Computers & Security 123 (2022) 102949

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

An automated closed-loop framework to enforce security policies from anomaly detection

João Henriques[a,b,c], Filipe Caldeira[a,b,c], Tiago Cruz[a,*], Paulo Simões[a]

[a] University of Coimbra, CISUC, Department of Informatics Engineering, Coimbra 3030-290, Portugal
[b] Informatics Department, Polytechnic of Viseu,Viseu 3504-510, Portugal
[c] CISeD – Research Centre in Digital Services, Polytechnic of Viseu, Portugal

ARTICLE INFO

Article history:
Received 13 June 2022
Revised 11 September 2022
Accepted 5 October 2022
Available online 8 October 2022

Keywords:
Automation
Policy as code
Decision trees
Machine learning
Zero-touch network and service management (ZSM)

ABSTRACT

Due to the growing complexity and scale of IT systems, there is an increasing need to automate and streamline routine maintenance and security management procedures, to reduce costs and improve productivity. In the case of security incidents, the implementation and application of response actions require significant efforts from operators and developers in translating policies to code. Even if Machine Learning (ML) models are used to find anomalies, they need to be regularly trained/updated to avoid becoming outdated. In an evolving environment, a ML model with outdated training might put at risk the organization it was supposed to defend.

To overcome those issues, in this paper we propose an automated closed-loop process with three stages. The first stage focuses on obtaining the Decision Trees (DT) that classify anomalies. In the second stage, DTs are translated into security Policies as Code based on languages recognized by the Policy Engine (PE). In the last stage, the translated security policies feed the Policy Engines that enforce them by converting them into specific instruction sets. We also demonstrate the feasibility of the proposed framework, by presenting an example that encompasses the three stages of the closed-loop process.

The proposed framework may integrate a broad spectrum of domains and use cases, being able for instance to support the *decide* and the *act* stages of the ETSI Zero-touch Network & Service Management (ZSM) framework.

1. Introduction

Due to the growing complexity and scale of IT systems, there is an increasing need to automate and streamline routine maintenance and security management procedures, to reduce costs and improve productivity. As a result, approaches such as the European Telecommunications Standards Institute (ETSI) Zero-touch network & Service Management (ZSM) (ETSI, 2019) are becoming increasingly popular.

Such approaches enable greater consistency and uniformity and contribute to significantly enhancing the efficiency of Operations & Maintenance (O&M) activities. Moreover, they may result in cost savings and a significant reduction in human errors. Similar approaches also occur in software development practices, with the widespread adoption of agile techniques for reducing the time allocated to the software development cycle and IT operations, leading to the well-known concept of DevOps (Bass et al., 2015). The addition of security management as a third pillar, complementing development and operations, characterizes the emerging field of DevSecOps.

In the scope of DevSecOps methodologies, policies are a fundamental instrument to accelerate the application of best practices, since they potentially enable the automated adaptation of code and operations to cope with new threats, changes in the network topology, new services, etc. Policies can express the desired system behavior in high-level general terms, and be later translated into specific lower-level rules applicable to the configuration of each specific component of the system.

Once security holes are found, the design, implementation and application of specific security policies require significant efforts from operators and developers. They need to design the policies and translate them to rules, code or other artifacts. This burden in-

* Corresponding author at: University of Coimbra, CISUC, Department of Informatics Engineering, Coimbra 3030-290, Portugal.
E-mail addresses: jpmh@dei.uc.pt (J. Henriques), tjcruz@dei.uc.pt (T. Cruz).

https://doi.org/10.1016/j.cose.2022.102949

creases even more in the case of large organizations. Also, the verification of policies by humans is time-consuming, and the required time significantly increases with the complexity of the infrastructure. This is aggravated by the fact that rules may not exist *a priori*, being created and evolved as data becomes available (Decker et al., 2020).

Frequently, policies are enforced by directly embedding them in source code. Many existing policies or Access Control Lists (ACLs) are set by the use of options in user interfaces, which is not an easily repeatable or versionable task. This is inefficient and makes it difficult to keep up to date inventories, also hampering automated testing. Moreover, ACLs usually lack support for auditing or checking if policies are being violated.

Translating policies expressed in natural language into formalized documents, in formats understandable by both humans and machines, can be challenging. Such formalized documents provide guidance and enhance readability, testability and reportability. However, such documents are still high-level, lacking the specific mappings into the configurations and tools used in the target domain, making it difficult to directly convert them into actionable actions.

A possible approach to overcome this problem is to take the concept of putting code in a high-level language to manage and automate the enforcement of policies, known as Policy as Code (PaC). This is a relatively new concept that helps decoupling the enforcement decisions from business logic policies. Describing policy logic directly in code, rather than depending on a natural language, may help documenting the reasons for those policies, by extending them using comments. PaC may help converting configuration policies into readable formats easily editable, auditable and reproducible by IT managers. Further, they can be translated into intermediate languages recognized by Policy Engines (PEs). PaC offers the opportunity to have policies incrementally refined and versioned, to support automating activities. Similar to code, it is possible to include in PaC the programming constructs that determine decisions, helping to automate the enforcement of policies. Moreover, PaC may be reviewed and checked by automated tests, reducing the need of human-based testing operations. The PaC concept can be applied to different domains, such as security, software development and IT operations rules and processes.

