SuperViz
Supervision et orchestration de la sécurité
Postdoc report (PO3.1): Symptomatic Response to Network Attacks

Identifiant projet	ANR-22-PECY-0008
Acronyme projet	SuperViz
Nom du projet	Supervision et orchestration de la sécurité
Numéro du livrable	L3.4
Titre du livrable	Postdoc report (PO3.1): Symptomatic Response to Network Attacks
Date prévue	M36
Date de soumission	15/09/2025
Editeurs	Thomas Marchioro
Contributeurs	Alexis Olivereau (postdoc supervisor)
Revision	1.0 (9 janvier 2026)
Mots clé	Réponse automatique aux attaques, alertes des systèmes de détection d'intrusion, intelligence artificielle, pare-feu
Keywords	Automated counter-easures, intrusion detection system alerts, artificial intelligence, firewall

– Piloté par

– Financé par

– Opéré par

Table des matières

Introduction

This report summarizes the research efforts of the postdoctoral project PO3.1, which was part of Lot 3 within the Superviz project (ANR-22-PECY-0008) supported by the French National Research Agency under the France 2030 label. The primary objective of PO3.1 was to design and implement a prototype of an AI-based intrusion response system capable of reacting to ongoing network attacks in real time. The motivation for pursuing this line of research lies in the growing difficulty of providing timely and effective responses to such attacks within modern, complex network environments.

Today's network architectures generate enormous volumes of traffic, and as a result, cybersecurity analysts working in security operation centers (SOC) are overwhelmed with alerts produced by monitoring tools such as Intrusion Detection Systems (IDS) [14]. Recent reports have emphasized that this flood of alerts not only prolongs response times but also reduces analysts' ability to reliably distinguish between false positives and genuine threats. Consequently, the effectiveness of current defensive strategies is diminished.

This highlights a significant shortcoming in existing aggregation and analysis tools, which at present are inadequate in translating incoming alerts into actionable responses. To address this gap, the research efforts of PO3.1 were directed toward a network-level approach. Specifically, the goal was to design a system (or rather, a family of systems) capable of processing IDS alerts and transforming them into network-level countermeasures.

Over the course of this postdoctoral project, the research objectives evolved based on newly identified gaps and opportunities. A fundamental issue that emerged was the absence of standardized testbeds for evaluating the performance of intrusion response systems. A review of the existing literature revealed that many studies rely on ad hoc simulations or custom-built testbeds for evaluation.

This practice introduces several limitations. In particular, the networks and attack scenarios used in such evaluations are often oversimplified in order to facilitate simulations. As a result, intrusion response systems are assessed under unrealistic conditions, which makes it difficult to compare results across different research efforts. Furthermore, while the cybersecurity community has produced a wealth of IDS datasets over the years, these remain largely unused in the evaluation of intrusion response.

To address these shortcomings, an additional goal was added to project : the development of an evaluation approach for our proposed intrusion response systems that leverages existing IDS dataset. This direction not only strengthens the evaluation methodology of PO3.1 but also contributes to Superviz by proposing a foundation for more consistent benchmarking in the field.

Problem description and research questions

The research conducted within PO3.1 was guided by two central research questions.

RQ1 : How can IDS alerts be translated into practical response actions ?

RQ2 : How can existing IDS datasets be leveraged to evaluate intrusion response solutions ?

In addressing these questions, the project maintained a scope centered on network-level alerts and responses. To this end, this research introduced the concept of network intrusion response systems (NIRS) : a family of systems designed to aggregate alerts generated by network IDS (NIDS) and transform these alerts into automated rule updates for firewalls or intrusion prevention systems (IPS). Additionally, a unified evaluation protocol was introduced for assessing NIRS performance using public IDS datasets. The proposed evaluation protocol models intrusion response as a binary decision problem, where a network connection can be either accepted or blocked by the existing set of firewall/IPS rules. While the problem remains fundamentally different from intrusion detection—since it requires to accept or block *future* traffic based on *present* alerts—this evaluation framework naturally induces performance metrics that are analogous to the concepts of true positives and false positives in intrusion detection. This, in turn, allows to compare different NIRS solutions and calculate objective metrics for their performance.

Link with other activities within Superviz

The research carried out within the scope of PO3.1 aligned with broader efforts in Superviz, particularly concerning the analysis and exploitability of existing IDS datasets. In addition to proposing a methodology that enables the use of public IDS datasets for intrusion response evaluation, the work included a detailed examination of the datasets themselves.

This analysis focused on understanding the characteristics of different datasets and assessing how these traits influence their suitability for the proposed evaluation protocol. Attention was also given to identifying and addressing potential limitations, such as inherent biases, missing information, or inconsistencies, all of which can significantly undermine the validity of benchmark evaluations. By explicitly accounting for these issues, the project contributes not only a technical methodology but also a critical perspective on the strengths and weaknesses of the resources commonly employed in intrusion detection and response research.

Chapitre 1

Background

1.1 Intrusion detection systems

Intrusion Detection Systems (IDS) are security tools designed to monitor hosts or networks in order to detect malicious activity. IDS can be classified along two main dimensions. The first dimension relates to their scope. IDS that operate at the host level are referred to as host-based IDS (HIDS), while those that monitor network activity are called network IDS (NIDS). These terms, however, can sometimes be misleading. For example, NIDS may be deployed to monitor traffic associated with a single host. The fundamental distinction is that NIDS rely exclusively on network-level data—mainly network traffic—to identify intrusions, whereas HIDS use host-level information such as system logs or application data.

The second dimension of classification concerns the detection methodology. Signature-based IDS identify malicious activity by matching incoming traffic against a database of known attack patterns. By contrast, anomaly-based IDS detect threats by identifying deviations from established models of normal traffic behavior. Each approach has its strengths and limitations : signature-based systems are effective in recognizing known attacks, while anomaly-based systems are better suited to identifying zero-day attacks or other attacks whose signature is unavailable.

Given that this research focuses on network-level attack signals and corresponding response actions, NIDS were identified as the most appropriate foundation for the intrusion response framework developed in this project.

1.1.1 Machine learning-based NIDS

The most widely used network intrusion detection systems (NIDS) are signature-based, with tools such as Snort, Suricata, and Zeek being standard in both research and practice. These systems rely on expert-crafted rules to match incoming traffic against known attack patterns. However, the growing diversity and sophistication of attacks has made it increasingly difficult to maintain comprehensive rule sets. Moreover, the notion of what constitutes an “attack” is

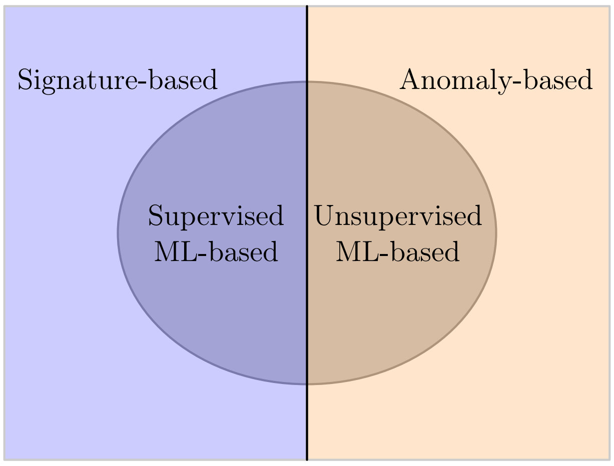


FIGURE 1.1: Venn diagram of NIDS categories.

highly context-dependent. For example, a level of traffic that would overwhelm a small business in a distributed denial-of-service (DDoS) scenario may simply correspond to the release of a new popular series for a large-scale streaming service.

To address these limitations, machine learning (ML) techniques have become increasingly popular in intrusion detection. Instead of relying on manually defined signatures, ml-based approaches learn patterns of benign and malicious traffic directly from data. From an ML perspective, network intrusion detection can be formulated as a binary classification problem : given a set of features representing traffic, the model predicts whether it is benign (typically encoded as 0) or malicious (typically encoded as 1).

While ML-based NIDS move away from static rules, they are not all anomaly-based in the strict sense described earlier. Instead, they can be divided into two broad families : supervised and unsupervised. Supervised methods require labeled traffic data containing examples of both normal activity and attacks. Models such as random forests and gradient-boosted trees fall into this category. These systems remain, in a sense, signature-based, as they learn to recognize attacks from past examples. In contrast, unsupervised approaches operate on unlabeled traffic, under the assumption that the majority of the observed data is normal. They infer patterns of typical network behavior and flag abnormal deviations as potential intrusions. The most widely used unsupervised NIDS rely on autoencoders to iteratively compress and reconstruct traffic, comparing the reconstruction error to a threshold to identify anomalies.

Supervised ML-based NIDS generally achieve strong performance in closed-world settings, where the training data closely matches the environment in which they are deployed. Unsupervised ML-based NIDS are typically less accurate but more adaptable, since they can be trained directly on the traffic of the monitored network. This makes them suitable for open-world scenarios, where labeled data is unavailable. The taxonomy of NIDS introduced in the paragraphs above is summarized in fig. 1.1.

