International Journal of Computer Applications (0975 - 8887)
Volume 187 - No.130, July 2026

A Hybrid and Adaptive Architecture for Explainable Intrusion Detection in Constrained IoT Environments

Nkondock Mi Bahanag Nicolas
University of Yaounde I
Faculty of Science
Department of Computer Sciences

Bayem Jacques Narcisse
University of Yaounde I
Faculty of Science
Department of Computer Sciences

Atsa Etoundi Roger
University of Yaounde I
Faculty of Science
Department of Computer Sciences

ABSTRACT

Internet of Things (IoT) environments are increasingly vulnerable to cyberattacks, necessitating effective and explainable intrusion detection solutions. This paper proposes a hybrid and adaptive architecture for explainable intrusion detection in constrained IoT environments. The approach combines workflow mining techniques with security rule-based detection to identify intrusions. It allows to avoid false positives during intrusion detection. Here, explainability also use methods to provide insights into detection decisions. More, the solution is hybrid due to the use of several techniques and also because it allows the detection of intrusions locally or through the Internet. The defined architecture offers advantages such as scalability and flexibility.

General Terms

Intrusion detection, IoT

Keywords

Intrusion Detection, Hybrid Architectures, Adaptative Architectures, Explainable Intrusion Detection, Constrained IoT Environments.

1. INTRODUCTION

The current digital landscape is marked by an unprecedented proliferation of the Internet of Things (IoT), with forecasts reaching 40 billion devices by 2030. These objects, essential to Smart Cities, connected health, and Industry 4.0, generate massive volumes of sensitive data, creating a vast and complex attack surface. Unlike traditional computer networks, IoT devices often operate in constrained environments (limited power, low computing power, limited memory). This hardware "lightweight" makes it difficult to host sophisticated defense mechanisms, leaving these nodes vulnerable to various network attacks (DDoS, code injection, zero-day attacks). [1][2][3] Traditional security solutions facing a dual challenge :

(1) Performance vs. Resources : Traditional IDS based on signatures[4] or deep learning are often too resource-intensive to run locally on sensors.

(2) The "Black Box" Problem : While Artificial Intelligence (AI) [5][6] improves accuracy, the opacity of complex models hinders the confidence of analysts who struggle to interpret the reasons for an alert.

To overcome these limitations, research is shifting towards hybrid approaches, combining signature detection (for speed) and AI-powered behavioral analysis (for unknown threats). Adaptability is becoming crucial for adjusting the system's responsiveness in real time according to the context (e.g., dynamic sliding windows) and optimizing energy consumption. The integration of Explainable AI (XAI), via methods like SHAP or LIME, is now recognized as a key element for transforming model predictions into human-understandable decisions. In an IoT environment, this transparency is vital for validating autonomous decisions made in real time and ensuring network resilience.
This paper proposes an explainable intrusion detection architecture based on workflow mining and security rules that enable intrusion detection in a constrained IoT environment. The following section presents the framework of the architecture.

2. STATE OF THE ART

Intrusion detection in constrained IoT environments requires reconciling three constraints : limited resources, traffic heterogeneity, and the need for explainable alerts. Existing research is structured around four main areas.

(1) **Lightweight IDS for Microcontrollers** Two strategies are favored. Rule engines inspired by Snort/Suricata, ported to ContikiOS or FreeRTOS, guarantee consistent detection time and a small memory footprint ($<$ 10 KB RAM), but are limited to known attacks. TinyML allows for the use of compressed models : SVM, KNN, or quantified Random Forest achieve $>$ 90% F1 on IoT-23 with less than 20 KB RAM. However, these approaches suffer from a lack of adaptability to traffic changes.[7][8][9]

(2) **Hybrid Signature + Anomaly Architectures** Hybridization aims to detect both known and zero-day attacks. Cascading schemes place rules on the front line to filter out benign traffic, then a lightweight SVM or Isolation Forest classifier handles the remaining traffic, reducing CPU load by up to 60%. Other work combines federated learning and edge detection to preserve data confidentiality. These approaches have two limitations : retraining often remains centralized

in the cloud, and the merging of decisions degrades the readability of the alert.[10][11][12].

(3) **Explainability of IoT IDS** Trust in the system requires understandable decisions. Intrinsically interpretable models, such as quantized decision trees, offer complete transparency on TON_IoT with nearblackbox performance. Post-hoc SHAP or LIME methods can explain neural network decisions on Edge-IIoTset, but their computational cost makes them difficult to use on sensors. Extracting rules from a trained model is an intermediate approach, at the cost of reduced fidelity.[13][14] [15]

(4) **Adaptability to Conceptual Drift**
IoT traffic evolves with the addition of new devices and protocols. Incremental learning via Hoeffding trees allows for inline updates on the gateway without complete retraining. Drift detectors like ADWIN trigger retraining only when there is a significant change in the distribution. Regarding rule bases, clustering-based approaches dynamically add new signatures, but at the risk of increasing system redundancy and complexity.[16][17] [18] [19]
Identified gap
Despite these advances, no work to date proposes a unified architecture combining lightweight hybrid detection, full edge adaptation, and native explainability compatible with the constraints of IoT devices. This combination constitutes the main scientific challenge addressed by our work.

