Extracting and Evaluating Similar and Unique Cyber Attack Strategies from Intrusion Alerts

Stephen Moskal*, Shanchieh Jay Yang*, and Michael E. Kuhl†

*Department of Computer Engineering, †Department of Industrial and Systems Engineering
Rochester Institute of Technology, Rochester, NY
Email: {sfm5015, jay.yang, mekie}@rit.edu

Abstract—Intrusion detection system (IDS) is an integral part of computer networks to monitor and detect threats. However, the alerts raised by these systems are often overwhelming to security analysts, making it difficult to uncover the steps an attacker took to compromise one or more systems in the network. This work presents a novel approach that aggregates IDS alerts and forms sequences of attack activities and their corresponding probabilistic models. This allows comparison of attack sequences to offer insights for unique as well as similar attack behaviors. We aggregate alerts by performing a Gaussian filter on specific alert attributes and model attackers using a suffix-based probabilistic model. We compare sequences generated from ten independent attacking teams with similar objectives demonstrating how our process uncovers similarities and uniqueness between the attacks that was not obvious. The sequences revealed by our process creates meaningful sequences that offers insights on how the attacking teams exploit a network.

I. INTRODUCTION

The constant threat of cyber attackers targeting networks require administrators to implement active tools like Intrusion Detection Systems (IDS) to monitor a network for suspicious activity. IDS’s scan network traffic for matches to a defined set of rules describing known malicious activity as “signatures.” If a packet or a series of packets matches a signature, an alert is generated and logged for a security analyst to evaluate. Security analysts may receive hundreds of critical alerts per day out of a total of possibly hundreds of thousands of lesser important alerts [1]. The mass quantity of alerts is due to general or imperfect rule sets, where a potential 18% false positive rate has been reported [2]. This makes it difficult and time consuming for an analyst to understand if the alerts is a legitimate threat or not.

The volume and the noisy alert logs not only makes it difficult to understand the impact of a single alert but a cyberattack typically consists of multiple actions leading to a final goal, which is often what an analyst tries to prevent by analyzing the alerts. Created by Lockheed Martin [3], a cyber attack kill chain describes a set of steps in which an attacker takes from the beginning stages of an attack to executing final objective. The actual steps of the kill chain varies depending on the context of the attack, whereas a typical cyber kill chain involves three phases: discovery, escalation, and exfiltration [4]. The analyst’s objective here is to “cut the chain”, stopping the attacker as early as possible in the kill chain to minimize the impact of the attack on the network. Due to the massive volume of alerts and generic rules creating high amount of false positives, it is difficult for an analyst to assess when, where and how an attack actually transpired over time.

This work aims at systematically processing the noisy and large IDS alert logs to aid in uncovering the steps the attacker takes to achieve an objective. We present a novel approach to process and represent cyber alert data which is used to reconstruct an attack activity sequence. We use a data set consisting of 10 independent attacking teams in a controlled penetration-testing competition environment and propose a method to model the attack processes of the independent teams to uncover similarities and/or unique strategies among the attackers. Our process reduces the amount of unnecessary alerts and exposes sequences that analysts can understand the interesting behaviors and unique characteristics of an attacker based off of the comparison between teams.

We reduce the large and noisy IDS alerts to a summarized sequence of alert activities types by leveraging key characteristics of cyber data like its repetitiveness, categorical attributes, and the network structure. By using this cyber-aware context and alert aggregation, we uncover similarities between different attacks hidden in the noisy raw data. The aggregated alerts are represented as a sequence of events for each team and modeled using a suffix based model to understand the process an attacker takes leading up to a high severity alert. Critical subsequences are defined from these severe alerts to evaluate likelihoods of sequences occurring against each other attacker to determine the uniqueness of the sequences and reveal interesting patterns that were once hidden.

II. RELATED WORK

The research category that this work primarily falls under is known as “alert correlation” where alerts are processed to reduce noise, extract patterns, or evaluated against other observables to help uncover the true action the attacker performed. Due to the wide adoption of IDS’s, extensive surveys and studies have been performed on the types of alert correlation types [5]–[7], techniques to minimize the amount of false alarms in IDS’s [8], intrusion response systems using alert correlations [9], and multistage attack anomaly detection for advanced persistent threats (APT) [10].