1.1.2 Network flow analysis

An ML-based NIDS is essentially a function $\mathcal{D} : \mathbb{X} \to \{0, 1\}$ where $\mathbb{X}$ is the feature space representing the input traffic and $\{0, 1\}$ is the (binary) set of classes. The features used to encode traffic greatly vary across the various proposed solutions, and they may describe different elements of network traffic. For example, a feature vector x may describe a single packet, an aggregation of packets, or a time series. However, the most common approach in the literature is to utilize aggregated metrics that describe an entire connection between two hosts, which are called *network flows*. A network flow x is a tuple

$$x = (t, \texttt{src\_ip}, \texttt{src\_port}, \texttt{dst\_ip}, \texttt{dst\_port}, \texttt{protocol}, \dots) \tag{1.1}$$

where t is the first packet's timestamp in the flow, `src_ip` (`src_port`) and `dst_ip` (`dst_port`) are the the source and destination IP addresses (ports), and `protocol` is the network protocol used by the two hosts to communicate. Other information included in a flow can vary depending on the tool used for monitoring the network traffic (e.g., CICFlowMeter, Zeek, NFStream[1]). Tools that extract flow information from network traffic are commonly called flow-meters.

1.2 Intrusion response

1.2.1 Intrusion prevention versus intrusion response

It is important to distinguish the role of intrusion prevention and intrusion response. Intrusion prevention focuses on blocking attacks before they reach their target systems. A common example is a firewall, which can filter traffic at the network perimeter by allowing or denying connections based on parameters such as IP addresses, ports, or protocols. In addition, some IDS that operate at the packet level, such as Snort, can operate in prevention mode as intrusion prevention systems (IPS). In this configuration, the system does not simply generate alerts but actively drops traffic that matches a detection rule.

The essential distinction between IPS and IRS lies in the timing of their actions. An IPS operates preemptively, attempting to stop malicious traffic before it ever reaches the protected system. An IRS, by contrast, focuses on responding once an attack is already underway, adapting its countermeasures based on the alerts and evidence generated during the incident. A form of intrusion response may consist in updating the IPS configuration according to the received alerts.

1. `https://github.com/UNBCIC/CICFlowMeter`, `https://zeek.org/`, `https://www.nfstream.org/`

Chapitre 2

Related work

A preliminary analysis of the existing literature revealed that intrusion response systems greatly vary on many aspects. In this review section, we focus on three of them, namely the types of response action, evaluation methodologies, and metrics.

2.0.1 Response actions

The first dimension that we investigate is the range of response actions that have been proposed to mitigate attacks. We found that these can be categorized into three main groups based on the level of control they exercise on the network. More specifically, some solutions require privileged access to many (or all) hosts and components in the network. Others require to set up additional software or hardware components. Clearly, this can impact the feasibility of their deployment and the willingness of network administrators to adopt such solutions.

Firewall-based responses requires the least level of control. They typically, interact with existing routers or firewalls [13] to block or redirect traffic.

SDN-based responses require the presence of a software defined networking (SDN) infrastructure. This allows fine-grained and dynamic control over network traffic, e.g., by dynamically rerouting packets or applying rate limits [11, 12]. On the other hand, the infrastructure requirements limit their applicability. At the time of writing, only a minority real-world networks implement SDNs.

Host-level responses assume the ability to intervene directly on hosts, either via some standalone application running on each host, or via some form of centralized control. Response actions include patch deployment, service restart, process isolation, or migration [5, 20]. Such approaches offer rich recovery options but require privileged access and strong orchestration capabilities.

2.0.2 Evaluation methodologies

The methods used to evaluate IRS is also not standardized. This is an important problem, especially when statistical approaches – such as machine learning-based ones – need to be com-

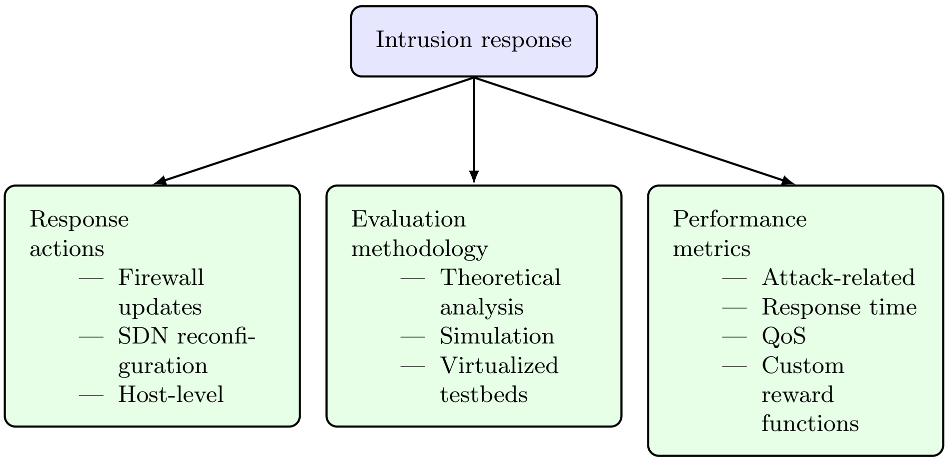


FIGURE 2.1: Aspects of intrusion response investigated in the literature review.

pared. To emphasize how relevant this is, it may be useful to draw a comparison with IDS evaluation. In the case of IDS, the common approach is to utilize benchmark datasets such as CIC-IDS 2017 or UNSW-NB15. The datasets are splitting into training and test sets, and the IDS model is first trained on one set and evaluated on the other. It is now well-established that temporal splitting should be used to analyze the IDS behavior for non-stationary traffic. Also, later works have also implemented concept drift analysis and robustness against adversarial examples [2]. This evaluation allows to effectively gauge the strength of an IDS model based on benchmark results. Even if two papers use different datasets, one can still make a comparison based on known characteristics or biases of the various datasets.

In the case of IRS, some papers focus on providing a theoretical analysis of their proposed IRS solution [7, 19]. Theoretical analyses require to make abstractions and assumptions to frame the proposed IRS within a mathematical model. However, an analysis that is purely theoretical does not allow to determine the strength of the proposed solution is real-world deployments, where the assumptions made may not hold.

Other works use simulated environments [20], which are used to model essential aspects of a system. However, most of these simulators are not publicly available, with notable exceptions such as MiniCPS [1] and CyberBattleSim [17]. This limits the reproducibility of the results and does not allow comparison with other IRS solutions.

Virtualized testbeds are the closest tools to real-world testing, creating more realistic scenarios using virtual machines [13], containers, or network emulation platforms (e.g., GNS3 as in [5]). However, testbeds are also rarely released publicly. Additionally, their scope is often limited to small networks due to hardware limitations.

2.0.3 Metrics

Another aspect where IRS evaluation falls short compared to IDS is the performance metrics. Intrusion detection is modeled as a binary classification problem [1]. This binary model yields a set of metrics that can be used to evaluate the IDS output. It is well understood that there are two types of correct predictions (true positives and true negatives) and two types of wrong predictions (false positives and false negatives). The metrics used to evaluate an IDS (accuracy, precision, true-positive rate, false-positive rate, etc.) are meant to provide a complete statistical description of these four output types.

In the case of IRS, intuition suggests that some analogous metrics should exist. However, in practice there is a lot of nuance when it comes to measure the impact of intrusion response techniques on a system. This is also linked with the diverse nature of IRS solutions. Obviously, attack-related indicators are often reported, such as the percentage of blocked attacks [13] or the final stage reached in a multi-stage attack scenario [5].

Another aspect that is occasionally analyzed is response time, which assesses how quickly the system reacts to a threat [6].

QoS-related indicators are adopted in some cases to quantify the impact of the response on legitimate traffic. These include metrics such as preserved throughput, increased latency, or false positive disruption [11].

IRS solutions for cyber-physical or industrial settings also measure physical-level indicators, which reflect the safety of physical processes (e.g., likelihood of tank overflows [12] or power load issues [7]).

Finally, some specific types of solutions RL-based and game-theoretic IRS use custom reward or utility functions. These are used to guide the decision-making process, but they are also employed as evaluation metrics. These functions typically integrate multiple heterogeneous objectives and tend to be system-specific [11, 20]. According to the principle “when a measure becomes a target, it ceases to be a good measure,” these latter metrics should not be used for evaluation.

2.1 Research gaps and opportunities

One conclusion that can be drawn from the above review is that IRS solutions suffer from a lack of standardized practices. Drawing again a comparison with IDS (more specifically a network IDS (NIDS)), when a new NIDS solution is proposed, the context is always well understood : the system is meant to analyze network traffic – likely in the form of network flows – and generate predictions ; it is run on common benchmark datasets (CIC-IDS 2017, UNSW-NB15, etc.) ; and,

1. In some cases both binary and multiclass analyses are carried out, where each class represents a different attack or family of attacks. However, the binary analysis is consistent across the vast majority of proposed solutions, while the multiclass analysis is less common.

finally, its predictions on these datasets are categorized according to the dataset's ground-truth labels.

Since all the above information is known and widely established, this allows to conduct a deeper analysis on more nuanced aspects, such as "Is the NIDS model robust against concept drift ? Is it prone to adversarial inputs ? How do dataset biases impact performance metrics ?"

In the case of IRS, the common setting is missing, and (almost) each paper comes with a different set of response strategies, evaluation methodologies, and performance metrics. This is a large obstacle to the creation of a common body of knowledge on intrusion response – especially in the case of ML-driven solutions.