3. CONCEPTUAL FRAMEWORK

In this section, we propose a framework of hybrid and adaptive architecture for explainable Intrusion Detection in Constrained IoT Environments. We begin by specifying the different concepts handled. These are the different classes of objects that we model in our constrained IoT environment so that they function normally. We represent them by Abstract Data Types. As a formalism, we represent concepts by tuples for which we provide the signature of the concept manipulation operations as well as the axioms. The framework is obtained based on the different models constructed. We are updating the proposal we made in [1] for the resolution of our problem. For the rating, we adopt the following bases :

(1) **Concepts** are represented by character strings, the first of which is capitalized.

(2) **Operations** have a lowercase first character.

(3) **The expression : func : A1*A2... An→B** denotes the signature of the function **func** and is understood as follows : The operation **func** takes as parameters the concepts **A1, A2, ..., An** and returns **B**.

(4) As types of concept manipulation operations, we have :
   (a) **Constructors,**
   (b) **Observers,**
   (c) **Transformers,**
   (d) **and Predicates**

(5) **Predicates** are functions that returns a Boolean value (true or false)
In our case, the proposed conceptualization is designed in such a way that it can allow the resolution of our problem. In addition, the literature does not agree on a consensus that allows a unique definition of each of these concepts. Here, we have proposed a conceptualization that takes into account most of the aspects related to the Information Systems implemented within companies in order to allow us to address our problem.

3.1 Definition of concepts.

3.1.1 The Task

. In the operation of a given system, many activities are performed in the form of tasks. Observing these tasks allows us to understand how processes evolve within the automated information systems. Formally, an activity or simply task can be represented by a tuple in the following way :
< **Task, TaskName, TaskDesc, Pre, Pos** >
Where :
Task is the task identifier,
TaskName The name of the task,
TaskDesc the task description,
Pre is the Pre-condition of the task, in other words, what conditions the execution of a task and Post, the post-condition. What happens after a task is executed.
The different tasks that run in an environment manipulate a set of resources and can also modify their state.
As an example of tasks on a file, we could have :

(1) **Opening**

(2) **Creation**

(3) **Modification**

(4) **Closing**

(5) **Deletion**

The **Precondition** here indicates the prerequisite for the file in question to be able to open.
The **Post-condition** provides information on the state of the resource (file here) after the task is executed. Functions of the Task class.

Constructors :

task :→Task
//Function that allows you to create a default task.

task : TaskName∗ TaskDesc *Pre∗ Pos →Task
//Function that allows you to create a task based on parameters.

Observers :

taskName : Task→TaskName
//Function that takes a task as input and returns its name.
taskDesc : Task→TaskDesc
//Function that takes a task as input and returns its description.
pre : Task→Pre
//Function that takes a task as input and returns the precondition for its completion.

pos : Task→Pos
//Function that takes a task as input and returns the post-condition after its completion.

Transformers :

setTaskName : Task *TaskName→Task
//Function that takes a task and a name as input, then returns a new task.

setTaskDesc : Task *TaskDesc→Task
//Function that takes a task and a description as input, then returns a new task.

setPre : Task *Pre→Task //Function that takes as input a task and a precondition for its completion then returns a new task.

setPos : **Task*Pos→Task**
//Function that takes as input a task and a post-condition for its execution then returns a new task.

setParam : Task ∗ TaskName ∗ TaskDesc ∗ Pre ∗ Pos→Task
//Function that takes a task and a set of parameters as input, then returns a new task.

Predicates :

isTaskTrue : Task→Bool
//Function that takes a task and says whether it was executed successfully.
isPreTrue : Task→Bool
//Function that takes a task and says whether its precondition is true.
isPosTrue : Task→Bool
//Function that takes a task and says whether its postcondition is true.

Properties of class functions Task :

$\forall$**t : Task, isTaskTrue(t)** $\Longrightarrow$ **isPreTrue(t)**
// For any task t, if it is true, then its precondition is true.

$\forall$***t : Task, isPosTrue(t)*** $\Longrightarrow$ ***isTaskTrue(t)***
//For any task t, if its post-condition is true, then the task is true.

3.1.2 The Quality of Service
. When technologies are implemented within an Information System, the main objective for a company is customer satisfaction and improving the quality of service. This notion in our case, which finds its interest in IT security, takes on its meaning when the security rules and mechanisms are respected. Thus, we model the concept of quality of service according to the criteria chosen by the people concerned. In other words, there is no universal notion that guarantees the quality of service but we describe it as the degree of satisfaction of an activity or a process executed in the environment of an Information System. Based on these observations, we have come to propose the following model of quality of service :

$<$ **QoS, Cr, Val** $>$

Where :
Cr represents a list of criteria considered for evaluating a quality of service,
QoS the identifier of the quality of service,
Val the set of values that can be assigned to these criteria.
It is worth noting that the quality of service we have just defined applies to the resources of the Information System because they are the ones being observed and used. An example here might be a server for which the quality of service could be assessed :

(1) Criteria can include speed, efficiency, robustness, etc., depending on sensitivity (since they cannot be the same in all environments).
(2) The values are those assigned to these criteria. For speed, the following

values can be used depending on the response time :

— **Very Slow**
— **Slow**
— **Average**
— **Fast**
— **Very Fast**
— **Exceptional**

The functions of the QoS class :

Constructors :

qoS: **→QoS**

//Function that allows you to create a default Quality of Service.
qoS : Cr * Val→QoS
//Function that allows you to create a Quality of Service based on parameters.

