On Security Policy Migrations

JORGE LOBO, UPF/ICREA/Imperial College
ELISA BERTINO, Purdue University
ALESSANDRA RUSSO, Imperial College

There has been over the past decade a rapid change towards computational environments that are comprised of large and diverse sets of devices, many of them mobile, which can connect in flexible and context-dependent ways. Examples range from networks where we can have communications between powerful cloud centers, to the myriad of simple sensor devices on the IoT. As the management of these dynamic environments becomes ever more complex, we want to propose policy migrations as a methodology to simplify the management of security policies by re-utilizing and re-deploying existing policies as the systems change. We are interested in understanding the challenges raised answering the following question: *given a security policy that is being enforced in a particular source computational device, what does it entail to migrate this policy to be enforced in a different target device?* Because of the differences between devices and because these devices cannot be seen in isolation but in the context where they are deployed, the meaning of the policy enforced in the source device needs to be re-interpreted and implemented in the context of the target device. The aim of the paper is to present a formal framework to evaluate the appropriateness of the migration.

CCS Concepts: • **Security and privacy → Formal methods and theory of security**;

Additional Key Words and Phrases: security policies; policy migration

ACM Reference Format:
Jorge Lobo, Elisa Bertino, and Alessandra Russo. 2020. On Security Policy Migrations. In *25th ACM Symposium on Access Control Models and Technologies (SACMAT'20), June 10–12, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3381991.3395613

1 INTRODUCTION

There have been technological developments such as cloud computing and virtualization that have facilitated the view of IT infrastructure as a common-pool of resources. This tendency continues to attract attention due to the proliferation of mobile computational devices and the fast development of the Internet of Things where shared resources and distributed computation are the norm. Sharing resources makes economic sense and it is a management paradigm that can be useful in many situations. There have been, for example, many efforts developing the concept of vehicular micro-cloud computing [15, 37]. In such an environment an IT cloud is put together with the resources from a collection of vehicles maybe parked in the same parking lot or moving together in the same geographical area to host cloud services to devices close by. There is also growing interest in coalition operations that bring together organizations with varying degrees of prior relationships that want to share resources to achieve a common goal such as providing services for large multi-national events or emergency situations. In these environments the computational resources are heterogeneous,

Manuscript submitted to ACM

their numbers can vary over time and there might not be previous knowledge of the schedule of availability. The same uncertainty applies to the users of the resources.

One of the management challenges in these environments is how to provide the appropriate security to the resources of the different participants in the collaboration. In other words, how to ensure the confidentiality, integrity and availability of the hardware, software and data shared by the participants. A tunable security on a par with the elasticity expected from these federated systems will be tantamount to their widespread deployment.

Let's consider the following example. There is a server holding a database that is accessed by many clients. As protection from SQL injection attacks, the server also runs a Snort network security monitor [14] to prevent any suspicious access to the database. The server is malfunctioning or it is in a vehicle that is moving away from the clients and the database needs to be moved to another location. The new server is less powerful and it does not have a Snort monitoring available. Instead, for security, only a restricted set of trusted source IP addresses will be allowed access to the database which the server will enforce using iptables or ebtables [12].

In the context of collaborative activities, Williams et al. [6, 38] have proposed a policy-based management architecture specifically designed to provide this kind of flexibility. Policies in the context of IT system management are application independent directives that regulate the operations of the system [2, 3]. Security policies are, therefore, directives to ensure the confidentiality, integrity and availability of a system like preventing SQL injection attacks in the database example above. The collaborative architecture proposed has been designed based on the concept of *generative policies* introduced in [35]. Generative policies are templates specified as partially defined context-sensitive grammars deployed in all the devices participating in the activity and that each device, when requested, will use the grammars to generate its local policies according to its local context. Hence, our example above will be easily realized by having a grammar that generates the appropriate enforcement according to the capabilities of the device. Observe that the policy migration in the example is not the result of simply changing some configuration parameters in the server. The policy in the second device is "weaker" than the first policy in the sense that if a SQL injection attack is initiated from a client from a trusted IP address the server and database will be compromised. But, perhaps the benefit of having the database available to the trusted clients overrides the risk of the unlikely attack. In addition, there are clients that were able to access the database in the first server that the new policy implementation might reject. These changes on security cannot be done lightly. The consequences should be well understood.

This brings us to the formulation of the following problem:

> *Given a security policy that is being enforced in a particular source device, how can this policy be enforced in a different target device?*

The answer can be, in many cases, that it cannot because the enforcement mechanisms in the second device are different or limited, or access to the information required to enforce the policy is limited or unavailable. However, one could cautiously find a realization of a policy in the second device that is stricter than the original policy, meaning that all the executions that were not allowed by the first policy are not allowed by the second policy, but a few extra executions are also rejected. There might be less cautious solutions that let a few policy violations pass to ensure the system is always available to correct executions. The main question is how to find the *appropriate implementation*. Let's assume, for example, there is a device that does deep packet inspection to enforce a policy over http accesses and the policy is now going to be enforced by a device that can only inspect TCP headers going to port 80; the new implementation cannot be exact because of the different computational capabilities of the devices.

The contribution of this paper is twofold. We first present a formal framework to characterize different types of migrations. Since the result of a migration is constrained by the computational capabilities of the source and the target devices, the framework must distinguish the concept of policies from the mechanisms needed to enforce the policies. Orthogonal to the computational differences between devices, the contexts where the migrations occur will also affect the migrations. A migration can happen from a device that is merely being replaced by a second device. This is the case of the vehicular micro-cloud example where a database and its access control policies are migrated to another device in the same micro-cloud, there is no change of context. However, the migration can depend on the context where the policy will be enforced. For example, we want to replicate the database server in different vehicular micro-cloud. Hence, the framework also differentiates migrations happening in the same context versus migrations happening between different contexts. This second type of migrations is the most challenging one. In the second part of the paper, we list the few results in policy migration that already exist. They are all for migrations within the same context. We then identify different results in other areas of security research that can be useful to develop both, same context and context-dependent migrations. These include policy mining, policy learning and policy similarity analysis. We also briefly discuss how we can evaluate the quality of the migrations.

Although we will frame the migration problem in fairly general terms, in the second part we will be addressing migrations within two loosely defined classes of security policies: access control policies which are meant to process access request to resources and decide whether to allow or reject the request, and network policies that typically classify and filter network traffic as in firewalls and intrusion detection systems.