In general, ML may help generating source code (Hireche et al., 2022; Murali et al., 2017; Riftadi et al., 2019; Yuan and Banzhaf, 2018). Specifically in domain of anomaly detection, Decker et al. (2020) described a real-time evolving solution based on a fuzzy rule-based classification model for log-based anomaly detection. Henriques et al. (2020) also highlighted how ML models can generate sets of rules at scale from unknown data. Overall, these works inspired the approach presented in this paper.

It is evident that the evolving threat landscape requires the introduction of new approaches for deployment, monitoring and assessment of security policies. Inspired by ZSM Zero-touch principles and the aforementioned works, we propose a continuous automated closed-loop relying on three stages. Firstly, to extract the Decision Trees (DTs) from ML models to identify the anomalies. Secondly, translating them to policies. Thirdly, enforcing them along with the different system components. This continuous closed-loop makes it possible update policies along time with the most recent data. The ML model produces DTs that identify the anomalies to be translated to PaC in a language recognized by the PE. This way, it is possible to reduce the human effort associated to defining and writing the policies to be enforced.

The remainder of this paper is organized as follows. Section 2 introduces the background and related concepts. Section 3 addresses related work. Section 4 presents the proposed closed-loop framework. Section 5 describes a proof-of-concept implementation of the proposed framework. Section 6 describes the experiments we performed to assess the proposed framework. Section 7 provides an overall discussion of the framework and validation experiments. Finally, Section 8 concludes the paper.

2. Background

This section starts by presenting base concepts such as policies and PaC, followed by a discussion of several technologies that support PEs.

In our framework, policies specify the conditions under which particular activities should be allowed, to enable logic-based enforcement decisions. Policies include conditions such as rules providing fine-grained control and governing activities for a specific domain (e.g., network security policies; periods under which deployments are allowed), representing the conduct to be evaluated

Policies may cover a large number of use cases. For example, to follow the best practices of data security according to the Payment Card Industry Data Security Standard (PCI-DSS) Payment Card Industry Security Standards Council (2022), or to enforce the best secure coding practices, such as the Open Web Application Security Project OWASP (2022), the Computer Emergency Response Team (CERT) C Secure Coding Standard Seacord (2008), the Common Weakness Enumeration CWE (2022) and the Common Vulnerabilities and Exposures (CVE) CVE (2022) recommendations, the Defense Information Systems Agency's National Vulnerability Database (DISA NVD) NIST (2022a) and the Common Vulnerability Scoring System (CVSS) NIST (2022b).

The PaC concept was inspired by Donald Knuth's notion of literate programming (Knuth, 1984), driven by the need to document programs to non-technical people. PaC also takes the best practices from Infrastructure as Code (IaC) on the automatic configuration of system dependencies (Rahman et al., 2019). Conceptually speaking, IaC relies on scripted workflows that are used to configure software systems and cloud instances at scale, in a secure manner. However, despite the evident potential of IaC for security purposes, recent literature reviews (Rahman et al., 2019) found no works specifically addressing security applications.

PaC should be learnable and writable by humans with no programming skills, including those responsible for implementing, updating and auditing them. PAC's machine-readable language can be applied programmatically to improve efficiency along with the development and deployment cycles. PaC allows to automatically audit the deployed systems and check their compliance, to detect gaps and quickly apply fixes. This efficiency results from the use of libraries of policies as templates for new applications and infrastructure environments. PaC also reduces the number of errors, because code and deployments can be tested before being run, decreasing implementation/deployment risks and costs. With a test sandbox, IT managers can also check policy changes against the entire policy stack, to ensure (i) modifications do not break the existing rules and (ii) there are no situations not covered by any rules.

Moreover, PaC leverages the application of consistent and accountable processes over time. Since policies are encoded in text files, it is possible to manage their lifecycle by using a Version Control System (VCS) such as git, taking advantage of features such as history, diffs, pull requests, and a central location for storing policies across platforms and applications. The VCS contributes to reusing code and helps to define modular policies that can be aggregated into comprehensive Policy Engines, to test policies on an isolated test or development environment before deploying them to production systems. The policies maintained by VCS can integrate the existing CI/CD development pipelines to automate approval, to ensure software compliance and to enable a tight feedback loop between developers and reviewers.

PaC can help documenting policies (which become self-documented), controls and best practices. It can be used to define the security policies to be enforced, including firewall rules, applications, resources or data access controls, data encryption rules, and code provenance restrictions. Thus, PaC also helps Software Bill of Materials assessment and tracking, in the scope of software supply chain risk management.

Enforcing policies is as important as defining and documenting them. Similarly to software compilers, PEs translate PaC into implementations (e.g., network security configuration, autorization control policies or Kubernetes cluster parameters) in different environments. PEs provide the capability to systematically check if a rule is broken. A PE includes the mechanisms to automatically check logical inconsistencies, syntax errors, and missing dependencies. The PE takes decisions by evaluating inputs against policies and data. PEs should be generic enough to be applied to different scenarios, combining context-specific data with the higher-level policies, to enforce them according to each specific context.

