xNIDS: Explaining Deep Learning-based Network Intrusion Detection Systems for Active Intrusion Responses

Feng Wei
University at Buffalo

Hongda Li
Palo Alto Networks

Ziming Zhao
University at Buffalo

Hongxin Hu
University at Buffalo

Abstract

While Deep Learning-based Network Intrusion Detection Systems (DL-NIDS) have recently been significantly explored and shown superior performance, they are insufficient to actively respond to the detected intrusions due to the semantic gap between their detection results and actionable interpretations. Furthermore, their high error costs make network operators unwilling to respond solely based on their detection results. The root cause of these drawbacks can be traced to the lack of explainability of DL-NIDS. Although some methods have been developed to explain deep learning-based systems, they are incapable of handling the history inputs and complex feature dependencies of structured data and do not perform well in explaining DL-NIDS.

In this paper, we present xNIDS, a novel framework that facilitates active intrusion responses by explaining DL-NIDS. Our explanation method is highlighted by: (1) approximating and sampling around history inputs; and (2) capturing feature dependencies of structured data to achieve a high-fidelity explanation. Based on the explanation results, xNIDS can further generate actionable defense rules. We evaluate xNIDS with four state-of-the-art DL-NIDS. Our evaluation results show that xNIDS outperforms previous explanation methods in terms of fidelity, sparsity, completeness, and stability, all of which are important to active intrusion responses. Moreover, we demonstrate that xNIDS can efficiently generate practical defense rules, help understand DL-NIDS behaviors, and troubleshoot detection errors.

1 Introduction

Deep Learning-based Network Intrusion Detection Systems (DL-NIDS) [22, 34, 37, 46, 51] have recently shown promising potential and demonstrated outstanding detection performance [41, 62, 64, 82], one key enabler of which is that DL-NIDS can detect unseen attacks [14] thanks to their strong ability to capture complicated patterns and trivial deviations [25, 40]. Despite the benefits of DL-NIDS, they are insufficient to actively respond to what they detect [3] because of the semantic gap between their detection results and actionable interpretations [70] as depicted in Fig. 1. Moreover, misdetections generated by DL-NIDS require network operators to spend a significant amount of time and effort on troubleshooting, resulting in high error costs [6]. These drawbacks are rooted in the design of DL-NIDS [70] since they maintain complex structures to capture different network traffic patterns, which unfortunately lead to low explainability [27]. Consequently, it is difficult to understand why DL-NIDS make certain decisions, which makes troubleshooting detection errors difficult and lowers the confidence of corresponding intrusion responses.

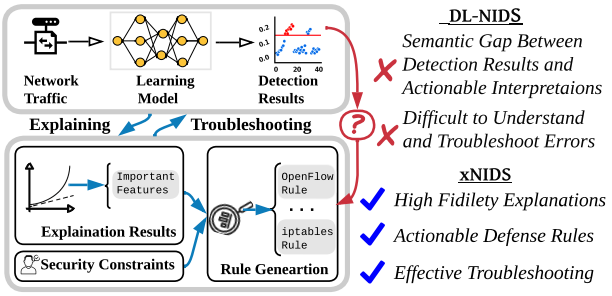


Figure 1: An illustration of how xNIDS explains DL-NIDS and enables active intrusion response.

Several methods [7, 19, 23, 55, 65, 67, 89] have been developed to explain results from deep learning models. Although it is difficult to explain the entire prediction process globally, existing methods provide a local explanation for a specific input sample and the decision on it by highlighting the most important features [55]. For example, Gradients [67], IG [73], LRP [7], LIME [55], and SHAP [43] are used to explain image classifiers [7, 8] and Natural Language Processing (NLP) models [56, 78]. LEMNA [27], which is specifically designed for deep learning-based security applications, can explain binary code analysis tools [63] and malware detection tools [68].

However, existing methods, including LEMNA, are incapable of explaining DL-NIDS for the following reasons: (1)

deep learning models for image classification or malware detection make decisions based on a single input sample, *e.g.*, an image, a piece of binary code, etc. Such a model can be denoted as $y_t = f(\mathbf{x}_t)$, where $\mathbf{x}_t$ is a sample represented by a d-dimensional feature vector $(x_1, ..., x_d)^T$. However, the input samples of DL-NIDS, *e.g.*, network packet headers, form a time series and are interdependent with each other. Accordingly, DL-NIDS make the decision based on the current input sample as well as k history inputs, which is denoted as $y_t = f(\mathbf{x}_t, \mathbf{x}_{t-1}, ..., \mathbf{x}_{t-k})$ [46, 70]. Existing explanation approaches [27, 43, 55], unfortunately, are insufficient to take the history input samples into account. For example, the typical sampling methods for local exploration in these approaches only sample instances in the vicinity of $\mathbf{x}_t$ and ignore the vicinities of $\mathbf{x}_{t-1}$, ..., and $\mathbf{x}_{t-k}$; and (2) existing explanation approaches are insufficient to capture the complex dependencies among the features since they either assume features within the input are independent [43, 55, 73] or adjacent features have similar contributions [27]. Those assumptions may hold for image classification tasks where each byte in an image independently represents the color of a pixel, and malware detection tasks where a piece of binary code can be interpreted as the beginning of an instruction. However, the fields in network packet headers of DL-NIDS input samples have well-defined meanings and complex dependencies. For example, `TCP.flag` is a sub-feature of `TCP`, which means if the `TCP.flag` feature is valid, then the `TCP` feature is valid as well. Also, adjacent features do not necessarily contribute similarly. For example, the `TCP` and `UDP` features are adjacent but mutually exclusive.

Moreover, there is no research on utilizing the explanation results from DL-NIDS to generate defense rules for active intrusion responses. Existing methods, such as FIRMA [54], utilize network signatures and blacklists to detect and respond to network intrusions, respectively. However, those response methodologies (*e.g.*, blacklists) are too strict and their generated rules can only support specific defense tools. In fact, it is crucial to generate *accurate* defense rules that can only alert the traffic influenced by attacks [3], since an over-specific defense rule may lead to false negatives in intrusion responses, whereas an over-generic defense rule may result in false positives in intrusion responses. Also, it is challenging to generate *practical* defense rules that can be used by heterogeneous defense tools, such as iptables [32] and Pfsense [53], which have different rule syntax and granularity.

In this paper, we present XNIDS, a new framework that explains DL-NIDS and uses its explanation results to generate actionable defense rules. The explanation method in XNIDS addresses the aforementioned challenges by (1) finding a small number of history inputs that lead to a prediction in the vicinity of the original prediction and sampling around each of the history inputs; and (2) capturing feature dependencies of the structured data with feature groups and sparse group lasso [66]. The defense rule generator in XNIDS enables active intrusion responses by introducing defense rule scopes, security constraints, and a unified defense rule representation for generating accurate and practical defense rules.

To evaluate XNIDS, we apply it to four state-of-the-art DL-NIDS: (1) the autoencoder-based Kitsune [46]; (2) the Long-Short-Term-Memory (LSTM) based bot detection system, ODDS [33]; (3) the Recurrent Neural Networks (RNN) based RNN-IDS [82]; and (4) the deep autoencoder (AE) based AE-IDS [64]. Our evaluation results show XNIDS outperforms existing approaches in terms of fidelity, sparsity, completeness, and stability in explaining DL-NIDS. We also showcase that XNIDS can help troubleshoot the detection errors of DL-NIDS. Furthermore, our experiments show that XNIDS can generate effective defense rules for four real-world defense tools, including iptables [32], OpenFlow [50], Pfsense [53], and Squid [72], to defend various attacks.

The key contributions of this paper are as follows:

- We design a novel explanation method dedicated to explaining DL-NIDS. By approximating and sampling the history inputs and capturing the feature dependencies with sparse group lasso, our explanation method generates high-fidelity, sparse, complete, and stable explanation results for DL-NIDS.
- We present a defense rule generation methodology to enable active intrusion responses. We introduce defense rule scopes and security constraints to make rules accurate and design a unified rule representation to make defense rules applicable to heterogeneous defense tools.
- We evaluate our framework, XNIDS [1], with four state-of-the-art DL-NIDS, demonstrate the effectiveness of its explanation method, and showcase how it can help understand DL-NIDS behaviors, troubleshoot detection errors, and enable active responses.

2 Motivations and Challenges

In this section, we discuss why DL-NIDS explanation and active intrusion response are important but challenging.

2.1 Why are DL-NIDS explanation and active intrusion response important?

Semantic Gap. *Explaining DL-NIDS bridges the semantic gap between their detection results and actionable interpretations.* Though DL-NIDS can detect various attacks, they lack the capability to actively respond to what they find [3] due to the semantic gap between their detection results and actionable interpretations [70]. The detection results of DL-NIDS are usually presented as numerical values or even simpler labels, but network operators need to interpret the reasons

[1] The source code of XNIDS is available at https://github.com/CactiLab/code-xNIDS.git.

Table 1: Comparison of the state-of-the-art explanation methods with XNIDS. ● means totally support, ◐ means partially support, while ○ means not support.

Explanation methods	MLP	CNN	RNN	History Input	Feature Dependency
LRP [7], DeepLift [65] Gradients [67], IG [73], CADE [81], DeepAID [28]	●	●	◐	○	○
GradCAM [60], CAM [88], Occlusion [19], RTIS [10], GuidedBP [71]	●	●	○	○	○
MEME [36]	○	○	●	◐	○
LIME [55], SHAP [43], LEMNA [27], QII [11]	●	●	◐	○	◐
XNIDS	●	●	●	●	●

behind those scores for proper responses. For example, network operators need to know which network entity, such as host or flow, to act on. Explaining the detection results in a detailed and human-understandable way, e.g., which features are more important, gives network operators confidence in the detection results and also provides them with an interpretation that can be used to build the corresponding response.

Troubleshoot Errors. *Explaining the DL-NIDS detection results helps troubleshoot classification errors.* The cost of misclassification in DL-NIDS is very high, in which an attacker can penetrate the network with a single false negative, and false positives swamp the network operators with further manual analysis [6]. Troubleshooting DL-NIDS, however, is difficult since the models are getting more complicated and incomprehensible for humans. Explaining the detection results regarding inputs with important features reduces the human effort for troubleshooting.

Active Response. *Active intrusion response helps block or shunt malicious traffic promptly.* DL-NIDS detects a wide range of attacks that require network operators to conduct a time-consuming manual analysis before proper responses. However, a delayed intrusion response is costly since attackers may have penetrated the network. Therefore, it is imperative to enable active intrusion response based on the detection results of DL-NIDS and corresponding explanations.

2.2 Why are existing explanation approaches insufficient in explaining DL-NIDS?

Although existing methods [7,73] have achieved great success in explaining batch data, they are insufficient in explaining DL-NIDS. We summarize the-state-of-the-art explanation methods in Table 1. Existing methods interpret the individual detection result of target deep learning model by generating an importance score vector $\beta = (\beta_i, ...\beta_d)^T$ to describe the importance of each feature of $\mathbf{x}_t = (x_1, ..., x_d)^T$ regarding $f(\mathbf{x}_t)$. Based on whether or not knowing the inner workings of the target models, these explanation methods can be categorized into whitebox [19,60,65,88] or blackbox approaches [27,43,55]. In the following, we show five representative explanation methods and analyze their limitations.