In the next section we will define the class of policies we will cover. We will also recall the definition of *execution monitor* as their enforcement mechanism from [32]. In Section 3, we formulate the concepts of *migrations* and policy *approximations*. Section 4 presents an overview of the existing methods to help with migrations. Section 5 discusses how to evaluate migrations that results in approximations of the desired policy, and we conclude the paper discussing some of the open problems in policy migration.

2 POLICIES AND EXECUTION MONITORS

It is fairly understood that a general abstraction to capture a large class of security policies is to define a policy as a property that must hold in all possible executions of a system, where an execution is defined as a finite or infinite sequence of events such as a sequence of system actions, or system states, or state-action pairs [5, 22, 32, 36]. These sequences are sometimes called execution *traces*. Take for example, an operating system where files have owners assigned to them, and only the owner of a file can read the file. An execution in this system is a sequence of state-action pairs, and the execution *satisfies* the policy if there is no pair in the execution in which the action is to “read" a file in a state where the owner of the file doesn’t match the executor of the action. As another example, we can have a network card in a server in which no traffic coming from IP address d and going to port p must get into the server. An execution comprises the sequence of packets passing through the card. The execution satisfies the policy if there is no packet in the sequence with source IP address d and destination port p in its TCP header. There are also policies that can specify properties that depend on several events in the trace. Let’s say, that in the same operating system there is a file that can be read by a fixed set of users of the system but each user can only read it once. In this case the policy is not satisfied if either there is a state-action pair where the action is “read" and the executor of the action is not in the set of allowed readers, or there is a previous state-action pair in the execution where the action “read" was executed by the same reader.

We call an *execution monitor* for a policy $\mathcal{P}$, a machine or a program that takes as input an execution and decides if the execution satisfies $\mathcal{P}$. Most implementations of execution monitors, more than simple monitors are enforcers of policies. This means that the monitoring is happening in real time and either executions are stopped as soon as the monitor realizes that a violation of the policy is about to happen, or a corrective action is executed by the monitor (e.g., a program trying to read a file with the wrong permissions is aborted, or a packet with inappropriate headings is dropped). In addition, monitor implementations are policy-independent in the sense that they are programmable to enforce different policies. For simplicity of presentation, we assume all monitors stop executions if violations of policies are detected.[1]

There is a large variability among the policies that different monitors can implement. This is heavily determined by the system events (i.e., the kind of execution events the monitor observes) and the computational capabilities the monitor has to detect policy violations. Nevertheless there are a few necessary conditions that a policy must have for a monitor to exist [17]. To explain the conditions we need a more precise characterization of policies.

For the time being we will ignore the details of what an event is, and only assume that there is a countable set of events E which an execution monitor EM is able to monitor. We call E the *domain* of EM. Let E^ω be the set of all finite or infinite sequences of events from E. *Executions* of the system monitored by EM are sequences in E^ω. We denote by ϵ the empty execution.

A *policy* $\mathcal{P}$ defines a set of acceptable or correct executions, and we denote that set by $\mathcal{P}(E^\omega) \subseteq E^\omega$. A basic condition that holds for all policies that can be enforced by an execution monitor, EM, is that they can be described in terms of single executions. That is, there must be a predicate P over executions that for any execution X, $P(X)$ is true if and only if $X \in \mathcal{P}(E^\omega)$. This condition lets EM take decisions by only looking at the current execution of the system independently of other potential executions. Note that in some cases, policies (e.g., policies based on differential privacy [13]) might be defined in terms of a set of executions, and not individual executions. But in this paper we only consider policies that can be enforced by execution monitors, i.e., evaluating conditions over single traces. A second condition that must hold is that these predicates P over executions must be *safety* properties [19]. Informally, a safety property identifies and stipulates that a "bad thing" does not happen during the execution. For the implementations of execution monitors, this condition implies that if $P(X)$ is false then there is a finite sequence of events $e_1 e_2 \dots e_i$, prefix of X such that $P(e_1 e_2 \dots e_i)$ is also false. Let's now give a formal definition of execution monitor.

Definition 2.1 (Execution monitor). Given a countable set of events E and a predicate P, an *execution monitor* EM programmed to enforce P over sequences of monitored events from E is an automaton defined by:

- A set Q of (protection) states
- A partial transition function $\pi : Q \times E \rightarrow Q$
- An initial state $q^I \in Q$

We denote such an automaton with EM_P and referred to it as an *implementation* of P.

In an execution trace $e_1 e_2 e_3 \dots e_i \dots$, monitored by an execution monitor EM_P, events e_i, will be input to the monitor, $EM(e_i)$, one by one in the order they occur. The state of the monitor after the observation of the event e_i may depend on previous events, and we will denote it by $EM[e_1, \dots, e_i]$. If $\pi(EM[e_1, \dots, e_{i-1}], e_i)$ is defined then

$$EM[e_1, \dots, e_i] = \pi(EM[e_1, \dots, e_{i-1}], e_i).$$

[1] This is a simplified view of what a monitor can do since a full characterization of execution monitors must consider monitors that can't prevent events that violate policies [5].

Otherwise $EM[e_1, \ldots, e_i] = \perp$, and $EM[] = q^I$. The case when $EM[e_1, \ldots, e_i] = \perp$ can be interpreted as an interruption of the execution of the system at the execution of $EM(e_i)$. In practice, an execution monitor EM_P monitoring an execution that does not satisfy a property P may take other actions when e_i is about to happen and $P(e_1 e_2 \ldots e_i)$ is false, to avoid the effects of e_i or to alert other parts of the system. This is done for the smallest i in the execution. Other restrictions may be imposed on P since properties enforced by an execution monitor cannot depend on the whole prefix of an execution; otherwise an unbounded amount of memory to store any of the prefixes that violates the policy would be needed by the monitor. There are other practical considerations like time and space complexity (e.g., how difficult is to implement π or how large is the set Q) needed for checking the property P that limit the policies that an execution monitor can enforce.

Definition 2.2 (Correct implementation). An execution monitor EM_P enforces a property P or is a *correct* implementation of P, if the following condition holds:

$$\forall X \in E^\omega\colon\ P(X) \equiv (EM_P[X] \neq \perp).$$

Let's look at a few examples. In typical stateless IP firewalls only input from the current event is used: this event has the TCP header of the packet that is arriving to the firewall. States encode tables of TCP header patterns. The policy fixes one of these tables as its initial state, q^I, and the firewall implements efficient and intricate algorithms to check whether the header in the input packet matches one of the patterns in the table and then decide whether or not to drop the packet.[2] The transition function always returns q^I or is undefined.[3] In a signature-based intrusion detection system like Snort [14], the set of execution prefixes that do not satisfy the property defines a regular language. The execution monitor is therefore programmed as a deterministic finite state automaton (DFA) able to recognize the "complement" of such language. The transition function of such automaton will return undefined whenever the monitored execution prefix includes a signature-based attack and therefore violating the property. These signatures are defined using more than the TCP headers and the *EM*s need the computational capability to do deep packet inspection. Anomaly-based intrusion detection systems typically assign probabilities to prefixes and if the probability of a prefix is over certain threshold the execution is rejected. States will be encoding data structures where statistical information about the network traffic is collected and updated by the transition function. Anomaly-based intrusion detection is, by construction, a safety property, but it is approximating a policy that is partially known. They may or may not do deep packet inspection.