Another aspect often missing in existing solution is a mapping between alerts raised by NIDS and the proposed response action. ML-based NIDS, according to benchmarks, yield highly accurate predictions, which however are often difficult to explain. Few works have investigated whether it is possible to find an adequate response to these alerts, despite the lack of explainability [18].

These gaps led to the identification of the two research questions, RQ1 and RQ2, described in the introduction of this report.

Chapitre 3

Proposed solution : NIRS

To address the problem of the lack of standardization in intrusion response systems, it is necessary to introduce more scoped definitions of IRS. As this research focuses on bridging the gap between network IDS and intrusion response, this naturally leads to defining a family of IRS that reasons and acts at the network level.

In analogy with network IDS (NIDS), systems that belong to these family are named *network intrusion response systems (NIRS)*.

3.1 Philosophy and assumptions of NIRS

The core idea of NIRS is to extract network-level rules, more specifically firewall rules, based on NIDS predictions. Clearly, these systems should account for possible mistakes among these predictions. On the other hand, they should also operate under the assumption that the majority of these alerts are correct.

Let us consider a set of connections labeled by a NIDS. This set would look as the examples shown in Table 3.1. Assuming a flow-level analysis, each connection is characterized by an initial timestamp t, source IP, destination IP, and other indicators. Additionally, each connection is associated with a predicted label $\hat{y}$ (either "normal" ($\hat{y} = 0$) or "attack" ($\hat{y} = 1$)).

With this setting in mind, the objective of NIRS is to update the firewall rules to mitigate possible ongoing attacks. Since the alerts analyzed by NIRS are related to flows that have already bypassed the firewall rules currently in place. Therefore, the rule update should aim to block future alerts of similar nature.

Considering the example in Table 3.1, it is easy to see that most alerts relate to `src_ip` 18.184.104.180. It appears that this IP is attempting to connect to TCP port 22 (default SSH port) of different IP addresses in the monitored network. Multiple alerts towards this port in a short time period may be a sign of an SSH bruteforce attempt. There is also one alert from 52.29.177.21, directed towards port 8000 of one host in the monitored network. However, several connections of similar nature where not flagged as alerts. Therefore, this might simply be a false

t	`src_ip`	`src port`	`dst_ip`	`dst_port`	`protocol`	$\hat{y}$
1556172725	52.59.177.21	42966	192.168.1.30	8000	tcp	1
1556173067	52.59.177.21	42987	192.168.1.30	8000	tcp	0
1556173461	18.184.104.180	34532	192.168.1.30	8000	tcp	0
1556173488	52.59.177.21	43374	192.168.1.30	8000	tcp	0
1556340863	52.59.177.21	43693	192.168.1.30	8000	tcp	0
1556340869	18.194.169.124	50204	192.168.1.32	8000	tcp	0
1556340880	18.184.104.180	43862	192.168.1.30	22	tcp	1
1556340887	18.184.104.180	43904	192.168.1.30	8000	tcp	0
1556340895	18.184.104.180	45016	192.168.1.32	22	tcp	1
1556340953	18.184.104.180	45124	192.168.1.32	22	tcp	1
1556548974	74.125.109.8	11	192.168.1.34	-	icmp	0
...	...	...	...	...	...	...

TABLE 3.1: Example of a NIDS output.

positive.

Based on this information, NIRS should find the most fitting rule update. It should ensure that the update would prevent most of the alerts without disrupting regular traffic. This task is far from trivial. Even after the above discussion, it is still unclear what kind of new rules should be put in place using the information available in Table 3.1. Should the rule update block all TCP connections with destination port 22 ? This might disrupt legitimate SSH connections. Should it block only connections from 18.184.104.180 ? The attackers may use a proxy or a VPN to change their IP address.

Since the decision is not trivial, different types of heuristics or AI systems could be employed to select the rule update. The definition of NIRS that is provided below aims to encapsulate all these possible systems under a common setup, allowing to evaluate and compare them.

3.2 Definition of NIRS

As NIRS are meant to update firewall rules, these should be formalized first. A set of firewall rules, or ruleset, can be defined as a map $r : \mathbb{X} \to \{0, 1\}$. Given a connection x this is either *let through* ($\hat{b} = 0$) or *blocked* ($\hat{b} = 1$) by the ruleset, i.e. $r(x) = \hat{b}$. This definition can be applied to both packets and network flows with some caveats. If x is a packet, firewall rules that are based on rates cannot be represented if x is the only input to r. However, this can be solved by including a “state” variable σ to the input. If x is a network flow, there might be cases where a rule blocks only some of its packets. In such cases, the flow is considered *blocked*, while it is considered *let through* only if none of its packets are blocked.

Since the ruleset can be updated over time, this means that the function r is time-dependent. The notation r_t can be used to denote the ruleset function at time t. A network intrusion response

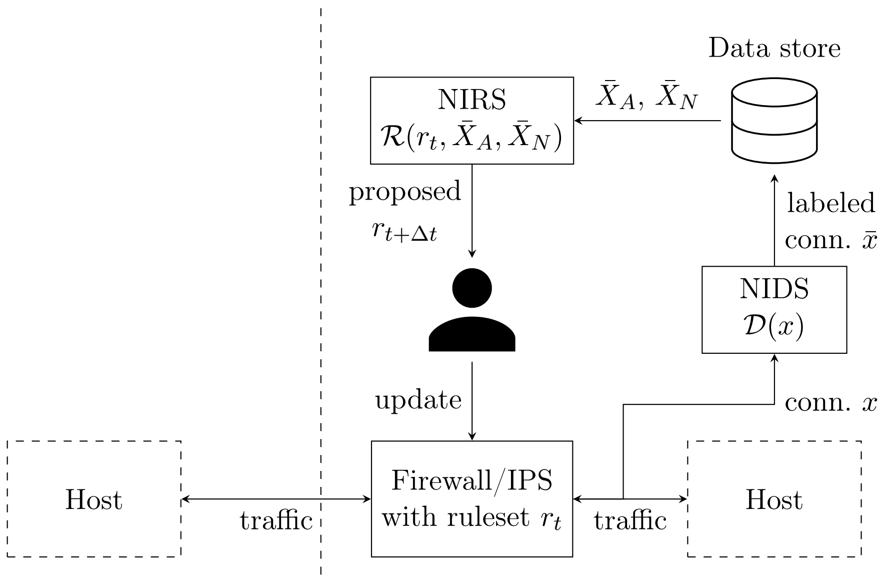


FIGURE 3.1: Proposed network intrusion response workflow.

system (NIRS) can thus be defined as a function $\mathcal{R}$ that maps

$$(r_t, \bar{X}_A, \bar{X}_N) \mapsto r_{t+\Delta t} = \mathcal{R}(r_t, \bar{X}_A, \bar{X}_N) \tag{3.1}$$

with input :

- r_t : The network ruleset at time t.
- $\bar{X}_A$: A set of network connections $\{\bar{x}_{A,1}, \bar{x}_{A,2}, \ldots\}$ that were labeled as alerts by a network intrusion detection system (NIDS).
- $\bar{X}_N$: A set of network connections $\{\bar{x}_{N,1}, \bar{x}_{N,2}, \ldots\}$ that were labeled as normal traffic by a NIDS.

and output :

- $r_{t+\Delta t}$: An updated network ruleset. $\Delta t > 0$ is the time elapsed between the the enforcement of two rulesets.

3.3 Deployment and workflow

While the formal definition is meant to be general and flexible, it is sensible to also provide a general workflow for the practical deployment of NIRS. This workflow is shown in fig. 3.1. Network traffic between two hosts – one internal and one external to the monitored network – passes through a firewall with ruleset r_t. Any connection x that is not dropped by the firewall[1] is monitored by the NIDS $\mathcal{D}$, which labels it as an alert or normal traffic. The labeled connection

1. It is worth noting that while the discussion is focused around firewall, the provided definition of ruleset applies also to IPS. Therefore, NIRS can be integrated with tools like Snort (operating in IPS mode).

$\bar{x}$ is added to a data store. Alerts and normal connections are queried from the data store and processed by the NIRS $\mathcal{R}$, which proposes a ruleset update $r_{t+\Delta t}$. The proposed update is then evaluated by a security analyst, who finally decides whether to update the current ruleset. Clearly, when evaluating $\mathcal{R}$, the human in the loop is ignored and the proposed updates are always assumed to be accepted. While the use of response systems to dynamically update firewalls is not new [13], framing these solutions using a formal framework helps facilitating their comparison.

3.4 Evaluation metrics

The impact of NIRS ruleset on network connections can be divided into four categories, mirroring the conventional classification of NIDS outputs into true positives, true negatives, false positives, and false negatives. Specifically, given a NIRS actions on a portion of network traffic, we can count the number of connections that are : correctly accepted (CA), wrongly accepted (WA), correctly blocked (CB), and wrongly blocked (WB). These values can be used to calculate NIRS performance metrics : the *correct block rate (CBR)*, i.e., the proportion of malicious traffic that is correctly blocked

$$\mathrm{CBR} = \frac{\mathrm{CB}}{\mathrm{CB} + \mathrm{WA}} \tag{3.2}$$

and the *wrong block rate (WBR)*, i.e., the proportion of benign traffic that is incorrectly blocked

$$\mathrm{WBR} = \frac{\mathrm{WB}}{\mathrm{WB} + \mathrm{CA}} \tag{3.3}$$

These metrics are analogous to TPR and FPR, respectively, for NIDS. Both CBR and WBR can be computed either per-packet or per-flow. Other metrics such as precision, balanced accuracy, and F_1-score can also be computed.

