Soteria: An Approach for Detecting Multi-Institution Attacks

Saif Zabarah*, Omar Naman*, Mohammad A. Salahuddin*, Raouf Boutaba*, Samer Al-Kiswany*#
*University of Waterloo, Canada
#Acronis Research, Canada

***Abstract*—We present Soteria, a data processing pipeline for detecting multi-institution attacks. Soteria uses a set of Machine Learning techniques to detect future attacks, predict their future targets, and ranks attacks based on their predicted severity. Our evaluation with real data from Canada wide academic institution networks shows that Soteria can predict future attacks with 95% recall rate, predict the next targets of an attack with 97% recall rate, and detect attacks in the first 20% of their life span. Soteria is deployed in production and is in use by tens of Canadian academic institutions that are part of the CANARIE IDS project.**

I. Introduction

Multi-institution attacks look for vulnerabilities at large number of nodes located at multiple institutions. These attacks cause significant financial loses and information leaks. For instance, the loss caused by NotPetya attack exceeds $10 billion [1], and the WannaCry ransomware attack affected more than 200,000 computers in 150 countries [2] causing millions in damages.

Defending against multi-institution attacks is complicated because the target nodes are managed by tens of independent security teams. Detecting these attacks requires timely information sharing between institutions and analysis of potential threats. This is further complicated by the following. First, the vulnerabilities an attacker can exploit continuously change making it harder to automate the defence mechanism. Worst yet, it may take months until vulnerabilities are patched. For instance, the WannaCry ransomware attack targeted a vulnerability in old Windows versions, for which a patch had been released more than two months before the attack [2]. Second, the attacks often happen in a short period of time. For instance, our data set (Section IV) shows that attacks can initiate millions of connections in just 24 hours. This short duration leaves little time for the cybersecurity personnel to detect, analyse, and deploy a defence mechanism. Third, large number of attacks happen at the same time. As the current process for analysing the attacks involves cybersecurity personnel, the number of attacks that can be inspected in time is limited. This prolongs the attack detection time and increases the time window in which an attack can cause damage.

For academic institutions, defending against multi-institution attacks is harder because they operate large and constantly changing networks (e.g., research projects or students spawning new nodes and services), and they have smaller budgets and cybersecurity teams. This makes academic institutions a prime target for attacks. For instance, in Canada, cybercrime caused an average of $9.25 million in losses per academic institution in 2019 [3] and 46% of the Canadian institutions reported a cybersecurity incident in 2017, which was the second highest impacted sector in Canada [4].

The current defence technique against these attacks is inadequate. The main approach relies on sharing intelligence between cooperating institutions and using public databases that list recent IoCs. This approach is slow to detect an attack and the information shared is often limited due to regulatory and privacy policies.

We present Soteria, a novel data processing pipeline for detecting multi-institution attacks. Soteria overcomes the shortcomings of the current defence approach. Soteria collects minimum information from cooperating institutions, mainly information about connections to an institution. It then uses a novel combination of machine learning (ML) techniques to detect current attacks and predict future attacks. Soteria also predicts the next targets of an attack and identifies the larger-scale attacks (i.e., the more severe attacks). These findings help focus each institution's limited resources on the most severe attacks that are targeting them now or in the near future.

Soteria is carefully designed to be able to scale to hundreds of institutions and detect attacks in a timely manner. Soteria uses graph analysis to extract features, linear regression to detect future attacks, and time series analysis to predict the next targets of an attack. We use a bidirectional long short-term memory recurrent neural network with attention mechanism (ABiLSTM) to predict the future targets of an attack. Finally, to predict the severity of an attack we capture static and dynamic features of the generated graphs and use normalization and reduction techniques to compute a severity indicator.

Through our study of the dataset and the exploration of different techniques we learnt a number of insights. We found that to accurately predict the next target of an attack, the used mechanism should capture: 1) The relationships between institutions, as institutions with similar characteristics (e.g., institution size, services offered, and security posture) are usually targeted together. 2) The sequence of the attack; 3) The level of activity of an attacker. One would expect that ML techniques that predict events that occur together, such as a co-occurrence matrix [5], to be efficient in detecting Multi-institution Attackers (MIA). Surprisingly, based on our experiments, these techniques are not efficient; this is because a co-

occurrence matrix does not capture the sequence or the level of activity of an attack. Techniques, such as a unidirectional LSTM which predict a sequence of events performed better, but did not achieve high accuracy in predicting the next target because attacks do not follow the same sequence in every attack incident.

It was interesting that a simple linear regression model over the right set of features is highly effective in detecting an attack and nodes that contact more than one institution are, with high probability, initiating an attack. Surprisingly, using a short history of recent connections detects attacks faster than when using the data from the last 24 hours.

Soteria has been deployed in production for the last year as part of the Canadian Network for the Advancement of Research, Industry and Education (CANARIE) Intrusion Detection System (IDS) program. CANARIE [6] is a Canada wide backbone network connecting academic institutions. The CANARIE IDS is serving over 100 institutions in Canada. Over the last year Soteria has identified numerous severe multi-institution attacks.