Observers :

cr : **QoS →Cr**
// A function that takes a Quality of Service as input and returns its criteria can take.

Transformers :

setCr : Qos * Cr →QoS

//Function that takes as input a Quality of Service and a set of criteria, then returns a new Quality of Service.

setVal : QoS * Val →QoS
//Function that takes as input a Quality of Service and a set of values, then returns a new Quality of Service.
setValCr : QoS ∗ Val ∗ Cr→QoS
//Function that takes as input a Quality of Services, a set of values and a set of criteria, then returns a new Quality of Services.

Predicates :

isQoSTrue : QoS→Bool
//Function that takes a Quality of Service and says whether it is compliant.
isValIn : Val ∗ QoS→Bool
//Function that takes a Quality of Service and values and says whether the latter can be taken by the Quality of Service criteria.
isPosTrue : Cr * QoS ß Bool
//Function that takes a Quality of Service and says whether the criteria in the parameter are part of it.

Properties of class functions QoS :

$\forall$**v** : **Val**, $\forall$***q : QoS***, ***isValIn(v,q)*** $\Longrightarrow$ **v** $\in$ ***val(q)***

// For any value v and any quality of service q, if the function isValIn(v,q) is true, then v is a value that can take the criteria of q.

$\forall$**c** : **Cr**, $\forall$**q :QoS**, **isCrIn(c,q)** $\Longrightarrow$ **c** $\in$ **cr(q)**

//For any criterion ***c***, for any quality of service ***q***, if the function **isCrIn(c,q)** is true, then **c** is a criterion of **q**.

3.1.3 The resource
. As mentioned above, computer security, and particularly intrusion detection, is implemented to protect access to the information system. Access, which must be in accordance with the security

policy, aims to protect the resources within the information system. These resources are the target of computer attacks. Formally, we describe a resource as follows :

$<$ **Res, ResDesc, SubRes, Type** $>$
Where :
Res is the resource identifier,
ResDesc the description of the resource,
Type the type of resource (human, material, software, etc.),
SubRes The sub-resources of the resource in question.
Note : In this model, we describe resources recursively.
An example is an intrusion detection within an environment [3] .For the case of Host-based intrusion detection, the resource can be a local disk for which access is governed by a security policy. If we consider detection at the disk level in question, then it will involve controlling the disk at its root and its subfolders and this for the entire disk tree. Important : A sub-resource is also a resource.
A resource is either a hardware device or a software construct that facilitates programmed computer tasks. For example, an operating system uses the resources provided by the hardware to create the environment in which applications run and the user interacts with the software to complete computing tasks. One of the main resources is RAM. If there is not enough physical RAM for the computing tasks at hand, performance suffers significantly because the operating system will ”swap” portions of inactive RAM to storage on the hard drive while it processes active RAM. It continues the process of swapping RAM in and out of storage on the hard drive as needed to continue its tasks. If this happens excessively, it is clear that more RAM is needed to ensure the CPU has the resources it needs to operate efficiently. This is a basic example. In a multitasker, managing operating system resources is detailed and complex and is the primary task of the operating system kernel. A computer's resources include things like RAM, CPU usage, and hard drive usage. Imagine opening the Google Chrome browser. The actual application resides on the hard drive and therefore needs to run and read data. There also needs to be some RAM usage for the program to run. The same goes for graphics cards in games. CPU usage is pretty much the biggest factor. When there's a lack of RAM, things become super slow. However, if the CPU is failing but still struggling, there may be a bottleneck in things like the GPU. Resources are exactly what the computer needs to be able to perform whatever action you do
In most cases, the term ”system resources” is used to refer to the amount of memory or RAM your computer has. System resources can also refer to the software installed on your computer. This includes programs, utilities, fonts, updates, and other software installed on your hard drive. For example, if a file installed with a certain program is accidentally deleted, the program may not open. The error message may read, ”The program could not be opened because the necessary resources were not found.”

Functions of the Res class :

Constructors :

res :→Res
//Function that allows you to create a default Resource.

res : ResDesc $*$ SubRes $*$ Type→Res
//Function that allows you to create a Resource based on parameters.

Observers :

resDesc : Res→ResDesc
//Function that takes a Resource as input and returns its description.
subRes : Res→SubRes
//Function that takes a Resource as input and returns its set of sub-resources.
type : res→Type
//Function that takes a Resource as input and returns its Type.

Transformers :

setResDesc : Res $*$ ResDesc→Res
//Function that takes a Resource and a description as input, then returns a new Resource.
setSubRes : Res $*$ SubRes→Res
//Function that takes as input a Resource and a set of sub-resources, then returns a new Resource.
type : Res $*$ Type→Res
//Function that takes a Resource and a type as input, then returns a new Resource. **setParamRes : Res $*$ ResDesc $*$ SubRes $*$ Type→Res**
//Function that takes as input a Resource, a set of resource parameters, then returns a new Resource.

Predicates :

isFinalRes : Res→Bool
//Function that takes a Resource and says if it is terminal (knowing that there are intermediate resources).
isResTrue : Res→Bool
//Function that takes a Resource and says if it is compliant.
isSubResEmpty : Res→Bool
//Function that takes a Resource and says if its sub-resources are empty.