IG. As shown in Equ. (1), Integrated Gradients (IG) [73] calculates the importance score β_i of feature x_i by accumulating the gradients regarding x_i along the path $x_i - x_i'$. It is evident that the important score β_i of feature x_i and β_j of feature x_j $(i \neq j)$ are independently calculated.

$$\beta_i = (x_i - x_i') \int_0^1 \frac{\partial f(\mathbf{x} + \alpha(\mathbf{x} - \mathbf{x}'))}{\partial x_i} d\alpha \tag{1}$$

LRP. As shown in Equ. (2), Layer-wise Relevance Propagation (LRP) [7] calculates the importance score β_i for feature x_i by tracing back its contribution to the detection layer by layer. Similar to IG, β_i^l and β_j^l $(i \neq j)$ are independently calculated in LRP. ARRAS [5] uses LRP to explain LSTM, but it also ignores the feature dependencies by omitting the gate neuron.

$$\sum_i \beta_i^1 = \sum_i \beta_i^2 \cdots = \sum_i \beta_i^L = f(\mathbf{x}) \tag{2}$$

LIME and SHAP. As shown in Equ. (3), LIME [55] and SHAP [73] assume each feature x_i is independent while calculating the corresponding important score β_i by *lasso* [17,76]. Consequently, the explanations of LIME and SHAP are independently selected features. The difference here is that when calculating the weights vector π, LIME uses cosine similarity while SHAP uses Shapely Values [61].

$$\underset{\beta}{argmin} \left\{ \pi ||\mathbf{y} - \mathbb{X}\beta^T||_2^2 + \lambda \sum_{j=1}^{d} ||\beta_j||_1 \right\} \tag{3}$$

LEMNA. As shown in Equ. (4), LEMNA [27] uses a mixture of K number of regression models to calculate the important score β_i. In order to address the adjacent feature dependencies, LEMNA assumes adjacent features have similar importance score and encourages the flatness of the vector β by fused lasso [77] regarding the penalty $||\beta_j - \beta_{j-1}||_1$.

$$\underset{\beta}{argmin} \sum^{K} \pi ||\mathbf{y} - \mathbb{X}\beta^T||_2 \;\; s.t. \sum_{i=1}^{d} ||\beta_i||_1 \leq s_1, \;\; \sum_{j=2}^{d} ||\beta_j - \beta_{j-1}||_1 \leq s_2 \tag{4}$$

In summary, most of the existing explanation methods (*e.g.* IG, LRP, GuidedBP, GradCAM, or CAM) explain Multilayer Perceptron (MLP) and Convolutional Neural Networks (CNN) well but are insufficient to explain Recurrent Neural Networks (RNN), since they assume that the features are independent [47]. LEMNA [27] is designed to explain security applications, but the assumption of LEMNA may not always hold. MEME [36] approximates RNN models via decision trees and MLP. Instead of selecting relevant features for a specific decision, MEME adopts concept extraction for a model explanation. For local explanation, MEME directly uses LIME [55], hence it has the same assumptions as LIME. Consequently, as shown in Table 1, those existing explanation methods are insufficient to explain DL-NIDS, considering the history inputs and feature dependencies of structured data.

2.3 Challenges in Explaining DL-NIDS

Since DL-NIDS makes decisions based on the current input samples as well as history inputs, to properly explain the detection results, we assume datasets that contain relevant history samples are available. We summarize the major challenges in explaining DL-NIDS as follows:

Ch1: ***How to consider history inputs?*** Existing explanation methods [27, 43, 55] interpret a detection result by highlighting the features that significantly contribute to the detection result. However, they are insufficient to interpret the detection results regarding history inputs. Schlegel et al. [58] assess the quality of selected explanation methods [55, 65, 67, 73] on time series with a fixed number of inputs. Their experiments show a considerable accuracy decrease for all the evaluated explanation methods when applied to time series. Therefore, they argue that there is a need to design more suitable explanation methods on time series for better explanations.

We identify two issues with respect to considering history inputs to explain DL-NIDS. First, it is likely to get degenerated explanations if only a fixed number of history inputs are considered, since different attacks may rely on different numbers of inputs (*e.g.*, `DDoS`, `OSscan`), whereas it is also infeasible to consider all the history inputs. Second, the current input may have more influence on the detection results of DL-NIDS than those history inputs, especially old ones.

Ch2: ***How to capture complex feature dependencies in structured data?*** Most of the existing approaches, such as IG, LRP, and LIME, are insufficient to leverage the structure information in data by assuming features are independent within the input [47], which leads to poor explanation fidelity. Other work, such as SHAP, has the feature-group (*e.g.*, superpixel) setting but assumes each feature inside the group has equal credits [42, 61]. LEMNA addresses this problem by assuming that adjacent features have similar contributions to the detection results. However, the inputs for DL-NIDS are well-structured, *e.g.*, IP frame, with complex feature dependencies.

2.4 Challenges in Generating Defense Rules

Some approaches use the results from signature-based NIDS for active intrusion responses [54]. However, no existing work uses the explanation results of DL-NIDS for intrusion responses. We identify two key challenges in enabling effective active responses with explanation results from DL-NIDS.

Ch3: ***How to balance precision and generalization for practical defense rule generation?*** Even if an explanation approach can identify the important features, *e.g.*, `IP` address, it is still challenging to generate accurate defense rules with these features. If the generated rules are too fine-grained, *e.g.*, only matching specific flows, it results in overfitting and an overwhelming number of defense rules. On the other hand, if the generated rules are too generic, they may disrupt benign traffics. For example, explanation methods identify an IP address, protocol, and port number as important features. An over-generic rule that blocks all packets from the specific protocol *e.g.*, TCP, may be effective but disruptive.

Ch4: ***How to generate universally applicable defense rules for different defense tools?*** Although the functionalities of network defense tools, *e.g.*, firewall and intrusion response system, are similar, they use different formats and rule granularity. For example, the rules in OpenFlow [50] and iptables [32] have much different syntax. To block `TCP.SYN` flood, the rule for OpenFlow looks like `<nw_src=192.168.1.10, tcp, tcp.syn, actions = drop, priority = 1, hard_timeourt=60>`, whereas the rule for iptables looks like `<iptables -A INPUT -p tcp --syn -m limit --limit-burst 3 -j RETURN>`. The rules generated by explainable DL-NIDS should be applicable to different network defense tools.

3 Explaining Detection Results of DL-NIDS

In this section, we first introduce the goal of our explanation method. Then, we address **Ch1** and **Ch2** by (1) approximating the history inputs; (2) synthesizing instances around history inputs by Weighted Random Sampling (WRS) [16]; (3) dividing features within each input to groups regarding their correlations; and (4) selecting important features in a sparse manner both on the group level and the feature level. Finally, we integrate the techniques discussed above into a proper explanation model to derive explanation results by approximating the detection results of DL-NIDS.

3.1 Notation and Design Goal

Notation. We use bold lowercase letters, such as $\mathbf{x}$, to represent a vector and bold capital letters, such as $\mathbf{X}$, to represent a matrix or a sequence of vectors. We denote by $\mathbf{x}_t$ the current input and $(\mathbf{x}_{t-1}, ..., \mathbf{x}_{t-k})$ or $\mathbf{X}_{t,k}$ its k history inputs. An DL-NIDS is denoted by $f(\mathbf{x}_t, \mathbf{x}_{t-1}, ..., \mathbf{x}_{t-k})$ or $f(\mathbf{x}_t, \mathbf{X}_{t,k}) \in \mathbb{R}$. Denote by $g(\mathbf{x}_t, \mathbf{X}'_{t,m}) \in \mathbb{R}$ a local approximation to $f(\mathbf{x}_t, \mathbf{X}_{t,k})$ considering l history input, since the k used in $f(\cdot)$ is unknown to $g(\cdot)$. We define the explanation result as $\mathbf{e}$, which contains the desired important features. We denote by $||\cdot||_1$ the generic and $||\cdot||_2$ the l_2 norm of a vector. Denote by $\mathcal{L}(\cdot)$ the loss function that measures how faithful one model locally approximates another. $\phi(\cdot)$ denotes the sparsity of a vector.

Our Goal. Given an input $\mathbf{x}_t$, our explanation method needs to find proper history inputs $\mathbf{X}'_{t,m}$ and a high-fidelity, sparse, complete, and stable explanation result $\mathbf{e}$. In our design, $\mathbf{e}$ is determined by the parameters β of $g(\cdot)$. To this end, we formulate our explanation method by the following:

$$\underset{g}{argmin}\left\{\underbrace{\mathcal{L}(f,g)}_{\text{Fidelity}} + \underbrace{\lambda \cdot \phi(\beta)}_{\text{Sparsity}}\right\} \quad s.t. \quad \underbrace{\left\{||f(\mathbf{x}_t, \mathbf{X}'_{t,m}) - y_t||_1 < \delta\right\}}_{\text{History inputs}} \quad (5)$$

3.2 Approximating History Inputs

DL-NIDS usually adopts sliding window or RNN to capture the aggregation information of benign and abnormal traffic, which can be described as $y_t = f(\mathbf{x}_t, \mathbf{X}_{t,k})$. An explanation results derived only from $\mathbf{x}_t$ or a fixed number of history inputs are insufficient, resulting in low fidelity explanations. Also, it is insufficient to consider all the historical inputs.

To effectively find a small number of inputs to approximate the relevant history inputs, our approach has two steps. In step 1, we find a small l for which $||f(\mathbf{x}_t, \mathbf{X}_{t,l}) - f(\mathbf{x}_t, \mathbf{X}_{t,k})||_1 < \delta$ and δ is a small deviation regarding the original detection score. We start with a configurable value for l. If $||f(\mathbf{x}_t, \mathbf{X}_{t,l}) - f(\mathbf{x}_t, \mathbf{X}_{t,k})||_1 \geq \delta$, we update $l = 2l$. If $||f(\mathbf{x}_t, \mathbf{X}_{t,l}) - f(\mathbf{x}_t, \mathbf{X}_{t,k})||_1 < \delta$, we update $l = \lfloor l/2 \rfloor$. There are two termination requirements for this process: the deviation δ, which determines the precise of the approximation; and the number (U) of the largest times for updating l. We repeat this process until one of the termination requirements is satisfied. In step 2, from the identified l history inputs, we choose the most relevant ones by removing some history inputs and checking if the detection result still satisfies the δ requirement. Intuitively, it is possible that some history inputs within the range l are irrelevant. We denote the output of this step as $\mathbf{X'}_{t,m}$, which is an $m \times d$ submatrix of $\mathbf{X}_{t,l}$ and the $'$ indicates the chosen history inputs are not necessarily consecutive ones in the original time series. To remove the irrelevant inputs, we introduce two filters: (1) a host filter, which removes the inputs from the same host; and (2) a protocol filter, which removes the inputs of the same protocol. We first apply the host filter to the hosts one by one within l; we then apply the protocol filter to the protocols within l. If the host or protocol information is unavailable, we skip Step 2.