PaC and PE can be used in IaC platforms to enforce infrastructure provisioning and deployment policies such as container cluster parameters and constraints in workload placement. IaC software might query the PE to take decisions before provisioning (e.g. depending on the type of node, storage, network dependencies, and application being targeted) – thus, they also help restricting access to infrastructure and enforcing rationalization policies.

Several tools are available for implementing PEs. Kyverno (2022a), for instance, is designed specifically for Kubernetes, managing policies as Kubernetes resources which can be generated, validated and mutated. Pulumi Crossguard (2022) works with cloud management tools for AWS, Azure, Google Cloud and Kubernetes. The Open Policy Agent (2022) is open-source and includes a high-level declarative language for writing PaC.

Azure PaC Microsoft (2022) is one of the few PaC software tools currently available for cloud environments. It can be used to define policies affecting firewall rules, application, resource or data access limits, data encryption rules, or code provenance constraints (among others), which are stored on a VCS and tested upon change.

Sentinel Kyverno (2022b) is a policy language and a framework designed to be integrated into applications, providing an automated test framework enabling continuous integration. HashiCorp Consul (2022), Nomad (2022), Terraform Liyanage et al. (2022), and Vault Project (2022) rely on Sentinel functionalities.

A recent example of a standard built upon a closed-loop management approach is ETSIs Zero-touch Network and Service Management (ZSM) ETSI (2019); Liyanage et al. (2022), an End-to-End (E2E) reference architecture that uses feedback-driven processes to achieve intelligent automated and management functionalities.

3. Related work

This section discusses previous work addressing automated and dynamic policy-based approaches somehow related with the scope of our proposal.

Moore and Childers (2013) presented a ML solution to automatically generate program affinity policies that consider program behavior and the target machine. Similarly, Quiroz et al. (2010) relied on unsupervised algorithms to capture the dynamic behavior of systems and the hidden relationship between the high-level business attribute space, and the low-level monitoring space. Similarly, Pelaez et al. (2016) used supervised models to capture the dynamic behavior.

Johansen et al. (2015) proposed a mechanism for expressing and enforcing security policies for shared data expressed as stateful meta-code operations defined in scripting languages interposed in the filesystem.

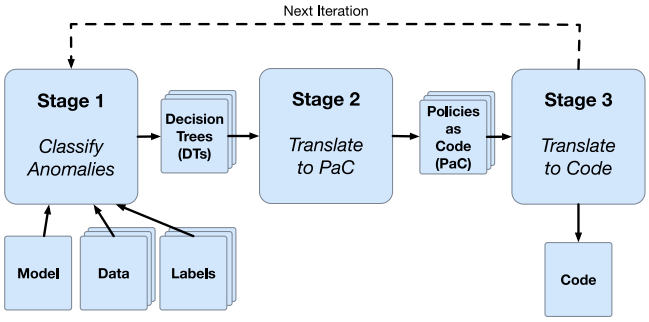

Fig. 1. Proposed Framework.

Gheibi et al. (2021) reviewed the state of the art on the use of ML in self-adaptive systems based in the traditional Monitor-Analysis-Planning-Executing (MAPE) Kephart and Chess (2003) feedback loop. Weyns et al. (2021) presented an approach combining MAPE and Control Theory to produce better adaptive systems.

Finally, the more recently contributions on use of ML models supporting the automation of self-adaptive IT operations has emerged a new field (AIOps) IBM (2022); Litoiu et al. (2021) while their contributions have been organized in a taxonomy by Notaro et al. (2021).

Out proposal suggests going a step further in the AIOps automation approach, by extending it to the security field (AISecOps). As explained next, it introduces a translation stage integrated within a closed feedback loop pipeline for simultaneously filling the gap and leveraging the benefits of decoupling ML model training and the security policies to be enforced.

4. Proposed framework

This section presents the proposed closed-loop framework that allows to create a workflow that automates the end-to-end process that goes from the classification of anomalies to translational policy rule generation and subsequent enforcement. As illustrated in Fig. 1, the proposed continuous closed-loop model S^n relies on a three-stage loop which is applied along n iterations, as formulated in (1).

$$S^n = \{S_1^n, S_2^n, S_3^n\} \tag{1}$$

The adoption of a closed-loop helps reducing the security risks arising from organizations with outdated security rules. The continuous workflow keeps deployed rules updated, by taking into account the most recent monitoring data to adjust the notion of anomaly, and to automatically adjust deployed rules based on the retrained ML models (more specifically DTs, in the case of our proposal) generated in this way.

The first stage (S_1), automatically takes into consideration new incoming data to classify security anomalies. A DT model fits the data to classify the anomalies. At the second stage (S_2), the previously generated DTs are translated into PaC rules in a format recognized by the PE. These rules bring together the conditional logic and the granular controls. Finally, at the third stage (S_3), the produced PaC is enforced by PE. The next cycle may be triggered periodically or based on specific events which, by their nature, might require rule adjustments. Next, we discuss in more detail each stage.