Sadoddin et al. [5] described three categories of alert correlation algorithms: 1) Similarity-based, 2) Knowledge-based,
and 3) Statistical-based. Similarity-based methods rely on a set of rules defining the relationships between features where the rules can be simple and explicitly defined [11], [12] to self-defining feature relationships using machine learning [13] and artificial neural networks [14]. Knowledge-based algorithms uses specific context like an predetermined attack scenario to create attack graphs [15], [16] or uncovering attack scenarios using previous attacks [12], [17]. Lastly, statistically-based models focus on the statistics of each alert and develop behavioral patterns and associations with no contextual information [18]. Sadoddin et al. [5] admits that this classification is not encompassing and some methods may be a mixture of these three categories. This is the case with this work where our method uses a similarity-based approach to aggregate alerts, a knowledge-based approach to develop our sequences, and a statistical model to compare attack sequences.

III. METHODOLOGY

This work processes overwhelming and noisy intrusion alerts and transforms them into summarized sequences of attack activities. These activity sequences can be used to characterize the cyber attacks in the form of probabilistic models. Comparing the probabilistic models enables the identification of unique and similar attack patterns. Fig. 1 shows the overall system architecture with the inputs and outputs of each subcomponent. The following subsections detail the design of these subcomponents.

**A. Data Preprocessing**

We define the IDS alert observables \( o \in O \) as a tuple of multiple attributes such as a time stamp, source IP, destination IP, category of attack, attack signature, source port, destination port, and severity. To reduce excessive repeated alerts caused by automated scripts and scanning, repeated \( o \) with identical attributes within a 1-second time interval are reduced into a single alert. To understand how the attributes in \( o \) change over time, we represent \( O \) as a time-series based on the attributes. We define \( X(\Gamma) = x_1, \ldots, x_n \) where \( x_j \) is the number of alerts that satisfy a set of criteria \( \Gamma \) within a fixed time interval \( w_j = [t_0, t_0 + j \cdot \Delta] \), where \( t_0 \) is the earliest time stamp of the data and \( \Delta \) is a fixed time interval for the histogram bins. Let \( \lambda \in \Lambda \) be the alphabet describing a set of unique numeric symbols for all alert categories in the data set. Likewise, let \( d \in D \) be the set of destination IP’s. The criteria \( \Gamma \) is used to focus on analyzing the observable based on the “where”, “how”, “what”, or a combination of them. For this paper, we focus on the “how” and “where”. Particularly, \( \lambda \) gives the type of scanning and exploits the attackers are using in-conjunction with \( d \) to investigate the attack at each target. The resulting time series demonstrates the intensity of attacks over time that satisfy the criteria \( \Gamma \).

**B. Reducing High Frequency Noise**

Even with the removal of repeated alerts the resulting time series of specific alerts can still be noisy, preventing the extraction of the likely attack actions and attack sequences. Not all actions result in consistent alert arrival times and exhibit peaky behavior similar to high frequency noise. To combat this noise, we apply a 1-D discrete Gaussian convolution filter defined as \( L(X, \sigma) \) [19] using the scale-space implementation of a Gaussian kernel in [20]. To obtain \( \sigma \) with respect to \( \Delta \), the cutoff frequency for a discrete Gaussian filter in the frequency domain is given by (1):

\[
f_c = \frac{F_s}{2\pi\sigma}
\]

where \( F_s = 1/\Delta \) is the sampling rate of the data and \( f_c \) is interpreted as the maximum time interval between alerts, signifying the alerts originate from separate actions. (1) is used to solve for parameter \( \sigma \) in terms of \( \Delta \) and \( f_c \) and the filter is applied to \( X \) given by (2).

\[
\hat{X} = L(X, \sigma)
\]

Once the smoothing is applied and the noise is reduced, the resulting \( \hat{X} \) gives a representation where attack intensity over time is shown with clearer trends. These rising and falling trends enables the definition of the attack activities and thus attack sequence extraction.