3 POLICY MIGRATIONS

We assume we have two (not necessarily different) devices, a *source* and a *target*, and some semantic knowledge related to the properties P_s and Pt, which we call *context*. The context may be the same between the source and target properties, or variable, in which case some extra knowledge is needed to relate P_s and Pt. We consider policy migration in each of these two cases separately.

3.1 Same context

The implementation $EM^s_{P_S}$ of a policy P_s by an execution monitor EM^s in the source device is characterized by four components: the set of input events E_s, the set of protection states Q_s, the transition function π_s, and the initial state q^I_s.

[2]Dropping the packet is done instead of stopping the execution.
[3]We will not consider administrative operations that can change policies on the fly in this paper.

The migration of the policy to an execution monitor EM^t in the target device requires finding a policy P_t that EM^t can implement. An instance of this situation is a scenario where EM^s is the Linux iptables utility program implementing a TCP firewall policy P_s, and the policy will be migrated to an instance of the intrusion detection system Snort running in the same device. In other words, EM^t is Snort. It would be possible to emulate the iptables rules implementing P_s using a DFA that only looks at TCP headers but at a higher level of the protocol stack since Snort reorders and reassembles IP fragments and TCP segments. This changes P_t. The set of events E_s in P_s are IP packets; events, E_t, in P_t are TCP segments. But there is a mapping $T : E_s^\omega \to E_t^\omega$ that let us establish a precise characterization of P_t in terms of P_s.

Definition 3.1 (Faithful migration). Let P_s be a property for a source device and P_t a property for a target device. The property P_t is a *faithful* migration of P_s if the following condition holds:

$$\forall X \in E_s^\omega: \quad P_s(X) \equiv P_t(T(X))$$

In contrast, if the monitors switch roles and EM^s is Snort programmed to detect SQL injection attacks from a fixed range of IP addresses, it is not possible to have a faithful implementation of the policy using iptables since it requires to do deep packet inspection and maintain states which the iptables application can't do. By simply looking at E_s^ω and E_t^ω we are not always able to characterize the potential limitations of the migration, as in the case of Snort as EM^s, and iptables as EM^t. All the information in an IP segment that Snort manipulates is available in the form of IP packets in the executions of iptables, but computationally, the iptables are limited. To capture computation, we will extend the definition of faithful migrations of policies to implementations.

Definition 3.2 (Faithful implementation). Let $EM^t_{P_t}$ be the implementation of P_t in EM^t. We say that $EM^t_{P_t}$ is a *faithful* implementation of $EM^s_{P_s}$ if there is mapping $T : E_s^\omega \to E_t^\omega$ such that the following condition holds:

$$\forall X \in E_s^\omega: \quad (EM^s_{P_s}[X] = \bot) \equiv (EM^t_{P_t}[T(X)] = \bot)$$

We would like to get a faithful implementation of P_s in EM^t, but if there are constraints in Q_t that don't exist in Q_s, for example, the number of states is bounded by a constant, a faithful implementation might not exist. In the case of Snort acting as EM^s, the minimum number of states in Q_s depends on the states needed to implement the automata that recognize the regular languages defined in P_s, and that number can be larger than 1. In an iptables instance, Q_t has a single state, the table of firewall rules. This means that if EM^t is the iptables utility, we can have two states, $q_s^0, q_s^1 \in Q_s$ in Snort, two executions X, Y, and an event $e \in E_s$ such that $EM^s_{P_s}[X] = q_0$, $EM^s_{P_s}[Y] = q_1$, $\pi_s(q_s^0, e) = \bot \neq \pi_s(q_s^1, e)$, but $\pi_t(q_t^I, T(e))$ can only map to a single value.

Even if the type of execution monitors in the target device and the source device is the same, faithful implementations might not exist. Again let's take Snort as an example. If both execution monitors are implemented using Snort, but the target device runs in a network where the traffic is encrypted, the target monitor might not be able to observe the same in executions if the source device gets traffic in an area of the network where traffic is not encrypted. An event e_s in E_s is mapped into an encrypted event e_t in E_t, but the transition function in EM^t cannot operate on e_t in the same way EM^s operates on e_s. This can be understood as the possibility of having two different events $e_s^0, e_s^1 \in E_s$ with the same headers but different payloads such that $EM^s_{P_s}[Xe_s^0] = \bot$ and $EM^s_{P_s}[Xe_s^1] \neq \bot$ but that $T(Xe_s^0) = T(Xe_s^1)$. Hence, similar to the case of states, there may no exist a *faithful* implementation $EM^t_{P_t}$ of $EM^s_{P_s}$.

Having $E_s = E_t$ and $Q_s = Q_t$ doesn't ensure the existence of a faithful implementation either, if there are computational limitations in π_t when compared to π_s. Given $e \in E_s$, and $q \in Q_s$, and denote by $|e|$, and $|q|$ the size required to represent e and q respectively, let's say as strings or in bits, and assume $|e| + |q| = n$. If the complexity of computing $\pi_s(q, e)$ is $\Theta(f_s(n))$, but any π_t implementable in EM^t cannot be of a computational complexity larger than $O(f_t(n))$,

where $O(f_s(n)) > O(f_t(n))$, then there may no exist a *faithful* implementation $EM^t_{P_t}$ of $EM^s_{P_s}$, unless EM^t uses more bits to represent either event in E_t or states in Q_t. For example, let's take (u, o) to be the event e indicating that users u wants to access object o, let q be a labeled graph as in Relationship-based Access Control [25] and $\pi_s(q, (u, o)) \neq \perp$ if there is a path in the graph from u to the owner of the object o. If EM^t can only implement a transition function π_t that is constant or sub-linear with respect to the size of the graph then the path cannot be computed.

A cautious solution is to try to find a policy P_t that rejects all executions that don't satisfy the policy P_s. Hence, we can define safe implementations as follows.

