Online Symbolic Learning of Policies for Explainable Security

Arthur Drozdov*, Mark Law†, Jorge Lobo‡, Alessandra Russo*, and Mercion Wilathgamuwage Don§
*Imperial College London, United Kingdom
arthur@drozdov.net, a.russo@imperial.ac.uk
†ILASP Limited, Grantham, United Kingdom
mark@ilasp.com
‡Universitat Pompeo Fabra & ICREA, Barcelona, Spain
jorge.lobo@upf.edu
§Cisco Systems, Oslo, Norway
mwilathg@cisco.com

***Abstract*—Statistical Machine Learning (ML) has been proved to be an invaluable tool in many areas including privacy and security. On the other hand, recent advances in the field of Symbolic Learning have included novel scalable algorithms that learn highly accurate classifiers encoded as logic programs. In this paper we advocate adding Symbolic Learning to the security and privacy ML toolset. Through an example in anomaly detection, we present a framework for developing systems capable of performing symbolic-based learning of security policies. Our framework, called Online Learning of Anomaly detection Policies from Historical data (OLAPH), uses a symbolic learning system and a domain-specific function for scoring candidate rules to guide the learning process towards the best policies for anomaly detection. The learned policies are fully explainable since the underlying symbolic learning system is inherently explainable: there is a one-to-one mapping between the learned symbolic rules and the anomaly detection policies. The online feature of OLAPH uses a notion of policy confidence to decide when to relearn the policy and what data to relearn the policy from. OLAPH has been evaluated on a dataset of network requests from a commercial security provider, and shown to have a strong anomaly detection performance in addition to the usability and explainability benefits induced by its symbolic learning approach.**

***Keywords*-online learning, symbolic learning, explainable artificial intelligence, anomaly detection, security.**

I. Introduction

In light of the scientific and technological advances in machine learning, and in particular, artificial neural computation, researchers and practitioners in many fields, including privacy and security, have been reevaluating their methods and approaches to incorporate machine learning techniques into their problem solving toolsets. There has been an explosion of research papers reporting on the application of neural networks and deep learning to address security issues that range from network security, to software reliability and access control, just to name a few [1]. However, the general application of deep and other statistical learning methods still faces several challenges. They require large amounts of training data which can be a significant impediment for privacy and security applications. Evolution of the systems to be protected can make the data used for training, and therefore the trained security system, obsolete. Mistakes made by the trained security systems can be difficult to detect and explain. On the other hand, recent advances in the field of Symbolic Learning have seen the development of novel and scalable algorithms and systems that can learn highly accurate classifiers from a limited amount of training data, which can be explained to and checked by humans [2, 3]. These systems are particularly suited to address the shortcomings of deep and statistical learning methods.

In this paper we show, through a fully developed system for anomaly detection, how Symbolic Learning can be used to generate explainable policies in a data-efficient manner. Specifically we propose a general framework, called *Online Learning of Anomaly detection Policies from Historical data* (OLAPH) that uses a Symbolic Learning system [3] to learn policies for anomaly detection from online data. Anomaly detection is a very practical problem and it is at the core of many Cisco network security products. A common process for network anomaly detection is shown in Figure 1.

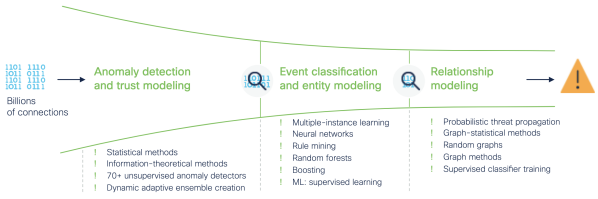

Fig. 1: Cisco Secure Network Analytics funnel for classification [4]

The definition of an anomaly requires expert domain knowledge to create and maintain. This overhead encourages the adoption of techniques that can learn anomalies from historical data. But, instead of using deep learning methods that suffer of the aforementioned limitations, we advocate the use of symbolic learning to learn policies expressed in rule form. Our proposed OLAPH framework performs symbolic online learning of anomaly detection policies and continuously re-

assesses and adapts the learned policies to be able to handle distribution shifts in the input data. Although general and able to learn any type of security policy from structured data, in this paper we present an instantiation of our framework to the network policies domain, as network traffic exhibits many of the challenging aspects of real-time data with anomalies, such as high dimensionality, noisiness and anomaly class imbalance.

To tackle the problem of learning anomaly detection policies from online data our framework provides novel solutions to key challenges. Firstly, control over the *level of generality* (or specificity) of the learned policies [5]. One of the key features of symbolic learning systems is their ability to generalise from examples. Following the Occam's razor principle[1], they learn "shortest" rules as optimal solutions for a given learning task. But this is not desirable in the security domain where the common, best practice, principle of least privilege requires very specific policies. Consider, for instance, the network access requests to an application shown in TABLE I. Assuming

TABLE I: Example network requests

src	src port	dst	dst port	method	access
192.168.0.1	30300	192.168.0.3	8080	GET	Allow
192.168.0.1	40500	192.168.0.3	8080	GET	Allow
192.168.0.2	20300	192.168.0.3	9090	POST	Deny

the default behaviour to be deny access, the *most general policy* (concise rule) that would cover the allowed requests would be the following policy:

src	access
192.168.0.1	Allow

But such a policy is not specific enough from a security standpoint, as it leaves room for many requests, based on different attributes, to be allowed. On the other hand, the most specific (or least privilege) policy that would comply with the network requests in TABLE I, is the one given in TABLE II.

TABLE II: Least privilege policy

src	src port	dst	dst port	method	access
192.168.0.1	30300	192.168.0.3	8080	GET	Allow
192.168.0.1	40500	192.168.0.3	8080	GET	Allow

Both the most general and the most specific policies are not ideal, as they would produce respectively a high number of false allows and false denies. The ideal solution would be to learn policies that are more general than the least privilege policies, by using a subset of the attributes in the data, but that still capture most of the behaviour from the data. An example is the anomaly policy given in TABLE III which defines

TABLE III: Anomaly policy

src	dst	dst port	method	access
192.168.0.1	192.168.0.3	8080	GET	Allow

normal access requests and for which any (future) denied requests are considered to be classified as *anomalous*. Our proposed OLAPH framework controls the level of generality of a learned anomaly policy by means of a domain-specific optimisation criterion that scores candidate rules and guides the learning process towards the best anomaly policies for a given dataset.