4.1. First stage

The first stage (S_1) takes as input: the DT ML family of algorithms $M_{DT}()$ (e.g. Random Forest, XGBoost); input data D_S organized according to the schema S; and optional labels J (e.g., in case

of supervised learning models) to obtain the DTs as T_S, according to (2).

$$S_1 : (M_{DT}, D_S, J) \rightarrow T_S \tag{2}$$

The realization of this first stage can be based, for instance, on the unsupervised learning model proposed in Henriques et al. (2020). This model identifies the DTs classifying the anomaly R_a, and non anomaly R_n events from unlabeled data. as denoted in Algorithm 1. Therefore, the overall list of DTs

Algorithm 1 Unsupervised Learning Model.

INPUT: D_S, Data

```
clusters ← 2
K ← KMeans(clusters)
Y ← K.Train(D_S)
X ← XGBoost(D_S, Y)
X.train()
ypred ← X.Predict(D_S)
R_1, R_2 ← X.DecisionTrees()
for all i ∈ ypred do
    if ypred_i > 0.5 then
        ypred_i^1 ← 1
        k2 ← k2 + 1
    else
        ypred_i^1 ← 0
        k1 ← k1 + 1
    end if
end for
if k1 > k2 then
    R_a ← R_2
    R_n ← R_1
else
    R_a ← R_1
    R_n ← R_2
end if
```

OUTPUT: R_a, Anomaly Decision Trees
OUTPUT: R_n, Non Anomaly Decision Trees

T_S combines the R_a and R_n to be included as input for the second stage, according to (3).

$$T_S = R_a \bigcup R_n. \tag{3}$$

It should be noted that our framework does not propose to separate the rules and then to gather them again. Instead, the union presented in (3) denotes the ability of the framework to integrate classification models. This is achieved by integrating the resulting rules from an unsupervised learning model into the framework Decision Tree (T_S) set. In this case, we highlight that T_S can plug a binary classification model by integrating the anomalies (R_a) and non-anomalies (R_n) rules into the T_S set.

4.2. Second stage

The second stage represents the key contribution of the proposed framework. A mapping function $S_2()$ receives as input the DTs T_S produced by the first stage and outputs policies P_S, according to (4).

$$S_2 : T_S \rightarrow P_S \tag{4}$$

Each policy P_S is defined by a set of rules, as per (5).

$$P = \{R_i\}_{i=1}^{n} \tag{5}$$

Each policy has associated an identification, a name, a description, and a level of enforcement $P^{(i,n,m)}$. It denotes a logical disjunction of n Boolean rules R_i, as described in (6).

$$P^{(i,n,m)} = R_1 \vee R_2 \vee \cdots \vee R_n = \bigvee_{i=1}^{n} R_i \tag{6}$$

According to the circumstances, a rule R_i denotes the conjunction of either positive or negative disjunctions of specific attribute levels, as denoted by (7).

$$R_i = \bigwedge_{k} S_k \tag{7}$$

The policies P target the domain data D_S (including the events $e_k \in D_S$) expressed using the schema S, according to (8). The schema S is a set of features $a_k \in S$.

$$D_S = \bigcup_{k=1}^{n} e_k \tag{8}$$

A set of logical operators (eg. AND, OR, NOT) helps defining complex rules R_i, and $\bigvee$ and $\bigwedge$ represent the Boolean algebra operators OR and AND. Using these operators, it is possible to construct other operators, such as “CONTAINS”, ”IN”, ”IS”, or ”MATCHES”. Moreover, l_k^j refers to one of the logical parts of a statement S_k about the jth attribute. Thus, the statement is composed of two distinct parts (9).

$$S_k = n_k \bigvee_{j} l_k^j \tag{9}$$

The first part is the disjunction of level values with l_k^j the jth level of the attribute a_k. The second part is the parameter $n_k \in [1, \neg]$, which allows negating (logical operator NOT) the disjunction when set to $\neg$. The user enters specific rules specifying the levels l_k^j and the parameters n_k, as expressed in (10).

$$R_i = \bigwedge_{k} (n_k \bigvee_{j} l_k^j) \tag{10}$$

A policy P will be checked by function $X()$ with data D_S and a set of rules or a policy P. This check produces a Boolean classification telling whether the Policy is being met or not (11).

$$X : (P, D_S) \rightarrow K \tag{11}$$

It should be noticed that the model can have different levels of enforcement $L = \{l_w, l_s, l_m\}$. At (default) mandatory level l_m, the policy must be complied, regardless of the circumstances and can not be overridden. In the warning level l_w, the failure of policies is allowed and just produces a warning to the user. The intermediary soft level l_s applies to policies that can be overridden to support the configuration of exceptions. Therefore, the enforcement levels $l \in L$ are also input to function $X()$, as described in (12).

$$X : (P, D_S, l) \rightarrow K, l \in L \tag{12}$$

4.3. Third stage

In the third and final stage (S_3), the PE translates the policy P_S resulted from the previous stage (4) into native code C_p, expressed in a programming language p to be deployed for enforcement purposes (13).

$$S_3 : P_S \rightarrow C_p \tag{13}$$

5. Proof-of-concept implementation

This section presents a Proof-of-concept (PoC) implementation of the framework, which demonstrates its practical feasibility by producing PaC rules from the identification of anomalies to be enforced by a PE.

