Evaluating Explanation Methods for Deep Learning in Security

Alexander Warnecke, Daniel Arp, Christian Wressnegger and Konrad Rieck
Technische Universität Braunschewig, Germany

Abstract—**Deep learning is increasingly used as a building block of security systems. Unfortunately, neural networks are hard to interpret and typically opaque to the practitioner. The machine learning community has started to address this problem by developing methods for explaining the predictions of neural networks. While several of these approaches have been successfully applied in the area of computer vision, their application in security has received little attention so far. It is an open question which explanation methods are appropriate for computer security and what requirements they need to satisfy. In this paper, we introduce criteria for comparing and evaluating explanation methods in the context of computer security. These cover general properties, such as the accuracy of explanations, as well as security-focused aspects, such as the completeness, efficiency, and robustness. Based on our criteria, we investigate six popular explanation methods and assess their utility in security systems for malware detection and vulnerability discovery. We observe significant differences between the methods and build on these to derive general recommendations for selecting and applying explanation methods in computer security.**

1. Introduction

Over the last years, deep learning has been increasingly recognized as an effective tool for computer security. Different types of neural networks have been integrated into security systems, for example, for malware detection [21, 24, 34], binary analysis [10, 42, 51], and vulnerability discovery [31]. Deep learning, however, suffers from a severe drawback: Neural networks are hard to interpret and their decisions are opaque to practitioners. Even simple tasks, such as determining which features of an input contribute to a prediction, are challenging to solve on neural networks. This lack of transparency is a considerable problem in security, as black-box learning systems are hard to audit and protect from attacks [7, 36].

The machine learning community has started to develop methods for interpreting deep learning in computer vision [e.g., 5, 44, 52]. These methods enable tracing back the predictions of neural networks to individual regions in images and thereby help to understand the decision process. These approaches have been further extended to also explain predictions on text and sequences [4, 22]. Surprisingly, this work has received little attention in security and there exists only a single technique that has been investigated so far [22].

In contrast to other application domains of deep learning, computer security poses particular challenges for the use of explanation methods. First, security tasks, such as malware detection and binary code analysis, require complex neural network architectures that are challenging to investigate. Second, explanation methods in security do not only need to be accurate but also satisfy security-specific requirements, such as complete and robust explanations. As a result of these challenges, it is an unanswered question which of the available explanation methods can be applied in security and what properties they need to possess for providing reliable results.

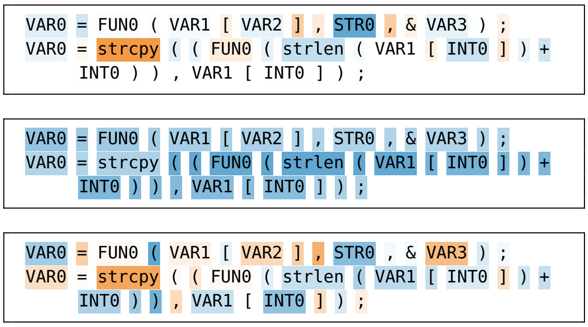
In this paper, we address this problem and develop evaluation criteria for assessing and comparing explanation methods in security. Our work provides a bridge between deep learning in security and explanation methods developed for other application domains of machine learning. Consequently, our criteria for judging explanations cover general properties of deep learning as well as specific aspects that are unique to the domain of security.

General evaluation criteria. As general criteria, we consider the *descriptive accuracy* and *sparsity* of explanations. These properties reflect how accurate and concise an explanation method captures relevant features of a prediction. While accuracy is an evident criterion for obtaining reliable results, sparsity is another crucial constraint in security. In contrast to computer vision, where an analyst can examine an entire image, a security practitioner cannot investigate large sets of features at once, and thus sparsity becomes an essential property when non-graphic data is analyzed.

Security evaluation criteria. We define the *completeness*, *stability*, *robustness*, and *efficiency* of explanations as security criteria. These properties ensure that reliable explanations are available to a practitioner in all cases and in reasonable time—requirements that are less important in other areas of deep learning. For example, an attacker may expose pathologic inputs to a security system that mislead, corrupt, or slow down the computation of explanations. Note that the robustness of explanation methods to adversarial examples is not well understood yet, and thus we base our analysis on the recent work by Zhang et al. [53] and Dombrowski et al. [14].