But clearly, learning from examples means that a classifier is as good as its training dataset. This leads to the second main challenge of learning anomaly policies from data: learned policies can quickly become *obsolete*. For example, an updated application can make new network requests, which are then incorrectly denied by the learned policy, or even more troubling, new incoming traffic from unknown source may use new unseen attributes and therefore be incorrectly allowed. Our OLAPH framework addresses this challenge of distribution shifts in the data by relearning policies when the *confidence* in their decisions becomes low. The question becomes, how do we measure policy confidence? That is, how do we measure how likely a policy is to make a misclassification? In practice, data are unlabelled, so we do not have a ground truth of "normal" and "anomalous" requests that we could use to evaluate the accuracy of our learned policy over time. Our framework OLAPH uses as proxy a notion of *distance* between new incoming requests and the requests observed and used to learn anomaly policies so far. New requests with abnormally high (or low) distance to the observed data would trigger a relearning step of anomaly policies. Our framework OLAPH can perform this relearning online, exploiting the efficiency of the symbolic learner that it uses [3].

The overview of our proposed general framework OLAPH is depicted in Figure 2. In communication with a policy

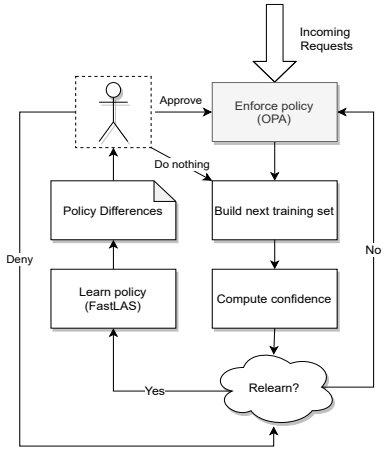

Fig. 2: OLAPH system overview

enforcer, OLAPH continuously updates a policy with rules that capture contextual and group anomalies. It computes the policy confidence to both build the next example dataset and decide whether to relearn. The user can look at differences between existing and relearned policies, leveraging upon the explainability feature of OLAPH, and decide whether to approve the relearned policy, which leads to enforcement, or

[1]The simplest explanation is the best one [6].

not. In our reference implementation of OLAPH for network anomaly detection we use OPA as the policy enforcer (see [7]) and the symbolic learning system FastLAS [3] as the *Learn policy* component. We have evaluated our system against three established anomaly detection baselines from the Scikit-learn software library.

In summary, the contributions of the paper are:

1) *A general framework, OLAPH,* for online learning of explainable security policies.
2) A *domain-specific scoring function* specifically designed for anomaly detection with configurable generalisation parameters.
3) A *definition of policy confidence* for deciding when and with what data to relearn anomaly policies.
4) An *online learning algorithm* that aims to maximise the policy confidence over time.

The paper is structured as follows. Section II describes the *Learn policy* main component of our framework, Section III presents how the rest of the OLAPH components fit together to enable the online learning aspect of the framework, evaluation results are shown in Section IV, and OLAPH is compared to Cisco's Webex platform third-party anomaly detector in Section V. The related work and final remarks can be found Sections VI and VII, respectively.

II. Learning Policies For Anomaly Detection

This section describes in more detail the *Learn policy* component of our OLAPH framework. This builds upon the symbolic learning system FastLAS [3], a scalable system for learning programs, that outputs a classifier expressed as set of rules, from a set E of examples. When the attributes defined in an example $e \in E$ are satisfied by a learned program, we say that e is *covered*. The learning of such programs is performed by FastLAS within the scope of a search space S_M of possible programs defined by a *language bias* that indicates which attributes may be used in the rules of the learned program. A set E of examples and a language bias S_M define together a symbolic learning task and the FastLAS system solves it by learning a program, within the search space S_M, that covers every positive example in E and none of the negative examples. For more details about the FastLAS system the reader is referred to [3].

In the context of OLAPH, attributes expressed in an example are sets of attribute-value pairs and the learned rules are symbolically represented as

$$\texttt{allow} \leftarrow \texttt{attr}_1(\texttt{val}_1), \ldots, \texttt{attr}_k(\texttt{val}_k)$$

where each $attr_i$ and val_i belong to the set of attributes and values that appear in the examples. So, to learn anomaly detection policies, OLAPH automatically generates from a set D of access requests and their decisions a symbolic learning task $\langle E, S_M \rangle$ for FastLAS. Each example $e \in E$ encodes a request in D as a set of attribute-value pairs, and S_M defines a search space sufficient to find a policy (i.e. set of rules) that covers every encoded request in E. The *Learn policy* of OLAPH is therefore composed of three steps as depicted in Figure 3. In our implementation, the requests in D are JSON

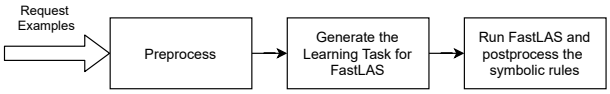

Fig. 3: OLAPH policy learning pipeline

objects and the learned policies are translated to Rego rules. For example, the network request in Figure 4a includes user attributes *source address*, *port* and *headers*, resource attributes *destination address*, *port* and *path*, and the operation *method*.

```
{
    "source": {
        "address": "192.168.0.1",
        "port": 30300
    },
    "destination": {
        "address": "192.168.0.3",
        "port": 8080
    },
    "method": "GET",
    "path": ["api", "v1"],
    "headers": {
        "User-Agent": "Firefox"
    }
}
```

(a) JSON request

```
allow {
    input.source.address = "192.168.0.1"
    input.source.port = 30300
    input.method = "GET"
    input.path[0] = "api"
    input.headers["User-Agent"] = "Firefox"
}
```

(b) Rego rule

Fig. 4: Input and output examples of *Learn policy*

A. Preprocess

The preprocessing step restructures the JSON requests into a flattened representation of attribute-value pairs that do not contain nested lists or objects. For example, the access request given in Figure 4a would be transformed into the following flattened representation:

$$\{\langle \texttt{source}, \texttt{address} \rangle : \text{“192.168.0.1”},$$
$$\langle \texttt{source}, \texttt{port} \rangle : 30300,$$
$$\langle \texttt{destination}, \texttt{address} \rangle : \text{“192.168.0.3”},$$
$$\langle \texttt{destination}, \texttt{port} \rangle : 8080,$$
$$\langle \texttt{method} \rangle : \text{“}GET\text{”},$$
$$\langle \texttt{path}, 0 \rangle : \text{“}api\text{”},$$
$$\langle \texttt{path}, 1 \rangle : \text{“}v1\text{”},$$
$$\langle \texttt{headers}, \text{“User-Agent”} \rangle : \text{“}Firefox\text{”}\}$$

A request can have any number of attributes and the running time of FastLAS depends on the number of attributes in the data. Hence, a *feature selection* function is applied over the preprocessed training set of requests, which selects a configurable maximum number of features. To make sure that the features with the largest potential impact are selected, we rank the features by kurtosis, a measure of the *tailedness* of a feature's distribution. Features with heavy tails rank more highly, as their values have a sharper deviation in the peak. These features are considered to be more anomalous and are therefore selected.