3.5 Evaluation methodology

3.5.1 Evaluation setup

Like other types of intrusion response systems, NIRS can be evaluated using interactive testbeds. However, a key advantage of NIRS is that they can also be evaluated on public NIDS datasets. A NIDS dataset $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^{n}$ is a sequence connections x_i and their ground-truth labels y_i. A NIRS can be evaluated on these datasets by iterating over labeled connections (x_i, y_i). Their ground-truth label of each connection x_i is compared to the decision b_i, taken according to the ruleset r_t in place at timestamp t_i.

As mentioned in §1.2, an important aspect to be considered in NIRS evaluation is that intrusion response is not retroactive. A ruleset update happens as a consequence of NIDS alerts.

However, the updated ruleset will be set in place only after the alert has ended. In other words, the connection raising the NIRS alert is not blocked by the update ; only future connections are affected. This time dependency should be taken into account during the evaluation. Consequently, flows should be processed in order of their timestamps to emulate a real-time scenario. Furthermore, the specifics of the NIRS implementation should also be considered in the evaluation. For example, for a NIRS $\mathcal{R}$ that updates r_t at a constant rate Δt, the connections can be evaluated in batches covering time ranges of size Δt, as described by Algorithm 1.

For each connection x_i with timestamp $t_i \in [t, t + \Delta t]$, the algorithm first checks whether it is blocked by the current ruleset r_t (i.e., if $b_i = r_t(x_i)$ is 1). After verifying whether x_i is blocked, one of the counters (CB, WB, CA, WA) is updated depending on the values of b_i and the ground truth y_i. If x_i is blocked, no further step is needed. Otherwise, x_i is given as input to the NIDS $\mathcal{D}$ to obtain a prediction $\hat{y}_i = \mathcal{D}(x_i)$. x_i is then added to $\bar{X}_A$ or $\bar{X}_N$ depending on the value of $\hat{y}_i$. When all the connections in the current time range have been processed, the NIRS $\mathcal{R}$ is invoked to update the ruleset. It should be noted that this algorithm does not account for the ruleset-installation time—i.e., the time τ employed by $\mathcal{R}$ to generate and install a new ruleset. In our experimental setup, this does not impact the evaluation, as $\Delta_t \gg \tau$ for both our baseline NIRS. However, more complex NIRS may need to account for it by extending the time range of r_t to $[t, t + \Delta_t + \tau]$, but using only $[t, t + \Delta_t]$ to update $\bar{X}_A$ and $\bar{X}_N$.

3.5.2 Limitations

While metrics such as the CBR and WBR provide a quantitative measure of a NIRS's performance, they are not sufficient on their own to determine whether an attack has been effectively mitigated. In the case of DoS, a high rate of blocked connections typically correlates with reduced service disruption, offering a relatively clear indication of successful mitigation. However, this relationship does not hold across all attack types, and more nuanced evaluation protocols are needed when focusing on specific attack vectors. Still, CBR and WBR remain valuable as general-purpose metrics, allowing for broad comparisons across different families of attacks. Another aspect to be considered is related to NIDS datasets. While dataset-based evaluation is probably one of the strongest advantages of NIRS, it still presents limitations. Datasets do not account for adaptive adversaries who deliberately attempt to evade detection and response mechanisms. Modeling such adversaries is complex, especially within the constraints of pre-collected data. This limitation is not unique to NIRS, but rather affects most data-driven approaches to cybersecurity.

3.6 Including NIRS in more complex solutions

NIRS are not intended to function as standalone components, much like how NIDS should not be the only systems in place to monitor the network. In network security, it is widely reco-

Algorithm 1 NIRS evaluation on a dataset $\mathbb{D}$ with constant Δt

```
Require: Dataset $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^{n}$ of connections and ground-truth labels, NIDS $\mathcal{D}$, NIRS $\mathcal{R}$,
    update interval $\Delta t$.
 1: Initialize empty sets $\bar{X}_A \leftarrow \emptyset, \bar{X}_N \leftarrow \emptyset$
 2: $t_{\min} \leftarrow \min_{i=1,\dots,n} t_i$, $t_{\max} \leftarrow \max_{i=1,\dots,n} t_i$
 3: Initialize time $t \leftarrow t_{\min}$
 4: Set initial ruleset $r_t$                    ▷ E.g., initialize $r_t$ to always output 0 (accept all traffic)
 5: Initialize CB← 0, WB← 0, CA← 0, WA← 0
 6: while $t \leq t_{\max}$ do
 7:     for each $x_i$ where $t \leq t_i \leq t + \Delta t$ do
 8:         $b_i \leftarrow r_t(x_i)$                        ▷ Check if $x_i$ is blocked by current ruleset
 9:         if $b_i = 1$ then
10:             if $y_i = 1$ then CB←CB+1                    ▷ Compare with ground truth
11:             else WB←WB+1
12:             end if
13:         else
14:             if $y_i = 1$ then WA←WA+1
15:             else CA←CA+1
16:             end if
17:         end if
18:         $\hat{y}_i \leftarrow \mathcal{D}(x_i)$
19:         if $\hat{y}_i = 1$ then $\bar{X}_A \leftarrow \bar{X}_A \cup \{x_i\}$      ▷ Update $\bar{X}_A, \bar{X}_N$ based on the NIDS output
20:         else $\bar{X}_N \leftarrow \bar{X}_N \cup \{x_i\}$
21:         end if
22:     end for
23:     $r_{t+\Delta t} \leftarrow \mathcal{R}(r_t, \bar{X}_A, \bar{X}_N)$                ▷ Update ruleset
24:     $t \leftarrow t + \Delta t$                                  ▷ Update current time
25: end while
26: return CB, WB, CA, WA
```

gnized that no single solution can address all scenarios ; instead, components must be integrated into broader, more complex architectures tailored to the specific needs of the system [16]. For instance, in large enterprise environments where the volume of alerts can overwhelm security analysts, NIRS could serve as alert aggregation tools that distill raw NIDS alerts into actionable summaries and predefined response strategies. In such deployments, the output of a NIRS might be redirected into a Security Information and Event Management (SIEM) platform or a log search engine like Elasticsearch. Whitelisting mechanisms could also be integrated to avoid blocking critical connections.

Chapitre 4

NIRS implementation

The NIRS presented in this report are based on a common window-based analysis of NIDS outputs. According to the general definition given in §3.2, NIRS require a ruleset r_t to update and two sets of connections : $\bar{X}_N$ for regular connections, and $\bar{X}_A$ for alerts.

4.1 Window-based analysis

In the setup chosen for our experiments, the two sets $\bar{X}_N$ and $\bar{X}_A$ are selected using two windows, as shown in fig. 4.1.

The normal traffic window is a sliding window of fixed duration ΔT_N, meaning only the connections observed in the last ΔT_N seconds are used as $\bar{X}_N$. Alerts, instead, are grouped to obtain $\bar{X}_A$ using an incremental window with a timeout of ΔT_A. The alert window continues to expand as long as new alerts arrive within ΔT_A seconds of the previous one ; each new alert resets the timeout. Once the timeout is exceeded without new alerts, the window is flushed, under the assumption that alerts separated by more than ΔT_A seconds likely originate from unrelated incidents. The NIRS $\mathcal{R}(r_t, \bar{X}_A, \bar{X}_N)$ processes the contents of these two windows $(\bar{X}_A, \bar{X}_N)$ along with the current firewall ruleset r_t and produces an updated ruleset r_{t+1}.

4.2 Firewall updates via iptables

There are many possible ways of implementing a firewall ruleset to be used and updated by NIRS. For Linux systems, one of the most natural choices is `iptables`. `iptables` is a Linux utility that controls how network packets are handled by the kernel's `netfilter` framework. One of its key functions is regulating traffic through a set of chains.

Using iptables, two possible implementations of firewall rulesets are possible. The first makes use of the `FORWARD` chain, which applies to packets being routed through a system acting as a router or gateway. By default, this chain accepts all incoming traffic, but stricter control can be enforced by applying `DROP` rules, which silently discard packets that match the rule. For example,

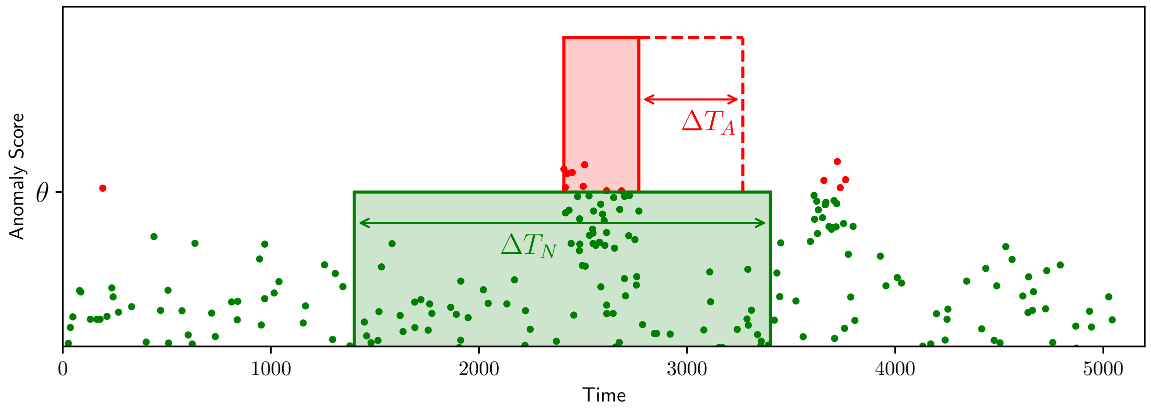


FIGURE 4.1: Window system used to select samples from $\bar{X}_A$ and $\bar{X}_N$ in our baseline examples.

a rule like `iptables -A FORWARD -s 10.2.0.1 -j DROP` blocks all forwarded traffic originating from source IP `10.2.0.1`. Other types of rules targeting destination addresses or specific ports can be used to prevent communication with certain hosts or services. A second possibility is to apply firewall rules directly at the host level. In that case, the `INPUT` and `OUTPUT` chains should be used to filter incoming and outgoing packets respectively.