This PoC was developed for spam detection use case scenarios, in email systems. According to these scenarios, an IT manager dictates a high-level rule to block suspect (spam) messages. Nevertheless, the objective is not to require the IT manager to specifically express how messages are classified as spam.

5.1. First stage

First, a DT classification model $M_{DT}()$ fits the incoming data. In our PoC, for instance, we used a labeled dataset of emails Biswas (2022) (originally created from Cohen, 2022) to train the model, obtaining DTs from the anomaly classification process.

Function $M_{DT}()$ is used to train a ML model with the email dataset as input data D_S and corresponding labels J in schema S, to obtain T_S (cf. Eq. (2)). The resulting DTs T_S provide the logical steps for classifying anomalous emails (label 0) and non-anomalous emails (label 1), as illustrated in Listing 1.

5.2. Second stage

Next, Sentinel Kyverno (2022b) is used as the domain-agnostic policy language. A mapping function was implemented to translate the previous DTs T_S into Sentinel policies P_S, therefore filling the role of the S_2 function from (4). These Sentinel policies are sets of rules defined with key-value pairs, with the main rule with a test.

Listing 2 shows the Sentinel policy to classify class 0 (spam email), while Listing 3 represents the Sentinel policy to classify class 1 (regular email).

In our PoC we created an instance of the sklearn Pedregosa et al. (2011) *DecisionTreeClassifier* algorithm and then it was initialized with "maximum depth" set to 20. The dataset fit to this model was split with 80% for training and 20% for tests. Each word in the email dataset corresponds to a distinct feature. The function *export_text()* provided the rules from the DTs resulting from the training stage.

```
|--- feature_13 <= 0.50
|   |--- feature_916 <= 0.50
|   |   |--- feature_92 <= 0.50
|   |   |   |--- feature_37 <= 0.50
|   |   |   |   |--- feature_418 <= 0.50
|   |   |   |   |   |--- feature_36 <= 0.50
|   |   |   |   |   |   |--- feature_81 <= 0.50
|   |   |   |   |   |   |   |--- feature_104 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_68 <= 0.50
|   |   |   |   |   |   |   |   |   |--- feature_107 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- feature_1139 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |   |--- feature_1139 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |--- feature_107 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- feature_535 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |   |--- feature_535 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1
```

Listing 1. Decision Trees for Email Classification.

```
main = rule {(feature_13 <= 0.50 and feature_916 <= 0.50 and
    feature_92 <= 0.50 and  feature_37 <= 0.50 and
    feature_418 <= 0.50 and feature_36 <= 0.50 and
    feature_81 <= 0.50 and feature_104 <= 0.50 and
    feature_68 <= 0.50 and  feature_107 <= 0.50 and
    feature_1139 > 0.50)
    or
    (feature_13 <= 0.50 and feature_916 <= 0.50 and
    feature_92 <= 0.50 and  feature_37 <= 0.50 and
    feature_418 <= 0.50 and feature_36 <= 0.50 and
    feature_81 <= 0.50 and feature_104 <= 0.50 and
    feature_68 <= 0.50 and  feature_107 > 0.50 and
    feature_535 <= 0.50)}
```

Listing 2. Sentinel Policy for class 0 (spam email).

```
main = rule {(feature_13 <= 0.50 and feature_916 <= 0.50 and
    feature_92 <= 0.50 and feature_37 <= 0.50 and
    feature_418 <= 0.50 and feature_36 <= 0.50 and
    feature_81 <= 0.50 and feature_104 <= 0.50 and
    feature_68 <= 0.50 and  feature_107 <= 0.50 and
    feature_1139 <= 0.50)
    or
    (feature_13 <= 0.50 and feature_916 <= 0.50 and
    feature_92 <= 0.50 and feature_37 <= 0.50 and
    feature_418 <= 0.50 and feature_36 <= 0.50 and
    feature_81 <= 0.50 and feature_104 <= 0.50 and
    feature_68 <= 0.50 and feature_107 > 0.50 and
    feature_535 > 0.50)}
```

Listing 3. Sentinel Policy for class 1 (regular email).

5.3. Third stage

Finally, the previously produced PaC P_S is translated to a language C_p recognized by the PE, according to the function S_3 referred in (13).

A test folder was created for the policy to be run, and a file with that policy defined in JavaScript Object Notation (JSON) format is stored in that folder. Since Sentinel allows to define one policy per class (anomalies and non-anomalies), two policies were created. Finally, policies were moved to a Github repository to streamline the PoC with versioning, continuous deployment and pull request capabilities.

For real-use scenarios, the PoC can be integrated into CI/CD tool-chains. Within a continuous integration pipeline, for example, it is possible to run a specific command translating a Sentinel PaC into an artifact containing the email rules that the email server understands.

6. Validation

The validation of the proposed framework is not straightforward, because its potential benefits result mainly from the operational gains obtained over time, in terms of cost of keeping rules updated and (indirect) accuracy improvements – which are not easy to measure.