Definition 3.3 (Safe implementation). $EM^t_{P_t}$ is a *safe* implementation of P_s by EM^t if the following condition holds:

$$\forall X \in E^\omega_s \colon (EM^s_{P_s}[X] = \perp) \Rightarrow (EM^t_{P_t}[T(X)] = \perp).$$

Recall that because P_s is a safety property, rejected execution traces, X, can be limited to finite sequences. Therefore, safe implementations of a policy can also be limited to enforce safety properties.

3.2 Variable Context

There is another more challenging class of policy migration. Take, for example, an execution monitor EM^s that looks at the headers of TCP packets and these packets are being produced within a closed Local Area Network (LAN). EM^s will only observe executions limited to sequences of packets with source and destination IP addresses within the ranges of IP addresses in the LAN. If the LAN of the target device is different, the headers will be different. The policy P_s can be implemented by $EM^s_{P_s}$, in terms of the IP address in the LAN, i.e. an event $e \in E^s$ appears in an execution only with headers in the LAN. In this class of migrations there is no mapping T from executions from the source device to target device. Thus, we cannot relay on faithful or safe implementations which depend on T to find P_t. Some extra semantic knowledge must be made available in order to relate P_s to P_t. In our example, this semantic knowledge must let us find out how the IP address in the LAN of the source device can be interpreted in the new LAN for EM^t to implement P_t. This semantic knowledge can come in many forms. For example, P_s can filter packets coming from a range of IP addresses that are known to belong to certain group of devices, let's say, the devices that belong to members of a particular department in the organization that owns the LAN. If in the new LAN, this grouping can also be realized, i.e., associate IP addresses to devices that belong to the same department, then P_t can be derived from that knowledge. P_s could also be doing load balancing splinting the range of addresses evenly into two output ports for forwarding. Then, P_t can do the same but for the range of addresses in the new LAN. We will refer to this extra semantic information as *the contexts*, C_s and C_t, of the executions in the source and the target devices. A context is a collection of attributes of execution monitors running in the device, that might be available to compute the migration. For an execution monitor EM, there can be attributes about E (as a set or attributes of its members), and about the device where EM will run. Attributes available in a context will depend on the settings where the migration occurs. For example, a device might be embedded in a network and propositions of the network topology (e.g., size of the network, connectivity between nodes) can be part of its execution monitors context; or the device is moving and its location is a context attribute. Before we define a context migration, we will need to assume that the policy $\mathcal{P}$ is context dependent, $\mathcal{P}(C_s, E^\omega_s) \subseteq E^\omega_s$, $\mathcal{P}(C_t, E^\omega_t) \subseteq E^\omega_t$ and that a safety property, P^c, exists such that (1) for any execution trace $X \in E^\omega_s$, $P^c(C_s, X)$ is true iff $X \in \mathcal{P}(C_s, E^\omega_s)$, and (2) for any execution trace $X \in E^\omega_s$, $P^c(C_t, X)$ is true iff $X \in \mathcal{P}(C_t, E^\omega_t)$.

Definition 3.4 (Context-dependent faithful migration). Let P_s and P_t be a source property and a target property respectively and let P^c be a safety property. P_t is a *context-dependent faithful* migration of P_s, under a safety property P^c if the following condition holds:

$$(\forall X \in E_s^\omega : P_s(X) \equiv P^c(C_s, X)) \wedge (\forall X \in E_t^\omega : P_t(X) \equiv P^c(C_t, X))$$

Definition 3.5 (Context-dependent faithful implementation). Let $EM_{P_t}^t$ be an execution monitor for the target property P_t. $EM_{P_t}^t$ is a *context-dependent faithful* implementation of a correct implementation, $EM_{P_s}^s$ of P_s, under a safety property P^c if the following condition holds:

$$\forall X \in E_t^\omega : P^c(C_t, X) \equiv (EM_{P_t}^t[X] \neq \perp)$$

In other words, we don't refer to $EM_{P_s}^s$ but to P^c that directly tells us the policy we need to implement in EM^t: $EM_{P_t}^t$ must be a correct implementation of P^c under the context C_t.

Like in migrations over the same context, there might not be faithful context-dependent implementations because of constraints in EM^t and E_t. We can also define *context-dependent safe* implementations as follows:

Definition 3.6 (Context-dependent safe implementation). $EM_{P_t}^t$ is a *safe* implementation of P_s in the context C_t by EM^t if the following condition holds:

$$\forall X \in E_t^\omega : (EM_{P_t}^t[X] \neq \perp) \Rightarrow P^c(C_t, X)$$

The challenge of policy migration with same context is to figure out how to get T. There are similar challenges for policy migrations with variable context since π_s may use properties of the events that are not available in the events in E_t, or constraints in Q_t or the computational capabilities of the π_t don't allow faithful implementations of P^c. But contexts complicate re-implementations even more because the only information we might have about P^c is $EM_{P_s}^s$, i.e., how P^c is implemented in EM^s.

An implementation that rejects all the executions is a trivial safe implementation of P_s in the target device but of little use. One should look for implementations that best approximate a faithful implementation. That could be interpreted as minimizing the number of correct executions rejected. But it might not be easy to come up with good safe approximations. Rejections of correct executions affect another axis of security: availability. Hence, for the sake of availability, non-safe approximations might need to be considered.

The need for policy migration has been discussed by Bertino et al. [6] in the more general setting of managing next generation federated distributed systems. They introduced a policy framework where they envision distributed management environments where devices can "learn from their environments, be aware of other devices in the environment, self-organize with these devices, and manage themselves." They identify semantic knowledge and advanced cognitive capabilities (e.g., learning and risk assessment) as core features to support, and they made a compelling argument for incorporating such features in any next generation distributed management system. In the next section, we will explore how semantic knowledge and cognitive capabilities play a central role in finding policy approximations.

4 CURRENT APPROACHES TO MIGRATION

As mentioned in the introduction, we look at policy migrations between access control monitors or between network traffic security monitors. We separate the migration methods into two classes. We will first look at migrations where the context of the source device is the same as the context of the target device but the monitors are different. Then, we

will look at migrations where the monitors are the same but the contexts are different. No one has addressed directly the case where contexts and monitors are different simultaneously, and although this case is less common, one could approach such scenarios by first, assuming the contexts are the same, doing a first migration between the different monitors, and then do a second migration assuming that the monitors are of the same type but the contexts change. At these early stages of development on policy migration, we can't predict if this migration composition would be effective.