3.3 Sampling Around History Inputs

DL-NIDS considers history inputs when making decisions; however, the latest inputs may influence the result more than the old ones. The intuition is that the influence of the old inputs should decrease; otherwise, DL-NIDS may suffer from exploding gradient and insufficient convergence [24]. For example, LSTM-based systems use forget-gate to discard the long-term dependencies [30], reducing the influences of the old inputs. Based on this intuition, we sample unevenly in the vicinity of the history inputs: (1) we first assign larger weights to the latest history inputs to demonstrate their more significant influences on the detection result; (2) we then shift the synthesized samples towards the latest history inputs by WRS. In WRS, the probability of each item (p_i) to be selected is determined by its relative weight:

$$p_i = \frac{\mathcal{D}(i)}{\sum_{j=t-1}^{t-m} \mathcal{D}(j)} \tag{6}$$

where $0 < p_i \leq 1$ is the probability for the ith input to be selected, m is the number of history inputs, $\mathcal{D}$ is a decay function to assign weights to each history input based on its arrival order. For instance, $\mathcal{D}(i)$ is the weight for ith input. Typical decay functions include exponential, Gaussian, and linear. We denote by $\mathbf{Z}_{t,m}$ the synthesized samples from $\mathbf{X'}_{t,m}$ following WRS:

$$\mathbf{Z}_{t,m} \sim \mathcal{W}(\mathbf{X'}_{t,m}, \mathbf{p}) \qquad \mathbf{p} \in (0,1)^m \tag{7}$$

where $\mathbf{p}$ is the probability vector for $\mathbf{X'}_{t,m}$ determined by a decay function $\mathcal{D}$. Note that we weigh the history inputs based on their arrival order, so features of the latest history inputs have a higher probability to be selected, while features of the same input have the same probability to be selected. Additionally, if history inputs and current input have a similar influence on the detection result, we can assign constant weight to history inputs. Consequently, WRS will be reduced to random sampling.

3.4 Capturing Feature Dependencies

DL-NIDS usually applies correlation-based methods (*e.g.*, clustering) or domain knowledge to handle feature dependencies of structured data for better detection performances. To capture the feature dependencies of the structured data for an explanation, we first divide the features into several groups based on their correlations. Then, we apply a sparse group lasso to the feature groups to achieve a sparse explanation.

Feature Groups. Unlike an image or a block of binary code, where each feature has the same type (*e.g.*, pixel or hex value), the data samples used for DL-NIDS are well-structured and have strict formats. For example, Kitsune [46] uses dozens of features from the IP header as input; among them, `TCP.srcport` is a sub-feature of `TCP`, which means if the `TCP.srcport` feature is valid, then the `TCP` feature is supposed to be valid as well. At the same time, the `UDP` is a mutex feature of `TCP`, which means if the `UDP` feature is valid, then the `TCP` feature is supposed to be invalid. Note that valid means the value of this feature is meaningful. Invalid means the value of this feature is unset. To achieve high-fidelity explanations, we address the feature dependency challenge in structured data by dividing the features of the input into several groups regarding their correlations.

$$\begin{gathered} \mathbf{x}_t^q = 1_{A_q} : \mathbf{x}_t \\ \sum_{q=1}^{M} ||\mathbf{x}_t^q||_1 = ||\mathbf{x}_t||_1 \;\; and \;\; \mathbf{x}_t^i \cdot \mathbf{x}_t^j = 0 \;\; (i \neq j) \\ 1_{A_q} : \mathbf{x}_t \to \{0,1\}, \quad j \mapsto \begin{cases} 1, j \in A_q \\ 0, j \notin A_q \end{cases} \end{gathered} \tag{8}$$

where $\mathbf{x}_t^q$ is q^{th} group of $\mathbf{x}_t$, M is the number of groups within $\mathbf{x}_t$, and $1_{A_q} : \mathbf{x}_t$ is q^{th} indicator function, which determines whether a feature in $\mathbf{x}_t$ belongs to group $\mathbf{x}_t^q$. Here each element inside the indicator function is either 0 or 1, representing absence or presence, respectively. We need M indicator functions to divide $\mathbf{x}_t$ into M groups. Especially, each feature must only be present in one group.

To determine the indicator functions, we consider three scenarios: (1) the grouping strategy used by the target DL-NIDS is available, which is usually determined by the domain knowledge. Then we adopt the same grouping strategy to xNIDS; (2) the dataset that contains relevant history samples are available, but the grouping strategy used by the target DL-NIDS is unclear. Then we calculate the correlations of features with clustering methods to create indication functions; and (3) the grouping strategy and dataset are unavailable, and xNIDS is forced to set the size of each group to one. Consequently, sparse group lasso will be reduced to lasso.

Additionally, we use the same indicator functions for $\mathbf{x}_t$ to divide each input $\mathbf{x}_i$ inside history inputs $\mathbf{X}'_{t,m}$ to M groups since every input for DL-NIDS follows the same well-defined structures. We denote the overall number of groups for $\mathbf{x}_t$ and $\mathbf{X}'_{t,m}$ as $Q = M \times (1+m)$.

Sparse Group Lasso. To achieve sparse explanations, we need to minimize $\phi(\mathbf{e})$, namely, choose the most relevant features from inputs as explanation results, while omitting those that do not significantly contribute to the detection results. Sparse group lasso is a regression method that allows predefined groups of features to be selected into or out of a model together, where all the features of a particular group are either included or excluded. More importantly, it has the desired effect of group-level and feature-wise sparsity [66]. Therefore, we use sparse group lasso to enable xNIDS to meet the sparsity requirements in explaining DL-NIDS. sparse group lasso allows us to find the important explanatory factors in predicting the corresponding detection result, where each explanatory factor may be comprised of a group of features within the inputs. At the same time, sparse group lasso helps us to achieve the sparse effects on both the group level and the feature level. We model the regression problem as follows:

$$\underset{\beta}{argmin}\left\{ ||f-g||_2^2 + \underbrace{(1-\alpha)\lambda\sqrt{p_q}\sum_{q=1}^{Q}||\beta_q||_2}_{\text{Group Sparsity}} + \underbrace{\alpha\lambda||\beta||_1}_{\text{Feature Sparsity}} \right\} \quad (9)$$

where β_q is a vector, which contains the coefficients for the features in qth group; p_q is the size of qth group; $\beta = (\beta_1 ... \beta_Q)$; and $\alpha \in [0,1]$ a convex combination of the lasso and group lasso penalties. To achieve group-level sparsity, we minimize $\sum ||\beta_q||_2$, namely exclude more groups by making $||\beta_q||_2 = 0$. To achieve feature-level sparsity, we minimize $||\beta||_1$, excluding more features by making $||\beta_i||_1 = 0$.

3.5 Model Development

We integrate the designs discussed above into a consolidated explanation model that explains the results of a DL-NIDS.

First, we use the proposed method in Section §3.2 to approximate the proper history inputs $\mathbf{X}'_{t,m}$. Then, combined with the decay function and weighted random sampling, we can tune the sampling strategy for history inputs. Finally, following the concept of linear approximation, we use linear components to approximate the local decision boundary of the DL-NIDS. Specifically, the key idea here is to utilize a local linear model to approximate the individual decision boundary around $y_t = f(\mathbf{x}_t, \mathbf{X}_{t,k})$. The approximation procedure is described as follows:

$$f(\mathbf{x}_t, \mathbf{X}_{t,k}) = f(\mathbf{x}_t, \mathbf{X}'_{t,m}) + \delta = g(\mathbf{x}_t, \mathbf{X}'_{t,m}) + \varepsilon \quad (10)$$

where $f(\cdot)$ is the non-linear DL-NIDS, $g(\cdot)$ is the local approximation method, and ε is a small deviation between the approximation and the true detection result. Guided by feature groups, we translate $g(\cdot)$ into the following equation:

$$g(\mathbf{x}_t, \mathbf{X}'_{t,m}) = \sum_{q=1}^{M} \mathbf{x}_t^q \beta_{t,q}^T + \sum_{i=t-1}^{t-m}\sum_{q=1}^{M} \mathbf{x}_i^q \beta_{i,q}^T \quad (11)$$

where the first component is for current input and the second component is for history inputs. $\mathbf{x}_i^q$ is the q^{th} group from input $\mathbf{x}_i$, and $\beta_{i,q}$ contains the corresponding coefficients for x_i^q. M is the number of feature groups within one input. m is the number of history inputs.

Finally, by taking Equ. (10) as the seed input to create n synthesized samples following our uneven sample strategy for Equ. (9), we formalize our explanation model as the following regression problem:

$$\begin{aligned} \underset{\beta}{argmin}\Bigg\{ & \frac{1}{2n}||\mathbf{y} - g(\mathbf{z}'_t, \mathbf{Z}'_{t,m})||_2^2 \\ & + (1-\alpha)\lambda \sum_{q=1}^{Q}\sqrt{p_q}||\beta_q||_2 + \alpha\lambda||\beta||_1 \Bigg\} \end{aligned} \quad (12)$$

where $\mathbf{z}'_t$ and $\mathbf{Z}'_{t,m}$ are the synthesised samples for the current input $\mathbf{x}_t$ and history inputs $\mathbf{X}'_{t,m}$, respectively. $\mathbf{y}$ is a vector, containing n detection results. β_q^T contains the coefficients for the features in q^{th} group. Q is the total number of groups. p_q is the size of q^{th} group and λ is a tuning parameter. $\alpha \in [0,1]$ a convex combination of the lasso [76] and group lasso [83].

To solve the objective function in Equ. (12), sparse group lasso needs to repeat two loops: a group-level outer loop and a feature-level inner loop. The group-level outer loop recurrently checks whether the group's coefficient is a zero vector. If a group's coefficients are a nonzero vector, then the feature-level inner-loop revises each parameter within the vector. The sparse group lasso repeats the two loops until the parameters converge. A detailed solution to the problem is shown in Appendix A.