Properties of functions of the class Res :

(1) $\forall$**r :SubRes, r** $\in$ **Res**

(2) $\forall$ **r :Res, isFinalRes(r)** $\Longrightarrow$ **isSubResEmpty(r)**

3.1.4 The Event, the Log

. An event extends the notion of a task by providing information about a set of information related to an execution. The event mainly describes which resource is executing a task, which resource(s) are manipulated for this execution, and the period in which this occurs. This period describes the start date and the end date.
Formally, an event can be represented by the following tuple :
$<$ **Event, Task, ResResp, ResUsed, Period, QoS** $>$
Where :
Task is the task identifier,
ResResp represents the resource that executes the task,
ResUsed describes the set of resources used for this event,
And **Period** is the time interval during which the event occurs.
Here, the period is broken down into a start date and an end date.
A Log is simply a collection of events. In other words, an event file or event log is a file in which line after line of events relating to a specific resource or set of resources are stored.

Functions of the Event class :

Constructors :

event :→Event
//Function that allows you to create a default event.

event : Task * ResResp * ResUsed * Period * QoS→Event
//Function that allows you to create an event based on parameters.

Observers :

task : Event→Task
//Function that takes an event as input and returns the associated task.
resResp : Event→ResResp
//Function that takes an event as input and returns the responsible resource.
resUsed : Event→ResUsed
//Function that takes an event as input and returns the resource used.
period : Event→Period
//Function that takes an event as input and returns the execution period.
qoS : Event→QoS
//Function that takes an event as input and returns the Quality of Service.

Transformers :

setTask : Event * Task→Event
//Function that takes as input an event and a set of criteria, then returns a new event.
setResResp : Event * ResResp→Event
//Function that takes an event and a resource as input, then returns a new event.
setResUsed : Event * ResUsed→Event
//Function that takes an event and a resource, then returns a new event.
setPeriod : Event * Period→Event
//Function that takes an event and a Period, then returns a new event.
setQoS : Event * QoS→Event
//Function that takes an event and a Quality of Service, then returns a new event.

Predicates :

isEventTrue : QoS→**Bool**
//Function that takes an event and says if it is compliant.
isPeriodValid : $Period$→**Bool**
//Function that takes a Period and says if it is valid, i.e. the start occurs before the end.
isResRespRecognize : ResResp→Bool
//Function that takes a Resource and says if it is in our resource set.

Properties of the functions of the Event class :

$\forall$**v : Val** $\wedge$ $\forall$**q : QoS, isValIn(v, q)**$\Longrightarrow$ **v** $\in$ **val (q)**
// For any value v and any quality of service q, if the function isValIn(v,q) is true, then v is a value that can take the criteria of q.
$\forall$**c : Cr, q : QoS, isCrIn(c, q)**$\Longrightarrow$ c cr(q)
// For any criterion c, and any quality of service q, if the function **isValIn(v,q)** is true, then c belongs to **cr(q)**

3.1.5 The workflow
. A workflow represents a flow of work. In other words, it is a series of tasks performed by manipulating resources, some producing documents that can be used by others, with the ultimate goal being the provision of a specific service.
In another way, a workflow can be considered as an instance of a business process.
The goal of workflow observation is to ensure the traceability of documents processed during the execution of a process. Formally, we describe a workflow as follows :
$<$ **Wf, TaskSet, ResSet, Period, BeginingPost, EndingPost, QoS** $>$
Where,
Wf is the workflow identifier,
TaskSet all the tasks completed,
Period the overall workflow execution time interval (which can also be estimated by the sum of the execution times of the individual tasks),
QoS is the quality of service produced,
BeginingPost the starting post of the workflow and,
EndingPost The last post in the workflow.
A workflow is a defined series of tasks within an organization to produce a final result.
Sophisticated workgroup computing applications allow you to define different workflows for different types of work. At each stage of the workflow, a person or group is responsible for a specific task.

3.1.6 The business Process
. A business process is a set of tasks or activities that follow one another and are executed by manipulating a set of resources to produce a specific result and achieve a specific degree of satisfaction in terms of quality of service.
We describe a business process by the following tuple :
$<$ **Bp, TaskSet, Period, ResSet, QoS** $>$
Where :
Bp is the business process identifier,
TaskSet the set of tasks used,
Period The process execution period,
ResSet The set of resources required to execute the process,
QoS The Quality of Service of the process.
As a side note, analyzing a set of workflows allows us to understand the associated business process.
Thus, we can also define a business process Bp as follows :
$<$ **Wf_Set, QoS** $>$
Where Wf_Set represents a set of workflows.
Now that we have described the different concepts that we are going to manipulate for our modelling, the next step is to present the model that will relate these multiple notions.