4.1 Same contexts, different monitors

$[C_s = C_t, EM^s \neq EM^t]$: there has been significant amount of work over the years in the access control community comparing the expressive power of different access control models. From this work we can get (1) how to do faithful migrations from monitors enforcing mandatory and discretionary access control policies to an execution monitor that enforces role-based access control policies [29]; (2) how to do faithful migrations from monitors enforcing role-based access control policies to an execution monitor that enforces relationship-based access control policies [30]; and (3) how to do faithful migrations from monitors of any of these types of policies to an execution monitor that enforces attribute-based access control policies [4, 18]. Migrations in the other directions (e.g., from a monitor enforcing an attribute-based access control policy into a monitor that can enforce role-based access control policies) have not been considered because a faithful migration might not exist, and it is only recently with this new-generation of federated systems that non-faithful migrations raises as a problem worth exploring.

Migration between network monitors has been less common. There are several reasons. If the monitoring is done inside a general purpose computational device, e.g. a server, monitors that work at the kernel level with access to the hardware process traffic very fast because they can get to the packets as they arrive at the physical network interfaces. In addition, they tend to process traffic at what is known as Layer 4 or Transport Layer of the network, they don't access data at higher layers such as the Session or the Application layer – decisions are made one packet at the time by order of arrival. The iptables utility is such a type of monitor. Monitors like Snort that look at the content of the traffic at higher layers usually run at the operating system (aka user) level. They are slower because packets need to move from the kernel to the user level memory, but at the user level there is more processing capacity (memory and CPU time) to implement more expressive policies that can be conditioned on a collection of packets like TCP segments or sessions. If the device is not a server but a network component like a router, a switch or a gateway, the device needs to make only routing decision on large amounts of traffic very fast. Historically, policies in these devices have dealt with traffic at Layer 4 or below – they need to provide high throughput, hence, any policy based on information only available a higher layers has been typically done at the end points of the traffic flow path. The functions implemented by each type of network device were very specific, networks were very stable, and there was no incentive to replace one network device of one type with another of a different type. With faster machines and modern operating systems where kernels can be updated without stopping the device, and with the new developments in network virtualization, network components are able to do more processing over traffic and implement more complex policies. These network components are also many times programmable. In the server side, there have been efforts for migrations or partial migrations of policy enforcement from the user level to the kernel level. Tools like fwsnort [31] that translate Snort policies into iptables are now being developed. The capabilities of iptables are being expanded but the translations are still not faithful. These are engineering efforts that, so far, put aside the question of the quality of the result.

Learning approximations: An approach to non-faithful migration of access control policies is policy mining. In policy mining, one starts with a set A of authorization decisions, i.e. a finite set of pairs $(X, P_s(X))$ for executions X in E_s^ω, and

an objective function (e.g., prefer safe policies that minimize rejections of correct executions) over all policies that can be implemented in EM^t, and works to find an approximation of the policy P_t such that $EM^t_{P_t}$ is most likely to maximize the objective function [10]. This schema is generic and can be applied to any of the access control models mentioned above. If we don't have a set A, we might be able to generate one using $EM^s_{P_s}$ to subsequently mine P_t. This method can also be useful in situations where there is no access to P_s, but there is a set A, for example in log files of the source device. Examples of concrete realizations of such schema are reported in [26, 28] for role-based access control, [21, 39**?**] for attribute-based access control, and [8] for relationship-based access control.

Learning approximations needs to guarantee that the mined target policies are general enough to allow correct decisions in a variety of unseen execution traces, but at the same time not too permissive to accept execution traces that are clearly violations of the target property. Defining or choosing a learning process capable of striking a balance between the number of false positives (i.e., unwanted execution traces that the learned target policy would accept), and the number of false negatives (i.e., desirable execution traces that the target policy would not support) is a challenge, in particular because available examples of policy decisions tend to be incomplete and skewed toward instances of correct decisions rather than instance of wrong decisions. Association rule learning tend to mine from example target policies P_t that are effective only with respect to decisions and execution traces similar to the one already observed. Such learned policies would have a low value of false positive, but be at the same time too specific (overfit the examples), and therefore have a high value of false negatives. True positive rate of such policies would be low and their true negative rate would be high. Symbolic learning approaches, on the other hand, tend to overgeneralise. They would compute target policies that are too permissive: perform well on unseen acceptable execution traces, so reporting a low false negative rate, but, at the same time, wrongly allowing behaviours that should not be accepted, so reporting high false positive rate. In such cases, learned target policies would have high true positive rate and low true negative rate.

A way to control over-generalisation during learning is normally done by considering examples of what would be wrong policy decisions. But log files do not include such negative examples as they reflect what a policy allows or denies, and not what a policy should not allow or not deny. To compensate for the lack of negative examples, learning techniques, such as symbolic learning, can make use of a scoring mechanism, as part of their objective function [21]. This scoring mechanisms express optimisation criteria that, taken into account during the learning process, control the level of generalisation performed during the learning, This helps learn target policies that are general but at the same time not over-permissive, so striking a good balance between their true positive and true negative rates.

Scoring optimisation function can also provide a way for tilting such balance, depending on the specific context in which the policy migration has to take place. For instance, in contexts such as access control, when log files are sparse, more specific rules that guarantee tighter security should be preferred. In this case, during learning the optimisation scoring mechanism would give more preference to target policies that would not wrongly allow access to resources as this might be more dangerous that wrongly denying access. Similarly, in contexts where authorisation is time critical, wrongly denying access could be more dangerous than wrongly allowing access. Scoring optimisation functions provides a way to specify whether the learning search for target policies should be biased towards more or less general policies.

Mining tools, filtering-rule generalization and other symbolic techniques described in [1, 16] could also be adapted for migrations but it is a research topic. And the programmablity of network switches in SDN might make the early explorations of reinforcement learning for routing [7] and other machine learning techniques [33] alternatives that can be considered to support migration of network monitors. We will expand on machine learning and migration in the next section.