4 Generating Defense Rules

After extracting the important features that the DL-NIDS uses to make a specific decision, xNIDS generates defense rules based on these features. A detailed example of how xNIDS generates defense rules is shown in Fig. 2.

xNIDS addresses **Ch3** by (1) defining the defense rule scope to confine where a rule should apply; (2) analyzing explanations to determine the rule scope; and (3) considering

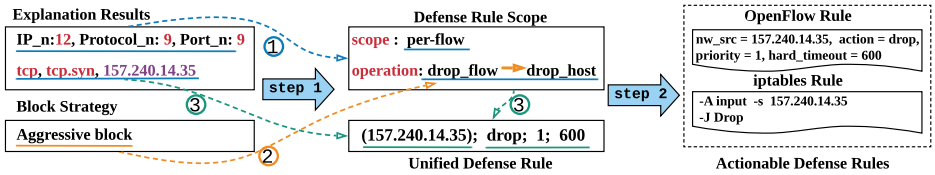


Figure 2: An example illustrates how xNIDS generates defense rules. The key idea is to ① determine the *defense rule scope* by analyzing *explanation results*, ② modify the *operation* concerning *block strategy*, and ③ generate unified defense rules by creating a corresponding entity with values from the *important features*. Finally, xNIDS automatically translates the *unified defense rules* into *actionable defense rules* (*e.g.*, *OpenFlow rules and iptables rules*).

the security constraints to improve the adaptability of the defense rules. xNIDS also addresses **Ch4** by defining a unified defense rule representation to abstract common entities and operations of rules in different defense tools.

4.1 Defense Rule Scope

To balance precision and generalization, we introduce the defense rule scope to confine where a rule should apply to. The precision metric requires that defense rules only affect the malicious traffic while leaving the benign traffic intact. The generalization metric requires that the defense rules should alert all the malicious traffic involved in an attack. To this end, we first introduce the definition of defense rule scope. We then introduce how to determine defense rule scopes by analyzing explanations.

Defining Rule Scopes. The scope of a defense rule can be defined at three levels: (1) *Per-flow Scope.* The per-flow defense rule can only influence the network packets from the specific flow. For example, a defense rule used to terminate a `TCP` connection is considered a per-flow defense rule. (2) *Per-host Scope.* The per-host defense rule can block multiple flows from the same host. For example, a defense rule used to block a bot is considered a per-host defense rule. (3) *Multi-hosts Scope.* The multi-hosts defense rule can block multiple flows from multiple hosts. For example, a defense rule used to block `SYN` requests from a set of hosts (*e.g.*, botnet) is considered a multi-hosts defense rule.

Analyzing Explanation Results. We determine the proper scope of a defense rule by analyzing the corresponding explanations. As shown in Table 2, we define the statistical information as `S`, which contains five fields: `IP_pool`, `IP_n`, `MAC_n`, `Port_n`, and `Protocol_n`. Statistical information is critical for generating practical defense rules. If it is unavailable, xNIDS doesn't generate defense rules.

Table 2: Details of the Statistical Information

Field	Description
`IP_pool`	The IPs involved in inputs
`IP_n`	The largest number of packets from the same IP
`MAC_n`	The largest number of packets from the same MAC
`Port_n`	The largest number of packets from the same Port
`Protocol_n`	The largest number of packets from the same Protocol

We decide the per-host and the multi-host scopes by checking which field in the statistical information (`S`) has the greatest value. For example, if `Protocol_n` or `Port_n` has the greatest value, it means the anomaly traffic belongs to the same protocol but not from a single host. In other words, multiple hosts are likely involved in this attack. Therefore, the defense rule should have a multi-hosts scope. Otherwise, if the anomaly traffic comes from the same host, the corresponding defense rule should have a per-host or per-flow scope. In such a situation, if the important features contain multiple protocols or ports, which means the host uses multiple protocols to launch the attack, the defense rule has a per-host scope, or else the defense rule has a per-flow scope.

4.2 Security Constraint

The network environment often varies from site to site, and network operators tend to have vague security requirements for their network environments. As a result, a defense rule that works fine on one site may be totally unusable on another site. In response, it is important to adjust the defense rules according to security constraints dynamically. To make the defense rule adaptable to different network environments, we introduce security constraints that are configurable for network operators. We design two mechanisms, whitelist and block strategy, to ensure network usability and maintain appropriate block rates, respectively.

Whitelist. Network traffic from whitelisted critical services will not be affected by any other defense rules.

Block Strategy. This mechanism is the interactive channel between network operators and xNIDS. Network operators can choose a proper block strategy for the unified defense rule generation procedure based on their requirements and knowledge. We further define three primary options: (1) *passive block.* xNIDS only generates defense rules that block malicious flows. The goal of this strategy is to minimize the influence of defense rules; (2) *assertive block.* Network operators tend to trust the corresponding DL-NIDS and the rules generated by xNIDS; and (3) *aggressive block.* xNIDS tends to block malicious activities by blocking the hosts directly. The goal of this strategy is to quickly eliminate malicious hosts from the network environment.

4.3 Unified Defense Rule

To support various defense tools that use different rule syntax, we introduce a unified defense rule representation, which serves as the bridge between the explanations and actionable defense rules.

Each unified defense rule comprises four primary components: ⟨`entity`, `action`, `priority`, `timeout`⟩, which describes an action to perform on a subset of the network traffic based on a set of network attributes [3]. An `entity` is a target (*e.g.*, a connection), where the defense action is applied. An `action` is the specific defense operation to perform on all of the entity's network traffic. The `timeout` field is dedicated to controlling the valid time of a defense rule. If a network traffic instance matches multiple unified defense rules, only the highest-priority rule is applied. In this scenario, the `priority` component, which is by default decided by an incremental order, can explicitly specify the operation order to resolve the potential conflicts from a set of unified defense rules. The syntax of unified rule representation is shown in Appendix B.

Entity. The unified defense rule can define a wide variety of entities: network flows, connections, IP addresses, network prefixes, and layer-2 MAC addresses. The unified rule specifies those entities through the 8-tuples, with the support for wildcards as tuple elements.

Action. The unified defense rule uses a list of primitive actions to specify how to operate the corresponding network traffic. The intuitive actions are `Drop` and `Allow`, which are used to enforce explicit access control upon packets from matched entities. `Modify` aims to change attributes (*e.g.*, dest IP) of packets from a specific entity. `Whitelist` is a preventive method to leave an entity unaffected by any other defense rules. This serves as a preventive measure to ensure the usability of the targeted network.

Generating Unified Defense Rule. To generate defense rules, we need to create the `entity` by filling the corresponding fields with the matched values from the *important features* with respect to the defense rule scope and security constraints.

To relate the defense rule scope with the defense action, we define two basic operations: `drop_flow` and `drop_host`. For per-flow rules, we block malicious flows with the `drop_flow` operation. For per-host rules, we block malicious hosts with the `drop_host` operation. For multi-host rules, we block malicious flows with the `drop_flow` operation recursively.

To corporate with different block strategies, we modify the operations, respectively. For the passive block, we modify the `drop_host` operation to the `drop_flow` operation. For the aggressive block, we modify the `drop_flow` operation to the `drop_host` operation. While for the assertive block, we keep the operations unchanged.

Finally, we can create an `entity` based on the `operation`. For example, when generating a multi-host rule with `drop_flow` operation, we need to create the flow `entity` recursively. To prevent rule conflicts, we set the `priority` as an incremental number. The `timeout` parameter is configurable concerning different deploy environments.

5 Implementation and Experiment Setup

xNIDS Implementation. There are three groups of hyperparameters that are configurable in xNIDS. For approximating the history inputs, in step 1, we set the search termination condition $\delta = 1e-2$, and the update times to 10. For sampling around history inputs, we set the decay function to Gaussian. For capturing feature dependencies, we follow the group strategies of the target DL-NIDS and set the mixing parameter $\alpha = 0.05$ and the tuning parameter $\lambda = 0.2$ for sparse group lasso. We implement xNIDS with the Python package asgl [45] and use grid search to tune the parameters to improve the performance and test the sensitivity of the parameters. We conduct our experiments on an HPC cluster provided by our university. The hardware we employ has four Intel E5-2630 v3 8-core CPUs at 2.4 GHz and two 10 GB network interface cards.

Target DL-NIDS. We use four state-of-the-art DL-NIDS as target systems: Kitsune [46] with its published dataset; ODDS [33] with the CIC-DoS 2017 dataset [20, 35]; and both RNN-IDS [82] and AE-IDS [64] with the NSL-KDD dataset [49]. For the strategy to divide training and testing data, we follow the original setting in those papers.

Comparison Baselines. We compare xNIDS with five state-of-the-art explanation methods: LIME, SHAP, LEMNA, IG, and LRP. The details are shown in Appendix C.2.

Actionable Rule Generation. xNIDS can translate unified defense rules into concrete defense rules that are then deployed to different defense tools through the following steps. First, an actionable defense rule template is created based on the specific syntax of the object defense tool. Then, the related fields in the actionable defense rule template are filled with the appropriate values from the unified defense rules. Finally, the generated defense rules can be deployed automatically or manually to corresponding defense tools.

6 Evaluation

We first evaluate the explanation component of xNIDS in terms of fidelity, sparsity, completeness, and stability considering the general criteria used by [27, 55] and security-related criteria introduced in [79]. We then evaluate the defense rule generation of xNIDS regarding practicability, accuracy, and efficiency. We also showcase how xNIDS can help understand DL-NIDS behaviors, troubleshoot detection errors, and facilitate active intrusion response.

6.1 Evaluation of Explanation

In the following experiments, we first scale the importance score vector β to the range $[0, 1]$. We then rank the features based on the importance scores. A larger importance score β_i demonstrates a high relevance of feature x_i.

6.1.1 Fidelity

This experiment evaluates how faithful the explanation method captures the important features that contribute to a specific detection result. We adopt the Descriptive Accuracy (DA), which is defined as $DA_k(\mathbf{x}, f) = f(\mathbf{x}|modify(k))$ [79], to evaluate the fidelity of explanation methods regarding DL-NIDS. We consider two scenarios for $modify(k)$: (1) for the anomaly samples, $modify(k)$ nullifies k important features to zero as described in [79]; and (2) for the benign samples, we find it insufficient to evaluate the fidelity of explanation methods against benign samples using the same operation. This finding is actually consistent with a claim in [79] "setting benign features to zero usually does not impact the prediction". To effectively evaluate the descriptive accuracy for benign samples, we follow the fidelity test approach proposed in [27] and slightly change the metric from [79] by replacing the top-k features of the benign sample with corresponding features from the nearest anomaly sample. We find if explanation methods can accurately select the benign features of benign samples, replacing those features with the corresponding anomalous features likely leads to misclassification. We then feed those samples to the DL-NIDS and calculate the Average Descriptive Accuracy (ADA) on the whole dataset. We expect a significant decrease in ADA from the fidelity test if the selected important features are relevant to the detection result.

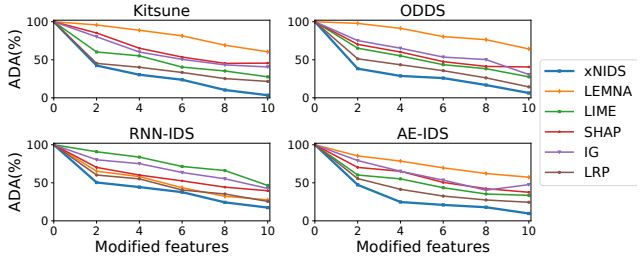


Figure 3: A steep decline in ADA means a good explanation.

Table 3: Area under the ADA curves from Fig 3.