**C. Attack Sequence Generation**

Ideally, an attack sequence would comprise of individual actions, including various exploits or scans the attacker performs. From the perspective of the observable data, recovering the exact attack sequence is nearly impossible due to the imperfect nature of the sensors and the potentially evading tactics. The smoothed \( \hat{X} \) gives rising and falling trends the attacker performs using each attack type; a change in such trends are likely indicative of the attacker switching from one type of attack to another. Intuitively, the intensity of a given attack category may rise as the attacker builds up a specific attack stage. When the intensity peaks and begin to fall, it signals the attacker wrapping up that stage and potential start of another one. Given this intuition, we define the Attack Activity as follows.

**Definition 1.** An Attack Activity in \( \hat{X} \) is a tuple \( A = \{j_p, x_j, \lambda, d\} \) describing a local maximum at time \( j_p \) with a
magnitude $x_t$ for an attack category $\lambda$ corresponding to a target IP address $d$.

An attack activity describes a grouping of similar alerts that are likely to be a result of the same action by the attacker. The attack activity captures the time interval of a local maximum in $X$ used to estimate when an attacker begins to switch to another action. Some actions require more attempts or more time to complete their intended objective and we propose that the peak alert activity describes this point where the objective is completed. Where as the first instance of the alert in an alert activity may not indicate when the attacker obtained the required information to move on.

A collection of attack activities representing the steps the attacker took over time is an attack sequence. An attack sequence is denoted as $s = s_1, s_2, ..., s_n \in \Lambda$ where $n$ is the length of the sequence and the count of categories in $s$ is $s$. $X(d, \lambda)$ represents a time series for the criteria of $d$ and $\lambda$ in $\Lambda$, whereas $\hat{X}_d$ is a set of $\hat{X}(d, \lambda) \forall \lambda \in \Lambda$ used as input to generate a single sequence for $d$ in Algorithm 1.

Algorithm 1: Sequence generation for a given $\hat{X}(d)$

```python
function CreateSequence (\hat{X}(d));
    Input : X for an ip
    Output: Sequence $s$ for an ip
2     $s$, attk acts = []
3     for all $\lambda$ in $\Lambda$ do
4         attk acts.append(findAttackActivities(\hat{X}(d, \lambda))
5     end
6     sortByTimeInterval(attk acts)
7     for all $A$ in attk acts do
8         s.append(A, \lambda)
9     end
```

We then use these sequences of attack activities to create subsequences to investigate the series of activities preceding an action identified by a high severity alert. This is akin to the process a security analyst would take, starting with the highest priority alerts and tracing back to understand the previous events. Using the severity of an attack category we create high severity subsequences $s_{sev} : s_{sev} \subset s$ containing activities between two high severity categories. Suricata defines three severity levels: high, medium, and low to roughly describe the potential impact of an alert. We use this notion of high severity sequences to define a significant event where the attacker has performed something malicious and evaluate if the process to leading up to that point is consistent or unique for the attacker and against other attacker’s.

D. Model Generation

Given the definition of an attack sequence above, we define the suffix of a sequence $s$ of length $n$ is represented as $\text{suffix}(s) = s_2, s_3, ..., s_n$ and the set of all suffixes as $\text{suffix}^*(s) = s_1, ..., s_n | 1 \leq i \leq n$. A suffix tree $M(N, E)$ contains a set of nodes $N$ and edges $E$ where a single node and edge between two nodes is defined as $n \in N$ and $(n_x, n_y) \in E$, respectively. The children of $n$ is denoted as $C(n)$ and the count of observations for $e$ is $\text{count}(e)$. The construction of $M$ for a given sequence $s$ is given in Alg. 2.

Algorithm 2: Suffix tree generated for a input sequence $s$

```python
function BuildSuffixTree (s);
    Input : An attack sequence $s$
    Output: A suffix tree $M$
2     curr_node ← $n_{root}$;
3     for all $s_i$ in suffix($s$) do
4         for all $\lambda$ in $s_i$ do
5             if curr_node.hasChild($\lambda$) then
6                 increment(count(curr_node, $\lambda$));
7                 curr_node ← C(curr_node)[\lambda];
8             else
9                 C(curr_node)[\lambda] ← N(\lambda);
10                curr_node ← C(curr_node)[\lambda];
11         end
12     end
13 end
```