With the help of these criteria, we analyze six recent explanation methods and assess their performance in different security tasks. To this end, we implement four security systems from the literature that make use of deep learning and enable detecting Android malware [21, 34], malicious PDF files [50], and security vulnerabilities [31], respectively. When explaining the decisions of these systems, we observe significant differences between the methods in all criteria. Some methods are not capable of providing sparse results, whereas others struggle with structured security data or suffer from unstable outputs. While the importance of the individual criteria depends on the particular task, we find that the methods IG [48] and LRP [5] comply best with all criteria and resemble general-purpose techniques for security systems.

To demonstrate the utility of explainable learning, we also qualitatively examine the generated explanations. As an example for this investigation, Figure 1 shows three explanations for the system VulDeePecker [31] that identifies vulnerabilities in source code. While the first explanation method provides a nuanced representation of the relevant features, the second method generates an unsharp explanation due to a lack of sparsity. The third approach provides an explanation that even contradicts the first one. Note that the variables VAR2 and VAR3 receive a positive relevance (blue) in the first case and a negative relevance (orange) in the third.



```
VAR0 = FUN0 ( VAR1 [ VAR2 ] , STR0 , & VAR3 ) ;
VAR0 = strcpy ( ( FUN0 ( strlen ( VAR1 [ INT0 ] ) +
       INT0 ) ) , VAR1 [ INT0 ] ) ;
```

```
VAR0 = FUN0 ( VAR1 [ VAR2 ] , STR0 , & VAR3 ) ;
VAR0 = strcpy ( ( FUN0 ( strlen ( VAR1 [ INT0 ] ) +
       INT0 ) ) , VAR1 [ INT0 ] ) ;
```

```
VAR0 = FUN0 ( VAR1 [ VAR2 ] , STR0 , & VAR3 ) ;
VAR0 = strcpy ( ( FUN0 ( strlen ( VAR1 [ INT0 ] ) +
       INT0 ) ) , VAR1 [ INT0 ] ) ;
```

Figure 1: Explanations for the prediction of the security system VulDeePecker on a code snippet from the original dataset. From top to bottom: LRP, LEMNA, and LIME.

Our evaluation highlights the need for comparing explanation methods and determining the best fit for a given security task. Furthermore, it also unveils a notable number of artifacts in the underlying datasets. For all of the four security tasks, we identify features that are unrelated to security but strongly contribute to the predictions. As a consequence, we argue that explanation methods need to become an integral part of learning-based security systems—first, for understanding the decision process of deep learning and, second, for eliminating artifacts in the training datasets.

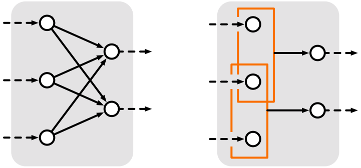
The rest of this paper is organized as follows: We briefly review the technical background of explainable learning in Section 2. The explanation methods and security systems under test are described in Section 3. We introduce our criteria for comparing explanation methods in Section 4 and evaluate them in Section 5. Our qualitative analysis is presented in Section 6 and Section 7 concludes the paper.

2. Explainable Deep Learning

Neural networks have been used in artificial intelligence for over 50 years, yet concepts for explaining their decisions have just recently started to be explored. This development has been driven by the remarkable progress of deep learning in several areas, such as image recognition [29] and machine translation [49]. To embed our work in this context, we briefly review two aspects of explainable learning that are crucial for its application in security: the *type of neural network* and the *explanation strategy*.

2.1. Neural Network Architectures

Different architectures can be used for constructing a neural network, ranging from general-purpose networks to highly specific architectures. In the area of security, three of these architectures are prevalent: *multilayer perceptrons*, *convolutional neural networks*, and *recurrent neural networks* (see Figure 2). Consequently, we focus our study on these network types and refer the reader to the books by Rojas [38] and Goodfellow et al. [20] for a detailed description of network architectures in general.



(a) MLP layer (b) CNN layer (c) RNN layer

Figure 2: Overview of network architectures in security: Multilayer perceptrons (MLP), convolutional neural networks (CNN), and recurrent neural networks (RNN).