System	Kitsune	ODDS	RNN-IDS	AE-IDS
LIME	0.509	0.531	0.770	0.521
SHAP	0.643	0.578	0.593	0.593
LEMNA	0.830	0.856	0.525	0.748
IG	0.608	0.618	0.690	0.623
LRP	0.409	0.427	0.507	0.438
xNIDS	**0.316**	**0.325**	**0.430**	**0.331**

As shown in Fig. 3, xNIDS has the steepest ADA decrease in the fidelity test compared with baseline methods. This considerable decrease confirms that the features selected by xNIDS are highly relevant to the detection results. We then use Table 3 to summarize the area under curve for the ADA curves from Figure 3. We observe that xNIDS outperforms the baseline methods regarding the AUC. Intuitively, when selecting relevant features, xNIDS captures the feature dependencies via sparse group lasso while baseline methods ignore them. In summary, this experiment confirms that xNIDS can generate more faithful explanation results regarding selecting important features.

Table 4: Comparison of setting benign features to zero and replacing them with anomalous features.

System	Dataset	Setting to zero	Replacement
Kitsune	Kitsune	0.084	0.239
ODDS	CIC-DoS2017	0.028	0.218
RNN-IDS	NSL-KDD	0.071	0.279
AE-IDS	NSL-KDD	0.059	0.281

To justify our adaptation of the DA for evaluating the fidelity of explanation methods against benign samples, we compare the anomalous rate of the two operations: 1) setting benign features to zero; and 2) replacing benign features with anomalous features. As shown in Table 4, we observe that the highest anomalous rate by setting benign features to zero is around 0.084, while the lowest anomalous rate for replacing the benign features with anomalous features is 0.218. On average, by replacing benign features with anomalous ones, we can significantly improve the anomalous rate by a factor of at least four on all datasets.

6.1.2 Sparsity

To further evaluate the explanation results, we measure the sparsity of the explanation. The desired explanation method should select a limited number of features as explanation results since that would be more convenient for network operators to conduct intrusion analysis and defense. To evaluate sparsity, we follow the Mass Around Zero (MAZ) criteria introduced in [79]. Since the important scores $\beta = (\beta_i, ...\beta_{\bar{d}})^T$ are scaled to the range $[0,1]$, we first fit β to a half-normalized histogram h, we then calculate the MAZ by $MAZ(\beta) = \int_0^1 h(x)dx$ for $\beta \in [0,1]$

We present the MAZ curves in Fig. 4. We also calculate the Area Under Curve (AUC) and show the information in Table 5. We expect a steep slope near zero from the MAZ curve and a large AUC, if the explanation method can assign zeros to most of the features and achieve sparse explanations.

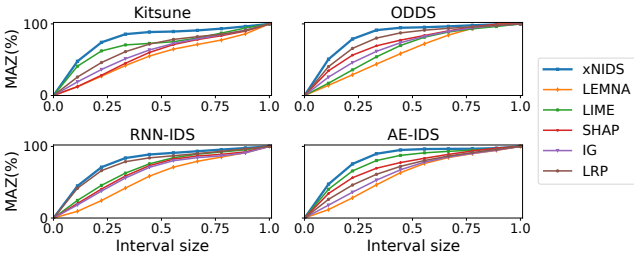


Figure 4: Sparsity of explanation methods. A MAZ distribution peaking around 0 represents a better explanation.

Table 5: Area under the MAZ curves from Fig 4.

System	Kitsune	ODDS	RNN-IDS	AE-IDS
LIME	0.650	0.762	0.745	0.667
SHAP	0.685	0.647	0.680	0.760
LEMNA	0.542	0.599	0.569	0.604
IG	0.577	0.713	0.637	0.632
LRP	0.605	0.680	0.655	0.708
xNIDS	**0.774**	**0.814**	**0.775**	**0.806**

As shown in Fig 4, we observe that xNIDS has the steepest slopes, which means xNIDS assigns the majority of important scores close to zero. Similarly, we observe that xNIDS has the largest AUC in Table 5, which confirms that xNIDS outperforms the baseline methods regarding the sparsity criteria. Especially, LEMNA has the poorest performance for the sparsity test. The reason behind that is LEMNA assumes that adjacent features have similar contributions to the detection results, which is inapplicable to the DL-NIDS. In contrast, xNIDS achieves sparse effects on both group-level and feature-level by adopting sparse group lasso techniques and is suitable for explaining DL-NIDS.

6.1.3 Completeness

An explanation is complete if it can create proper results for all possible input samples. We first follow the definition from Warnecke et al. [79] to evaluate the completeness of explanation regarding single input $\mathbf{x}_t$. We then extend the completeness criteria to evaluate the completeness of explanation regarding the history inputs $(\mathbf{x}_{t-1},...,\mathbf{x}_{t-k})$. Warnecke et al. [79] argue that it is essential to synthesize a fraction (5%) of samples detected as the opposite class to achieve completeness. We denote benign samples as $\mathbf{x}^b = (x_1^b,...,x_d^b)^T$ and malicious samples as $\mathbf{x}^m = (x_1^m,...,x_d^m)^T$. They also argue that it is easy to synthesize benign samples around malicious ones. However, it is challenging to synthesize malicious samples around benign ones by simply setting benign features to zero ($x_i^b = 0$), which usually cannot flip the detection results. To address this issue, when sampling around benign, instead of always setting features to zero, we replace a fraction of them with malicious features ($x_i^b = x_i^m$). Our experiments show that by replacing benign features with those malicious features, we can efficiently satisfy the 5% requirement.

When evaluating the completeness considering history inputs, we observe that, in some cases, the current input $\mathbf{x}_t$ contains all the information needed for the explanation. For example, packets look similar in a `UDP` stream. Thus, it is possible to get non-degenerated explanations from the current input. However, we are likely to get degenerated explanations for other cases if we ignore the history inputs. For example, there are various protocols involved in the whole process of OS scan, but the current input only contains one protocol. Consequently, we may be unable to mark other protocols as relevant. We also observe that creating non-degenerated explanations for anomaly samples is more critical. From the network operator's perspective, we are only required to prevent the anomalies while leaving the benign intact. Intuitively, we measure the completeness regarding history inputs by calculating the percentage of anomaly samples, which contain enough information for non-degenerated explanations.

Table 6: Samples remaining regarding the history inputs.

	Kitsune	**ODDS**	**RNN-IDS** and **AE-IDS**
Anomaly	0.739	0.638	0.696

We can observe that at least 26% of the anomaly samples don't contain enough information for non-degenerated explanations. Since the baseline methods ignore history inputs, they are likely to generate degenerated explanation results for attacks that are involved in a various number of inputs. In contrast, xNIDS can dynamically approximate the history inputs and outperform existing methods regarding completeness.

6.1.4 Stability

We further evaluate the stability of the explanation results following the definition in [79]. From a network operator's perspective, the explanation results should be stable. Hence, the desired explanation method should generate similar results for the same samples among multiple tests. Since IG and LRP are deterministic, the explanation results from IG and LRP for the same samples don't vary among multiple tests. Consequently, we focus on the stability of perturbation-based methods LIME, SHAP, LEMNA, and xNIDS.

To examine the stability of explanations, we calculate the intersection size of the top K features of the explanation results for the same input regarding different tests. We denote the explanation result of the first test as $\beta^1 = (\beta_i^1,...\beta_{\bar{d}}^1)^T$, and for n_{th} test as $\beta^{\mathbf{n}} = (\beta_i^n,...\beta_{\bar{d}}^n)^T$. $top(\beta^n)$ denotes the top K features of β^n. Then the stability is measured as $stability = \frac{1}{K}||\{top(\beta^1) \cap top(\beta^2)... \cap top(\beta^n)\}||_1$. A stable explanation method should have a *stability* score close to 1 [79], which means $||top(\beta^i) \cap top(\beta^j)||_1$ is close to K.

Table 7: Average *stability* scores of explanation methods.

System	**Kitsune**	**ODDS**	**RNN-IDS**	**AE-IDS**
LIME	0.424	0.640	0.442	0.470
SHAP	0.563	0.473	0.482	0.543
LEMNA	0.473	0.520	0.357	0.371
xNIDS	0.850	0.830	0.914	0.807

We can observe that the lowest score for xNIDS is 0.807, which is still higher than the highest score (0.640) of LEMNA, SHAP, and LIME. Intuitively, xNIDS can generate more stable explanations by: (1) utilizing the feature groups to synthesize samples more reliably; and (2) concentrating the samples by assigning higher probabilities to the latest inputs.

6.1.5 Sensitivity and Tuning of Hyper-Parameters

We evaluate how the performance of xNIDS would change if the parameters are set differently. We test a large set of parameter configurations and summarize key conclusions here.

For parameter tuning, we use grid search to find the ideal set. We set ranges for each parameter following the descriptions in [21] [31] [66]: $\delta = [1e-4, 1e-1]$, $U = [1, 100]$, $\alpha = [1e-2, 1]$, and $\lambda = [1e-2, 1]$. We tune these parameters on the training dataset and fix them on the testing dataset. For approximating the history inputs, in step 1, if δ is too small and the update times (U) are too large, xNIDS will become insufficient since the searching space will increase exponentially. By setting the update times to 10, we can efficiently approximate the history inputs (up to 1024) in the

experiments. We apply the host and protocol filters to Kitsune and ODDS while skipping step 2 for RNN-IDS and AE-IDS since the host or protocol information is unavailable in the NSL-KDD dataset. For sampling around history inputs, we find Gaussian works well for all the experiments. For capturing feature dependencies, we follow the same group strategies of the target DL-NIDS, which improve the performance of xNIDS. For sparse group lasso, the larger we set λ, the sparser the explanation will be. We find that $\lambda = 0.2$ achieves the best overall performance in our experiments. Additionally, in case of an increasing number of features and the unavailableness of grouping information, the sparse group lasso component of xNIDS will be reduced to lasso.

6.1.6 Summary of Explanation Evaluation

Table 8: Overall Comparison of explanation methods. ● means strong, ◐ means medium, while ○ means weak.

Criteria	LIME	SHAP	LEMNA	IG	LRP	xNIDS
Fidelity	◐	◐	○	◐	◐	●
Sparsity	◐	◐	○	○	○	●
Completeness	○	○	○	◐	◐	●
Stability	○	○	○	●	●	◐
Rule Generation	/	/	/	/	/	●

We summarize the performance of xNIDS and the baseline methods over the target DL-NIDS and present an overview of the evaluation of explanation in Table 8. We rank each method following three scores for each evaluation criterion. We observe that DL-NIDS achieves a higher ranking than LIME, SHAP, and LEMNA in all the criteria. For IG and LRP, xNIDS has a higher ranking regarding fidelity, sparsity, and completeness and has comparable performance regarding stability. Overall, xNIDS has the best performance since it considers the history inputs and feature dependencies.

6.2 Evaluation of Rule Generation

We evaluate the practicability, accuracy, and efficiency of the rule generation mechanism in xNIDS regarding supported defense tools, covered attacks, and latency. We use both the Kitsune dataset and CIC-DoS2017 dataset in this experiment, since the NSL-KDD dataset doesn't have enough traffic information for rule generation.