Once the set of features is selected the flattened representation of attribute-value pairs is encoded into logical atoms. For example, the attribute-value pair $\langle \texttt{source}, \texttt{address} \rangle$: “192.168.0.1”, given in the above flattened request would be encoded as the logical atom `source_address`(“192.168.0.1”).

B. Generate the Learning Task for FastLAS

Once the preprocessing step is completed, a symbolic learning task can be automatically generated. Let $\mathcal{F}_D$ be the set of selected features over a set D of access requests, the language bias for the search space S_M is given by all possible atoms $attr_i(val_j)$ that can be constructed from every $attr_i \in \mathcal{F}_D$ and possible value val_j that it has in D. Any combination of these atoms can appear in the body of a learned rule for `allow`. Clearly such a search space is very large, given the combinatorial number of possible rules that can be constructed. OLAPH addresses this challenge by defining a *domain-specific scoring function* and exploiting the FastLAS guarantee of learning solutions that are optimal with respect to any given scoring function. Specifically, given a set D of access requests, the selected set of attributes $\mathcal{F}_D$, and the encoded set E of examples, let $max\_attr(D) = max\{size(d) \mid d \in D\}$ and $min\_attr(D) = min\{size(d) \mid d \in D\}$, where $size(d)$ is the number of attributes in an access request $d \in D$. For a rule r, let $attrs(r)$ be the set of attributes in r. So, we define the score of a rule, $score_D(r)$ as follows:

$$score_D(r) = (|attrs(r)| - \lfloor \frac{max\_attr(D) + min\_attr(D)}{2(g+1)} \rfloor)^{2k} + c$$

where $g, k, c \geq 0$, are integers, and together constitute the configurable generalisation parameters of the *Learn policy* component of OLAPH. Given the above scoring function and the learning task $T_D = \langle E, S_M \rangle$, generated from a training set D of access requests, FastLAS solves the task T_D by learning an *optimal* anomaly policy P^*_D, over the policies in the power set of the search space $\mathcal{P}(S_M)$, that minimises the sum of the scores of the rules that are included in the policy. Formally:

$$P^*_D = \underset{P_D \in \mathcal{P}(S_M)}{argmin} \sum_{r_i \in P_D} score_D(r_i)$$

In solving the above optimisation problem, FastLAS learns an anomaly policy that minimally generalises from the least privilege policy. The three parameters guide the generalisation of the learned rules. The constant c determines the minimal score a rule can get. For the case of OLAPH, $c > 0$, as if $c = 0$, FastLAS is not encouraged to generate a small number of rules. The expression inside the exponent determines the lengths (i.e. no. of attributes) of the rules with minimal score. If $g = 0$ rules with exactly $\lfloor (max\_attr(D) + min\_attr(D))/2 \rfloor$ attributes are considered the "best" rules. The further away the size of the rule is from this ideal value (either smaller or larger) the larger the score. Hence, the expression tries to keep the rules away from the extremes: very short rules that are too general or very long rules that are too specific. With larger g's, the rules with minimal score become shorter. Intuitively, g provides some control to increase the generalization. The 2 in the exponent is a simple way to keep the scores positive. The value of k tells us how strict the system should be with rules that deviate from the length of the minimal scoring rules, the larger the k the faster the scores increase when rule lengths differ from the ideal length of the "best" rules.

C. Run FastLAS and postprocess the symbolic rules

Once the FastLAS system is run on the generated learning task, it produces a set of symbolic rules. This is then postprocessed into a set of Rego rules. The translation is relatively simple, since Rego and the rule-based formalism are both declarative and almost semantically equivalent. It is, however, not obvious how one can automatically generate such rules from models resulting from statistical machine learning methods. As a simple illustration, consider the following example of a FastLAS output:

```
allow ←
  source_port(30300),
  destination_address("192.168.0.3"),
  destination_port(8080),
  path(1, "v1"),
  headers("User-Agent", "Firefox").
```

The corresponding Rego rule, shown in Figure 5, begins with a `package` declaration to namespace the rules, followed by optional imports to simplify the policy syntax, and then listing the rules themselves. To deny requests not covered by a learned policy, a *default* rule of the form `default allow = false` is also automatically added to the policy.

```
package olaph

import input.source
import input.destination
import input.path
import input.headers

default allow = false

allow {
        source.port = 30300
        destination.address = "192.168.0.3"
        destination.port = 8080
        path[1] = "v1"
        headers["User-Agent"] = "Firefox"
}
```

Fig. 5: Final Rego policy

III. Online Learning

Representation learning methods frequently provide confidence scores for predictions, such as the distance of a data point to the centroid of its assigned cluster in k-means clustering, or a point's reconstruction loss when using an autoencoder. In contrast, symbolic learning methods provide rules that result in a binary decision for a given data point. This alone, is not sufficient to determine if a distribution shift has occurred and a policy needs to be relearned. In our OLAPH framework we propose a method for assigning a confidence level to a policy based on a distance calculated incrementally between the incoming *windows* (batches) of requests and the training set used to learn the policy. This confidence value, applied to the current policy, is used to compute a *relearn indicator* that lets the system decide when an anomaly policy needs to be relearned. Concurrently, an updated set of examples must be assembled for relearning. This updated set will contain examples extracted from the incoming requests used to calculate the policy confidence. Therefore, the online learning algorithm of OLAPH includes

two main components: one for building the new training set, and a second one that decides when to relearn (see Figure 6).