Our evaluation with real data from the CANARIE network shows that external IPs communicating with more than one institution are 95% likely to be a multi-institution attacker. Our evaluation also shows that Soteria detects future attacks with up to 95% recall rate and within the first 20% of the attack's lifetime. Finally, Soteria detects and notifies 97% of institutions that will be targeted in the future before the attacker initiates a connection to that institution.

The rest of this paper is organized as follows. In Section II we survey related work. In Section III we detail the design of Soteria. In Section IV we evaluate the accuracy of predicting a future attack and the future targets of an attack. We present our concluding remarks in Section V.

II. Related Work

Reconnaissance Detection. Reconnaissance attacks try to scan systems looking for vulnerabilities. Previous efforts on reconnaissance detection are limited to detecting port scans. For instance, Udhayan et al. [7] detect port scanning attempts by applying a set of heuristics on the connection timing and TCP header fields. Allen et al. [8] note that port scanning tools leave detectable features in the generated requests. For instance, they may contain invalid data or header information. Given the short list of scanning tools, they explore inspecting packets for special markers to identify port scanning attempts.
Heavy Hitters Detection. Previous efforts attempted to detect large scale attacks known as heavy-hitter attacks. Heavy-hitters communicate with an unusually large number of hosts. The main challenge for detecting heavy hitters is handling large amounts of data. Previous efforts [9, 10] resorted to filtering out low cardinality hosts and sampling. Yang et al. [11] summarize traffic measurements by using mergeable data structures and aggregating the summaries at the operator center. The merged summary is used to detect heavy hitters.
Intrusion Detection Systems. IDSs monitors network traffic to detect malicious activities. Examples of IDS systems are ZEEK [12], Snort [13], and Suricata [14]. IDS is primarily for detecting attacker in a single institution, mostly based on network rules and policies. Soteria targets MIA attacks using ZEEK connection logs.
Current Approach for Detecting Multi-Institutional Attacks. The main approach for handling multi-institution attacks nowadays is through sharing attack intelligence between institutions [15]. Institutions may share knowledge, tools, operational details, and details about active attacks. The shared intelligence can come from peer institutions or from public databases of attacks. A number of databases offer information about known attacks and list malicious IPs, such as AbuseIPDB [16] and VirusTotal [17].

Unfortunately, while helpful, this approach does not adequately protect against multi-institution attacks. That is because this approach is often slow in responding to active attacks, the systems are large and continuously changing, and there is continuously a large number of attack attempts which overwhelms institution staff.

Soteria aims to overcome the shortcomings of the previous techniques. It relies on minimal information shared by institutions, namely only information about incoming connections to institutions. It uses this data to automatically detect multi-institution attacks. To help prioritize the analysis of potential attacks, Soteria identifies the most severe attacks. Further, it identifies the next potential targets for the attack. This information is used to notify the targeted institution of the most severe attacks before the attack reaches that institution.

III. System Design

Soteria data processing pipeline is organized into five stages: (i) feature extraction, (ii) attack detection, (iii) severity estimation, (iv) next target detection, and (v) report generation. Figure 1 shows the Soteria system architecture.

Institutions periodically submit logs of the recent communication activities on their networks. The feature extraction step (Figure 1) analyzes the data to extract a vector of features. The extracted features are leveraged by the attack detection step that uses an ML model to predict potential attacks. The ML model also outputs additional metrics to help with the next two steps.

The severity estimation step uses the metrics calculated in the previous steps to estimate the severity of the predicted attacks. This step helps identify severe attacks. The next target prediction step uses deep learning to identify the next targets

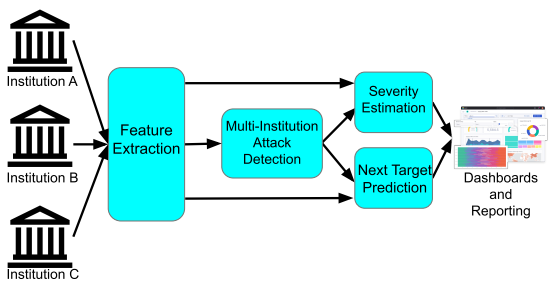


Fig. 1: Soteria Pipeline.

of the predicted attacks. Finally, Soteria combines the results of the attack detection, severity estimation, and next target prediction steps into user reports. The rest of this section details the design of each step.

A. Institutional Data

Sharing connection and infrastructure information between institutions is complicated due to regulatory restrictions and privacy-related concerns. This is the case for the academic institutions connected to CANARIE [6]. The institutions periodically share connections logs collected by ZEEK, a network security monitoring tool. In addition to ZEEK connection logs, each institution identifies the IP addresses it owns. We present the details of the data set we use in Section IV.

Each row in a ZEEK connection log lists information about a connection. In our work, we use three fields for each connection:

- $id.orig\_h$: IP address of the node starting the connection.
- $id.resp\_h$: IP address of the node responding to the connection.
- ts: the time stamp when the connection occurred.