6.2.1 Supported Defense Tools

As discussed in §4.3, unlike existing work that focuses on specific defense tools, the unified defense rule can support various network defense tools. This experiment presents the different defense tools whose native rules are accommodated by the unified defense rules.

As shown in Table 9, the unified defense rule is totally transferable to OpenFlow, which means we can translate all the generated unified defense rules into OpenFlow rules. Meanwhile, the unified defense rule is partially transferable to iptables, Pfsense, and Squid. We cannot set customized priority or timeout when using iptables, Pfsense, and Squid.

Table 9: Summary of four defense tools that are supported by unified defense rule. ● means totally transferable, while ◐ means partially transferable.

Defense Tool	Entity	Action	Priority	Timeout
OpenFlow [50]	●	●	●	●
iptables [32]	●	●	◐	◐
Pfsense [53]	◐	●	◐	◐
Squid [72]	◐	●	◐	◐

6.2.2 Covered Attacks

To demonstrate that the unified defense rule can accurately defend against a wide variety of network attacks, We show three representative examples regarding the defense rule scopes.

Per-flow. We use SYN DoS as an example to demonstrate how a unified defense rule can block malicious flows. A simple approach is to block the unidirectional flow from the malicious host. To minimize the effect, the unified defense rule specifies the `TCP.flag` filed as `SYN`.

```
R1:<entity(src_ip = 157.240.1.9, dst_ip =
157.240.1.3, TCP,  TCP_flags=SYN),
actions = drop,  priority = 1,  timeout = 6000>
```

Per-host. We use OS scan as an example to demonstrate how the unified defense rule can block a malicious host. The unified defense rule will block the infected device by IP or MAC address as long as possible.

```
R2:<entity(src_ip = 157.240.1.12,
src_mac = dc:a9:04:bc:7e:42 )
action = drop, priority = 3, timeout = MAX>
```

Multiple-hosts. We use Simple Service Discovery Protocol (SSDP) amplification as an example to demonstrate how the unified defense rule can block malicious flows from multiple hosts. The unified defense rule will block all the network traffic with the destination port as 1900, while allowing the benign hosts to use this service. By default, the multiple-hosts scope rule blocks the SSDP flows from the malicious hosts recursively. However, an alternative way is to shut down the SSDP port, while allowing benign hosts to use this service.

```
R3:<entity(src_ip=*, dst_port = 1900 )
action = drop, priority = 4, timeout = MAX>
R4:<entity(src_ip=157.240.1.13, dst_port = 1900 )
action = allow, priority = 5, timeout = MAX>
```

6.2.3 Latency

This experiment evaluates the latency of explaining detection results and actionable rule generation, respectively. When generating defense rules, one of the concerns is how fast it takes effect. Therefore, we should reduce the latency of the explanation and rules generation as much as possible.

Explanation Latency. In Fig. 5, over 90% of the explanation latency is under 600*ms*. This latency is determined by the speed of explanation methods and the target DL-NIDS.

Rule Generation Latency. In Fig. 5, the rule generation time varies for different defense tools. The largest latency

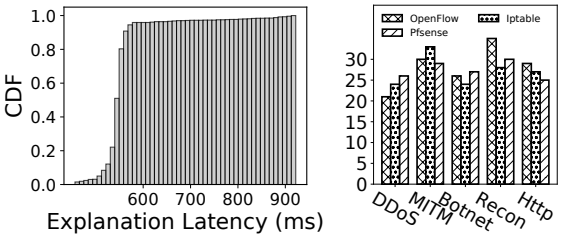


Figure 5: Left figure shows the latency of explanation. The right figure presents the rule generation time (ms) regarding defense tools. Recon, MITM, and HTTP represent Reconnaissance, Man in the Middle, and HTTP Flood, respectively.

(35*ms*) is the OpenFlow rule of the Reconnaissance attack. The average latency is about 25*ms*.

6.2.4 Comparison with Existing Work

To better demonstrate the benefits of the rule generation component of XNIDS, we provide a comparison of FIRMA [54] and XNIDS in Table 10.

Table 10: Comparison of FIRMA and XNIDS.

Criteria	FIRMA	XNIDS
Supported Tools	Two [69, 74]	Four Defense Tools
Covered Attacks	One scope	Three scopes
Latency	NA	Varies

As shown in Table 10, XNIDS can support more defense tools and generate defense rules with different scopes. In contrast, FIRMA only supports two defense tools and generates per-host rules. Moreover, FIRMA requires offline analysis for signature generations, while XNIDS can generate rules directly from the explanation of detection results of DL-NIDS.

6.3 Troubleshooting and Active Responses

This experiment demonstrates how XNIDS can help network operators understand DL-NIDS behaviors, troubleshoot detection errors, and enable active responses with case studies.

6.3.1 Understanding DL-NIDS Behaviors

We present two case studies to show how XNIDS helps network operators understand the detection results of DL-NIDS.

Reasons for Anomalies. We use OS to scan and HTTP flood as examples to demonstrate how DL-NIDS detects anomalies.

```
Explanations: xNIDS identifies LLMNR, NBNS, and SSDP
as the important features.
```

Network operators' understanding: this result matches the well-known description of OS scan attacks, namely the attacker exploits various protocols to scan the network to look for possible vulnerabilities.

```
Explanations: xNIDS identifies HTTP Referer as the most
important feature.
```

Network operators' understanding: this result matches the procedure of the HTTP flood attack in this experiment. The attacker uses Botnet to send tons of HTTP GET requests via the same hyperlink (Referer) to overwhelm the victim server.

Reasons for FN and FP. By analyzing the explanation results, we identify two reasons for false negatives: (1) *Inappropriate Parameter* e.g., threshold; and (2) *Inadequate Design* e.g., focusing on header information. We also identify three types of false positives: (1) *Infected Benign*, benign traffic from infected hosts are misdetected as anomalies; (2) *Disturbed Benign*, disturbed traffic from the benign hosts are misdetected as anomalies; and (3) *Unseen Benign*, unseen benign traffic is misdetected as anomalies, which is the most common error committed by DL-NIDS.

6.3.2 Troubleshooting Detection Errors

By analyzing the detection logic behind the DL-NIDS, we now present approaches to patching the DL-NIDS with respect to reducing detection errors.

To patch *infected benign*, we utilize the defense rule generation procedure to minimize the cost of detection errors. For example, we use the passive block strategy to generate per-flow scope rules to block the malicious TCP.SYN traffic precisely, while leaving the benign UDP flows unchanged. The advantage of this approach is that we don't need to retrain the DL-NIDS while reducing the errors at run-time. To patch *disturbed benign* error, we need to disturb the traffic intentionally to make the DL-NIDS more robust. For *unseen Benign*, we can reduce the FPR after retraining the DL-NIDS with the augmented data. As shown in Table 11, for *infected benign* error, XNIDS can reduce detection errors by 100%, to be specific, XNIDS only blocks the SYN DoS flows while it leaves the benign UDP flows unaffected. For *disturbed benign* and *unseen benign* errors, XNIDS can reduce detection errors by 65.36% and 96.79%, respectively. We observe that *disturbed benign* is the most challenging FP to reduce, which is consistent with the intuitive knowledge, considering that it is hard to mimic an unstable network environment.

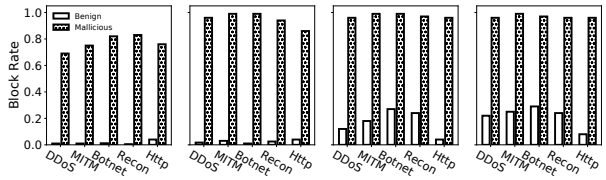
Table 11: Detection errors before and after troubleshooting. False positive denotes the number of benign samples that are detected as anomalies. Blocked denotes the number of benign samples that are dropped by defense rules.

Error Type	Before		After		Reducing Rate
	FP	Blocked	FP	Blocked	
Infected Benign	137583	136425	137583	0	**100%**
Disturbed Benign	45744	44371	16012	15369	**65.36%**
Unseen Benign	35676	35562	1192	1141	**96.79%**

6.3.3 Active Responses

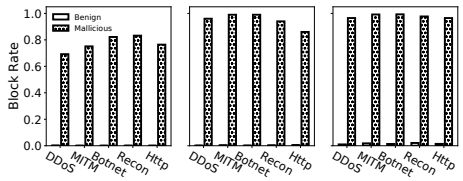
We showcase the block rates of XNIDS in three scenarios and compare it with the baseline method FIRMA.

In Fig. 6, before troubleshooting, the aggressive block (Fig. 6 (c)) has the highest block rate (avg. 97.34%) for malicious traffic, while many benign traffic also gets blocked.



(a) Passive (b) Assertive (c) Aggressive (d) FIRMA

Figure 6: Traffic block rates for XNIDS regarding different scenarios and FIRMA before troubleshooting.



(a) Passive (b) Assertive (c) Aggressive

Figure 7: Traffic block rates for XNIDS after troubleshooting.

Since the infected hosts may send benign and malicious traffic at the same time. Thus, if we block those hosts, we will block the benign traffic. Meanwhile, the passive block (Fig. 6 (a)) has the lowest block rate for benign (avg. 0.32%) and malicious traffic (avg. 74.28%) since we only block the malicious flows. If the infected hosts launch attacks with other protocols, they may bypass this defense rule. As a trade-off, the assertive block (Fig. 6 (b)) has a high block rate (avg. 95.72%) for malicious traffic and a high pass rate (avg. 98.54%) for benign traffic. We measure the block rate within an observation window when the defense rules are active. When defense rules expire, the block rate will be zero.

As shown in Fig. 7, after troubleshooting, the block rate for benign traffic is significantly reduced. Specifically, the benign traffic block rate for the passive scenario (Fig. 7 (a)) and assertive scenario (Fig. 7 (b)) is less than 0.04%. For the aggressive scenario (Fig. 7 (c)), the highest block rate for benign traffic is 0.2%, which is still relatively high. Thus, we would suggest not to use the aggressive strategy unless there are intense network attacks.

When comparing with XNIDS, we observe that FIRMA has a similar performance with the aggressive block before troubleshooting, which is the worst case of XNIDS, as shown in Fig. 6 (d). This result matches the intuition that we may increase the false positive rate by simply blocking the hosts, which may inappropriately block the benign traffic or, worse, block benign hosts. This experiment also proves that it is costly and dangerous to facilitate response directly based on the detection results. In contrast, XNIDS can generate more accurate defense rules by considering the rule scopes and security constraints. To further improve the effectiveness of active intrusion responses, we use the *whitelist* mechanism to avoid altering the critical services. We can explain the benign samples to generate *allow* rules to avoid blocking the benign.

7 Discussion

Robustness. Recently, existing works [12, 59] demonstrate that deep learning models and their coupled explanation models are vulnerable to the adversarial attack [2, 9, 39, 86], which is defined as $\{\exists\ \hat{\mathbf{x}} : ||\hat{\mathbf{x}} - \mathbf{x}|| \leq \sigma,\ s.t.\ f(\hat{\mathbf{x}}) f(\mathbf{x}) \leq 0\}$.