3.1.7 The security rule
. Security policy is established based on a company's strategy at a given time and can vary over time. It is defined by a list of rules such as those that define :

(1) The list of resources authorized to perform a task.
(2) The authorized period for using a given resource.
(3) The set of resources that can modify another resource during a predefined period. In this case, a rule can be conceptualized by the following tuple :

$<$ **Rule, Task, ResSet, PeriodB, PeriodEnd** $>$
1. Task $\rightarrow$ ResSet The list of resources authorized to perform a task.
2. Res $\rightarrow$ Period The period allowed to use a given resource.
3.Res $\rightarrow$ ResSet * Period The set of resources that can modify another at a predefined period.
In the implementation, at the validation prototype level, we used text files in which the separator is ";".
A set of parameters is enclosed in parentheses and separated by

commas.
If we consider our three rules and :

— Resources : Res1, Res2, Res3, Res4.
— Tasks : T1, T2, T3.
— Periods : P1, P2, P3, P4, P5.

The security policy can be represented as follows :

T2 ; Res1 ; Res4 ; Res3
Res3 ; P2 ; P3 ; P4
Res2 ;(Res3, P2) ; (Res4, P1)

In fact, these three example rules are initially expressed in natural language according to one of the security standards for defining security policy. We then translate them so that we can integrate them into our intrusion detection model.

3.1.8 Security Policy

. A Security Policy is a set of rules.
Conceptually, a security policy can be described as follows :
SP $<$ **SP, Rule_Set** $>$;
Where :
SP refers to the security policy identifier and **Rule** refers to a rule in the security policy.
As operations on the security policy, we can add a rule, modify one, add a set of rules, delete one or more rules. So, we have the following operations :

As constructors we have :

sP :→**SP**
//Function that allows you to create a default security policy.
sP : Rule_Set→SP
//Function that allows you to create a security policy based on parameters.

As setters we have :

addRule : Rule * SP→SP
//Function that takes as input a rule and a security policy, then adds that rule to the given security policy.
UpdRule : Rule * SP→SP
//Function that takes as input a rule and a security policy, then modifies that rule into the specified security policy.
addsetRules : SP * Rule_set→SP
//Function that takes a security policy and a set of rules, then adds them to the security policy.
delsetRules : SP * Rule_set→SP
//Function that takes a security policy and a set of rules, then removes them from the security policy.

3.1.9 The Information System

. The intrusion detection we carry out is done within an Information System. As we have defined it, it comes down to a set of resources, processes, and procedures put together for the purpose of collecting, processing, storing, or disseminating information. Considering this definition, we describe an Information System in this way :
$<$ **SI, Res_Set, Log_Set, Bp_Set, SP** $>$
Where :
Res_Set is a set of resources (of different natures),
Log_Set a set of event files and
BP_Set a set of business processes of the Information System considered.
SP the Information System Security Policy.

3.2 The Commutative Diagram of Concepts

To continue to explain our hybrid and adaptative framework for explainable intrusion detection in IoT constraints environments, we propose in this section a model relating the entities allowing us to carry out this detection using Workflow Mining, this is the commutative diagram of the framework concepts. It is worth noting that we model the security policy by taking into account the strategic objectives of a company in terms of quality of service. Subsequently, we show the formula used in a real way to detect intrusions in an Information System by solving our problem. Finally, the general algorithm is shown to describe the steps to follow to use our solution.
Figure 1 illustrates a commutative correlation diagram of the manipulated concepts.
As a definition, writing **X**→**Y** means that the element **X** can be totally or partially defined by the elements of **Y**.

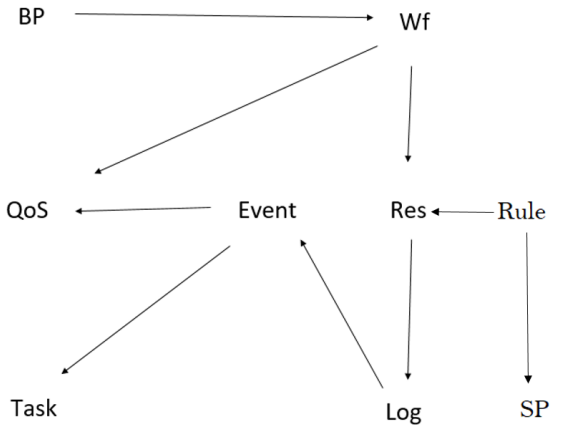

FIG. 1. Commutative diagram of the model's concepts

Properties :

(1) The transitivity property is respected, when, if $X \to Y$ and $Y \to Z$, then $X \to Z$
(2) The commutative diagram is well defined by the fact that it does not have a cycle.

3.3 The security policy model

The security policy depends on the enterprise and is a list of rules. Rules Generation follows the previous definition of concepts. For each of them, we generate getters (functions that return values of attributes) and setters (functions that allow modifications of attributes). We can also have certain rules specified by the information system management team like the following :

$<$ **SP= SP, R1, R2, R3... Rn** $>$
For instance :
R1 : Task$-$ $>$ **ResSet** The list of Resources able to perform a task.
R2 : Res$-$ $>$ **Period** The period authorized to use a given resource.

R3 : Res$-$ > ResSet * Period The set of resources that can modify another one at a predefined period.
We can then define a list of rules that represent security policy.

3.4 Detection Formula

An event ***Ev*** violates the security policy ***SP*** if it violates at least one of the rules described in ***SP***.
Here, we accept the notation ***EvSP*** to describe this fact.
Thus, in an Information System, the set ***ID*** of intrusions is obtained by the following formula :

$$ID = \cup Ev_{i,j} \epsilon (\cup Log(Res_j)), Ev_{i,j} SP$$

Where 1<=i<=n represents the event counter and 1<=j<=m is the resource counter.