B. Feature Extraction

To identify attacks on multiple institutions, we build a directed and weighted graph representing all the connections in the ZEEK connection logs. Each IP address represents a vertex. We add a directed edge from a source to a destination between two vertices that had one or more connections. Each edge has a weight. The weight is equal to the number of connections with the same direction between the two vertices. Vertices are labeled as internal vertex if they belong to an institution or external vertex otherwise. Unfortunately, this approach for generating a graph creates enormous graphs that are challenging to analyse in a timely manner. For instance, for our data set this approach resulted in an enormous graph with over 26.5 million vertices and 1.4 billion edges.

To reduce the graph size without losing information relevant to the attack we do the following. First, we remove all edges representing a connection that is initiated by an internal vertex because those vertices are trusted. Second, we represent each institution by a single aggregate vertex and remove all its internal vertices. The aggregate vertex represents all the IP addresses belonging to an institution. We add a directed edge from an external vertex to an aggregate vertex if the external vertex have contacted any of the internal address of that institution. Each edge has two weights: the number of connections the external vertex initiated to any of the internal vertices, i.e., `conn_count`, and the count of unique internal vertices the external vertex is connected to, i.e., `vert_count`. These steps significantly reduce the graph size.

In the feature extraction step (Figure 1) we compute the following for every external vertex:

- Outdegree (OD): the number of edges that begin from this specific vertex. For an external vertex, this equals the number of institutions it communicated with.

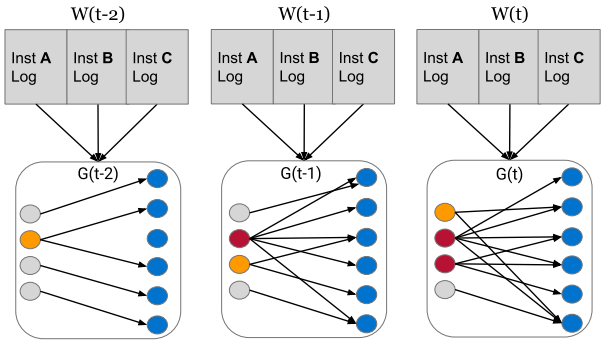


Fig. 2: Three graphs created over 3 consecutive time windows. Grey: Benign External IP; Orange: Early detection of MIA; Red: MIA.

- Outdegree weighted by number of connections ($ODW(connection)$): the summation of all the `conn_count` weights of all the outgoing connections of an external vertex.
- Outdegree weighted by number of vertices ($ODW(ip)$): the summation of all the `vert_count` weights of all the connections of an external vertex.
- Adjacency list ($V(adj)$): the adjacency list associates each external vertex with the collection of its neighboring institution vertices.

C. Tracking External IPs over time

Institutions periodically share their connection logs. To analyse the activities of external vertices over time we discretize the logs. We divide the time into windows. Each window is l hours long. We analyse the connections of each window separately. For each window $W(t)$ at time t we create a graph $G(t)$ and extract the four features as described in the previous section.

In Soteria we track the features of each external vertex of the latest N windows. N and l are configurable and we evaluate the efficiency of our approach while varying these two parameters in Section IV. Figure 2 shows an example of collecting logs from three institutions and dividing the time into three windows. A graph is built for each window.

We track the adjacency list ($V(adj)$) of an external IP starting from the window it becomes active even if this extends to more than the N latest windows. To avoid analysing a previous time window graph, we compute the cumulative adjacency list $V(cumltv)$ at t for external vertex V_x as follows:

$$V_{xt}(cumltv) = \begin{cases} V_{xt}(adj) & if\ t = 0 \\ V_{x(t-1)}(cumltv) \cup V_{xt}(adj) & if\ t > 0 \end{cases} \quad (1)$$

where $t = 0$ is the first window V_x becomes active. $V_{xt}(adj)$ is the adjacency list of V_x at time t. $|V_{xt}(cumltv)|$ is the count of all institutions contacted by V_x since it began.

For each external IP, we track OD, $ODW(connection)$, $ODW(ip)$, and $|V_{xt}(cumltv)|$ of the latest N windows in a $N \times 4$ matrix.

D. Detecting Multi-Institution Attacks

We found that linear regression is effective in identifying vertices that will launch a multi-institution attack. Linear regression fits the data to a straight line that relates one independent variable t to a dependent variable Y.

$$F(t) = Y = B + tS, \tag{2}$$

where t is the time window and Y is the feature of interest. $F(t)$ is the linear regression function that approximates Y. B is the value of $F(t)$ at $t = 0$ and S is the slope. S approximates the growth rate of a feature.

For each V_x, we keep track of OD, $ODW(connection)$, $ODW(ip)$, and $|V_{xt}(cumltv)|$ over N windows. For each V_x we fit each feature to a linear regression line. In doing so we can predict the future outcome of each feature. To fit the linear regression line we minimize the Residual Sum of Squares (RSS); a measure of the discrepancy between the data and the linear regression function.

$$RSS = \sum_{t=1}^{N} (Y(t) - F(t))^2 \tag{3}$$

We identify an IP as a MIA if it contacts $p(inst)$ institution or more. We set $p(inst)$ to 3 in our study. To predict if an IP V_x will become an MIA we predict if V_x will contact more than $p(inst)$ of institutions in the current or a future time window. To identify a current or future MIA we use:

$$V_x \; is \begin{cases} MIA & if \; |V_{xt}(cumltv)| \geq p(inst) \; or \\ & F_{|V_x(cumltv)|}(t+n) + k \geq p(inst) \\ not \; MIA & otherwise \end{cases} \tag{4}$$

where $F_{|V_x(cumltv)|}(t+n)$ is the predicted number of institutions V_x will contact by the time window $t+n$. k is a constant that is used to tune the prediction.