4.2 Different contexts, same monitors

$[C_s \neq C_t, EM^s = EM^t]$: context issues related to parametric configurations, like the IP address changes between LANs in the example mentioned earlier, can be handled by an auto-configuration system like the one proposed in [35]. They have proposed a framework in which generic policies are described by templates specified as partially defined context-free grammars in each target device that the device completes according to a local context. It then uses the completed grammar to generate the policies to be enforced by the target device [35]. The context is defined by an interaction graph which is an abstract description of the various entities within the environment that the device needs to interact with. In an interaction graph, a link defines a possibly many-to-many interaction of entities assigned to predetermined roles at the two ends of the link. Policies are meant to be constraints over these interactions. Hence, grammars are written from the point of view of a device that will be assigned one of these roles. In the grammar, there are non-terminals that represent generically an entity in a role. The grammar rule re-writing this non-terminal will only become available at run time when the actual set of devices playing that role are specified or discovered. Any target device may be fulfilling one or more roles in this interaction graph. Let's look at a simple example to illustrate the concepts. There is a server hosting a database that can be accessed by many clients. As protection for SQL injection attacks, the sever also runs Snort to prevent any suspicious access to the database. There are three roles in this example: Client, Network Monitor and Database. The interaction will be C $\rightarrow$ N, C $\leftrightarrow$ D. The server will have two roles, the role of a network monitor and the role of a database. The grammar associated with the database generates policies regarding the access privileges of the different clients (e.g., which client is the database administrator). The context then must provide a mechanism for the database system to identify the clients. But for simplicity, let's look at the partially defined grammar of the Network monitor. This will have a single grammar rule:

$$S \longrightarrow \text{“}\texttt{snort -l /var/log/snort -i}\text{"}\ C$$

In this rule, the non-terminal C must identify to the network monitor where the traffic can be monitored. The server, when deployed, can automatically check which of its network adapters is available for connections and add grammar rules that re-write C into the interfaces associated with the adapters. In this instance, no specific identification of the clients needs to be made, only the physical interfaces of the server that can be used for incoming traffic to the database. But in general, the context supplied to a device playing a particular role is the appropriate identification to the entities that can be at the other side of the links in the interaction graph fulfilling that role. This has two advantages. One is that policy parameters that only depend on the device where the policy will be instantiated can be an autonomous decision of the device. The second advantage is that changes on these parameters are easy to handle locally. For example, if the device decides to change the interface for the traffic, it can regenerate the policy with the new interface. And if this change is needed by the clients to access the database (e.g., the adapter has a different IP address), the change will be sent to the clients as a new context so that they can also regenerate their policies.[4]

There can also be context issues related to language. Let's assume there is a source monitor $EM^s_{P_s}$ that enforces an attribute-based policy, e.g., the events are attribute/value pairs and the policy is a predicate over these attributes to decide access to some resources. On the one hand, there is the problem of value and attribute naming differences due to changes in vocabulary (e.g., different local settings). The language changes from the source device to the target device, and hence, an attribute may need to be retrieved using a different name in the target device. We have studied policy similarities using ontologies [24]. A reverse process could be applied here to guide the search of P_t using an ontology to

[4]Clients that don't trust the database may want to apply policies to their interactions to the database.

drive the mapping. The contexts for the migration will be an ontology O_s that describe general concepts in the source device (this can be independent of P_s), and an ontology O_t for the target device. The migration will proceed as follows. First, a mapping between the concepts in O_s and O_t is selected. Then, using this mapping, attributes and attribute values in P_s are mapped according to the ontology mapping to generate P_t. This mapping might not be unique and we might get several possible P_t to choose from.

On the other hand, if attributes are the same and some attributes are missing in the events in E_t, it is possible that $\mathcal{P}$ was decomposed based on the context of the source and P_s uses attributes that, for privacy consideration, are not accessible from the target device. But it is also possible that the portion of $\mathcal{P}$ that needs to be enforced in the target device doesn't need the missing attributes, i.e., the context is different. If we have access to $\mathcal{P}$, we could expand the techniques of decomposition developed in [23] to generate P_t from $\mathcal{P}$ by doing syntactic analysis of $\mathcal{P}$ and its relationship with C_t. If neither case applies, and EM^t is expected to implement P_s, an approximation needs to be found. We should note that the ontology mapping mentioned above might not find mappings for all the attributes or attribute values also requiring alternative methods of approximation.

Learning approximations: We have seen how C_s and C_t can be useful to get to P_t. But it is possible that there is uncertainty in which are the elements in C_t or that there is no obvious correlation between C_t and the consequences it can have for P_t. An alternative is again to try policy mining to directly learn $EM^t_{P_t}$. The challenge though is that in contrast to the case where $C_s = C_t$, when the contexts are different, the availability of examples in the target context might be limited. In these scenarios, we need to use all the semantic information available, e.g., P_s, C_s, C_t and P^c, in order to learn from a small set of examples. We have developed incremental algorithms for learning grammars symbolically [20] that can repair a grammar using positive and negative examples of words in the language. Our experiments show that this can be significantly faster than learning the grammar from scratch. We could, for example, start with a grammar representation of the automata $EM^t_{P_s}$ and modify it with examples that apply to the target.

With the advances in machine learning, an important category of execution monitors is the class of monitors that implement policies using deep neural networks. Approaches have been proposed specifically for anomaly detection systems for network intrusion detection [33]. An initial evaluation of their use and comparison with decision trees for access control have been recently carried out [9]. However a major issue when using deep neural networks, as other machine learning algorithms, for security purposes is the scarcity of training data, especially when dealing with novel attacks. There is often very little data related to new malware or reliable data for network attacks. The new training data is often expensive and time-consuming to collect. Another issue is whether the neural networks might have to be retrained on a new data from scratch whenever a new malware or attack is identified, which will require a lot of computing resources and time. This issue is not limited to machine learning algorithms applied to security but to the more general issue of transfer learning. Transfer learning enables migrating the learned features and knowledge from a trained source neural network to a target neural network with minimal new training data. For example, an image classification neural network trained to detect 1000 different categories of objects can be re-purposed for a new specialized domain like identifying the various types of roses. Adapting such a general neural network for the task of classifying roses gives much better results than training a neural network on the rose image dataset from scratch. Experiments carried out by Singla et al. [34] on the UNSW-NB15 dataset for network intrusion detection systems [27] has shown that transfer learning is effective for policy migration as well. Singla et al. [34] has shown how an existing neural network trained on identifying a set of given attack types can be retrained to identifying novel attacks even when the training dataset for new attacks is small. The experimental results show that, whereas when trained on a

dataset with more than 300 samples the transfer learning approach has the same accuracy of a conventional approach (i.e., training the network from scratch), the former approach has much higher accuracy for smaller training datasets. For example, for a training dataset of 100 samples the transfer learning approach achieves 19.1% higher accuracy than the conventional approach when the new attack to be identified is a shell-code attack.