Multilayer Perceptrons (MLPs). Multilayer perceptrons, also referred to as *feedforward networks*, are a classic and general-purpose network architecture [39]. The network is composed of multiple fully connected layers of neurons, where the first and last layer correspond to the input and output of the network, respectively. MLPs have been successfully applied to a variety of security problems, such as intrusion and malware detection [21, 24]. While MLP architectures are not necessarily complex, explaining the contribution of individual features is still difficult, as several neurons impact the decision when passing through the network layers.

Convolutional Neural Networks (CNNs). These networks share a similar architecture with MLPs, yet they differ in the concept of *convolution* and *pooling* [30]. The neurons in convolutional layers receive input only from a local neighborhood of the previous layer. These neighborhoods overlap and create receptive fields that provide a powerful primitive for identifying spatial structure in data. CNNs have thus been successfully used for detecting malicious patterns in the bytecode of Android applications [34]. Due to the convolution and pooling layers, however, it is hard to explain the decisions of a CNN, as its output needs to be "unfolded" and "unpooled" for analysis.

Recurrent Neural Networks (RNNs). Recurrent networks, such as LSTM and GRU networks [9, 23], are characterized by a recurrent structure, that is, some neurons are connected in a loop. This structure enables memorizing information and allows RNNs to operate on sequences of data [16]. As a result, RNNs have been successfully applied in security tasks involving sequential data, such as the recognition of functions in native code [10, 42] or the discovery of vulnerabilities in software [31]. Interpreting the prediction of an RNN is also difficult, as the relevance of an input feature depends on the sequence of previously processed features.

2.2. Explanation Strategies

Given the different architectures and the complexity of many neural networks, decoding the entire decision process is a challenging task that currently cannot be solved adequately. However, there exist several recent methods that enable explaining individual predictions of a neural network instead of the complete decision process [e.g., 5, 22, 37, 48, 52]. We focus on this form of explainable learning that can be formally defined as follows:

Definition 1. Given an input vector $x = (x_1, \ldots, x_d)$, a neural network N, and a prediction $f_N(x) = y$, an explanation method determines why the label y has been selected by N. This explanation is given by a vector $r = (r_1, \ldots, r_d)$ that describes the relevance of the dimensions of x for $f_N(x)$.

The computed relevance values r are typically real numbers and can be overlayed with the input in form of a heatmap, such that relevant features are visually highlighted. An example of this visualization is depicted in Figure 1. Positive relevance values are shown in blue and indicate importance *towards* the prediction $f_N(x)$, whereas negative values are given in orange and indicate importance *against* the prediction. We will use this color scheme throughout the paper[1].

Despite the variety of approaches for computing a relevance vector for a given neural network and an input, all approaches can be broadly categorized into two explanation strategies: *black-box* and *white-box* explanations.

Black-box Explanations. These methods operate under a black-box setting that assumes no knowledge about the neural network and its parameters. Black-box methods are an effective tool if no access to the neural network is available, for example, when a learning service is audited remotely. Technically, black-box methods rest on an approximation of the function f_N, which enables them to estimate how the dimensions of x contribute to a prediction. Although black-box methods are a promising approach for explaining deep learning, they can be impaired by the black-box setting and omit valuable information provided through the network architecture and parameters.

White-box Explanations. These approaches operate under the assumption that all parameters of a neural network are known and can be used for determining an explanation. As a result, these methods do not rely on approximations and can directly compute explanations for the function f_N on the structure of the network. In practice, predictions and explanations are often computed from within the same system, such that the neural network is readily available for generating explanations. This is usually the case for stand-alone systems for malware detection, binary analysis, and vulnerability discovery. However, several white-box methods are designed for specific network layouts from computer vision and not applicable to all considered architectures [e.g., 44, 47, 52].

Black-box and white-box explanation methods often share similarities with concepts of adversarial learning and feature selection, as these also aim at identifying features related to the prediction of a classifier. However, adversarial learning and feature selection pursue fundamentally different goals and cannot be directly applied for explaining neural networks. We discuss the differences to these approaches for the interested reader in Appendix A.