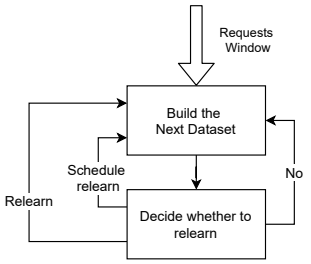

Fig. 6: OLAPH online learning algorithm

A. Build the Next Dataset

During the online iterative process, depicted in Figure 6, OLAPH considers the set of requests used to learn the current anomaly policy as the reference *trained set*, and monitors sequences of *windows* of new incoming requests until a relearn action is performed and a new policy is learned. After a new policy is learned the reference trained set is updated to the set of requests used to relearn the new policy. Between two relearn steps, OLAPH incrementally transforms the sequence of windows into a sequence of weighted windows such that for each request r appearing in a window there will be a pair $\langle r, \delta_r \rangle$ in the corresponding transformed weighted window. The value δ_r represents a distance of r to the reference trained set ts and is updated each time a new window in the sequence is processed. Distances are calculated as follows. Let's assume that at iteration t a window $\mathcal{W}_t$ of requests arrives and let $\langle w_1, \ldots, w_{t-1} \rangle$ be the current sequence of weighted windows. In the base case when $t = 1$, the sequence is empty and an initial policy is learned from the requests in $\mathcal{W}_t$. This window is then added to the sequence with each request weighted by a distance of 0 and $\mathcal{W}_t$ becomes the trained set ts. Otherwise, for $t > 1$, the distances of the requests in $\mathcal{W}_t$ to ts are calculated. First, a one-hot vector $\mathbf{e}_r$ of size n is used to encode each request r in $ts \cup \mathcal{W}_t$, with n given by the max number of attributes in the largest request included in the set. The distance δ_r of a request $r \in \mathcal{W}_t$ to ts is the minimal value of the Manhattan distances between the one-hot vector $\mathbf{e}_r$ and each of the one-hot vectors that encode the requests in ts. This is given by the function $dist(r, ts)$ defined in the equation below.

$$dist(r, ts) = min\{d(\mathbf{e}_r, \mathbf{e}_{r_{ts}}) \mid r_{ts} \in ts\}$$

where $d(\mathbf{v}_1, \mathbf{v}_2)$ denotes the Manhattan distance between vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. So, the weighted window w_t for the new window of requests $\mathcal{W}_t$ is defined as follows.

$$w_t = set\_weights(\mathcal{W}_t, ts) = \{\langle r, dist(r, ts) \rangle \mid r \in \mathcal{W}_t\}$$

If we denote by $\gamma\langle w_1, \ldots, w_{t-1} \rangle$, the sequence of weighted windows $\langle w'_1, \ldots, w'_{t-1} \rangle$, such that $\langle r, \gamma * \delta_r \rangle \in w'_j$ if and only if $\langle r, \delta_r \rangle \in w_j$, and by $\langle w_1, \ldots, w_{t-1} \rangle_{>th^l}$, the sequence $\langle w''_1, \ldots, w''_{t-1} \rangle$, such that $w''_j = \{\langle r, \delta_r \rangle \mid \delta_r > th^l \wedge \langle r, \delta_r \rangle \in w_j\}$, then the current sequence is decayed by

$$decay(\langle w_1, \ldots, w_{t-1} \rangle, \gamma, th^l) = \gamma(\langle w_1, \ldots, w_{t-1} \rangle_{>th^l}),$$

where γ is a *distance decay* constant parameter between 0 and 1, and requests are dropped if their distances to ts are below a dynamically computed threshold th^l. The next sequence is formed by appending the weighted window w_t to the decayed current sequence.

B. Decide whether to relearn

The constructed sequence of weighted windows represents a possible next training set of request examples that will be used by the symbolic learner to relearn the anomaly policy. Clearly, requests from more recent windows are likely to have a higher distance from the previous trained set, indicating that they are more likely to be anomalous, whereas older requests will have a decayed distance indicating that they are less likely to be anomalous. To decide whether to relearn the policy, a *relearn indicator* is computed over the next training set represented by the sequence of weighted windows. It is given by the average of the highest request distances in each of the windows included in the sequence. Let $ns_t = \langle w_1, \ldots, w_k \rangle$ be the next training set at iteration t composed of *non-empty* weighted windows in the sequence of weighted windows at the end of the iteration (note that $k \leq t$), and let $max\_dist_t(w_i) = max(\{\delta_r \mid \langle r, \delta_r \rangle \in w_i \wedge w_i \in ns_t\})$. We define the *relearn indicator* at iteration t, ri_t, as the mean of the maximum distances of the windows in the next training set ns_t (see equation below).

$$relearn\_indicator_t(ns_t) = \frac{\sum_{i=1}^{|ns_t|} max\_dist_t(w_i)}{|ns_t|}$$

Two *relearn thresholds* are then computed, th^h_t and th^l_t, using the mean and standard deviation of the relearn indicator values $ri_i \in rs_t$ accumulated since the previous relearn step. Algorithm 1 shows the detailed process for computing the relearn indicator and thresholds at each step.

Algorithm 1 Relearn indicator and thresholds at time step t: ri_t, th^h_t, th^l_t

Require: ns_t, rs_{t-1}
$ri_t \leftarrow relearn\_indicator_t(ns_t)$
$rs_t \leftarrow append(rs_{t-1}, ri_t)$
$th^h_t \leftarrow mean(rs_t) + 2 * std(rs_t)$
$th^l_t \leftarrow mean(rs_t) - 2 * std(rs_t)$

Figure 7 shows the band formed by the thresholds that are two standard deviations away from the mean of the relearn indicators. The standard deviation is the metric for policy confidence, as the average distance of incoming requests to the previous trained set over time is expected to be stable for a confident policy.