Updating rules at the host level would greatly increase the complexity of the evaluation, requiring to assess multiple rulesets and have the NIRS update them independently. This solution is also less practical in a real deployment, as it would require a NIRS to have control on the ruleset of each individual host. Therefore, the experiments run in this report focus on r_t being applied at the router level via the `iptables` `FORWARD` chain.

4.2.1 Assessing rule impact

In order to evaluate NIRS using public datasets, it is necessary to assess the impact of a dynamically updated ruleset on the traffic of that dataset. In the case of `iptables`, that requires to determine which connections are blocked based on the state of the `FORWARD` chain. One relatively straightforward way to do so is to replay the traffic stored in PCAP files on a virtual machine that simulates the router where the firewall rules are applied.

However, there are two major drawbacks in this approach. Firstly, the setup may be complex and resource demanding. Secondly, the experimental evaluation of a NIRS would be slow. Although tools like `tcpreplay` allow to speed up the traffic why replaying it, the total time would be still proportional to the time range of the dataset PCAP files.

A more efficient way to determine whether a connection is affected by the rules in place consists in using a custom parser. This allows to directly match a connections stored in tabular format (e.g., as network flows) against a ruleset. Indeed, implementing the parser is also a challenging task, especially considering the high expressiveness of `iptables` rules. To reduce

Description	Rule format
Block source IP or subnet	`-A FORWARD -s <src_ip>[/<subnet>] -j DROP`
Block destination IP or subnet	`-A FORWARD -d <dst_ip>[/<subnet>] -j DROP`
Block specific protocol for a destination IP/subnet	`-A FORWARD -d <dst_ip>[/<subnet>] -p <protocol> -j DROP`
Block specific protocol's port for a destination IP/subnet	`-A FORWARD -d <dst_ip>[/<subnet>] -p <protocol> -dport <dst_port> -j DROP`

TABLE 4.1: Allowed `iptables` rules formats.

the complexity of rule generation and parsing, the pool of allowable rules was constrained to a limited set of predefined formats, summarized in Table 4.1.

4.2.2 Rule parser

To match `iptables` rules with connections, the custom parser first converts the rule to a JSON representation. For example, the rule

```
iptables -A FORWARD -d 10.122.2.10 -p tcp --dport 22 -j DROP
```

is mapped to

```
{
"option": "-A",
"table": "FORWARD",
"src_ip": "any",
"dst_ip": "10.122.2.10",
"protocol": "tcp",
"src_port": "any",
"dst_port": "22",
"jump": "DROP",
}
```

The JSON rule is then matched against the relevant fields of a connection. Since the rules in the `FORWARD` chain apply to packets in either directions, flows are matched in case at least one packet in either directions is matched. In other words, the rule above also matches a connection with `src_ip` 192.14.2.10, `protocol` TCP, `src_port` 22, as long as the number of source-to-destination bytes in the original flow is > 0. Subnets are parsed as regular IP addresses, but a different matching function is used, which compares only prefixes.

Chapitre 5

Experimental results

5.1 Dataset

The dataset selected for NIRS evaluation was chosen to closely mirror the intended deployment setup. Specifically, datasets where network flows were collected at the router level were prioritize, since this facilitates matching them against the generated `iptables` rules. Several commonly used NIDS datasets (such as CIC-IDS 2017 [15], EdgeIIoTSet [3], and TON-IoT [9]) were deemed unsuitable for this use case. In CIC-IDS 2017, traffic is captured at the host level within a private subnet behind a NAT, and external attacks are recorded as originating from the router's IP address rather than the true source. While it is trivial to map internal private IPs to public ones, identifying the correct external attacker IPs that should replace the router IP is non-trivial. Similar issues are present in EdgeIIoTSet, where traffic is also collected individually at the host level. In the case of TON-IoT, all hosts are connected through a vSwitch within the same subnet, making the application of `iptables` for routing-based mitigation infeasible.

Instead, the UNSW-NB15 dataset [10] – also widely used in the NIDS literature – was eventually selected. In UNSW-NB15, traffic is captured by a router, monitoring interactions across three main subnets : subnet 1 (59.166.0.0/24) and subnet 3 (149.171.126.0/24) generate normal traffic, while attacks originate from subnet 2 (175.45.176.0/24). These subnets exchange traffic among themselves and with other hosts through two routers, (linking subnets 10.40.85.0/24 and 10.40.182.0/24), one of which is used to capture traffic. Our experimental evaluation assumes router 1 is a Linux machine acting as router, which is also running a NIRS instance to dynamically update the `iptables FORWARD` chain.

5.2 Hyperparameters

The practical implementation of NIRS requires to define a number of hyperparameters that determine how rules are selected and how often the ruleset r_t is updated.

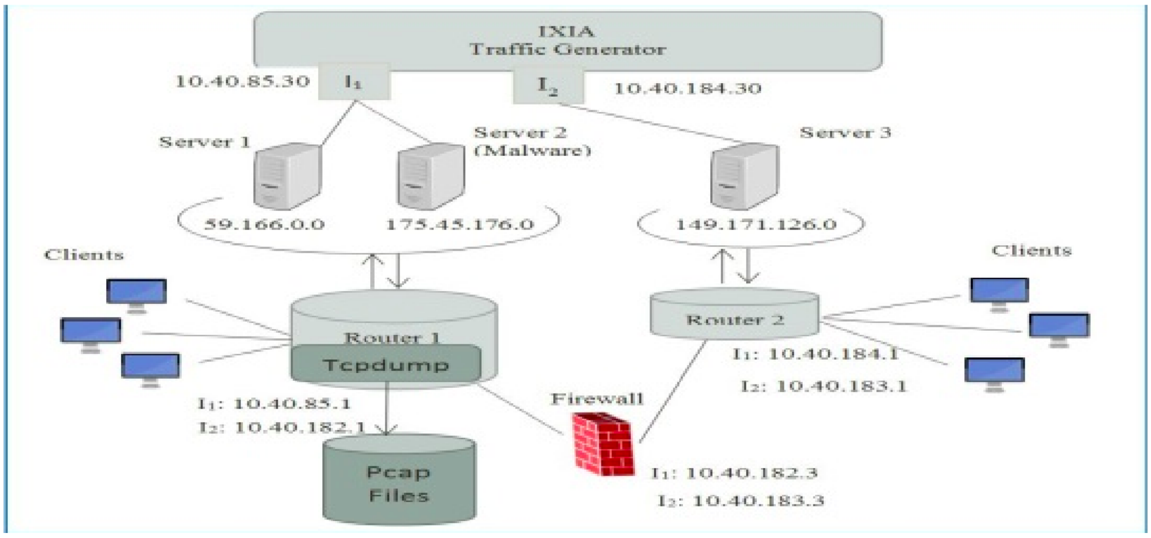


FIGURE 5.1: Setup used to collect UNSW-NB15 [10].

Hyperparameter	**Value**
Normal-traffic window size ΔT_N	10min
Alert window timeout ΔT_A	1min
Interval between NIRS updates Δt	30min
Max number of rules per ruleset n_{rules}	10

TABLE 5.1: Hyperparameters used in our evaluation of the baseline NIRS.

In all the experiments reported below, each ruleset update includes either zero or one additional rules. Each new rule is appended to the `FORWARD` chain. The case where no rule is added happens when the NIRS generates an invalid rule. To prevent an excessive growth of the ruleset, a cap of n_{rules} rules is enforced ; when this threshold is exceeded, the oldest rule is discarded. The `iptables` rules should be enforced on Linux machines that are used as firewalls.

Ruleset updates are performed at a constant rate, every Δt seconds. A lower value of Δ_t implies more timely rule updates, but also more frequent changes of the ruleset. The time range ΔT_N of the normal traffic window and the alert window timeout ΔT_A are also hyperparameters that can affect NIRS performance. These hyperparameters were fixed in all experiments to the values shown in Table 5.1.

5.3 Heuristic-based NIRS $\mathcal{R}_1$

The first implementation of NIRS tested in this work relies on simple heuristics. More specifically, the core idea is to simply block the IP address that caused most alerts.