The suffix-model for a single attacker $T$, $M_T$, is created using the set of $s \forall \hat{X}(d) \in \hat{X}(T)$. We then create a second opposing model containing for all other attack observations not a part of $T$ as $M_{all} = s \forall \hat{X}(d) \not\in \hat{X}(T)$ to determine if the testing sequence is unique or has been observed previously. We then use the the suffix tree models to determine a likelihood of observing $\lambda$ given some context node in $M$. The probability of a single observation $\lambda$ given a parent node $N$ is shown in (3).

$$P(\lambda|n) = \frac{\text{count}(n, \lambda)}{\sum_{i \in C(n)} \text{count}(e(n, i))} \quad (3)$$

To determine the likelihood of a sequence, Alg. 3 $l(s, M)$ calculates the likelihood of the current position in the tree and the next element in $s$.

Algorithm 3: Likelihood of a sequence given a suffix tree

```python
function SequenceLikelihood ($s$);
    Input : An attack sequence $s$
    Output: The likelihood of $s$ occurring in $M$
2     curr_node = N_R;
3     $l_s = 1$;
4     for $\lambda$ in reversed($s$) do
5         if curr_node.hasChild($\lambda$) then
6             $l_s \leftarrow l_s \times P(\lambda|curr_node)$;
7             curr_node ← curr_node[C(\lambda)];
8         else
9             return 0;
10        end
11 end
12 return $l_s$
```

51
To determine the similarities or uniqueness of a sequence \( s \), the likelihood of a sequence \( l_t = l(s, M_T) \) and \( l_{alt} = l(s, M_{all}) \) and the ratio of these two likelihoods is taken. \( l_t / l_{alt} = 0 \) signifies that the sequence is unique to attacker \( T \) and has not been observed in any other attacker, where as \( l_t / l_{alt} > 0 \) signifies a similarity among other attackers. We compute these likelihoods for each of the high severity sequences \( s_{sec} \) to determine the similar and unique sequences for each team which then can be evaluated for interesting and distinct behaviors based on these likelihoods.

IV. RESULTS AND ANALYSIS

A. Experiment Setup

The data under test in this work is collected from the 2017 National Collegiate Penetration Testing Competition (CPTC) [21] comprised of 10 student teams from different universities across US. The competition mimics a real world network for each team and the teams’ objective is to effectively and comprehensively discover, exploit, and document the vulnerabilities in the network. The data collected for this work is the Suricata IDS alert logs from each team’s network in a 12 hour time period where 9 unique source IP’s targeted a total of 425 IP’s resulting in 47876 alerts across all teams. Each team is comprised of multiple students and an expert red team consultant (coaching and advice only) and there is no communication between teams. This data is not exactly representative of actual cyber attacks as the objective of the competition is to be as comprehensive as possible to mimic a real penetration test. However the controlled network environment allows us to analyze the different strategies each team uses to achieve the same objective.

With a fixed network, a set attack duration, and the same objective, this data set gives us the opportunity to compare the attack sequences of each team directly without having to take in account of different environments. We process each team independently and then compare the likelihoods of the discovered sequences of alert categories independent of the network structure. The sequences generated in this work is presented as a list of alert categories id’s with the last entry identified as a high severity alert. The mapping between id to category description and severity is described in Table I.

We use these sequences to evaluate the similarities and differences of the preceding attack activities to common high severity alerts.

B. Method Analysis

To show how our method processes and aggregate alerts, identifies and compares sequences, and finds interesting sequences we present a series of metrics to demonstrate our findings. The total number of alerts after the preprocessing filtering step is reported as a baseline to compare against the number of attack activities determined; extracting a reduced and less overwhelming representation of the alert data. Then the number of unique IP addresses with high severity alerts is reported for each team signifying the team’s potential for being detected by a defender. Given the fixed network between teams, a lower number of IP’s with severe alerts may lead to the attackers being less likely of being detected but less comprehensive due to targeting less IP’s; where higher values may attract more attention from administrators. The number of attack activity sequences for each team also gives an idea of the attacker’s detectability as it relates directly to the number of high severity alerts reported and when compared to the number of IP’s targeted gives an estimate of the attacker’s persistence on a target. We theorize that a higher number of sequences than the number of targeted IP’s signifies the attacker targets the same IP address multiple times whereas a lower number of sequences signifies uses high severity actions multiple times on a target performs it in the same sequence.