When the relearn indicator exceeds the thresholds, this signifies an anomalous deviation of the next dataset to the previous trained set caused by new anomalous requests. At this point, one option is to immediately relearn a new policy from the constructed next dataset ns_t of new requests to cover the new anomalies. But anomalies, unlike outliers, can have

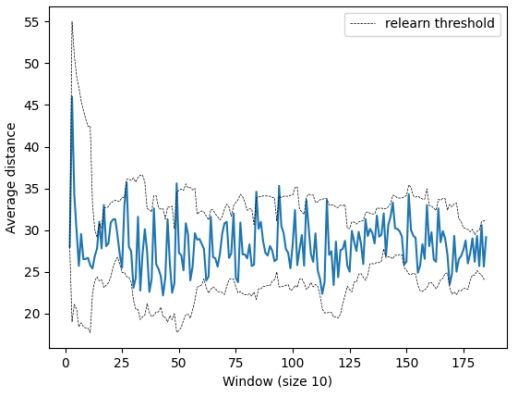

Fig. 7: Relearn indicator and thresholds over time

deviations that persist for a longer period, as they possibly characterise new behaviour that becomes permanent. So, to maximise the confidence in the relearned policy, the next dataset should contain as many of the anomalous requests as possible. A strategy for achieving this is to schedule the relearn to occur after the anomalous request distances have begun to stabilise, which happens when the relearn indicator value returns to the current threshold.

Algorithm 2 Decide whether to relearn given windows $\mathcal{W}_t$

Require: $ns_{t-1}, \gamma, th^l_{t-1}, ts, rs_{t-1}$
  $schedule_h$ = **false**
  $schedule_l$ = **false**
  while $\mathcal{W}_t \leftarrow$ **do**
    $w_t \leftarrow set\_weights(\mathcal{W}_t, ts)$
    $ns_{t-1} \leftarrow decay(ns_{t-1}, \gamma, th^l_{t-1})$
    $ns_t \leftarrow append(ns_{t-1}, w_t)$
    $ri_t \leftarrow relearn\_indicator_t(ns_t)$
    $rs_t \leftarrow append(rs_{t-1}, ri_t)$
    $th^h_t \leftarrow mean(rs_t) + 2 * std(rs_t)$
    $th^l_t \leftarrow mean(rs_t) - 2 * std(rs_t)$
    if $ri_t > th^h_t$ **and not** $schedule_h$ **and not** $schedule_l$ **then**
      $schedule_h \leftarrow$ **true**
    else if $ri_t < th^l_t$ **and not** $schedule_l$ **and not** $schedule_h$ **then**
      $schedule_l \leftarrow$ **true**
    else if $schedule_h$ **and** $ri_t \leq th^h_t$ **or** $schedule_l$ **and** $ri_t \geq th^l_t$ **then**
      $ts \leftarrow \{r \mid \langle r, \delta_r \rangle \in w \mid w \in ns_t\}$
      $relearn\_policy(ts)$
      $rs_t \leftarrow \{\}$
      $schedule_h \leftarrow$ **false**
      $schedule_l \leftarrow$ **false**
    end if
    $t \leftarrow t + 1$
  end while

The full online learning component of OLAPH is described by Algorithm 2, where ri_t is the newly computed relearn indicator, th^h_t and th^l_t are the high and low relearn thresholds, and rs_t the set of relearn indicators since the last relearning step. ns_{t-1} is the next dataset accumulated up to the previous iteration $t-1$. At the current step t, the window $\mathcal{W}_t$ of new requests is considered, its requests are weighted with their distance from the reference trained set ts, and the resulting weighted window w_t is appended to the decayed next training set constructed so far ns_{t-1} to form the next training set ns_t at step t. The relearn indicator ri_t is calculated and added to the current sequence of relearn indicators, rs_{t-1}, to get rs_t. Now the high and low relearn thresholds are computed with respect to rs_t. When the relearn indicator becomes higher than the high relearn threshold the relearn schedule flag $schedule_h$ is set to true to indicate that anomalies are starting to be detected. Similarly, the relearn indicator is checked with respect to the low relearn threshold and if it is lower than the low relearn indicator, then the schedule flag $schedule_l$ is set to true. Only when the anomalous request distances from the trained set begin to stabilise (i.e. $schedule_h = true$ and $ri_t \leq th^h_t$ or $schedule_l = true$ and $ri_t \geq th^l_t$) a relearn of the anomaly policy is performed. Then the schedule flags are reset, the accumulated next dataset used to learn the policy becomes the new reference trained set and the set of accumulated relearn indicators is also emptied. Scheduling tries to minimise the distance between new incoming requests and the reference trained set, when compared with not scheduling, as it collects more anomalous requests into the training set. The policy confidence is then higher for future requests, as these are more likely to be accumulated in the training set before relearning.

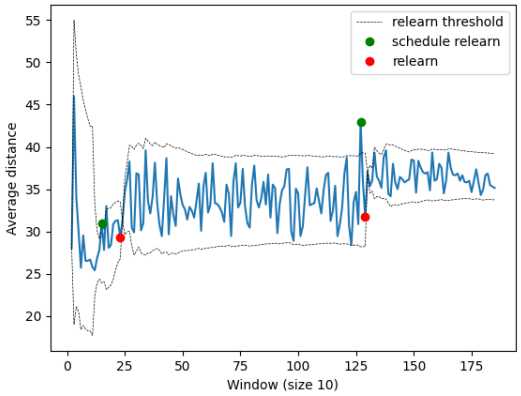

Fig. 8: Long-term memory

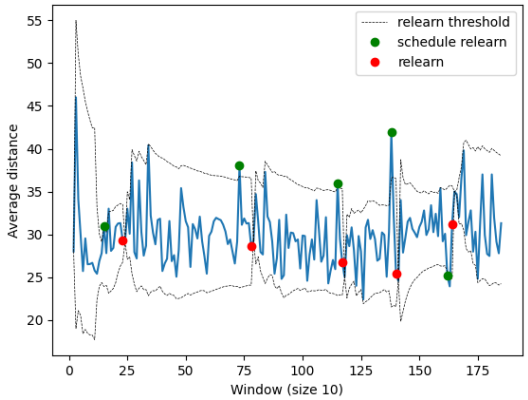

Fig. 9: No long-term memory

Once a policy has been relearned, the windows in the next dataset are collected into a single set of requests (without distances), and is passed through the policy learning pipeline from Section II to generate the new Rego policy.

C. Long-term memory of anomalies

Although the example requests in the next dataset ns_t would be covered by the policy, because of the decay, they will represent a **short-term memory** of the trained requests. For online learning, there needs to be a **long-term memory** of the most important anomalies over multiple relearns. That way, policies will eventually converge to an accurate model of network behaviour that is rarely relearned. Algorithm 2 is modified to have long-term memory by adding a special window of permanent requests to the next dataset when relearning that have the highest (if a high relearn) or lowest (if a low relearn) distances to the reference trained set and are not already permanent. Permanent requests are not decayed from the next dataset. Instead, they stay until the next dataset is full, after which the oldest window is dropped. The maximum number of windows in the next dataset can be configured as a long-term forgetfulness parameter of the system. The resulting improvement in policy confidence is shown by the narrower band around the relearn indicator in Figure 8 compared to Figure 9.