In addition to predicting which node will become an MIA, we use linear regression to compute the slopes or the growth of the features for each external vertex. The growth of these features is used in the following steps in the Soteria pipeline.

- $V_{xt}(cumltv)$: The cumulative set of institutions V_x contacted throughout its lifetime till time t.
- $S_{|V_{xt}(cumltv)|}$: Growth of the cumulative number of institutions V_x connects to throughout its lifetime till time t.
- S_{OD_x}: Growth of the number of institutions connected to V_x per time window.
- $S_{ODW(connection)_x}$: Growth of the number of outgoing connections of V_x per time window.
- $S_{ODW(ip)_x}$: Growth of the number of internal IPs communicated with per time window.

E. Severity Estimation

The attack detection step may identify hundreds of potential MIAs. Tasking security analyst to analyse these potential attacks in a timely manner is a daunting task. The severity estimation step computes a severity indicator for each potential MIA and uses it to identify the most severe MIAs. This step helps the security analyst to prioritize analysing attacks with high severity indicators.

The severity indicator uses all the static and growth features computed in the previous two steps of the pipeline. The severity estimation technique should have three properties. First, given the large number of external IPs, the severity indicator should be efficient to compute. Second, it should maintain the linearity of each feature, i.e., if feature X for IP1 is larger than X for IP2, this relation should be represented in the severity estimation mechanism. Third, it should tolerate highly skewed data.

We first considered normalizing each one of the seven features (three static and four growth) to the range $[0, 1]$. Adjacency list $V(adj)$ and cumulative adjacency list $V_{xt}(cumltv)$ are not included in the calculation but the size of these lists are included. The classical normalization approach of a feature X is:

$$X_{norm\_V} = \frac{X_V - X.min}{X.max - X.min} \tag{5}$$

where X_V is the value of a feature X for IP V and X_{norm_V} is the normalized value of X_V. $X.min$ is the smallest value for the feature among all IPs in the data set. $X.max$ is the largest value for X in the data set. While this approach is simple, it is not effective. This is because this approach does not handle highly skewed data well. For instance, the majority of attackers would create hundreds of connections, while an aggressive attacker may create hundreds of thousands of attacks which significantly skews $X.max$ and stretches the bounds of normalization. This causes the majority of values to be placed on the lower end of the normalized range and makes it hard to differentiate between attacks since the normalized values are very close.

To overcome this shortcoming, we use a robust scaler, as shown below:

$$X_{norm\_V} = \begin{cases} 0 & X_V < X.Q_1 \\ 1 & X_V > X.Q_3 \\ \frac{X_V - X.Q_1}{X.Q_3 - X.Q_1} & otherwise \end{cases} \tag{6}$$

where X_V is the value of a feature X for IP V and X_{norm_V} is the normalized value of X_V. The robust scaler finds the quartile for each feature X. $X.Q_1$ and $X.Q_3$ are the values of the first and third quartiles. The employeed Robust Scaler normalizes the skewed values less than the first and greater than the third quartiles to the values 0 and 1, respectively, then it normalizes the values between the first and third quartiles to the range $[0, 1]$. This approach effectively handles highly skewed data.

The normalized values of all features per external IP are added. To give an indicator with values in the range of $[0, 1]$, the aggregated value is then normalized. This approach effectively computes the severity indicator, preserves the linearity of the features, and handles skewed values.

F. Predicting Future Targets

In the next step we try to predict, for each MIA, which institution will be targeted next. Our first attempt to predict

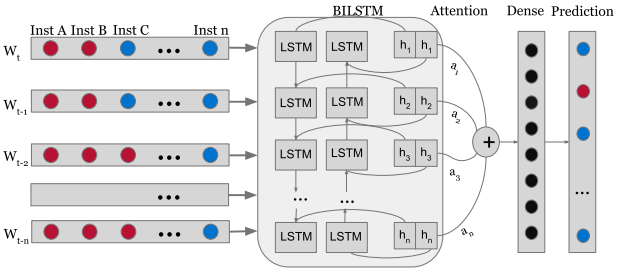


Fig. 3: Design of the model for next target prediction.

the future targets of each attack explored techniques to predict which institutions are often targeted together. We experiment with the co-occurrence matrix model which is successfully used in recommendation systems to predict the items that occur together. Our results show (Section IV) this approach is not effective in predicting future targets. This is because Co-occurrence matrix can only capture the relationship between institutions and does not capture the sequence of the attack, neither does it capture the future possible growth of the attack.