The likelihood of these sequences are then computed against \( M_T \) and \( M_{all} \) to determine the number of similar and unique sequences. We identified three metrics using the sequence likelihoods to determine interesting sequences: longest similar sequences, most similar sequences, and most likely unique sequences. The sequences with of the longest length and a non-zero likelihood against all other teams is denoted as the longest similar subsequence which gives an insight into sequences that were similar to another team and unlikely by chance due to the length. The most similar subsequences are the sequences with the highest \( l_{alt} \) for each team, these are likely to be the most “uninteresting” sequences as they are common across multiple teams. Lastly, the sequence with the highest \( l_t \) and a zero \( l_{alt} \) reveal sequences that are unique to the team and common amongst the team, potentially a signature sequence of the team. These metrics for each team are shown in Table II demonstrating the numerous amount of alerts reduced into smaller and more interesting sequences, making it easier for analysts to extract the key information in the data.

For most teams, the number of attack activities is on an
order of magnitude lower compared to the number of alerts for the team. Team 4 is shown to be the most unique by a significant amount and upon manual analysis, Team 4’s behavior is heavily scripted and the alerts arrived in the same bin for nearly all the sequences making it difficult for this process to separate the activities even with a low $\Delta$. Other teams employed scripting techniques, however Team 4’s approach differed the most evident from their uniqueness. Features like the most similar sequence gives insight into sequences that have less interesting meaning to them but also shows that some vulnerabilities are easy to accomplish as many teams are performing the same process and possibly achieving a similar results. In this case the most similar sequences are mostly uninteresting combinations of categories 5, 15, and 18 signifying a scan and a web app attack are not indicative of individual behaviors. Whereas the longest similar sequence may reveal a less trivial sequence that others have performed that is unlikely due to chance (i.e. Team 6). Conversely, the most likely unique sequences for each team reveal sequences that were performed multiple times by a single team, indicating that these are signature processes for each team. Our work demonstrates that these pattern observations would not be easy to reveal without significant processing leveraging cyber-contexts.

To demonstrate how our process reveals patterns, we varied the size of the the Gaussian filter to measure: 1) the effect on the number of attack activities compared to the raw data; and 2) to evaluate the filters effect on the comparisons of sequences between teams. Fig. 2 shows the relationship between the filter size for both the number of attack activities and the ratio between the total number of similar and unique patterns determined.

As expected, there is a point at which the filter size is too large creating an increase in the number of attack activities evident in filter sizes around 140 minutes. Unexpectedly, the unfiltered and small filter sizes (>5 minutes) do not exhibit the uniqueness one would expect when comparing noisy and seemingly random signals and instead there is a high amount of similarities. Upon inspecting the sequences generated for the low filter sizes we had two observations: 1) the similar sequences were short, obvious, and not useful for an analyst; and 2) the unique sequences were long and due to noise are likely to be unique. An example of this is shown in Table III where the same sequence is compared before and after smoothing in which a similarity was then found. The un-smoothed sequence exhibits the undesirable repetitive behavior discussed earlier and it is determined to be unique given its length and possibly inconsistent order. Once smoothing is applied to the data, the sequence found is more concise and $l_{\text{u}}$ increased signifying that this is a pattern