D. Generating policy differences

For administrative support, OLAPH produces after each policy relearn the difference between the old and the new policy, with individual line changes highlighted. At this point, the policy administrator can adjust the policy for desired and undesired new behaviours by adding examples to the learning task or changing the generalisation parameters. An example difference between two policies is shown in Figure 10.

```
package olaph

import input.destination
import input.headers
import input.path
import input.source

default allow = false

allow {
        source.port = 30300
        destination.address = "192.168.0.3"
        destination.port = 8080
        path[1] = "v1"
        headers["User-Agent"] = "Firefox"
}
```

```
package olaph

import input.destination
import input.headers
import input.path
import input.source

default allow = false

allow {

        destination.address = "192.168.0.3"
        destination.port = 9090
        path[1] = "v1"
        headers["User-Agent"] = "Firefox"
}

allow {
        source.port = 40300
        destination.address = "192.168.0.3"
        destination.port = 9090
        path[1] = "v2"
        headers["User-Agent"] = "Firefox"
}
```

Fig. 10: Difference between an old and a new policy

IV. Evaluation

We evaluate OLAPH against three established anomaly detection baselines from the Scikit-learn software library [8]: a One-Class Support Vector Machine (OC-SVM), an Isolation Forest (iForest), and Local Outlier Factor (LOF). We use anomaly classification labels from Cisco's Webex platform third-party anomaly detector (TP-AD) to evaluate OLAPH's anomaly detection performance compared to the baselines in scenarios with and without distribution shifts in the data. A multimodal dataset was collected from Cisco servers consisting of JSON documents summarising processes, containers and Kubernetes pods over the same period of time. The anomaly labels available from the TP-AD were for pod anomalies, so that was the subset of the dataset chosen for the test set in the evaluation.

The dataset of pod summaries is split into two sets: one set of normal behaviour, and one test set with normal and anomalous behaviour identified by the TP-AD labels. The data is flattened and feature selection is performed as in Subsection II-A. The evaluation covers both offline and online learning. In the offline mode, OLAPH learns a static policy from the normal behaviour, the baselines are fitted to the normal behaviour, and they both make anomaly predictions on the test set. The online evaluation tests OLAPH and the baselines all in an online setting, using the online learning algorithm described in Section III. The two evaluation steps are repeated for datasets with distribution shifts to compare the effects of the shifts on the performance of the online learning versus the offline learning. Two distribution shifts are considered. A small distribution shift where the normal set pod summary data is replaced with container summary data from the same time period, which uses a slightly lower-level representation. The container summary data has some overlapping attributes with the pod summary data, such as the image repository and the pod name. There is also an experiment with a large distribution shift, where the normal pod summary data is replaced with process summary data from the same period, which has a very small overlap with the original data.

The quality of the classifiers is measured through a direct comparison of their True Positive (TPR) and False Positive (FPR) rates. We plot the TPR against the FPR on an AUC-ROC curve for different classification thresholds. For the statistical baselines, the thresholds are set on the outputted anomaly probability scores, whereas for OLAPH, the threshold is controlled through the policy generalisation parameter g. We vary g from 0.25 to 2.5 in steps of 0.5, and fix $k = 4$ and $c = 1$ for all the experiments. The area under the AUC-ROC curve is how much the classifier can maximise the TPR (a.k.a the recall), while minimising the FPR, for different thresholds, evaluating how well the model can separate anomalous points from the normal points. The TP-AD anomalies are assumed to be the ground truth for the pod summary dataset in all the simulations.

A. Offline Learning

In this experiment, OLAPH and the baselines learn from the same 1000 pod summary data points from the normal set to predict anomalies in the test set of 1846 points, 5 of which are anomalous. Figure 11 shows OLAPH and OC-SVM having the best results, with OLAPH achieving an AUC of 0.98. This demonstrates that the policy learning pipeline from Section II is a good classifier in an offline learning setting with no distribution shift. On the other hand, iForest's slightly lower AUC of 0.83 can be explained by its difficulty fitting

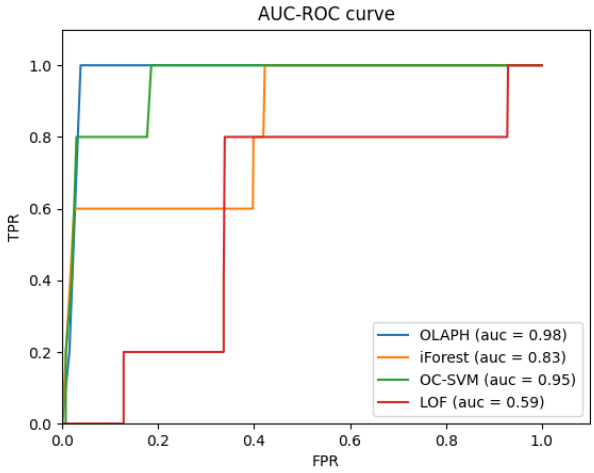

Fig. 11: Offline AUC-ROC with no distribution shift

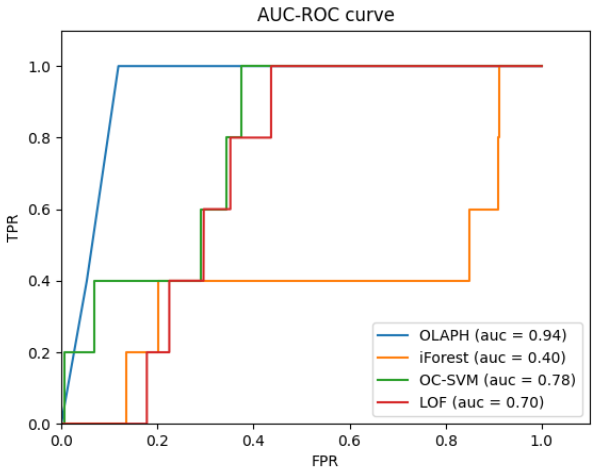

Fig. 12: Online AUC-ROC with no distribution shift

to the high-dimensional data points (caused by the one-hot encoding). Similarly, LOF's low AUC result is explained by its use of a distance measure to classify anomalies, which is not informative in the sparse high-dimensional scenario, even though Manhattan distance is used in the experiment to try and minimise this issue. The experiments with mild and major distribution shifts caused the classification performance to drop for all the models in the offline setting. For OLAPH, this happened because most of the learned rules of allowed behaviour no longer apply to the test data.