We also explored using the Long Short Time Memory (LSTM) model [18]. LSTM is effective in predicting sequences of events. This approach captures the growth of an attack and uses it to predict the next targets. This approach achieved better results than the co-occurrence matrix model but did not achieve high accuracy. This is because attackers do not always attack institutions in the same order and we need it to learn slight variations to these sequences. To overcome this challenge, we resorted to using bidirectional LSTM with an attention mechanism (ABiLSTM). ABiLSTM learns a sequence of events in both directions, forward and backward, to better predict targets despite variations in the order in which institutions are attacked. It also better capture relationships between institutions in a specific window and across time windows.

Model Design. Figure 3 shows the network structure of the ABiLSTM model we use. The model has the following stages: input encoding, BiLSTM network, attention mechanism, and an output layer. The rest of this subsection details the design of each of these stages. The presentation in the rest of this subsection is geared toward readers versed in ML methodology and can be skipped without loss of context.

Input Encoding. We first encode $V(cumltv)$ for each external IP using multi-hot encoding. For each time window, we create a bit map of size M which is the total number of institutions. An index in the bitmap corresponds to an institution. A bit is set if the institution has ever been contacted by this IP address. Since we look into the last N time windows, the input to the model is an $M \times N$ array representing $V(cumltv)$ in the last N time windows.

BiLSTM Model. Figure 3 shows the structure of the ABiLSTM model. The BiLSTM model uses an $M \times 2$ array of LSTM cells organized in M pairs. One LSTM in a pair learns the forward direction of a sequence while the other learns the backward direction. The output of the two LSTM blocks is concatenated. The pairs are organized in a stack (Figure 3). Each LSTM cell has three gates: input gate (i_t), output gate (o_t), and forget gate (f_t), where t is the window timestamp.

Equation 7 shows the input gate of a cell while Equation 8 generates a candidate vector. The combination of the input gate and the candidate vector controls the information stored in a cell at the current time window t.

$$i_t = \sigma(Z_i[h_{t-1}, x_t] + b_i), \tag{7}$$

$$\check{C}_t = \tanh(Z_c[h_{t-1}, h_t, x_t] + b_c), \tag{8}$$

where x_t represents the input of that cell at time t, h_t represents the output of the cell at time t and position m in the LSTM stack, h_{t-1} represents the outputs value one time step before the current time and $h_{t(m-1)}$ with the output value of the cell underneath it. Similarly, c_t and c_{t-1} represent the memory unit at time t and $t-1$. σ represents the sigmoid activation function, and $tanh$ represents the tangent function. Z_i and Z_c represent the weight matrices, and b_i and b_c are the bias values.

To decide whether to discard information from the previous time step and from the lower LSTM block a forget gate is used as shown below:

$$f_t = \sigma(Z_f[h_{t-1}, h_t, x_t] + b_f). \tag{9}$$

The memory value for this time step is calculated using Equation 10. Note that we use the memory value c_{t-1} from the previous time step.

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \check{C}_t \tag{10}$$

The output gate determines the value of the next hidden state. This state contains information on previous inputs. First, the output gate uses a sigmoid function to decide which portion of a cell state to return (Equation 10). We take the output of the output gate and perform the hadamard product ($\otimes$) with the output of the $tanh$ function of the memory value.

$$o_t = \sigma(Z_o[h_{t-1}, h_t, x_t] + b_o) \tag{11}$$

$$h_t = o_t \otimes \tanh(c_t) \tag{12}$$

h_t is the hidden state of the cell which is shared with the next layer of the model and the next LSTM cell. The output of the BiLSTM model will be a concatenation of the outputs of both direction models, which will hence forth be denoted as h_t.

Attention Layer. The BiLSTM hidden layer outputs h_t through the activation function to obtain the correlation coefficient u_t using Equation 13.

$$u_t = \tanh(Z_a h_t + b_a) \tag{13}$$

Where Z_a represents the weight matrix, and b_a represents the bias values. First, we assign weights that demonstrate the

importance of each output of the hidden layer by obtaining the weight coefficient a_t:

$$a_t = \frac{exp(u_t)}{\sum_{j=1}^{N} exp(u_j)} \tag{14}$$

Then we calculate the product of the weight coefficient and the output of the hidden layer to obtain the output vector v of the attention layer, as shown in Equation 15.

$$v = \sum_{N} a_t h_t \tag{15}$$

Dense and Output Layer. Finally, the prediction result is obtained through the output layer using a sigmoid function. The output layer contains as many neurons as there are institutions, each will produce an output for an institution. The model outputs the probability that each institution will be targeted. Because this is a classification problem we need to convert probabilities into binary values. Therefore, we can simply round the probabilities into integers using a threshold. This threshold is tuned to provide better predictions. This gives us a multi-hot encoding vector, similar to the input matrix. We reverse the encoding we did on the input which gives us a list of institutions that are most likely to be targeted next.

G. Dashboards and Reporting

The last step of the pipeline creates a dashboard and reports to present the findings to the security analysts. All the static and growth features as well as the severity metric are presented to the institutions. The severity metric is used to order the external IPs. The display of the list of MIAs is customised for each institution. The list of MIAs is split into three groups: a list of MIAs already targeting institutions, a list of MIAs that are predicted to attack in the near future, and the rest of the MIAs.