To fully assess the performance of the proposed framework, we would need datasets whose rules had evolved over a significant period of time (so that new types of cyberattacks or new types of spam email would start appearing only after some time), so that we could measure the improvements brought by the automated adjustment of the rules over time, and also the ability to preserve (or even increase) the system accuracy.

Since we had no such datasets available, we devised a different but still relevant experiment. Starting with a publicly available dataset with spam email Biswas (2022) (created from Cohen, 2022), we performed the following experiment:

- First, we split the dataset in six different blocks with similar sizes (block 0, block 1, block 2...). These blocks emulate the emails received during six consecutive periods (e.g., one week).
- We used the block 0 to train both our platform and a baseline system. This would be similar, for instance, to the initial training of the system with the emails from the previous week.
- Afterwards, we tested the accuracy of the trained system with block 1 as input – this could represent, for instance, the first week of emails with our framework running.

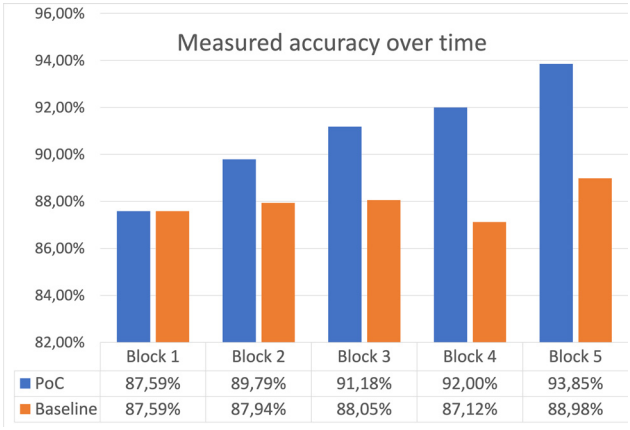

Fig. 2. Measured accuracy over time for PoC and baseline systems.

- Next, our PoC performed an automatic readjustment, based on the original training and on the updates induced by the inputs from block 1 (i.e. the first week). This corresponds to the first automatic readjustment of the rules. The baseline system used for comparison kept using the original training data.
- Then, we kept repeating the process for the next blocks, so that our PoC kept automatically refining the rules. This could correspond, keeping the analogy, to having 5 weeks of operation with weekly updates.

The accuracy obtained in each of these steps is presented in Fig. 2. Overall, these results are in line with what we expected. For the baseline system, accuracy remained stable (with slight natural fluctuations), around 87–89%. When using our approach, the system kept improving accuracy over time, since the data from the previous period was used to further refine the models. It should be noted, however, that in real world operations we expect results to be slightly different: while baseline (i.e. static) systems are expected to degrade their accuracy over time (due to the appearance of new types of spam or cyberattacks not present in the original training data), our approach is expected to preserve accuracy over time, adjusting to those changes.

7. Discussion

This work was inspired by the ideas of translating policies to code that are present in several works Decker et al. (2020); Hireche et al. (2022); Murali et al. (2017); Riftadi et al. (2019); Yuan and Banzhaf (2018), also aligning with the Zero-touch concept of the ETSI ZSM framework. It supports a closed-loop with the intelligence and automation of the tasks of monitoring and detecting the ongoing threats, to produce the security policies to be enforced.

The presented PoC, based on a simple but representative use case, shows how this approach can be applied in practice, to streamline the security operations associated with keeping spam email filters up-to-date. The first stage classifies spam emails as anomalies, extracting the DTs that identify spam messages as anomalies. Next, policy rules are generated, by means of translating those DTs into PaC. Finally, those PaC can be used by email servers to block new spam emails.

This process is cyclic, and can be triggered at regular time intervals or based on specific events. Emails classified by users (as spam or not spam) are used to progressively update applied policies. Automating these process reduces the operators' burden by streamlining routine maintenance and security management procedures.

The adopted policy engine in the proposed framework enables decoupling policies from the applications that will enforce them. Moreover, it may be integrated with other tools, for instance to identify threats and take automatic responses on stopping attacks in progress or introducing defensive actions.

The proposed framework helps automating repetitive operation tasks related with updating and enforcing policy rules. This potentially improves productivity and reduces the continuous effort of maintaining the systems' security up-to-date. Moreover, the time required to apply new security rules is shortened, reducing the time the systems are exposed to outdated policies.

Translating DTs into PaC contributes to the readability of those policy rules by human operators, while not requiring specific programming skills. The presented PoC can be generalized to fit other anomaly detection scenarios requiring frequent updates. The framework can also be applied to automatically update and enforce forensics and compliance auditing mechanisms.

Despite the potential benefits of the proposed framework, it should be noted that some drawbacks may arise. First, relying on an automatic enforcement from newly generated policies, generated from ML models, in some cases may result in a significant number of false positives. This may be attenuated by prior validation by humans before enforcing those policies, at the cost of some degradation in the process streamlining levels. Second, despite the benefits brought by PaC, some compromises apply regarding performance and flexibility. Performance can be compromised because, typically, PaC does not support unsafe operations (such as direct memory access) or operations (such as sub-process execution). In terms of flexibility, PaC may result in a limited offer in terms of programming languages.