Considering that our intrusion detection is resource-centric, $Ev_{i,j}$ represents the event Ev_i that occurred in the Information System related to resource j.

In other words, by browsing all the logs of the Information System ($Log(Res_j)$), the events considered as intrusive will be those that violate the security policy **SP** . Now, we will describe the new intrusion detection algorithm that we propose, based on Workflow Mining[20].

3.5 The proposed intrusion detection algorithm :

3.5.1 Description of the NKONDOCK-IDS algorithm's principle
. During the execution of processes within an Information System, numerous tasks are performed. To keep track of these executions, we record them in event files, also known as workflow logs. These event files are continuously analyzed in real time by comparing each new entry added to them against the security policy. As a reminder, the security policy is a set of security rules.
Each security rule specifies the actions authorized by certain users on resources. The explainable intrusion detection we perform within constrained IoT environments is resource-centric. Therefore, when an event occurs within the constrained IoT environment that violates at least one security policy rule, our model considers it intrusive. Furthermore, for explainability, our system identifies all violated rules and generates an alert with an explanation.

3.5.2 NKONDOCK-IDS Algorithm .

Algorithm 1: NKONDOCK-IDS Algorithm

$N, P, M, K \geq 0$

(1) Create event files numbered 1 to P
(2) Create an intrusion table of size M
(3) . Generate the contents of event files, each containing at most N entries
(4) Create the security policy, with at most K rules
(5) For each event file
— Browse each entry in the file
— Compare the entry to all security policy rules
— If a security policy rule is violated, Then
* The user mentioned on the current entry of the event file must be a user authorized in the security policy, to perform the task concerned on the resource involved (K) ;
* The execution date of the task on the resource involved in the event must be between the start date and the end date mentioned in the security policy as authorization for the execution of this task on the resource concerned.
— Add the entry (event), along with its explanation, to the intrusion table.
— Generate an alert **(EACH TIME, EXPLAIN THE VIOLATED SECURITY POLICY RULES)**

3.5.3 The complexity of Algorithm
. **a) Time complexity**

Best case :

The best-case scenario occurs when :
— There is 01 event file
— There is 00 security rule
— The event file has 00 entry
— The intrusion table has 00 entry
Calculating the complexity allows us to obtain the following :
C= 1①+1②+1*0③+1④+1*(0⑤*0)
= 3
The complexity is $O(1)$.

Worst case :
The worst-case scenario occurs when :
— There are P event files
— There are k security rules
— There are N entries in each event file
Calculating the complexity allows us to obtain the following :
C = P①+1②+NP③+K④+P(N+K) ⑤
= P+1+NP+K+PNK
= P(1+N+NK)+1+K

b) Space complexity

Best case :

C= 1①+1②+1③+1④+0⑤
= 4
The complexity is $O(1)$.

Worst case :

C= P①+1②+NP③+1④+0⑤
C= P(1+N) +2

4. IMPLEMENTATION

The reference open-source solutions (Snort, Suricata, Zeek for NIDS ; OSSEC, Wazuh for HIDS) each cover a part of the spectrum but impose an integration and operational burden that is sometimes heavy for small teams. Furthermore, most commercial IDSs operate as "black boxes" whose detection policy is hard to read, which limits the ability of teams to understand, adjust and audit their own rules. The central research question of this project is to design an adaptative, hybrid host/network IDS explicable and able to be efficient in IoT constrained environments.

4.1 Reference solutions

Solution	Family	Paradigm
Snort	NIDS	Signatures
Suricata	NIDS	Signatures + protocols
Zeek	NIDS	Behavioural
OSSEC	HIDS	Signatures + integrity
Wazuh	HIDS + NIDS	Hybrid
Fail2ban	Partial HIDS	Simple behavioural
NKONDOCK-IDS	**HIDS + NIDS**	**Declarative policy + signatures + behavioural + correlation**

black

4.2 Positioning of NKONDOCK-IDS

NKONDOCK-IDS distinguishes itself from the existing ecosystem on three points :

(1) **Cumulative declarative policy** : unlike Snort/Suricata rules oriented around *negative signatures*, our policy is a set of cumulative `allow/deny` rules evaluated in order, bringing the detection engine closer to a fine-grained access control system.

(2) **Seven-module hybrid pipeline** including a dedicated module for operational maintenance (purge, rotation, archiving), a kill chain correlation engine and a behavioural anomaly detector, three dimensions often neglected in academic IDSs.

(3) **Native multi-channel dissemination** : SMTP, Slack, Discord, Teams and Syslog SIEM with minimum-severity filtering, without requiring any additional connector.

4.3 Modular architecture of NKONDOCK-IDS

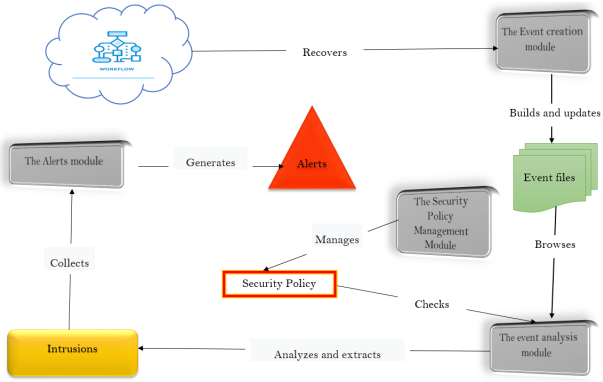

FIG. 2. Modular Architecture of NKONDOCK-IDS

4.4 Technical implementation

4.4.1 Environment and technology choices. The platform is developed in **Python 3.10+** with the **Flask** web framework. Data persistence relies on **SQLite** via the **SQLAlchemy** ORM. Network capture uses the **Scapy** library, and process monitoring relies on **psutil**. The whole system is designed to run on both **Linux** (Ubuntu, Debian, RHEL) and **Windows 10/11**.