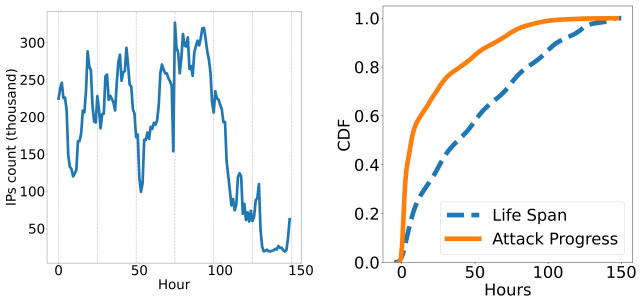
IV. Evaluation

In our evaluation, we evaluate the accuracy of Soteria in detecting future attacks, its accuracy in identifying the next targets, and the impact different configurations have on the detection performance.

A. Evaluation Setup

Dataset Details. Our dataset includes the ZEEK connection logs from 52 Canadian institutions connected to the CANARIE backbone network. We use the data collected over six days between the 25th and 30th of January 2022. The dataset consists of over 15.5 billion connections.

There were over 12 million unique external IP addresses initiating a connection to any of the institutions during the six days. Out of the 12 million, 2.7 million of them initiated connections to multiple institutions. The number of connections from external IP addresses per day fluctuates across the six days with noticeable drops during the weekends. The drop could be interpreted as a result of the institutions being less active during the weekends. Figure 4a shows the number of connections per hour.



(a) Number of connections per hour during the six days.

(b) CDF of the lifespan of an External IP and the attack progress.

Fig. 4: Life span of external IPs.

Each external IP is tracked from the moment it starts it first connection to an institution until it stops communicating with any institution. For external IPs that communicate with multiple institutions, Figure 4.b shows the life span and attack progress of external IPs. Life span is the total time the external IP was active. It is the time period between its first connection and last connection that IP made in the data set. The life span of 70% of the external IPs is less than 3 days. Fifty percent of the external IPs had a life span of less than one day. For each external IP we extract the full list of institutions it contacts. The attack progress line in Figure 4b shows the progress an external IP makes in contacting institutions in this list. The figure shows that an attacker contacts 70% of its target list within the first 24 hours.

Data Preprocessing. To simulate a stream of updates from institutions we split the connections into time windows. Given that a large percentage of attacks complete within 24 hours, we vary the window size l from 1, 3, 6, and 12 hours. Each window contains all the connections for that given time period. To simulate a real workload we split the data and then feed the data to the pipeline for analysis.

When we train the model we use multiple windows as input. Unless otherwise specified we use the current windows and 2 previous windows as input.

B. Labeling Multi-Institution Attackers

Unfortunately, the attackers in our dataset are not labeled. We attempted to utilize public databases to label our data but we found them unreliable as there is a high degree of false positives (i.e., an IP is recycled and is no longer malicious such as the case with malicious actors using cloud services) and false negatives (i.e., threat has not been reported by anyone). To overcome this shortcoming of our dataset we select 387 random samples of external IP addresses that contacted 3 or more institutions. We manually inspect the logs and interactions with each one of the IP addresses and identify MIAs. We found 369 real multi-institution attacks and 18 benign IP addresses. This indicates that 95% of external IP addresses that contact multiple institutions are malicious actors with a 95% confidence interval and a margin of error of 5%. In this paper, if an external IP address contacts more than 3 institutions we label it as MIA.

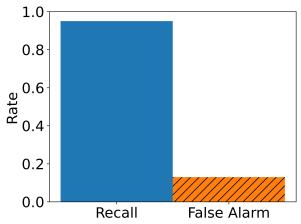


Fig. 5: Recall and false alarm of detecting MIA.

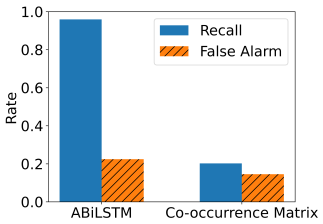


Fig. 6: Recall and false alarm of detecting next targets.

Metrics. We use the following two metrics in our evaluation: recall, which gauges how many of the true positive we have detected, and false alarm, which gauges how many of the true negatives we have misclassified as positives.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \quad (16)$$

$$FalseAlarm = \frac{FalsePositives}{TrueNegatives + FalsePositives} \quad (17)$$

Testbed. We conduct our experiments using a 17-node cluster. Sixteen nodes are used to ingest data and extract features. These nodes have an Intel(R) Xeon(R) Silver 4208 CPU with 32 cores, 188GB of RAM, and 48TB of storage space. One node is used to run the rest of Soteria pipeline. The node has an Intel(R) Xeon(R) Gold 5120 CPU with 56 cores, 376 GB of RAM, and a NVIDIA Tesla P40 GPU.