We evaluate the robustness of XNIDS based on existing literature. We create three types of practical traffic-based adversarial examples [29], 200 samples for each type and a total of 600, to test whether XNIDS can resist adversarial attacks. We assume that the attacker has limited knowledge about the targeted system and can only estimate and design a surrogate model based on domain knowledge. Our preliminary experiment shows that the evasion rates [2] for those three types of traffic-based adversarial samples are 33%, 49%, and 53%, respectively. Thus, XNIDS is relatively robust to those adversarial attacks by considering the history inputs.

To improve the robustness of XNIDS, we briefly discuss potential solutions based on existing literature since little is known about how to protect explanation methods against adversarial attacks. Zhang et al. [86] argue that to mitigate those adversarial attacks, we need to improve the robustness of both target deep learning models and their coupled explanation models. We would like to (1) improve the robustness of DL-NIDS by applying Generative Adversarial Networks (GAN) [26] to create a large dataset [59] for adversarial training [57, 87] considering the trade-off [84] between accuracy and robustness, and (2) improve the robustness of XNIDS with an ensemble of interpreters [86].

Adaptability. XNIDS addresses the adaptability problem by merging the security constraints configured by the network administrators into the defense rule generation process. However, fully autonomous systems that can self-adjust to dynamic environments are more desirable. To further address such a problem, we plan to adopt the concept of reinforcement learning (RL) [48, 75] in XNIDS to make it adaptive to different network environments.

Global Explanation. XNIDS and state-of-the-art explanation methods [18, 27, 55] can locally explain the detection results of DL-NIDS by generating an importance score vector, which is still relatively low level. At the same time, there is an additional demand for human-interpretable explanations that have more meaningful high-level features. We plan to investigate the concept of *global* model interpretability on a modular level [15, 38]. With the global explanation, we can investigate the high-level correlations of features and check whether the DL-NIDS are misled by the learned artifacts [4].

8 Other Related Work

Since we have discussed most of the related works in Section §2, we briefly discuss other related works here.

[2] The evasion rate represents how much the crafted attacks can bypass the target DL-NIDS [29].

Explainable Machine Learning. CADE [81] introduces a distance-based explanation method for security applications. Unfortunately, while it works well for malware detection, it suffers from an extremely low fidelity (1.41%) when applying to DL-NIDS. DeepAID [28] is a whitebox explanation method based on back-propagation, which focuses on neural network components of DL-NIDS and ignores both the feature extractor (FE) and the feature mapper (FM) of Kitsune. Moreover, by assuming each feature is independent, DeepAID is insufficient to capture the feature dependencies of structured data when explaining DL-NIDS.

Active Intrusion Responses. Researchers attempt to use signature-based or specification-based NIDS [52, 69] to enable active intrusion responses. For example, Amann et al. [3] introduced a framework, which enables active intrusion responses on top of Bro [52], and assesses its functionality through OpenFlow [44]. Xing et al. [80] designed a prototype to analyze network traffic with Snort [69] and performed prevention by utilizing OpenFlow. Zhang et al. [85] proposed a system to mitigate DDoS attacks via programmable switches. To the best of our knowledge, xNIDS is the first work towards active intrusion responses by explaining DL-NIDS.

9 Conclusion

In this paper, we present xNIDS, a novel framework that explains the detection results of DL-NIDS and generates actionable defense rules. xNIDS explains the detection results of DL-NIDS by considering history inputs and feature dependencies. Based on the explanation results, xNIDS can also troubleshoot the detection errors and generate actionable defense rules regarding defense rule scope and security constraints. The evaluation results show that xNIDS can generate high-fidelity, sparse, complete, and stable explanation results and actionable defense rules. Additionally, we showcase how xNIDS can help understand DL-NIDS behaviors and troubleshoot detection errors.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation (NSF) under Grant No. 2129164, 2114982, 2228617, 2120369, 2226339, and 2037798, and an Amazon Research Award for the proposal "Explaining Learning-based Intrusion Detection Systems for Active Intrusion Responses". Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF and Amazon.

References

[1] M. Alber, S. Lapuschkin, P. Seegerer, M. Hägele, K. T. Schütt, G. Montavon, W. Samek, K.-R. Müller, S. Dähne, and P.-J. Kindermans, "innvestigate neural networks!" in *Journal of Machine Learning Research*, 2019.

[2] E. Alhajjar, P. Maxwell, and N. Bastian, "Adversarial machine learning in network intrusion detection systems," in *Expert Systems with Applications*. Elsevier, 2021.

[3] J. Amann and R. Sommer, "Providing dynamic control to passive network security monitoring," in *Recent Advances in Intrusion Detection*. Springer, 2015.

[4] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and donts of machine learning in computer security," in *USENIX Security Symposium*, 2022.

[5] L. Arras, G. Montavon, K.-R. Müller, and W. Samek, "Explaining Recurrent Neural Network Predictions in Sentiment Analysis," in *the EMNLP Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Association for Computational Linguistics, 2017.

[6] S. Axelsson, "The base-rate fallacy and its implications for the difficulty of intrusion detection," in *ACM CCS*, 1999.

[7] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," in *PloS one*. Public Library of Science, 2015.

[8] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in *IEEE CVPR*, 2017.

[9] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in *ICML*, 2020.

[10] P. Dabkowski and Y. Gal, "Real time image saliency for black box classifiers," in *NeurIPS*, 2017.

[11] A. Datta, S. Sen, and Y. Zick, "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems," in *IEEE S&P*, 2016.

[12] G. S. Dhillon, K. Azizzadenesheli, Z. C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, and A. Anandkumar, "Stochastic activation pruning for robust adversarial defense," in *ICLR*, 2018.

[13] S. Diamond and S. Boyd, "Cvxpy: A python-embedded modeling language for convex optimization," in *The Journal of Machine Learning Research*. JMLR. org, 2016.

[14] B. Dong and X. Wang, "Comparison deep learning method to traditional methods using for network intrusion detection," in *Proc. of IEEE ICCSN*, 2016.

[15] M. Du, N. Liu, and X. Hu, "Techniques for interpretable machine learning," in *Communications of the ACM*. ACM New York, NY, USA, 2019.

[16] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," in *Information Processing Letters*. Elsevier, 2006.

[17] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," in *The Annals of statistics*. Institute of Mathematical Statistics, 2004.

[18] L. et al., "Shap," 2020. [Online]. Available: https://github.com/slundberg/shap

[19] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *IEEE CVPR*, 2017.

[20] C. I. for Cybersecurity, "Cic dos dataset (2017)." [Online]. Available: https://www.unb.ca/cic/datasets/dos-dataset.html

[21] J. Friedman, T. Hastie, and R. Tibshirani, "A note on the group lasso and a sparse group lasso," in *arXiv:1001.0736*, 2010.

[22] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," in *computers & security*. Elsevier, 2009.

[23] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *IEEE S&P*, 2018.

[24] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," in *Neural computation*. MIT Press, 2000.

[25] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016.

[26] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS*, 2014.

[27] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *ACM CCS*, 2018.

[28] D. Han, Z. Wang, W. Chen, Y. Zhong, S. Wang, H. Zhang, J. Yang, X. Shi, and X. Yin, "Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications," in *ACM CCS*, 2021.

[29] D. Han, Z. Wang, Y. Zhong, W. Chen, J. Yang, S. Lu, X. Shi, and X. Yin, "Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors," in *IEEE Journal on Selected Areas in Communications*, 2021.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," in *Neural computation*. MIT Press, 1997.

[31] Y. Ida, Y. Fujiwara, and H. Kashima, "Fast sparse group lasso," in *NeurIPS*, 2019.

[32] Iptable, "Iptables," 2020. [Online]. Available: https://linux.die.net/man/8/iptables

[33] S. T. Jan, Q. Hao, T. Hu, J. Pu, S. Oswal, G. Wang, and B. Viswanath, "Throwing darts in the dark? detecting bots with limited data using neural data augmentation," in *IEEE S&P*, 2020.

[34] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. of the 9th EAI BIONETICS*, 2016.

[35] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting http-based application layer dos attacks on web servers in the presence of sampling," in *Computer Networks*. Elsevier, 2017.

[36] D. Kazhdan, B. Dimanov, M. Jamnik, and P. Liò, "Meme: generating rnn model explanations via model extraction," in *NeurIPS Workshop*, 2020.

[37] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *International Conference on Platform Technology and Service*. IEEE, 2016.

[38] L. Kopitar, L. Cilar, P. Kocbek, and G. Stiglic, "Local vs. global interpretability of machine learning models in type 2 diabetes mellitus screening," in *Artificial intelligence in medicine: Knowledge representation and transparent and explainable systems*. Springer, 2019.

[39] A. Kuppa, S. Grzonkowski, M. R. Asghar, and N.-A. Le-Khac, "Black box attacks on deep anomaly detectors," in *Proc. of the 14th International Conference on Availability, Reliability and Security*, 2019.

[40] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," in *nature*. Nature Publishing Group, 2015.

[41] H. Li, F. Wei, and H. Hu, "Enabling dynamic network access control with anomaly-based ids and sdn," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2019.

[42] S. M. Lundberg, "Is it valid to aggregate shap values to sets of of features?" 2019. [Online]. Available: https://github.com/slundberg/shap/issues/933

[43] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *NeurIPS*, 2017.

[44] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," in *ACM SIGCOMM Computer Communication Review*, 2008.

[45] A. Mendez-Civieta, M. C. Aguilera-Morillo, and R. E. Lillo, "Adaptive sparse group lasso in quantile regression," in *Advances in Data Analysis and Classification*. Springer, 2021.

[46] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," in *Proc. of NDSS*, 2018.

[47] B. Mittelstadt, C. Russell, and S. Wachter, "Explaining explanations in ai," in *Proc. of the conference on fairness, accountability, and transparency*, 2019.

[48] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," in *arXiv:1906.05799*, 2019.

[49] NSL-KDD, 2010. [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html

[50] OpenvSwitch, "Openvswitch," 2020. [Online]. Available: http://vswitch.org/open-vswitch/

[51] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," in *ACM Computing Surveys (CSUR)*. ACM New York, NY, USA, 2021.

[52] V. Paxson, "Bro: a system for detecting network intruders in real-time," in *Computer networks*. Elsevier, 1999.

[53] Pfsense, "Pfsense," 2020. [Online]. Available: https://www.pfsense.org/

[54] M. Z. Rafique and J. Caballero, "Firma: Malware clustering and network signature generation with mixed network behaviors," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2013.

[55] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proc. of the 22nd ACM SIGKDD KDD*, 2016.

[56] ——, "Semantically equivalent adversarial rules for debugging nlp models," in *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018.