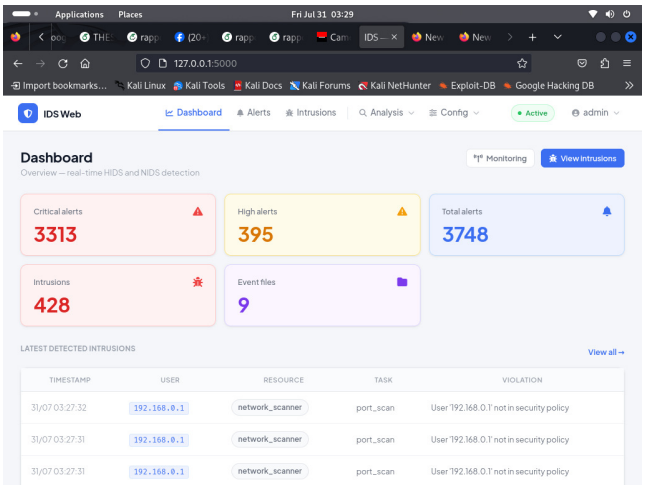

FIG. 3. Dashboard of NKONDOCK_IDS

4.4.2 Illustration of the validation. Here we have a view of dashboard of NKONDOCK_IDS

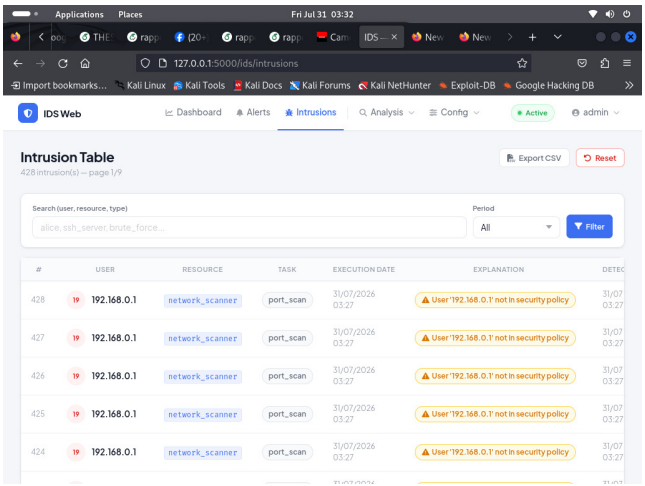

FIG. 4. Explainable Intrusion table of NKONDOCK_IDS

This capture presents an extraction of intrusion table with explanation
And here, we have an image of some security alerts

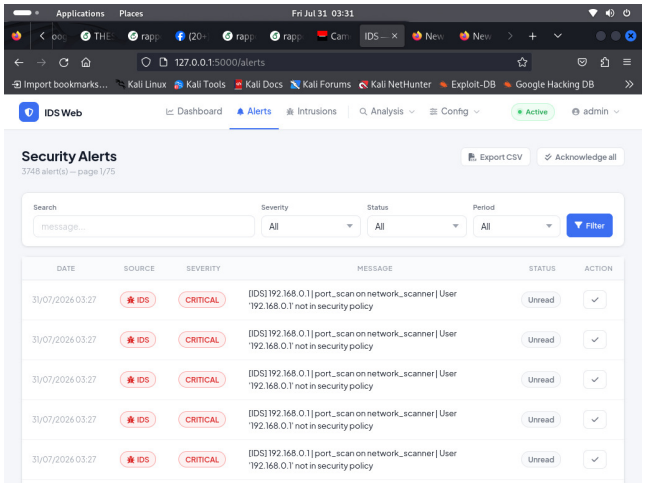

FIG. 5. Security alerts of NKONDOCK_IDS

5. CONCLUSION AND PERSPECTIVES

In conclusion, we reiterate that we focused on constrained IoT environments. These environments are increasingly used because they enable the connection of multiple objects equipped with technologies (sensors, software, chips) for data exchange over the internet without human intervention. During these exchanges, governed by a security policy, events can occur that violate this policy and are therefore considered intrusions. We have proposed an architecture that enables the detection of intrusions in constrained IoT environments. We emphasized the explainability of detections to allow users of tools based on our architecture to precisely understand the reason for each detection. Furthermore, the proposed solution is hybrid because it allows for the detection of explainable intrusions on isolated devices (generally servers) or on networks that connect multiple objects within a constrained IoT environment. Finally, our architecture is adaptive because the detection rules are not fixed; they follow a dynamic that is described by users according to their flexible definition of the security policy rules. This helps avoid false positives. Future work for this project would consist, on the one hand, of managing large volumes of data, particularly through big data, for the increased load related to the use of event files, and on the other hand, of proposing an equivalence class for security rules so that any new security rule (defined by a user of the tool based on our architecture) finds its equivalent in this equivalence class.