### TABLE II

**SUMMARY OF THE OUTPUT FOR EACH TEAM WITH A FILTER SIZE OF 60 MINUTES**

<table>
<thead>
<tr>
<th>Team ID</th>
<th># of Alerts</th>
<th># of Attk. Activities</th>
<th># of severe seq.</th>
<th># of similar seq.</th>
<th># of unique seq.</th>
<th>Longest Similar Seq.</th>
<th>Most Similar Seq.</th>
<th>Most Likely Unique Seq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4274</td>
<td>504</td>
<td>21</td>
<td>20</td>
<td>11</td>
<td>9</td>
<td>[5, 15, 15, 5, 15, 18]</td>
<td>[13, 15, 5, 13, 15, 18]</td>
</tr>
<tr>
<td>2</td>
<td>2923</td>
<td>587</td>
<td>22</td>
<td>11</td>
<td>7</td>
<td>4</td>
<td>[5, 15, 13, 18]</td>
<td>[13, 15, 5, 15, 18]</td>
</tr>
<tr>
<td>3</td>
<td>3353</td>
<td>517</td>
<td>24</td>
<td>35</td>
<td>20</td>
<td>15</td>
<td>[15, 15, 5, 15, 18]</td>
<td>[15, 15, 13, 15, 18]</td>
</tr>
<tr>
<td>4</td>
<td>7801</td>
<td>1324</td>
<td>37</td>
<td>106</td>
<td>6</td>
<td>100</td>
<td>[5, 15, 15, 18]</td>
<td>[15, 12, 3]</td>
</tr>
<tr>
<td>5</td>
<td>1912</td>
<td>502</td>
<td>22</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>[5, 15, 15, 18]</td>
<td>[15, 15, 5, 15, 15, 18]</td>
</tr>
<tr>
<td>6</td>
<td>8413</td>
<td>760</td>
<td>30</td>
<td>62</td>
<td>19</td>
<td>43</td>
<td>[11, 19, 5, 15, 5, 18]</td>
<td>[15, 5, 3]</td>
</tr>
<tr>
<td>7</td>
<td>4712</td>
<td>715</td>
<td>21</td>
<td>31</td>
<td>19</td>
<td>12</td>
<td>[15, 15, 5, 3, 15, 18]</td>
<td>[5, 15, 5, 15, 18]</td>
</tr>
<tr>
<td>8</td>
<td>7150</td>
<td>838</td>
<td>40</td>
<td>68</td>
<td>20</td>
<td>48</td>
<td>[5, 15, 5, 5, 14]</td>
<td>[5, 15, 5, 15, 15, 15, 18]</td>
</tr>
<tr>
<td>9</td>
<td>2233</td>
<td>542</td>
<td>23</td>
<td>13</td>
<td>4</td>
<td>9</td>
<td>[5, 15, 5, 15, 18]</td>
<td>[5, 9, 15, 5, 5, 15, 15, 18]</td>
</tr>
<tr>
<td>10</td>
<td>5105</td>
<td>556</td>
<td>27</td>
<td>30</td>
<td>16</td>
<td>14</td>
<td>[5, 15, 5, 15, 5, 3]</td>
<td>[15, 5, 15, 18]</td>
</tr>
</tbody>
</table>
the team performs more often and a similarity to another team was found. The more concise representation of attack sequences exposes patterns that is easier to analyze and more meaningful. Table IV shows example sequences before and after the smoothing process was applied.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>(T)</th>
<th>(l_{all})</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Smoothing</td>
<td>5, 5, 5, 5, 5, 5, 5, 5, 5, 15, 12, 5, 3</td>
<td>(0.0046)</td>
</tr>
<tr>
<td>Smoothing</td>
<td>5, 15, 12, 3</td>
<td>(0.0075)</td>
</tr>
</tbody>
</table>

The sequences shown with no smoothing exhibit repeated alert activities and contain vague categories like potentially bad traffic and misc activity. The examples shown applying smoothing exposes teams 3, 6, 7, 10 all showing the same process of performing an information leak type attack (11), gaining access to a web app (19), and then attacking the web app (18). Where as team 8 was unique using protocol command decodes (10) to then plant a Trojan (0) and team 9 brute forced a web app attack which was sensed as a denial of service attack (4). It was found that the uniqueness found when no smoothing was applied was due to sequences being excessively long where the average length of the unique sequences with no smoothing was 11.0 and 6.7 for the smoothed case.

V. Conclusion

In this paper we have presented a method to aggregate similar and redundant alerts to create sequences of attack activities to better identify the similarities or uniqueness of attacker behaviors. We have shown examples of the reduced sequences discovered after our filtration process and we created a suffix-based model to calculate likelihoods of sequences occurring to determine similarities against attackers under the same conditions. Our process also exposed patterns of alerts that provided insight into how attackers performed tasks that were once hidden within noise. Our next steps include expanding the sequence generation to include features like alert signature, source ip, etc. to add more context into the sequences generated which can be related to an attack model like a kill chain. The suffix model can also be expanded on to allow for prediction of alert activities which can be used to simulate the attacker’s actions and behaviors using a cyber attack simulator.

References