This can be accomplished by having the NIRS produce at each update a rule of the form

```
-A FORWARD -s {{src_ip}} -j DROP
```

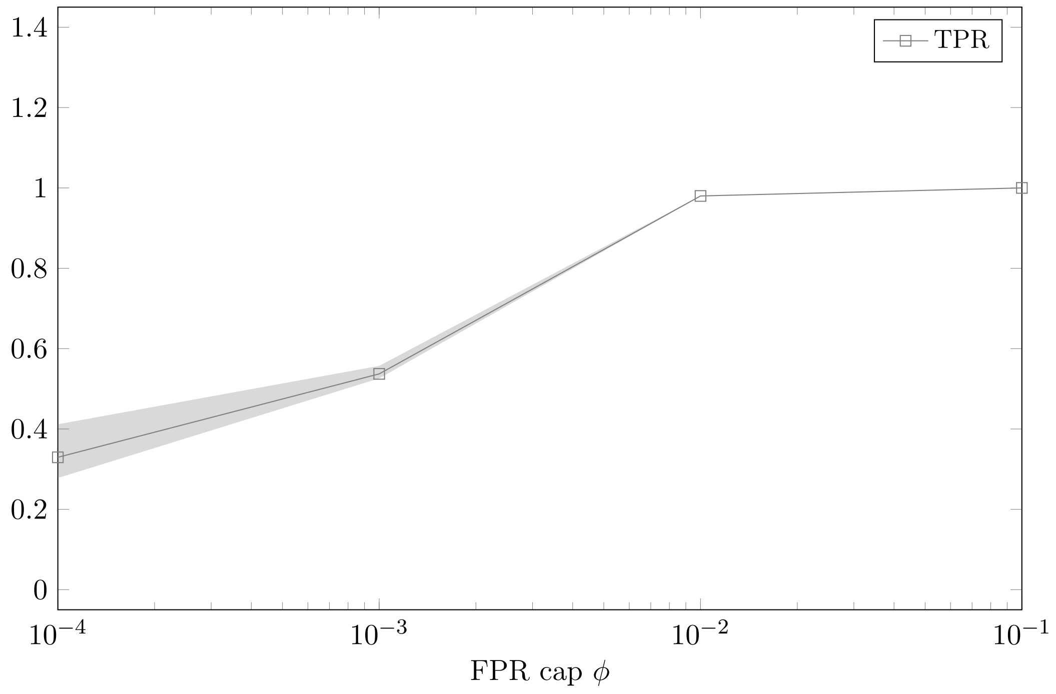


FIGURE 5.2: TPR versus ϕ.

The `src_ip` is selected as the most frequent IP among the alerts $\bar{X}_A$. However, since alerts may include false positives, the rule should also block a limited amount of benign traffic. Therefore, a threshold $\epsilon = 0.01$ is set for the occurrence of `src_ip` in the set of benign connections $\bar{X}_N$. If that limit is exceeded for the most frequent IP address in the alert window, the second most frequent IP is selected, and so on.

5.3.1 Ideal NIDS $\mathcal{D}^*$

The first experiment performed to assess $\mathcal{R}_1$ aims to evaluate the NIRS under ideal conditions. Specifically, Algorithm 1 was executed while simulating a perfect NIDS $\mathcal{D}^*$. This is done by assigning the correct labels to all connections, with no false positives or false negatives. This allows to isolate and evaluate the rule-generation and enforcement capabilities of the NIRS without interference from detection errors. Under these assumptions, $\mathcal{R}_1$ exhibited strong performance in terms of CBR and WBR, successfully mitigating the majority of malicious traffic. It generated rules that blocked the IP addresses of each attacker from the subnet 175.45.176.0/24. However, it also incorrectly blocked the IP address 149.171.126.12, which was a legitimate host that happened to be under attack and not generating benign traffic at the time of rule creation.

5.3.2 Random forest NIDS $\mathcal{D}_\phi$

In the second experiment, the ideal NIDS was replaced with a real ML-based NIDS $\mathcal{D}_\phi$. Specifically, a random forest classifier was trained on the first 30% of the connections. The random forest comprised 100 trees with maximal depth 10, a common choice of hyperparameters for this type of model. To provide an unbiased evaluation, the 30% of the data used for training was excluded from the evaluation when calculating TPR, CBR, and WBR.

A crucial part of this assessment consisted in measuring how much the TPR/FPR tradeoff of a NIDS affects NIRS performance. To do so, instead of using the Random Forest predicted labels directly, probability value p_i was extracted for each connection x_i derived from its Gini impurity. By setting a threshold θ to these probabilities, it is possible to enforce specific FPR caps and move across different points of the NIDS receiver operating characteristic (ROC) curve. Experiments were performed for $\phi = 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$ and repeated over 5 trials, each initialized with a PRNG seed corresponding to the trial number to ensure reproducibility. As shown in Fig. 5.2, the true positive rate (TPR) of the NIDS increases as ϕ increases, following the expected ROC curve pattern. The optimal operating point appears to be near $\phi = 10^{-2}$, where the TPR saturates to 1.

Due to the use of a fixed temporal split, the TPR variance across trials remains minimal ; however, the actual connections labeled as benign or malicious vary, causing downstream effects on the NIRS performance. These variations notably impact $\mathcal{R}_1$, as illustrated in Fig. 5.3. Since $\mathcal{R}_1$ operates under a constraint to block less than 1% of the normal window $\bar{X}_N$, false negatives can significantly reduce its CBR. The CBR remained below 60% across all trials. On the other hand, this conservative approach consistently maintained a low WBR, staying under 8%. A different tradeoff can be achieved by increasing ϵ. However, some brief experimentation with larger ϵ values (e.g., $\epsilon = 0.1$) revealed that this leads to blocking a large fraction of benign traffic, which in a real-world deployment would disrupt regular operations.

5.3.3 Lessons learned

This baseline NIRS implementation reveals that, as one would expect, it is difficult to achieve effective rule generation via heuristic approaches. While it would be possible to try different rule patterns – e.g., by blocking specific `dst_ip:port` or `dst_ip/protocol` combinations – this would increase the search space and problem complexity. Such a result is not surprising : in most problems that are based on pattern recognition (in this case, finding common patterns in alert windows), it is hard to encode human intuition into effective heuristics. This suggests the need to use machine learning for rule generation. In contrast to "hand-crafted" heuristics, machine learning approaches learn patterns based on a set of provided examples. This typically allows them to effectively replicate patterns that are normally easily identified by humans but that are difficult to transpose into algorithms.

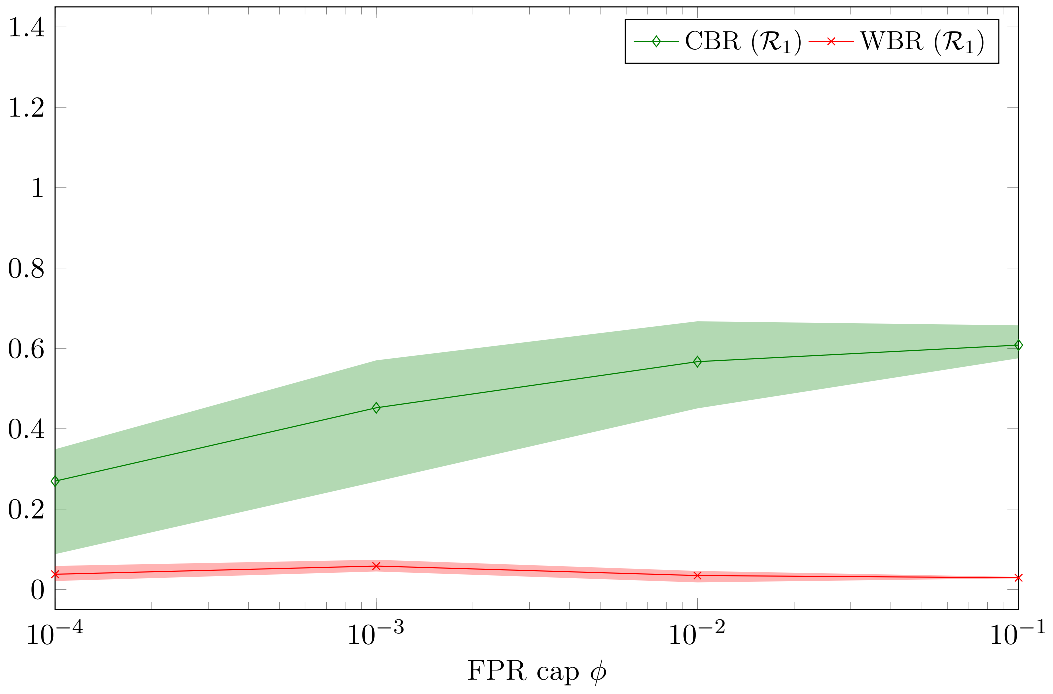
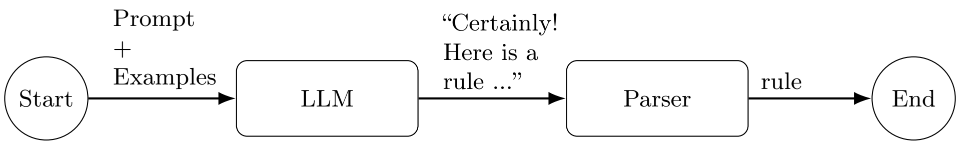


FIGURE 5.3: Correct Block Rate (CBR) and Wrong Block Rate (WPR) versus ϕ for $\mathcal{R}_1$



5.4 LLM-based NIRS $\mathcal{R}_2$

Since heuristic approaches seem to be limited in their effectiveness, the next NIRS baseline implementation was machine learning-based. Among the various possible choices available, large language models (LLMs) were identified as the most prominent one for `iptables` rule generation. More, specifically, open-weights LLMs like Llama or Qwen are pretrained on large corpi of textual data, allowing them to answer questions related to various topics, including computer networks, security, and firewalls. This makes them directly useable without need for additional training.