B. Online Learning

In the online learning experiments, the normal behaviour dataset size is increased to 10000 so that the online learning algorithm is able to collect data into its next dataset. The size of the next dataset at convergence is 500 data points at its maximum, which is half of the size of the normal set in the offline learning evaluation. OLAPH's online learning parameters are the same for all the scenarios: the maximum number of attributes are 30, the maximum next dataset size is 10 windows, the window size is 50 requests, and the decay is 0.9. OLAPH outperforms the baselines in scenarios with and without a distribution shift, although most of the baselines have an improved performance when combined with our online learning approach.

When learning online with no distribution shift, Figure 12 shows that LOF has a 19% increase in its AUC, compared with a decrease for the other baselines, as LOF is able to take advantage of the more up-to-date data points in the online learning dataset due to its localised algorithm. The other baselines show a reduction in performance, possibly due to the reduced next dataset size compared with the 1000 data points in the offline learning scenario.

In the case of a major distribution shift, the results in Figure 14 show that our online learning algorithm has a 58% increase in the AUC of the ROC curve for OLAPH compared with offline learning where its AUC was 0.5, and a 38% increase

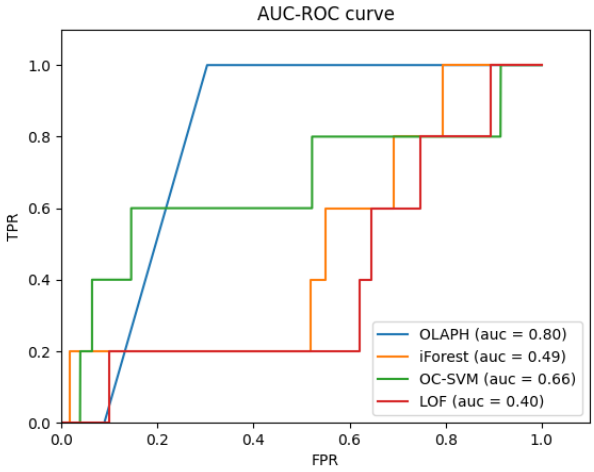

Fig. 13: Online AUC-ROC with a mild distribution shift

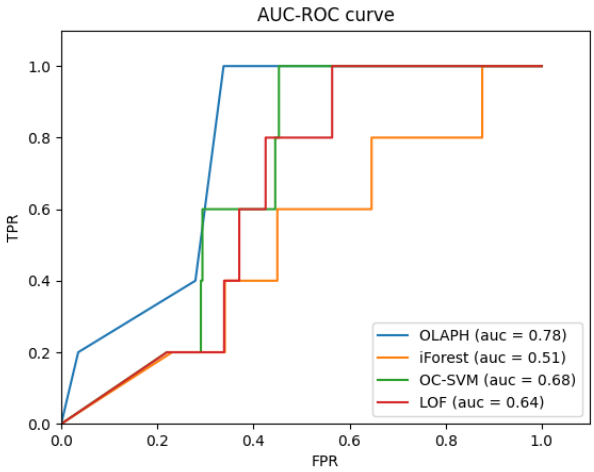

Fig. 14: Online AUC-ROC with a major distribution shift

in the case of a mild distribution shift where its offline AUC was 0.58, as shown in Figure 13. The baselines also see improvements when learning online in the scenario of a major distribution shift, with the best improvement of 51% for the OC-SVM from its offline AUC of 0.45. This shows that our online learning algorithm is able to benefit other machine learning algorithms too. These experiments show the effects of the distribution shift in the worse-case scenario, where the test set is the shifted dataset. In practice, OLAPH could have learned from more requests in the new distribution before making anomaly classifications. This would have made the anomaly detection performance closer to the scenario with no distribution shift, as the data in the next dataset would be exclusively from the new distribution.

V. Discussion

In this section we discuss the advantage of using OLAPH's versus Cisco's Webex platform third-party anomaly detector (TP-AD), in terms of its usability. The adoption of a Symbolic Learning system allows OLAPH to be both more flexible, as it provide the experts parameters with which to control the level of generalisation of the learned policies, and more explainable as it can highlight the differences between existing policies and new learned ones helping the experts to gain trust into the behaviour of the system. None of these features are offered by Cisco's existing security solution.

We have conducted a qualitative analysis on the effectiveness of the system in learning policies that are "closer" to those that experts would manually define, that is policies that include experts' desired characteristics. To do so, we have considered an example policy provided by Cisco's experts that allows certain requests to an application. This user-supplied target policy, for one of their applications, is shown in Figure 15. We have then run OLAPH statically and generated automatically a policy that is very similar to the user-supplied target one. This is given in Figure 16.

```
allow {
    input.attributes.destination.address.socketAddress.portValue == 3000
    input.attributes.request.http.protocol == "HTTP/1.1"
    input.attributes.request.http.method == "GET"
}
```

Fig. 15: Cisco's policy for requests to one of their applications

```
allow {
    input.attributes.destination.address.socketAddress.portValue = 3000
    input.attributes.request.http.method = "GET"
    input.attributes.request.http.protocol = "HTTP/1.1"
}
```

Fig. 16: OLAPH's policy for Cisco's application

The only difference between the above two policies is in the order of the body expressions of the rule. This a pure syntactic difference that does not affect their meaning: the two policies are semantically equivalent. Furthermore, modulo the ordering the learned policy includes the same desired characteristics indicated in the target policy. This has been made possible by controlling the parameters of OLAPH, described in Subsection II-B, for configuring the generalisation process.