3. Methods and Systems under Test

Before presenting our criteria for evaluating explanation methods, we first introduce the methods and systems under test. In particular, we cover six methods for explaining predictions in Section 3.1 and present four security systems based on deep learning in Section 3.2. For more information about explanation methods we do not evaluate in the paper [e.g., 12, 17] we refer the reader to the Appendix B.

3.1. Explanation Methods

Table 1 provides an overview of popular explanation methods along with their support for the different network architectures. As we are interested in explaining predictions of security systems, we select those methods for our study that are applicable to all common architectures. In the following, we briefly sketch the main idea of these approaches for computing relevance vectors, illustrating the technical diversity of explanation methods.

Gradients and IG. One of the first white-box methods to compute explanations for neural networks has been introduced by Simonyan et al. [44] and is based on simple gradients. The output of the method is given by $r_i = \partial y / \partial x_i$, which the authors call a *saliency map*. Here r_i measures how much y changes with respect to x_i. Sundararajan et al. [48] extend this approach and propose Integrated Gradients (IG) that use a baseline x', for instance a vector of zeros, and calculate the shortest path from x' to x, given by $x - x'$. To compute the relevance of x_i, the gradients with respect to x_i are cumulated along this path yielding

$$r_i = (x_i - x'_i) \int_0^1 \frac{\partial f_N(x' + \alpha(x - x'))}{\partial x_i} \mathrm{d}\alpha.$$

Both gradient-based methods can be applied to all relevant network architectures and thus are considered in our comparative evaluation of explanation methods.

LRP and DeepLift. These popular white-box methods determine the relevance of a prediction by performing a backward pass through the neural network, starting at

TABLE 1: Popular explanation methods. The support for different neural network architectures is indicated by ✓. Methods evaluated in this paper are indicated by *.

Explanation methods	MLP	CNN	RNN
Gradients* [44], IG* [48]	✓	✓	✓
LRP* [5], DeepLift [43]	✓	✓	✓
PatternNet, PatternAttribution [25]	✓	✓	–
DeConvNet [44, 52], GuidedBP [47]	✓	✓	–
CAM [54], GradCAM [8, 40]	✓	✓	–
RTIS [11], MASK [17]	✓	✓	–
LIME* [37], SHAP* [32], QII [12]	✓	✓	✓
LEMNA* [22]	✓	✓	✓

[1] We use the blue-orange color scheme instead of the typical green-red scheme to make our paper better accessible to color-blind readers.

the output layer and performing calculations until the input layer is reached [5]. The central idea of layer-wise relevance propagation (LRP) is the use of a conservation property that needs to hold true during the backward pass. If r_i^l is the relevance of the unit i in layer l of the neural network then

$$\sum_i r_i^1 = \sum_i r_i^2 = \cdots = \sum_i r_i^L$$

needs to hold true for all L layers. Similarly, DeepLift performs a backward pass but takes a reference activation $y' = f_N(x')$ of a reference input x' into account. The method enforces the conservation law,

$$\sum_i r_i = y - y' = \Delta y\,,$$

that is, the relevance assigned to the features must sum up to the difference between the outcome of x and x'. Both approaches support explaining the decisions of feed-forward, convolutional and recurrent neural networks [see 4]. However, as DeepLift and IG are closely related [2], we focus our study on the method ϵ-LRP.

LIME and SHAP. Ribeiro et al. [37] introduce one of the first black-box methods for explaining neural networks that is further extended by Lundberg and Lee [32]. Both methods aim at approximating the decision function f_N by creating a series of l perturbations of x, denoted as $\tilde{x}_1, \ldots, \tilde{x}_l$ by setting entries in the vector x to 0 randomly. The methods then proceed by predicting a label $f_N(\tilde{x}_i) = \tilde{y}_i$ for each $\tilde{x}_i$ of the l perturbations. This sampling strategy enables the methods to approximate the local neighborhood of f_N at the point $f_N(x)$. LIME [37] approximates the decision boundary by a weighted linear regression model,

$$\arg\min_{g \in \mathcal{G}} \sum_{i=1}^{l} \pi_x(\tilde{x}_i)\big(f_N(\tilde{x}_i) - g(\tilde{x}_i)\big)^2,$$

where $\mathcal{G}$ is the set of all linear functions and π_x is a function indicating the difference between the input x and a perturbation $\tilde{x}$. SHAP [32] follows the same approach but uses the SHAP kernel as weighting function π_x, which is shown to create *Shapley Values* [41] when solving the regression. Shapley Values are a concept from game theory where the features act as players under the objective of finding a fair contribution of the features to the payout—in this case the prediction of the model. As both approaches can be applied to any learning model, we study them in our empirical evaluation.