The second baseline evaluated in this work, $\mathcal{R}_2$, relied on an LLM that has access to $\bar{X}_A$ and $\bar{X}_N$. The LLM is prompted to produce one `iptables` rule at every call, using the template shown in fig. 5.4. The template is divided in a "system prompt," which is set when the LLM is initialized, and a "user prompt," which changes at every LLM call (once per NIRS update).

Since LLMs are limited by their context window (the number of tokens that an LLM is able to process at once), it is not feasible to feed an LLM the windows $\bar{X}_A$ and $\bar{X}_N$ in their entirety. Instead, only the last $k_A = 10$ alerts and $k_N = 10$ benign flows are added to the prompt. Also, LLMs can make mistakes while generating `iptables` rules. These can result in rules that are

$\mathcal{R}_2$ prompts

System prompt : You are a network security engineer. You are tasked with monitoring incoming malicious and benign traffic, and writing one iptables rule accordingly. You will observe examples of benign flows and malicious flows. You will also have access to the current iptables status. Based on this information, you will write one single iptables rule, which should be enclosed within `<rule></rule>` tags. Output only one iptables DROP rule to append to the FORWARD table. The rule must block most of the malicious flows and must not block most of the benign flows. Valid formats for the rule include :
`{{accepted_formats}}`
DO NOT USE RULES WITH DIFFERENT FORMATS. The `/<subnet>` is optional. Examples of valid rules :
`{{few_shot_examples}}`

User prompt : Malicious flows :
`{{malicious_flows}}`
Benign flows :
`{{benign_flows}}`
Iptables status :
`{{iptables_status}}`

FIGURE 5.4: System and user prompt formats used for $\mathcal{R}_2$.

ineffective or syntactically invalid. To handle the latter case, the `iptables` parser introduced in §4.2 is also used as rule validator. If a generated rule is invalid, the ruleset is kept at its current state (i.e., $r_{t+\Delta_t} = r_t$).

5.4.1 Preliminary LLM testing

Before proceeding with the implementation of $\mathcal{R}_2$, preliminary tests were performed on open-weights models, testing their reliability in responding to the system and user prompts defined above. Reasoning models such as `deepseek-r1:8b` and `qwen3:8b` often exhibited undesirable behavior, including excessive generation loops during rule synthesis. While `qwen2.5:7b` was able to produce syntactically valid `iptables` rules, it frequently failed to follow the required encapsulation format, substituting quotes in place of the specified `<rule></rule>` tags.

Ultimately, `llama3:8b` was selected due to its reliable formatting and overall consistency. The model managed to generate correctly formatted rules with reasonable latency without fine-tuning. Additionally, we confirmed that the model has inherent knowledge of `iptables`, as we were able to generate valid rules without providing examples.

The average rule generation time (including text generation and rule parsing) was ≈ 1.6 seconds on an Nvidia GTX 2080 Ti GPU. When no rule is in place, the `iptables` status is set to `[Empty]`. Otherwise, it displays the current list of rules.

5.4.2 Ideal NIDS $\mathcal{D}^*$

When repeating the ideal NIDS experiment, $\mathcal{R}_2$ immediately blocked the entire 175.45.176.0/24 subnet that includes all the attacking IP addresses. An example of output generated by `llama3:8b` is as follows :

```
<rule>-A FORWARD -s 175.45.176.0/24 -j DROP</rule>

This rule blocks all traffic from the malicious IP range
(175.45.176.0/24) to any destination, effectively blocking
most of the malicious flows while allowing benign traffic
to pass through.
```

This strategy yielded a high CBR of 97.55% while maintaining a relatively low WBR of 1.67%, showing promising results for LLM-driven intrusion response under perfect NIDS alerts.

5.4.3 Random forest NIDS $\mathcal{D}_\phi$

While $\mathcal{R}_1$ allows to only block one IP at a time, the LLM-based $\mathcal{R}_2$ allows broader mitigation actions, such as blocking entire subnets. This type of action appears to be the most frequently chosen by the LLM, likely to capture multiple address within the alert window.

While this strategy enabled a high CBR (ranging between 85% and 91%), it also disrupted around 45% of benign connections, making this NIRS unusable in real-world applications. Attempts to reduce this over-blocking through negative prompting (e.g., explicitly instructing the model not to block traffic from 59.166.0.0/24 ; or by warning it about the risk of false positives) were ineffective.

5.4.4 Example of LLM hallucinations

Besides its general tendency to produce broad rules, $\mathcal{R}_2$ exhibited hallucination behavior, claiming in multiple instances to block a single IP address while actually generating rules that blocked whole subnets. This bias toward subnet-level blocking may be attributed to the examples of `iptables` rules observed by `llama3:8b` during training. An example of such allucinations is the following :

```
<rule>-A FORWARD -s 59.166.0.3/24 -j DROP</rule>

This rule blocks traffic from IP address 59.166.0.3, which
is the source IP address for all malicious flows.
This should block most of the malicious traffic while
allowing benign traffic to pass through.
```

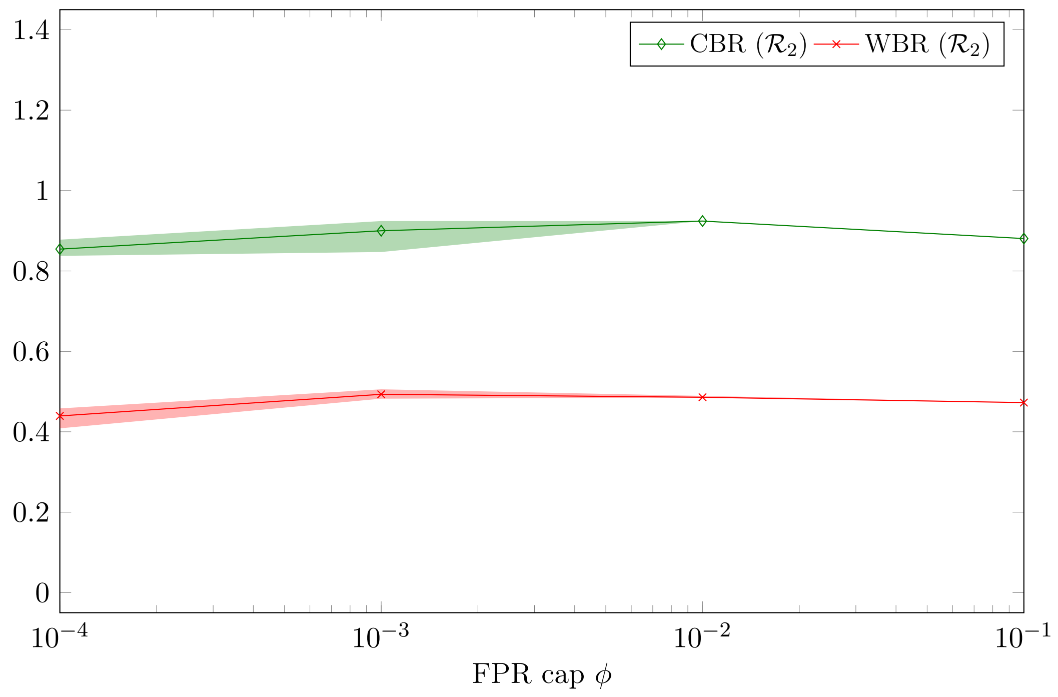


FIGURE 5.5: Correct Block Rate (CBR) and Wrong Block Rate (WPR) versus ϕ for $\mathcal{R}_1$ (left) and $\mathcal{R}_2$ (right).

While the the output claims that this rule only blocks 59.166.0.3, `iptables` prioritizes netmasks, meaning that this rule blocks any IP in the 59.166.0.0/24 subnet.

5.4.5 Result analysis

Figure 5.6 compares the TPR and the resulting CBRs for $\mathcal{R}_1$ and $\mathcal{R}_2$ on different types of attack for $\phi = 0.1$. Notably, even for low TPR values, the NIRS were often still able to block most of the malicious flows, as long as a sufficient number of alerts were generated. However, this effectiveness is also caused by the attack originating from a small address space. Further experimentation is required to understand how the NIRS perform against distributed attacks, where malicious activity may be spread across a wide range of sources [4], potentially diminishing the CBR.