To further evaluate whether the explanation of learned policies is "meaningful" for the user we have involved Cisco's experts to assess this quality of OLAPH. As shown in Section III-D, a distinct feature of OLAPH is to ability to generate explainable policies, which allows the user to identify the normal behaviour in the rules. The TP-AD does not provide this information. We have therefore sent the explanation of policy difference, generated by OLAPH on the pod summary dataset, to Cisco as part of a survey. Cisco Webex platform engineers were asked whether the context provided by the policy difference helped them to understand the new allowed behaviours better than their existing TP-AD system. Given a scale from 1, for not at all identifiable behaviour, to 5, for completely identifiable, OLAPH was given an average score of 4.7 by three respondents. One justification for the rating was:

> *The policy engine shows exactly what is considered normal/allowable, whereas the [third-party security] system is a blackboxed machine learning algorithm, and thus we have no visibility into why certain traffic is considered "normal".*

VI. Related Work

We are not aware of previous work on online symbolic learning. However, in this section we review recent methods for learning security policies that are either online or incremental, or related to learning attribute-based access control (ABAC) policies such as the Rego policies generated by OLAPH with some parametric generalization control.

A Reinforcement Learning (RL) framework – Jarvis-SDN – for network flow policies has recently been developed that encodes knowledge about existing malicious attacks in the agent's optimisation function as a security metric [9]. This is combined with functionality metrics, such as maximising packet throughput, using parameters to decide the tradeoff between usability and security. The result is a semi-supervised approach that learns policies online and achieves good accuracy when detecting known attacks, but has reduced accuracy for unknown or modified attacks. This could be due to Jarvis-SDN's reliance on supervised learning for its security metric, which the authors mention could be improved by including an anomaly detection component. Additionally, the policy is not explainable, due to the statistical machine learning approach.

Karimi et al. propose learning ABAC policies by extracting them from the *k-modes* unsupervised learning model [10]. Their approach consists of tuning hyperparameters to find the policy that maximises their policy quality metric composed of correctness and conciseness, as well as rule pruning and policy refinement. This is less efficient than OLAPH's approach, which uses the examples directly to find the solution of the scoring function optimisation problem. Similar to OLAPH's use of a least-privilege policy as the least general solution, Karimi et al. uses the *most complex policy* as the limit for the policy size metric. But they implement several features that can lead to higher quality learned policies. For example, a

more concise policy could use negated rule attributes, such as `not headers["User-Agent"] = "Firefox"`, or variables or ranges for attribute values (e.g. `headers["User-Agent"] = X` or `destination.port > 8000`). Negation, variables and ranges are all supported by FastLAS. The only challenge would be to decide how OLPAH integrates the changes to the task search space; i.e. exactly which attributes can appear negated and which values can be represented by variables or ranges.

Batra et al. generate ABAC policies using incremental maintenance of a previously mined policy [11]. The existing policy rules are updated by applying a set of given changes Δ to the dataset, which could consist of new or deleted access permissions, as well as new or deleted attribute values. The rules in the policy are then updated to take into account the changes. The overall approach is much faster than re-mining the policy. However, the Δ changes are assumed to be given instead of being detected automatically and it is a supervised learning algorithm with permission labels for requests in the dataset, which are not always available in practice.

Polisma learns ABAC policies from examples and contextual information [12]. It combines policy mining, statistical methods and machine learning to maximise the correctness and completeness of generated policies, while being robust to noisy examples. Polisma combines several rule mining techniques into one pipeline, whereas OLAPH delegates all the rule mining to the FastLAS policy learner. OLAPH is able to do this, as the desired level of generalisation and the use of certain attributes is achieved with FastLAS's domain-specific scoring function. OLAPH is not required to apply a machine learning technique to classify requests that are not covered by the policy, as its online learning algorithm detects requests that are not confidently covered and includes them in the next training set when relearning the policy.

VII. Conclusion and Future Work

Offline learning is not feasible in ML applications with large amounts of constantly evolving data. Current security providers in the industry cannot scale to the needs of a modern microservices environment consisting of multimodal data streams, as it is too costly. A goal of security is to have a *steady-state* representation of a system, but this is often impossible with the currently available offline learning tools. We have shown that OLAPH can fill this space in the domain of anomaly detection, while also being fully explainable, which is a significant advantage over existing tools. The OLAPH framework can be extended to other domains, since it is not coupled to the input data format, nor the output policy language, nor the scoring function. Cisco plans to use OLAPH to learn a common policy language from multimodal data streams for explainable online anomaly detection, and eventually policy enforcement. OLAPH is an open source project[2] and has a user guide for learning Kubernetes application access control policies[3]. We are eager to explore new uses of online symbolic learning with the security community.

[2]https://github.com/olaph-ai/olaph

[3]https://olaph-ai.github.io

References

[1] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, 54(2), March 2021.

[2] Mark Law, Alessandra Russo, and Krysia Broda. The ilasp system for inductive learning of answer set programs, 2020.

[3] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. Fastlas: Scalable inductive logic programming incorporating domain-specific optimisation criteria. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2877–2885, Apr 2020.

[4] Products Services, Cisco Enterprise, and White Papers. Cisco stealthwatch enterprise - cisco security analytics white paper, 2020. URL https://www.cisco.com/c/en/us/products/collateral/security/stealthwatch/white-paper-c11-740605.html.

[5] Jorge Lobo, Elisa Bertino, and Alessandra Russos. On security policy migrations. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, SACMAT '20, page 179–188, New York, NY, USA, 2020. Association for Computing Machinery.

[6] Duignan Brian. Occam's razor.

[7] Styra. *Open Policy Agent*, 2021. URL https://www.openpolicyagent.org/.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] Anand Mudgerikar, Elisa Bertino, Jorge Lobo, and Dinesh Verma. A security-constrained reinforcement learning framework for software defined networks. In *ICC 2021 - IEEE International Conference on Communications*, pages 1–6, 2021.

[10] Leila Karimi, Maryam Aldairi, James Joshi, and Mai Abdelhakim. An automatic attribute based access control policy extraction from access logs. *CoRR*, abs/2003.07270, 2020.

[11] Gunjan Batra, Vijayalakshmi Atluri, Jaideep Vaidya, and Shamik Sural. Incremental maintenance of abac policies. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, CODASPY '21, page 185–196, New York, NY, USA, 2021. Association for Computing Machinery.

[12] Amani Abu Jabal, Elisa Bertino, Jorge Lobo, Mark Law, Alessandra Russo, Seraphin Calo, and Dinesh Verma. Polisma - a framework for learning attribute-based access control policies. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve Schneider, editors, *Computer Security – ESORICS 2020*, pages 523–544, Cham, 2020. Springer International Publishing.