimensionality. Reference [Casas et al. 2012] presents an unsupervised NIDS based on Sub-space Clustering and Multiple Evidence Accumulation techniques, which shows improvement over previously used unsupervised approaches. Other than clustering-based methods, Principle Component Analysis [Lakhina et al. 2004], Genetic Algorithms [Abadeh et al. 2007] and Artificial Neural Networks [Wang et al. 2010] are all well-known intrusion detection techniques used in analyzing network traffic flows for signs of intrusions.

The proposed SID system, however, is different from conventional intrusion detection systems in that it emphasizes on detecting network threats earlier and more proactively. And there are only few research works in the literature that focus on early network intrusion detection. Zhou et al. [Zou et al. 2003] propose algorithms for early detection of the presence of a computer worm using Kalman filtering techniques that model the trend of the captured traffic. In reference [Siris and Papagalou 2006] the authors investigate statistical anomaly detection algorithms (i.e. Adaptive Threshold algorithm and Cumulative Sum algorithm) for the early detection of SYN Flooding, which is the most common type of DoS attack.

To the best of our knowledge, there were also few language-based approaches proposed for the task of network intrusion detection. Hofmeyr et al. [Hofmeyr et al. 1998] introduces an intrusion detection method using sequences of system calls as discriminators between normal and abnormal characteristics of running programs, which is similar to our approach in the sense of using behavioral structures as signs of malicious activities. Authors of [Rieck and Laskov 2006] propose to extract language features such as $n$-grams and words from connection payloads (byte sequences in their case) and apply unsupervised anomaly detection. And particular patterns in these $n$-gram language models can be traced back to attack semantics and utilized for automatic generation of attack signatures. This approach is different from ours in that they only used the local information of byte sequences while our approach models the long-distance dependencies between language patterns and attack activities for the purpose of detecting cyber threats in advance.

6. CONCLUSIONS AND FUTURE WORK

The novel intrusion detection system we have proposed here has the advantage of being able to detect network intrusions earlier than previous proposals in the field of network intrusion detection. It employs a language-based approach to discovering the behavioral semantics underlying network traffic flows and use them as early signs of incoming malicious network activities. In particular, the system models the long-distance dependency between the structural patterns in history network traffic and possible incoming cyber attacks. We have provided experiment results on the KDD99 dataset in terms of the preprocessing and early prediction components of the proposed system, which have verified the capability of our system in terms of early detection of network intrusions with considerably high precision and recall.

In the future, we plan to modify the algorithms deployed in our system to make them scale better on large datasets while at the same time discover the deeper semantics of network behavior. We are also interested in experimenting with real datasets to test its feasibility in real applications.
REFERENCES


Early Detection of Cyber Security Threats using Structured Behavior Modeling