5 EVALUATING APPROXIMATIONS

To evaluate approximations we have been borrowing the terminology from hypothesis testing/binary classification and refer to correct executions that are rejected as *false negatives* and the set of executions that violate the policy and are not detected as *false positives*. In a migration over the same context, an execution X is a false negative if $P_s(X)$ holds but $P_t(T(X))$ does not hold; X is a false positive if $P_s(X)$ does not holds, but $P_t(T(X))$ does hold. If analytical estimations of these values cannot be made, we can use a combination of logs from executions of $EM^s_{P_s}$ and synthetic traffic data to statistically estimate these values. The approach then defines an error matrix similar to the error matrix of binary classifiers with definitions such as *true positive and true negative rates, precision, accuracy, etc.* A safe policy will have a true negative rate (true negatives/(true negatives + false positives) equal to 1. A highly available system will have its true positive rate (true positives/(true positives + false negatives)), also called recall, close to 1. But we could further adapt the error matrix to accommodate specific security concerns. If semantic information is available with some qualitative measurement of the severity of the rejections to the availability of the system and the potential attacks caused by the violations not detected, they can be incorporated into the evaluation of the policy. We can also explore scenarios where semantic information exists specifying what violations cannot be allowed under any circumstances or what services cannot be disrupted to evaluate approximations.

In the case of context-dependent migrations, we cannot resort to P_s to define *false negatives* or *false positives*. We need P^c. An execution $X \in e^{\omega}_t$ is a false negative if $P^c(C_t, X)$ holds but $P_t(X)$ does not hold; X is false positive If $P^c(C_t, X)$ does not hold, but $P_t(X)$ does hold. If analytical estimations of these values cannot be made, but P^c is available, using P^c, we can use synthetic data to statistically estimate these values. However, it's possible that we don't have access to P^c either. Under these circumstances we need to generate sample traces for $EM^t_{P_t}$. This might not be an easy task without the intervention of a human administrator, and even for the administrator might be difficult. We can proceed as follows. From logs of executions of $EM^s_{P_s}$, we gather statistical correlations between correct and incorrect executions and attribute values in the context C_s. If there is an ontology mapping that can be applied from C_s to C_t, we can use the mapping to generate synthetic traffic based on the traffic of $EM^s_{P_s}$. If there is no mapping or the mapping is partial (e.g., we have the mapping between attributes but not between attribute values), the administrator is asked to help with the mapping. The questions to the administrator will be limited to the attributes that are found to be relevant to the policy decision.

6 FINAL REMARKS

Although policy migration has come into attention only recently, we can see how earlier work in policy analysis and learning can be adapted for the purpose of migrations. In the short term we need a better and more principled characterization to evaluate and compare policy approximations. Evaluations based on error/confidence matrices are good starting points but there are other aspects that are specific to security that might need to be incorporated. For example, we might want to make sure that a policy is safe for a particular subset of executions and be more flexible for others. For instance, one might be willing to tolerate some inappropriate read operations but not write operations. One can also define some kind of structural complexity and prefer less complex policies. For example, in a Role-based access

control system, one might want to minimize the number of roles used [26]. Similarly to the problem of explainabilty in machine learning, one might want approximations where decisions that don't match expectations can be explained to the system administrators.

Methods for evaluating policies can be studied first in migrations between access control models within the same context. One can start from the unified attribute-based access control model of [18] and develop a framework for coming with automatic translations of access control policies. [18] shows how discretionary (DAC), mandatory (MAC) and role-based (RBAC) access control models can have a unified representation under the attributed-based (ABAC) access control model. One can investigate different translations between any pair of these models. Given the differences in expressive power some of these translations will not be faithful.

We have evidence that policy migration through learning might work well with a few examples of labeled executions. This might allow us to start with a rough, mostly safe, approximation and work to incrementally update the approximations when more data become available. However such initial analysis has been carried out when the source training dataset and the target training dataset have the same set of features. Approaches are needed for dealing with differences in feature sets in the two datasets. One possible approach to address this issue would be to use adversarial generative networks for domain adaptation.

Another area where learning can help is by establishing relationships between contexts. For example, it might be possible that we have context information in the form of ontologies but no mapping between them. Providing labeled data to learn the mapping might be much easier than generating synthetic positive and negative execution traces.

There are two important aspects of policy migration that we have not mentioned but also need attention. To carry a migration we need to trust the migration process and the execution monitor. There is a need for the development of trusted migration protocols. This must include trusted communication channels among the participants in the migration (source, destination and any potential third party required during the migration) as well as the attestation of the execution monitor and its implementation of the policy. These issues have already arisen in the context of Trusted Platform Modules (TPMs) in Virtual Machines and Virtual Machine migrations. Trusted computing in Virtual Machines is based on a secure virtualization of the TMP supported by the host computer. But if the Virtual Machine migrates to a different physical server with a different TMP then a trusted protocol that preserved the trust of the execution in the new host machine must be followed. This is the topic of [11]. It is an open question whether such protocols can be independent of the execution monitors.

Acknowledgements

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. Jorge Lobo was also supported by the Spanish Ministry of Economy and Competitiveness under Grant Numbers TIN201681032P, MDM20150502, and the U.S. Army Research Office under agreement number W911NF1910432.

REFERENCES

[1] Muhammad Abedin, Syeda Nessa, Latifur Khan, Ehab Al-Shaer, and Mamoun Awad. 2010. Analysis of firewall policy rules using traffic mining techniques. *IJIPT* 5, 1/2 (2010), 3–22. https://doi.org/10.1504/IJIPT.2010.032611

[2] Dakshi Agrawal, Seraphin Calo, Kang-won Lee, Jorge Lobo, and Dinesh Verma. 2008. *Policy technologies for self-managing systems.* Pearson Education.

[3] Dakshi Agrawal, Kang-Won Lee, and Jorge Lobo. 2005. Policy-based management of networked computing systems. *IEEE Communications Magazine* 43, 10 (2005), 69–75.

[4] Tahmina Ahmed, Ravi Sandhu, and Jaehong Park. 2017. Classifying and comparing attribute-based and relationship-based access control. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, 59–70.

[5] David Basin, Vincent Jugé, Felix Klaedtke, and Eugen Zălinescu. 2013. Enforceable security policies revisited. *ACM Transactions on Information and System Security (TISSEC)* 16, 1 (2013), 3.

[6] Elisa Bertino, Seraphin Calo, Maroun Toma, Dinesh Verma, Christopher Williams, and Brian Rivera. 2017. A cognitive policy framework for next-generation distributed federated systems: concepts and research directions. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 1876–1886.

[7] Justin A. Boyan and Michael L. Littman. 1993. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In *Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993]*. 671–678. http://papers.nips.cc/paper/770-packet-routing-in-dynamically-changing-networks-a-reinforcement-learning-approach

[8] Thang Bui, Scott D. Stoller, and Jiajie Li. 2019. Greedy and evolutionary algorithms for mining relationship-based access control policies. *Computers & Security* 80 (2019), 317–333. https://doi.org/10.1016/j.cose.2018.09.011

[9] Luca Cappelletti, Stefano Valtolina, Giorgio Valentini, Marco Mesiti, and Elisa Bertino. 2019. On the Quality of Classification Models for Inferring ABAC Policies from Access Logs. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 4000–4007.