8. Conclusion

This work proposed a closed-loop framework aiming to reduce the evolving security risks organizations are exposed to, by streamlining the routine maintenance and management of security policies.

The presented PoC demonstrates how it can be applied in practice. Beyond the PoC scenario, the framework can be applied to a wide range of other use cases. In practice, any security monitoring scenario with evolving threats and evolving systems, where the criteria to identify anomalies need to evolve over time, can benefit from this framework. General policy based management scenarios, in dynamic environments, may also benefit from the proposed approach, since it enables the streamlining of access policies updates without requiring formal specification of those policy updates and/or their manual translation into code.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

João Henriques: Conceptualization, Methodology, Investigation, Data curation, Writing – original draft. **Filipe Caldeira:** Conceptualization, Methodology, Investigation, Writing – review & editing, Supervision, Funding acquisition. **Tiago Cruz:** Conceptualization, Methodology, Investigation, Writing – review & editing, Supervision. **Paulo Simões:** Conceptualization, Methodology, Investigation, Writing – review & editing, Supervision, Funding acquisition.

Acknowledgments

This work was partially funded by National Funds through the FCT—Foundation for Science and Technology, I.P., and the European Social Fund, through the Regional Operational Program Centro 2020, within the scope of the projects UIDB/05583/2020 and CISUC UID/CEC/00326/2020. It was also partially co-funded byFEDER, via the Competitiveness and Internationalization Operational Program (COMPETE 2020) of the Portugal 2020 framework, in the scope of Project Smart5Grid (POCI-01-0247-FEDER-047226).

Furthermore, would also like to thank the Research Center in Digital Services (CISeD) and the Polytechnic of Viseu for their kind support.

References

Agent, O. P., 2022. Open policy agent. https://www.openpolicyagent.org/.
Bass, L., Weber, I., Zhu, L., 2015. DevOps: A Software Architect's Perspective. Addison-Wesley Professional.
Biswas, B., 2022. Spam emails dataset. Visited on 2022-04-10. https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv.
Cohen, W. W., 2022. Ernron email dataset. Visited on 2022-08-19. https://www.cs.cmu.edu/~enron/.
Consul, 2022. Sentinel in consul. Visited on 2022-04-01. https://www.consul.io.
Crossguard, 2022. Crossguard. Visited on 2022-04-10. https://www.pulumi.com/crossguard/.
CVE, 2022. Common vulnerabilities and exposures. Visited on 2022-03-01. https://cve.mitre.org.
CWE, 2022. Common weakness enumeration. Visited on 2022-03-01. https://cwe.mitre.org.
Decker, L., Leite, D., Giommi, L., Bonacorsi, D., 2020. Real-time anomaly detection in data centers for log-based predictive maintenance using an evolving fuzzy-rule-based approach. In: Proceedings of the IEEE International Conference on Fuzzy Systems, pp. 1–8. doi:10.1109/FUZZ48607.2020.9177762.
ETSI, G., 2019. Zero-touch network and service management (ZSM); reference architecture. Technical Report. https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/002/01.01.01_60/gs_ZSM002v010101p.pdf.
Gheibi, O., Weyns, D., Quin, F., 2021. Applying machine learning in self-adaptive systems: a systematic literature review. ACM Trans. Auton. Adapt. Syst. 15 (3). doi:10.1145/3469440.
Henriques, J., Caldeira, F., Cruz, T., Simões, P., 2020. Combining k-means and xgboost models for anomaly detection using log datasets. Electronics 9 (7). doi:10.3390/electronics9071164.
Hireche, O., Benzaïd, C., Taleb, T., 2022. Deep data plane programming and ai for zero-trust self-driven networking in beyond 5g. Comput. Netw. 203, 108668.
IBM, 2022. Ibm pak for aiops. Visited on 2022-09-01, https://www.ibm.com/cloud/cloud-pak-for-watson-aiop.
Johansen, H.D., Birrell, E., van Renesse, R., Schneider, F.B., Stenhaug, M., Johansen, D., 2015. Enforcing privacy policies with meta-code. In: Proceedings of the 6th Asia-Pacific Workshop on Systems. Association for Computing Machinery, New York, NY, USA doi:10.1145/2797022.2797040.
Kephart, J.O., Chess, D.M., 2003. The vision of autonomic computing. Computer 36 (1), 41–50.
Knuth, D.E., 1984. Literate programming. Comput. J. 27 (2), 97–111.
Kyverno, 2022a. Kyverno. Visited on 2022-04-10, https://kyverno.io/.
Kyverno, 2022b. Sentinel. Visited on 2022-04-10, https://www.hashicorp.com/sentinel.
Litoiu, M., Watts, I., Wigglesworth, J., 2021. The 13th cascon workshop on cloud computing: engineering aiops. In: Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering. IBM Corp., USA, pp. 280–281.
Liyanage, M., et al., 2022. A survey on zero touch network and service management (ZSM) for 5g and beyond networks. J. Netw. Comput. Appl. 203, 103362. doi:10.1016/j.jnca.2022.103362.
Microsoft, 2022. Design azure policy as code workflows. Visited on 2022-04-05, https://docs.microsoft.com/en-us/azure/governance/policy/concepts/policy-as-code.
Moore, R.W., Childers, B.R., 2013. Automatic generation of program affinity policies using machine learning. In: Jhala, R., De Bosschere, K. (Eds.), Compiler Construction. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 184–203.
Murali, V., Qi, L., Chaudhuri, S., Jermaine, C., 2017. Neural sketch learning for conditional program generation. arXiv preprint arXiv:1703.05698.
NIST, 2022a. National vulnerability database. Visited on 2022-03-01, https://nvd.nist.gov/.
NIST, 2022b. Vulnerability metrics. Visited on 2022-03-01, https://nvd.nist.gov/vuln-metrics/cvss.
Nomad, 2022. Nomad. Visited on 2022-04-01, https://www.nomadproject.io.
Notaro, P., Cardoso, J., Gerndt, M., 2021. A systematic mapping study in aiops. In: Hacid, H., Outay, F., Paik, H.-y., Alloum, A., Petrocchi, M., Bouadjenek, M.R., Beheshti, A., Liu, X., Maaradji, A. (Eds.), Proceedings of the Service-Oriented Computing-ICSOC Workshops. Springer International Publishing, Cham, pp. 110–123.
OWASP, 2022. OWASP. Visited on 2022-03-01, https://www.owasp.org.
Payment Card Industry Security Standards Council, 2022. Payment card industry data security standard - requirements and testing procedures, v4.0.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al., 2011. Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12 (Oct), 2825–2830.
Pelaez, A., Quiroz, A., Parashar, M., 2016. Dynamic adaptation of policies using machine learning. In: Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 501–510. doi:10.1109/CCGrid.2016.64.
Project, V., 2022. Vault. Visited on 2022-04-01, https://www.vaultproject.io/docs/enterprise/sentinel.
Quiroz, A., Parashar, M., Gnanasambandam, N., Sharma, N., 2010. Autonomic policy adaptation using decentralized online clustering. In: Proceedings of the 7th international conference on Autonomic computing, pp. 151–160.
Rahman, A., Mahdavi-Hezaveh, R., Williams, L., 2019. A systematic mapping study of infrastructure as code research. Inf. Softw. Technol. 108, 65–77. doi:10.1016/j.infsof.2018.12.004.
Riftadi, M., Oostenbrink, J., Kuipers, F., 2019. Gp4p4: enabling self-programming networks. arXiv preprint arXiv:1910.00967.
Seacord, R.C., 2008. The CERT C Secure Coding Standard. Pearson Education.
Weyns, D., Schmerl, B., Kishida, M., Leva, A., Litoiu, M., Ozay, N., Paterson, C., Tei, K., 2021. Towards better adaptive systems by combining mape, control theory, and machine learning. In: Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE, pp. 217–223.
Yuan, Y., Banzhaf, W., 2018. ARJA: automated repair of java programs via multi-objective genetic programming. IEEE Trans. Software Eng. 46 (10), 1040–1067. doi:10.1109/TSE.2018.2874648.