[57] A. S. Ross and F. Doshi-Velez, "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[58] U. Schlegel, H. Arnout, M. El-Assady, D. Oelke, and D. A. Keim, "Towards a rigorous evaluation of xai methods on time series," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 2019.

[59] L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Mądry, "Adversarially robust generalization requires more data," in *NeurIPS*, 2018.

[60] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proc. of the IEEE ICCV*, 2017.

[61] L. S. Shapley, "A value for n-person games, contributions to the theory of games," 1953.

[62] M. Sheikhan, Z. Jadidi, and A. Farrokhi, "Intrusion detection using reduced-size rnn based on feature grouping," in *Neural Computing and Applications*. Springer, 2012.

[63] E. C. R. Shin, D. Song, and R. Moazzezi, "Recognizing functions in binaries with neural networks," in *USENIX Security Symposium*, 2015.

[64] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," in *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.

[65] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *ICML*, 2017.

[66] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, "A sparse-group lasso," in *Journal of Computational and Graphical Statistics*. Taylor & Francis Group, 2013.

[67] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *ICLR*, 2013.

[68] C. Smutz and A. Stavrou, "Malicious pdf detection using metadata and structural features," in *Proc. of the 28th annual computer security applications conference (ACSAC)*, 2012.

[69] Snort, "Snort," 2020. [Online]. Available: https://www.snort.org/

[70] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE S&P*, 2010.

[71] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *ICLR*, 2015.

[72] Squid, "Squid," 2020. [Online]. Available: http://www.squid-cache.org/

[73] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *ICML*, 2017.

[74] Suricata, "Suricata," 2022. [Online]. Available: https://suricata.io/

[75] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[76] R. Tibshirani, "Regression shrinkage and selection via the lasso," in *Journal of the Royal Statistical Society: Series B (Methodological)*. Wiley Online Library, 1996.

[77] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight, "Sparsity and smoothness via the fused lasso," in *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. Wiley Online Library, 2005.

[78] M. Toneva and L. Wehbe, "Interpreting and improving natural-language processing (in machines) with natural language-processing (in the brain)," in *NeurIPS*, 2019.

[79] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, "Evaluating explanation methods for deep learning in security," in *IEEE EuroS&P*, 2020.

[80] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, "Snortflow: A openflow-based intrusion prevention system in cloud environment," in *Second GENI research and educational experiment workshop*. IEEE, 2013.

[81] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xinyu, and G. Wang, "Cade: Detecting and explaining concept drift samples for security applications," in *USENIX Security Symposium*, 2021.

[82] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," in *IEEE Access*, 2017.

[83] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," in *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. Wiley Online Library, 2006.

[84] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan, "Theoretically principled trade-off between robustness and accuracy," in *ICML*, 2019.

[85] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *Proc. of NDSS*, 2020.

[86] X. Zhang, N. Wang, H. Shen, S. Ji, X. Luo, and T. Wang, "Interpretable deep learning under fire," in *USENIX Security Symposium*, 2020.

[87] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, "Improving the robustness of deep neural networks via stability training," in *IEEE CVPR*, 2016.

[88] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *IEEE CVPR*, 2016.

[89] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling, "Visualizing deep neural network decisions: Prediction difference analysis," in *ICLR*, 2017.

Appendix A Details of Explanation Methods

As described in Section §3.4, we utilize sparse group lasso (SGL) [21, 31, 66], where the objective function is the sum of an empirical loss and the penalty to derive the explanation results for DL-NIDS.

$$\underset{\beta}{argmin}\Big\{\underbrace{\frac{1}{2n}||\mathbf{y}-\sum_{q=1}^{Q}\mathbb{X}_q\beta_q^T||_2^2}_{\text{empirical loss}}+\underbrace{(1-\alpha)\lambda\sum_{q=1}^{Q}||\beta_q||_2+\alpha\lambda||\beta||_1}_{\text{penalty}}\Big\} \tag{13}$$

As discussed in Section §2, to explain DL-NIDS, we need to address the complicated feature dependencies of structured data (**Ch2**). Unfortunately, IG, LRP, LIME, and SHAP are insufficient to address **Ch2** since they assume each feature is independent. Moreover, LEMNA bonded with consecutive dependency through the adjacent penalty $||\beta_j-\beta_{j-1}||_1$. In contrast, our method can address **Ch2** by applying feature groups and selecting features among and within groups.

To minimize β, we need to solve the objective function in Equ. (13), which is convex. As a result, the optimal solution is characterized by subgradient equations. Simon et al. [66] argue that SGL promotes the desired sparsity both on group and feature levels by solving the subgradient equations.

$$\frac{1}{n}\mathbb{X}_k^T(\mathbf{y}-\sum_{q=1}^{Q}\mathbb{X}_q\hat{\beta}_q^T)=(1-\alpha)\lambda u+\alpha\lambda v \tag{14}$$

where Equ. (14) is the subgradient equation for k^{th} group $\mathbb{X}_k$. u and v are the subgradients of $||\hat{\beta}_k||_2$ and $||\hat{\beta}_k||_1$ evaluated at $\hat{\beta}_k$.

Friedman et al. [21] show that by utilizing block coordinate descent (BCD), SGL can iteratively update the parameters $(\hat{\beta}_k)$ for each group in $(\mathbb{X}_k)$ until convergence. BCD consists

of a group-level outer loop and a feature-level inner loop. The group-level outer-loop cyclically iterates through the groups and checks whether the group's coefficients $\hat{\beta}_k$ are a zero vector. If the k^{th} group's coefficients $\hat{\beta}_k$ turn out to be a nonzero vector, the feature-level inner-loop updates each parameter in $\hat{\beta}_k$. BCD repeats the process until the whole parameter vector converges.

$$\begin{aligned} \hat{\beta}_k^{new} &= (1 - \frac{t(1-\alpha)\lambda}{||S(\mathcal{M}_k, t\alpha\lambda)||_2}) + S(\mathbb{X}_k, t\alpha\lambda) \\ \mathcal{M}_k &= \hat{\beta}_k + \frac{t}{n}(\mathbb{X}_k^T\ r_{-k} - \mathbb{X}_k^T\mathbb{X}_k\hat{\beta}_k) \\ r_{-k} &= y - \sum_{l \neq k}^{Q} \mathbb{X}_q \hat{\beta}_q^T \end{aligned} \tag{15}$$

Equ. (15) is the inner-loop, where $t \geq 0$ is the step size and r_{-k} is the partial residual. $S(\cdot)$ is the coordinate-wise soft-threshold operator. i^{th} element of S is defined as $S(m,\gamma)_i = sign(m_i)(||m_i||_1 - \gamma)_+$. Coefficients of the k^{th} group are iteratively updated until convergence.

$$||S(\mathbb{X}_k^T\ r_{-k}, \alpha\lambda)||_2 \leq \sqrt{p_k}(1-\alpha)\lambda \tag{16}$$

Equ. (16) is the outer loop, which is used to check whether $\hat{\beta}_k$ is a zero vector or not. If Equ. (16) is satisfied, then Equ. (15) is skipped. Otherwise, repeat Equ. (15) until the parameters converge.

Appendix B Syntax of the Unified Rule

```
Notation: Integer n, Wildcard *

Entity        entity  ::=<IP, MAC, port, protocol, flag>
              IP      ::=<src_IP, dst_IP>
              MAC     ::=<src_MAC, dst_MAC>
              port    ::=<src_port, dst_port>
              protocol::=tcp | udp | icmp | arp | http | *
              flags   ::=tcp.syn | tcp.ack |tcp.fin | *

Action        action  ::=drop | allow | modify | whitelist
Priority      priority::=n
Timeout       timeout ::=n
Unified Rule  rule    ::=<entity, action, priority, timeout>
```

Appendix C Details of Experiments Setup

Here we provide details about the experiments in Section §5 and §6.

Appendix C.1 Target DL-NIDS

Kitsune. Kitsune [46] is an autoencoder-based intrusion detection system, which is composed of three major components. (1) Feature Extractor (FE) is responsible for extracting n features from the arriving packets to create a vector to describe the packets and which channel it comes from. FE uses a total of five-time windows, each creating 23 features, thus a total of 115 features. Note that those features are not derived from a single packet. In contrast, they represent the aggregation information of a series of packets involved in those five-time windows. (2) Feature Mapper is designed to divide the features into a set of smaller groups. (3) Anomaly Detector (AD) is responsible for detecting malicious activities. Kitsune has one main parameter for the auto-encoders, $m \in [1, 10]$, which is the maximum number of the features for any encoder of KitNET's ensemble [46]. Since $m = 10$ sometimes performs better than $m = 1$, we set $m = 10$ according to the paper.
ODDS. The bot detector of ODDS [33] is a Long-Short-Term-Memory (LSTM) based model. The input for the LSTM based detector is a matrix (IP sequence) of 30×5 (dimension $= 150$), which contains 30 HTTP Requests, each request has 5 features. The original dataset used in ODDS is collected by collaborating with Radware. Due to privacy concerns, it is not publicly available. As a result, we use the CIC-DoS2017 dataset to build the system. The total packets number of CIC-DoS2017 is $6,393,767$ with $562,669$ HTTP request.
RNN-IDS. we follow [82] to build an RNN-NIDS based on a widely used dataset NSL-KDD. The authors first used data numericalization and normalization approaches to pre-process the data and then feed them to the RNN model.
AE-IDS. we follow [64] to build an autoencoder-NIDS based on the NSL-KDD dataset. To be specific, we test the AE-IDS with the binary classification settings.

Table 12: Detection performance for the trained DL-NIDS

System	Dataset	Precision	Recall	FPR
Kitsune		0.9568	0.9972	0.001
ODDS	CIC-DoS2017	0.9348	0.9465	0.0352
RNN-IDS	NSL-KDD	0.9235	0.9723	0.0137
AE-IDS	NSL-KDD	0.9785	0.9874	0.0215.

Appendix C.2 Explanation Methods

LIME. We set the number of the synthesized samples $n = 500$. For $\pi_{\mathbf{x}}(\mathbf{z})$, we choose *cosine* similarity. We set the number of selected features of k-lasso as the dimension d of the inputs. Finally, we employ *sklearn.linear_model.Lasso* with l_1 prior as the regularizer to solve the regression problem.
SHAP. Since we treat DL-NIDS as blackbox models, we use KernelSHAP, a model agnostic method, to estimate SHAP values for any model. For the implementation, we make use of the open-source code provided by the authors [18].
LEMNA. Three hyper-parameters are configurable: the number of synthesized samples N, the number of mixture components M, and the threshold of the fused lasso s. According to the original paper, LEMNA is not sensitive to these hyper-parameters. In other words, changing the hyper-parameters does not significantly influence the performance of LEMNA [27]. Since the features are not independent in our experiment, we set $s_2 = 1e-4$, $N = 500$, and $M = 6$ according to the paper. We implement the fused lasso regression problem with Python Package cvxy [13].
IG and LRP. We use iNNvestigate [1] toolbox to implement IG and LRP. We set $\varepsilon = 1e-4$ for LRP and $N = 50$ for IG.