C. Detection of Future Multi-Institution Attacker

Using our dataset we measure the accuracy of our MIA detection step. For this evaluation, we use a window size l of 3 hours and we use a history N of 3 previous windows, and try to predict if an IP address will become a MIA in the next 24 hours. Figure 5 shows the recall and false alarm of our attack detection step. The figure shows that our linear-regression-based technique detected more than 95% of attacks with lower than 15% false alarms. All false alarms have a severity level of less than 25%, which are presented last in the list of threats to search.

D. Predicting the Next Target

We use our dataset to measure the accuracy of predicting the next target. We use a window size l of 3 hours and use a sequence N of 3 windows. Figure 6 shows the recall and false alarm of the next target prediction step. We compare the performance of using ABiLSTM and co-occurrence matrix. Figure 6 shows that our approach with ABiLSTM achieves 4.7 times higher recall rate. ABiLSTM achieves 95% recall rate with 20% false alarm rate while the co-occurrence matrix achieves only 20% recall rate with 15% false alarm rate. This asserts our previous discussion that BiLSTM's supersedes co-occurence matrix due to its added ability to learn data sequences and future growth of attacker.

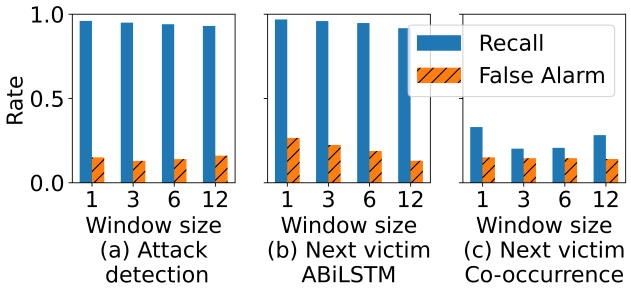


Fig. 7: Performance using three look back windows.

E. Effect of Window Size

In this section, we evaluate the effect of window size l on the accuracy and the speed the detection. We fix the number of windows N to 3 and vary the window size l between 1, 3, 6, and 12 hours. Figure 7.a shows the recall and false alarm rate for identifying future attacks. The results show a slight variation in the recall rate with smaller window sizes having better recall rates. Due to the shorter lifetime of the attackers and their rapid attacking rates, smaller windows are able to capture this type of behaviour. For instance, window size of 1 achieves 97% recall rate compared to 92% recall rate for window size of 12. There is no significant change in the false alarm rate. Figure 7.b and 7.c evaluates the effect of window size on the accuracy of predicting the next target. We evaluate both ABiLSTM and co-occurrence matrix. The figures show that changing the windows size does not significantly change the recall or the false alarm rate of ABiLSTM. For co-occurrence matrix, changing the window size changes the recall rate with the best being with a window size of 1, achieving 26% and the worst being 20% with a window size of 3. The results show under all window sizes ABiLSTM achieves 3.5 to 7 times higher recall rate without a significant increase in false alarm rate.

F. Effect of the Number of Windows

In the previous section, we kept the number of windows N fixed but varied the window size l. This results in each configuration processing a variable size of history. The number of connections in 3 windows of a 1 hour window size is much smaller than 3 windows of 12 hours windows size. In this section, we set the look back time to 24 hours, regardless of the window size. We use 24 windows with 1-hour long windows, 8 windows with a 3-hours window size, 4 windows with a 6-hours window size, and 2 windows of a 12-hour window size.

Figure 8.a shows the recall and false alarm rate for identifying future attacks. The results do not show a noticeable change in the recall or the false alarm rate with different window sizes.

Figures 8.b and 8.c evaluates the effect of window size on the accuracy of predicting the next target using both ABiLSTM and co-occurrence matrix. Similar to the previous results the figures show that changing the window size while using the history size does not bring significant change to the recall or false alarm of these techniques.

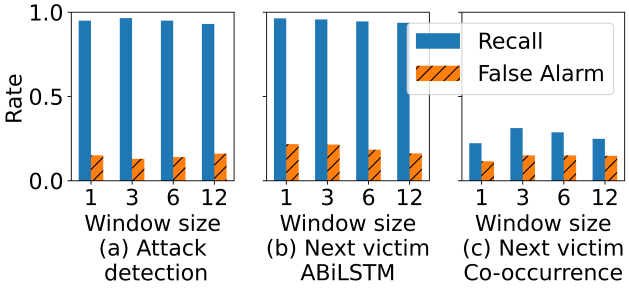


Fig. 8: Performance while looking back for 24 hours.

Interestingly, there is no noticeable change in results for both MIA detection and path prediction between fixing the number N to 3 or fixing the look-back time to 24 hours.

In general, we see slight improvement in recall with smaller number of windows and with smaller window sizes, which we attribute to the quick nature of these attacks. The performance gap between the smaller and larger windows is not large and that is because our comparison so far compares performance of predicting future attacks. This comparison does not highlight that larger windows are unable to capture attack progress as well as smaller windows. We evaluate the utility of different configurations in the following subsections.