João Henriques is a PhD student in Science and Information Technology at the University of Coimbra (UC) and Assistant Professor at the Department of Informatics Engineering at the Polytechnic Institute of Viseu (IPV). His research interests at the Center for Informatics and Systems (CISUC) at UC include forensic and audit compliance for critical infrastructures protection.

Filipe Caldeira is an Adjunct Professor at the Informatics Department of the Polytechnic Institute of Viseu, Portugal. He obtained his PhD degree in Informatics Engineering in 2014 from the Faculty of Sciences and Technology of the University of Coimbra. He acts as program director of the Informatics Engineering program since 2014. He is also a researcher at the Centre for Informatics and Systems of the University of Coimbra and at the CI&DETS research centre of the Polytechnic Institute of Viseu. He has been recently involved in some international and national research projects. His main research interests include ICT security, namely, policy-based management, trust and reputation systems, Security and Critical Infrastructure Protection.

Tiago Cruz received his Ph.D. degree in informatics engineering from the University of Coimbra (Coimbra, Portugal), in 2012. He has been an Assistant Professor in the Department of Informatics Engineering, University of Coimbra, since December 2013. His research interests include areas such as management systems for communications infrastructures and services, critical infrastructure security, broadband access network device and service management, Internet of Things, software-defined networking, and network function virtualization (among others). He is the author of more than 80 publications, including chapters in books, journal articles, and conference papers. Dr. Cruz is a senior member of the IEEE Communications Society.

Paulo Simões received the Doctoral degree in informatics engineering from the University of Coimbra (Coimbra, Portugal), in 2002. He is an Associate Professor in the Department of Informatics Engineering, University of Coimbra, where he regularly leads technology transfer projects for industry partners such as telecommunications operators and energy utilities. His research interests include network and infrastructure management, security, critical infrastructure protection, and virtualization of networking and computing resources. He has more than 150 publications in refereed journals and conferences. Dr. Simões is a senior member of the IEEE Communications Society.