5.4.6 Resilience to adversarial attacks

Any application that implements LLM-based decision making, especially when these can affect critical aspects of a system, potential exploitation (e.g., via prompt injection[8]) should be considered. In the case of NIRS, the risk can be effectively mitigated by adopting a predefined prompt template as done for $\mathcal{R}_2$. Since the only external inputs are numerical indicators or other objective information (such as IP addresses or protocols), there is no

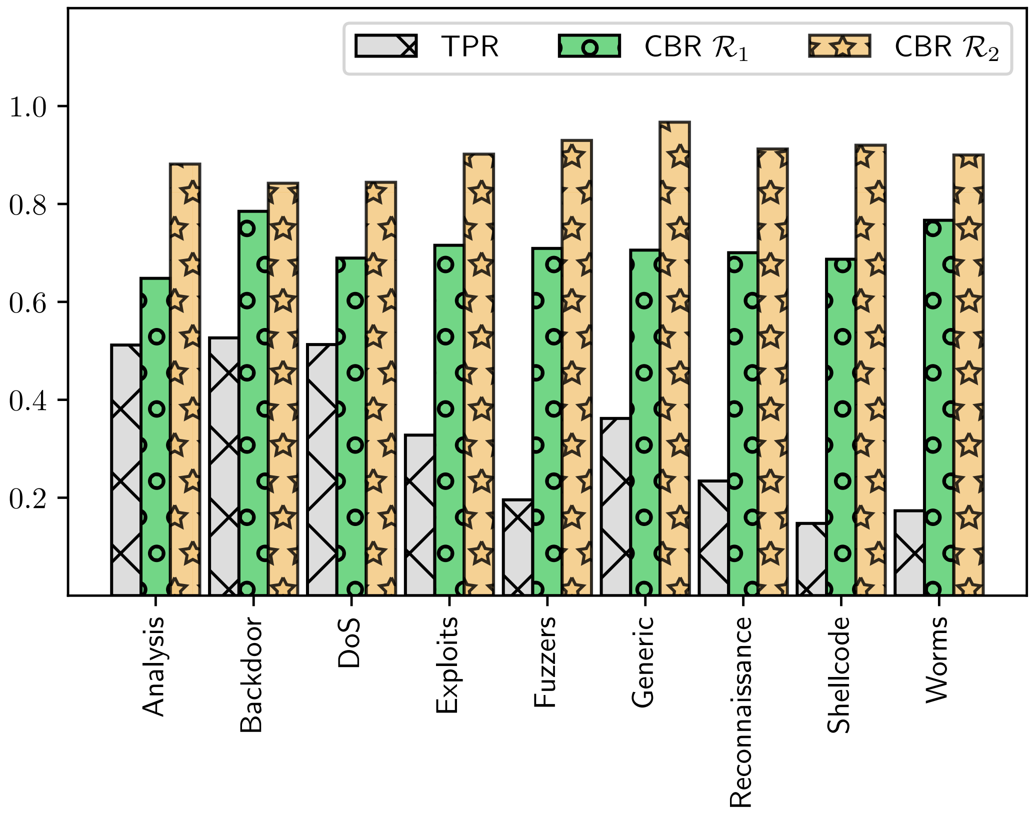


FIGURE 5.6: CBR by attack for $\phi = 0.1$.

Conclusion

This project identified and addressed research gaps in the intrusion response literature. To address these gaps, it introduced the concept of network intrusion response systems (NIRS), designed to transform IDS alerts into firewall/IPS rules. Along with the NIRS definition, an evaluation approach was proposed and implemented to leverage existing IDS datasets, tackling the longstanding problem of ad hoc and non-standardized testbeds in intrusion response research.

Two baseline NIRS, $\mathcal{R}_1$ and $\mathcal{R}_2$, were implemented and evaluated on UNSW-NB15. The first utilized a conservative blocking strategy that maintained low collateral damage (less than 8% of benign traffic disrupted) but achieved only moderate coverage, with correct block rate (CBR) under 60%. The LLM-based NIRS ($\mathcal{R}_2$) demonstrated much higher coverage (85-91% CBR) but frequently blocked legitimate traffic, with 45% wrong block rate (WBR).

Overall, the findings listed in this report allow to draw the following conclusions :

— The proposed definition and evaluation methodology effectively allow to analyze NIRS solutions.

— While neither of the two NIRS achieve high performance in terms of CBR/WBR tradeoff, they represent a preliminary step towards effective network-level response.

— The knowledge embedded in open-weights foundation LLMs such as Llama can be leveraged to translate NIDS alerts into firewall rules without additional training or fine-tuning. Even a simple implementation such as $\mathcal{R}_2$ can provide effective mitigation when the underlying NIDS is accurate enough.

— A number of challenges have been identified for LLM-based NIRS implementations : one is the need for more complex workflows to guide the LLM reasoning during the rule generation process ; another is enforcing LLM compliance with network specific constraints (e.g., compliance with negated prompting “Do not block critical network 59.166.0.3/24”).

The methodology and results described in this report are all reproducible using the code available at the following Github repository : `https://github.com/thomasmarchioro3/NIRS`.

Future work

While some important gaps have been addressed, new challenges emerged that could not be tackled within the scope of this project. LLM-based network intrusion response appears to be promising, but improving their workflow is paramount to achieve effective attack mitigation and avoid service disruption. More specifically, future work should focus on integrating LLMs with tools for assessment and validation of generated rules (before the firewall/IPS is updated). Additionally, the problem of compliance with negated prompting should be addressed. Preliminary experiments revealed that fine-tuning via direct preference optimization (DPO) strongly can be used to improve the model understanding of negated prompting. These results should be further validated via systematic experiments.

The limitations of the setup used in the experiments should also be considered. While `iptables` provides a concrete implementation of firewall rulesets, it makes the analysis limited to setups that fit this implementation. For this reason, many public datasets could not be used in the NIRS evaluation. More flexible rule formats, such as the unified defense rule proposed in [18], could be used to fully leverage the wealth of available IDS datasets.

Bibliographie

[1] D. Antonioli and N. O. Tippenhauer. Minicps : A toolkit for security research on cps networks. In *Proceedings of the First ACM workshop on cyber-physical systems-security and/or privacy*, pages 91–100, 2015.

[2] G. Apruzzese, P. Laskov, and J. Schneider. Sok : Pragmatic assessment of machine learning for network intrusion detection. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 592–614. IEEE, 2023.

[3] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke. Edge-iiotset : A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning. *IEEe Access*, 10 :40281–40306, 2022.

[4] H. Friji, I. Mavromatis, A. Sanchez-Mompo, P. Carnelli, A. Olivereau, and A. Khan. Multi-stage attack detection and prediction using graph neural networks : An iot feasibility study. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 620–627. IEEE, 2023.

[5] K. Hughes, K. McLaughlin, and S. Sezer. A model-free approach to intrusion response systems. *Journal of Information Security and Applications*, 66 :103150, 2022.

[6] S. Iannucci, Y.-C. Huang, and E. Bertino. A performance evaluation of deep reinforcement learning for model-based intrusion response. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 83–88. IEEE, 2019.

[7] Y. W. Law, T. Alpcan, and M. Palaniswami. Security games for risk minimization in automatic generation control. *IEEE Transactions on Power Systems*, 30(1) :223–232, 2014.

[8] Y. Liu, G. Deng, Y. Li, K. Wang, Z. Wang, X. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng, et al. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv :2306.05499*, 2023.

[9] N. Moustafa. A new distributed architecture for evaluating ai-based security systems at the edge : Network ton_iot datasets. *Sustainable Cities and Society*, 72 :102994, 2021.

[10] N. Moustafa and J. Slay. Unsw-nb15 : a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[11] T. V. Phan and T. Bauschert. Deepair : Deep reinforcement learning for adaptive intrusion response in software-defined networks. *IEEE Transactions on Network and Service Management*, 19(3) :2207–2218, 2022.

[12] A. F. M. Piedrahita, V. Gaur, J. Giraldo, A. A. Cardenas, and S. J. Rueda. Virtual incident response functions in control systems. *Computer Networks*, 135 :147–159, 2018.

[13] J. R. Rose, M. Swann, K. P. Grammatikakis, I. Koufos, G. Bendiab, S. Shiaeles, and N. Kolokotronis. Ideres : Intrusion detection and response system using machine learning and attack graphs. *Journal of Systems Architecture*, 131 :102722, 2022.

[14] D. Schlette, P. Empl, M. Caselli, T. Schreck, and G. Pernul. Do you play it by the books ? a study on incident response playbooks and influencing factors. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3625–3643. IEEE, 2024.

[15] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani, et al. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1(2018) :108–116, 2018.

[16] M. Sheeraz, M. A. Paracha, M. U. Haque, M. H. Durad, S. M. Mohsin, S. S. Band, and A. Mosavi. Effective security monitoring using efficient siem architecture. *Hum.-Centric Comput. Inf. Sci*, 13 :1–18, 2023.

[17] M. D. R. Team. Cyberbattlesim. `https://github.com/microsoft/cyberbattlesim`, 2021. Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei.

[18] F. Wei, H. Li, Z. Zhao, and H. Hu. {xNIDS} : Explaining deep learning-based network intrusion detection systems for active intrusion responses. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4337–4354, 2023.

[19] T. Yarygina and C. Otterstad. A game of microservices : Automated intrusion response. In *Distributed Applications and Interoperable Systems : 18th IFIP WG 6.1 International Conference, DAIS 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings 18*, pages 169–177. Springer, 2018.

[20] S. A. Zonouz, H. Khurana, W. H. Sanders, and R. Yardley. Rre : A game-theoretic intrusion response and recovery engine. *IEEE Transactions on Parallel and Distributed Systems*, 25(2) :395–406, 2013.