6. REFERENCES

[1] **Atsa Etoundi Roger, Nkoulou Onanena Georges, Nkondock Mi Bahanag Nicolas, et al**. "A Formal Framework for Intrusion Detection within an Information System based on Workflow Audit". International Journal of Computer Applications, **2013**, vol. 81, no 1, p. 1-10.

[2] **Nkondock Mi Bahanag Nicolas, and Atsa Etoundi Roger**. "An overview of Intrusion Detection within an Information System : The Improvment by Process Mining." Netw. Commun. Technol. 7.1 (2022) : 55-60.

[3] **Nkondock Mi Bahanag Nicolas, Bell Bitjoka Georges, and Emvudu Yves.** "A Framework for Intrusion Detection Based on Workflow Mining." American Journal of Computer Science and Technology 2.2 (2019) : 27-34.

[4] **Faeiz Alserhani**. "Intrusion detection and real-time adaptive security in medical IoT using a cyber-physical system design". Sensors, 2025, vol. 25, no 15, p. 4720.

[5] **Praveen, S. P., Sharma, K., Parashar, D., Murthy, V. S. N., Sirisha, U., Dewi, D. A. (2025)**. "Design of an iterative method for adaptive federated intrusion detection for energy-constrained edge-centric 6G IoT cyber-physical systems". Scientific Reports, 15(1), 41387.

[6] **Li, Yi, Wang, Haitao, and Xu, Guoming.** "Federated reinforcement learning–driven multi-task optimization for robust and ethical edge internet of things security". Scientific Reports, 2026.

[7] **Banbury, C., Zhou, C., Fedorov, I., Matas, R., Thakker, U., Gope, D., ... Whatmough, P. (2021)**. "Micronets : Neural network architectures for deploying tinyml applications on commodity microcontrollers". Proceedings of machine learning and systems, 3, 517-532.

[8] **Ortiz-Garcés, Iván, William Villegas-Ch, and Sergio Luján-Mora.** "Autonomous cyber-physical security middleware for IoT : anomaly detection and adaptive response in hybrid environments." Frontiers in Artificial Intelligence 8 (2025) : 1675132.

[9] **Alauthman, Mohammad, et al**. "Generative Adversarial Networks for Intrusion Detection Systems : A Comprehensive Survey of Applications, Challenges, and Research Directions." Arabian Journal for Science and Engineering (2026) : 1-25.G. Apruzzese et al., "Real-World IDS Evaluation," arXiv 2023.

[10] **Ogunseyi, Taiwo Blessing, et al.** "Performance Analysis of Explainable Deep Learning-Based Intrusion Detection Systems for IoT Networks : A Systematic Review." Sensors 26.2 (2026) : 363.S. Amin et al., "Incremental IDS," ICASSP 2021.

[11] **Adjewa, Frédéric, Moez Esseghir, and Leïla Merghem-Boulahia.** "From Edge Transformer to IoT Decisions : Offloaded Embeddings for Lightweight Intrusion Detection." Sensors 26.2 (2026) : 356.

[12] **Joshi, V. R., Assa-Agyei, K., Al-Hadhrami, T., Qasem, S. N. (2025)**. "Hybrid AI Intrusion Detection : Balancing Accuracy and Efficiency". Sensors, 25(24), 7564.

[13] **Upender, Talluri, et al.** "CyberDetect MLP a big data enabled optimized deep learning framework for scalable cyberattack detection in IoT environments." Scientific Reports 15.1 (2025) : 40865.

[14] **Upender, T., Neelakantappa, M., Rao, C. P., Gera, J., Reddy, V. L., Yamsani, N. (2025)**. "CyberDetect MLP a big data enabled optimized deep learning framework for scalable cyberattack detection in IoT environments". Scientific Reports, 15(1), 40865.

[15] **Karthik, M. G., Keerthika, V., Mantena, S. V., Siri, D., Yeluri, L. P., Lella, K. K., Ganesh, B. R. (2026)**. "Energy-efficient intrusion detection with a protocol-aware transformer–spiking hybrid model". Scientific Reports.

[16] **Tawfik, M. (2024)**. "Optimized intrusion detection in IoT

and fog computing using ensemble learning and advanced feature selection". Plos one, 19(8), e0304082.

[17] **Wu, Juan, Shuai Fu, and Mohammad Sarabi**. "Introducing a hybrid intrusion detection method for IoT-cloud environments based on ResNeXt and improved Ebola optimization search algorithm." Scientific Reports 15.1 (2025) : 37612.

[18] **AR, S., Katiravan, J. (2025)**. Enhancing anomaly detection and prevention in Internet of Things (IoT) using deep neural networks and blockchain based cyber security. Scientific Reports, 15(1), 22369.

[19] **Talukder, M. A., Khalid, M., Sultana, N. (2025)**. A hybrid machine learning model for intrusion detection in wireless sensor networks leveraging data balancing and dimensionality reduction. Scientific reports, 15(1), 4617.

[20] **Van der Aalst, W. M., Van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., Weijters, A. J. (2003).** "Workflow mining : A survey of issues and approaches. Data knowledge engineering", 47(2), 237-267.