[10] Carlos Cotrini, Luca Corinzia, Thilo Weghorn, and David A. Basin. 2019. The Next 700 Policy Miners: A Universal Method for Building Policy Miners. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 95–112. https://doi.org/10.1145/3319535.3354196

[11] Boris Danev, Ramya Jayaram Masti, Ghassan O Karame, and Srdjan Capkun. 2011. Enabling secure VM-vTPM migration in private clouds. In *Proceedings of the 27th Annual Computer Security Applications Conference*. 187–196.

[12] Bart de Schuymer and Nick Fedchik. 2003. Ebtables/Iptables interaction on a Linux-based Bridge.

[13] Cynthia Dwork. 2008. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*. Springer, 1–19.

[14] Christopher Gerg and Kerry J Cox. 2004. Managing security with Snort and IDS tools. (2004).

[15] Mario Gerla, Eun-Kyu Lee, Giovanni Pau, and Uichin Lee. 2014. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 241–246.

[16] Korosh Golnabi, Richard K. Min, Latifur Khan, and Ehab Al-Shaer. 2006. Analysis of Firewall Policy Rules Using Data Mining Techniques. In *Management of Integrated End-to-End Communications and Services, 10th IEEE/IFIP Network Operations and Management Symposium, NOMS 2006, Vancouver, Canada, April 3-7, 2006. Proceedings*. 305–315. https://doi.org/10.1109/NOMS.2006.1687561

[17] Kevin W Hamlen, Greg Morrisett, and Fred B Schneider. 2006. Computability classes for enforcement mechanisms. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 28, 1 (2006), 175–205.

[18] Xin Jin, Ram Krishnan, and Ravi Sandhu. 2012. A unified attribute-based access control model covering DAC, MAC and RBAC. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 41–55.

[19] Leslie Lamport. 1977. Proving the correctness of multiprocess programs. *IEEE transactions on software engineering* 2 (1977), 125–143.

[20] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. 2019. Representing and Learning Grammars in Answer Set Programming. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2919–2928. https://doi.org/10.1609/aaai.v33i01.33012919

[21] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. 2020. FastLAS: Scalable Inductive Logic Programming Incorporating Domain-specific Optimisation Criteria. In *The Thirty-fourth AAAI Conference on Artificial Intelligence, AAAI 2020, NY, NY, USA, February 7 – 12, 2020*. AAAI Press.

[22] Jay Ligatti, Lujo Bauer, and David Walker. 2005. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security* 4, 1-2 (2005), 2–16.

[23] Dan Lin, Prathima Rao, Elisa Bertino, Ninghui Li, and Jorge Lobo. 2008. Policy decomposition for collaborative access control. In *13th ACM Symposium on Access Control Models and Technologies, SACMAT 2008, Estes Park, CO, USA, June 11-13, 2008, Proceedings*. 103–112. https://doi.org/10.1145/1377836.1377853

[24] Dan Lin, Prathima Rao, Rodolfo Ferrini, Elisa Bertino, and Jorge Lobo. 2013. A Similarity Measure for Comparing XACML Policies. *IEEE Trans. Knowl. Data Eng.* 25, 9 (2013), 1946–1959. https://doi.org/10.1109/TKDE.2012.174

[25] Jorge Lobo. 2019. Relationship-based access control: More than a social network access control model. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 9, 2 (2019). https://doi.org/10.1002/widm.1282

[26] Ian Molloy, Hong Chen, Tiancheng Li, Qihua Wang, Ninghui Li, Elisa Bertino, Seraphin B. Calo, and Jorge Lobo. 2010. Mining Roles with Multiple Objectives. *ACM Trans. Inf. Syst. Secur.* 13, 4 (2010), 36:1–36:35. https://doi.org/10.1145/1880022.1880030

[27] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Military Communications and Information Systems Conference (MilCIS), 2015*. IEEE, 1–6.

[28] Qun Ni, Jorge Lobo, Seraphin Calo, Pankaj Rohatgi, and Elisa Bertino. 2009. Automating role-based provisioning by learning from examples. In *Proceedings of the 14th ACM symposium on Access control models and technologies*. ACM, 75–84.

[29] Sylvia L. Osborn, Ravi S. Sandhu, and Qamar Munawer. 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.* 3, 2 (2000), 85–106. https://doi.org/10.1145/354876.354878

[30] Edelmira Pasarella and Jorge Lobo. 2017. A Datalog Framework for Modeling Relationship-based Access Control Policies. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2017, Indianapolis, IN, USA, June 21-23, 2017*, Elisa Bertino, Ravi Sandhu, and Edgar R. Weippl (Eds.). ACM, 91–102. https://doi.org/10.1145/3078861.3078871

[31] Michael Rash. 2007. *Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort.* No Starch Press.

[32] Fred B Schneider. 2000. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)* 3, 1 (2000), 30–50.

[33] Ankush Singla and Elisa Bertino. 2019. How deep learning is making information security more intelligent. *IEEE Security&Privacy* 17, 3 (2019), 56–65.

[34] Ankush Singla, Elisa Bertino, and Dinesh C. Verma. 2019. Overcoming the Lack of Labeled Data: Training Intrusion Detection Models Using Transfer Learning. In *IEEE International Conference on Smart Computing, SMARTCOMP 2019, Washington, DC, USA, June 12-15, 2019*. IEEE, 69–74. https://doi.org/10.1109/SMARTCOMP.2019.00031

[35] D Verma, S Calo, Supriyo Chakraborty, Elisa Bertino, Christopher Williams, J Tucker, and Brian Rivera. 2017. Generative policy model for autonomic management. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 1–6.

[36] Mahesh Viswanathan. 2000. Foundations for the run-time analysis of software systems. (2000).

[37] Md Whaiduzzaman, Mehdi Sookhak, Abdullah Gani, and Rajkumar Buyya. 2014. A survey on vehicular cloud computing. *Journal of Network and Computer Applications* 40 (2014), 325–344.

[38] Christopher Williams, Elisa Bertino, S Calo D Verma, K Leung, and C Dearlove. 2017. Towards an architecture for policy-based management of software defined coalitions. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 1–6.

[39] Zhongyuan Xu and Scott D. Stoller. 2015. Mining Attribute-Based Access Control Policies. *IEEE Trans. Dependable Sec. Comput.* 12, 5 (2015), 533–545. https://doi.org/10.1109/TDSC.2014.2369048