G. Speed of Attack Detection

We evaluate how early our technique can detect an attack. We analyse the dataset and identify for each MIA the complete list of institutions it will attack. Figure 9 shows a box plot of the percentage of the MIA life at which Soteria detects the attack. We compare two configurations: windows size of 3 with a fixed number of windows of 3, and a windows size of 3 with a total look-back period of 24 hours. Figure 9 shows that using smaller window sizes allows for predicting the attack earlier. With a window size of 1 hour detecting the attacks before 20-40% of its life span compared to 75-85% with a window size of 12. Surprisingly using a fixed number of windows achieves better results. With a window size of 1, using 3 windows the attack is detected at around 20% of its life span while when using 24 hours the attack is detected when it is around 40% of its life span on average. This is because smaller windows help to detect an attack earlier, and a smaller number of windows speeds up the detection step. Under all window sizes using a fixed number of windows performs better on average.

V. Conclusion

We present Soteria a data processing pipeline for detecting multi-institution attacks. Soteria can detect current and future multi-institution attacks, rate the severity of the attacks, and predicts its future targets. Our evaluation shows that Soteria is able to identify future attacks and identify their future targets with high accuracy. Soteria is currently deployed in production as part of CANARIE IDS.

In the future, we plan to explore two directions. First, Tor nodes are often used by multiple clients. This complicates detecting MIAs. We plan to explore techniques to identify attacks originating from a Tor node. Second, an attacker can use multiple IPs, we plan to explore techniques to identify MIA using multiple IPs.

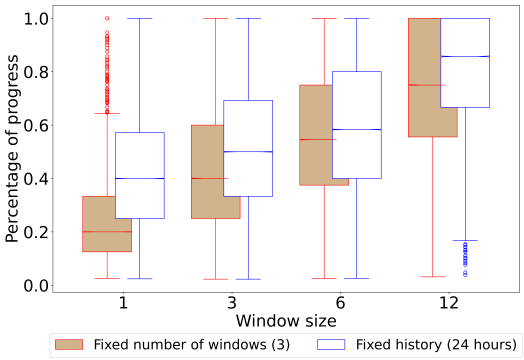


Fig. 9: The attack detection speed. The box plot shows when an attack is detected during its life span.

References

[1] Government Accountability Office. Cyber insurance—insurers and policyholders face challenges in an evolving market, 2021. https://www.gao.gov/assets/gao-21-477.pdf, Accessed: Jan. 2023.

[2] M. Akbanov and V. Vassilakis. Wannacry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms. *Journal of Telecommunications and Information Tech.*, 1:113–124, 2019.

[3] Accenture Security. Ninth annual cost of cybercrime study, 2021. https://www.digitalmarketingcommunity.com/researches/ninth-annual-cost-of-cybercrime-research-2019, Accessed: Jan. 2023.

[4] H. Bilodeau and M. Lari, M. Uhrbach. Cyber security and cybercrime challenges of canadian businesses in 2017, 2019. https://www150.statcan.gc.ca/n1/pub/85-002-x/2019001/article/00006-eng.htm, Accessed: Jan. 2023.

[5] T. Dunning and E. Friedman. In *Practical Machine Learning: Innovations in Recommendation*. O'Reilly, 2014.

[6] Canarie.ca, 2022. https://www.canarie.ca/, Accessed: Jan. 2023.

[7] J. Udhayan, M. Prabu, V. Krishnan, and R. Anitha. Reconnaissance scan detection heuristics to disrupt the preattack information gathering. In *International Conference on Network and Service Security*, 2009.

[8] W.H. Allen, G.A. Marin, and L.A. Rivera. Automated detection of malicious reconnaissance to enhance network security. In *Proceedings. IEEE SoutheastCon, 2005.*, pages 450–454, 2005.

[9] J. Cao, Y. Jin, A. Chen, T. Bu, and Z.-L. Zhang. Identifying high cardinality internet hosts. In *IEEE INFOCOM 2009*, 2009.

[10] N. Kamiyama and R. Mori, T. Kawahara. Simple and adaptive identification of superspreaders by flow sampling. In *IEEE INFOCOM*, 2007.

[11] Y. Liu, W. Chen, and Y. Guan. Identifying high-cardinality hosts from network-wide traffic measurements. *IEEE Transactions on Dependable and Secure Computing*, 13(5):547–558, 2016.

[12] The Zeek Project. conn.log - book of zeek, 2022. https://docs.zeek.org/en/master/logs/conn.html, Accessed: Jan. 2023.

[13] Cloud Cisco: Networking and Cybersecurity Solutions. Snort, 2022. https://www.snort.org, Accessed: Jan. 2023.

[14] The Open Information Security Foundation (OISF). Suricata, 2022. https://www.suricata.io/, Accessed: Jan. 2023.

[15] B. Feng. Threat intelligence sharing: What kind of intelligence to share?, 2021. https://www.concordia-h2020.eu/blog-post/threat-intelligence-sharing/ , Accessed: Jan. 2023.

[16] Marathon Studios Inc. Abuseipdb - ip address abuse reports, 2016. https://www.abuseipdb.com/, Accessed: Jan. 2023.

[17] virustotal.com, 2004. https://www.virustotal.com/, Accessed: Jan. 2023.

[18] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.