LEMNA. As last explanation method, we consider LEMNA, a black-box method specifically designed for security applications [22]. It uses a mixture regression model for approximation, that is, a weighted sum of K linear models:

$$f(x) = \sum_{j=1}^{K} \pi_j(\beta_j \cdot x + \epsilon_j).$$

The parameter K specifies the number of models, the random variables $\epsilon = (\epsilon_1, \ldots, \epsilon_K)$ originate from a normal distribution $\epsilon_i \sim N(0, \sigma)$ and $\pi = (\pi_1, \ldots, \pi_K)$ holds the weights for each model. The variables $\beta_1, \ldots, \beta_K$ are the regression coefficients and can be interpreted as K linear approximations of the decision boundary near $f_N(x)$.

3.2. Security Systems

As field of application for the six explanation methods, we consider four recent security systems that employ deep learning (see Table 2). The systems cover the three major architectures/types introduced in Section 2.1 and comprise between 4 to 6 layers of different types.

TABLE 2: Overview of the considered security systems.

System	Publication	Type	# Layers
Drebin+	ESORICS'17 [21]	MLP	4
Mimicus+	CCS'18 [22]	MLP	4
DAMD	CODASPY'17 [34]	CNN	6
VulDeePecker	NDSS'18 [31]	RNN	5

Drebin+. The first system uses an MLP for identifying Android malware. The system has been proposed by Grosse et al. [21] and builds on features originally developed by Arp et al. [3]. The network consists of two hidden layers, each comprising 200 neurons. The input features are statically extracted from Android applications and cover data from the application's manifest, such as hardware details and requested permissions, as well as information based on the application's code, such as suspicious API calls and network addresses. To verify the correctness of our implementation, we train the system on the original Drebin dataset [3], where we use 75 % of the 129,013 Android application for training and 25 % for testing. Table 3 shows the results of this experiment, which are in line with the performance published by Grosse et al. [21].

Mimicus+. The second system also uses an MLP but is designed to detect malicious PDF documents. The system is re-implemented based on the work of Guo et al. [22] and builds on features originally introduced by Smutz and Stavrou [46]. Our implementation uses two hidden layers with 200 nodes each and is trained with 135 features extracted from PDF documents. These features cover properties about the document structure, such as the number of sections and fonts in the document, and are mapped to binary values as described by Guo et al. [22]. For a full list of features, we refer the reader to the implementation by Šrndić and Laskov [50]. For verifying our implementation, we make use of the original dataset that contains 5,000 benign and 5,000 malicious PDF files and again split the dataset into 75 % for training and 25 % for testing. Our results are shown in Table 3 and come close to a perfect detection.

DAMD. The third security system studied in our evaluation uses a CNN for identifying malicious Android applications [34]. The system processes the raw Dalvik bytecode of Android applications and its neural network is comprised of six layers for embedding, convolution, and max-pooling of the extracted instructions. As the system processes entire applications, the number of features depends on the size of the applications. For a detailed description of this process, we refer the reader to the publication by McLaughlin et al. [34]. To replicate the original results, we apply the system to data from the Malware Genome Project [55]. This dataset consists of 2,123 applications in total, with 863 benign and 1,260 malicious samples. We again split the dataset into 75 % of training and 25 % of testing data and obtain results similar to those presented in the original publication.

VulDeePecker. The fourth system uses an RNN for discovering vulnerabilities in source code [31]. The RNN consists of five layers, uses 300 LSTM cells [23], and applies a word2vec embedding [35] with 200 dimensions for analyzing C/C++ code. As a preprocessing step, the source code is sliced into code gadgets that comprise short snippets of tokens. The gadgets are truncated or padded to a length of 50 tokens. For verifying the correctness of our implementation, we use the CWE-119 dataset, which consists of 39,757 code gadgets, with 10,444 gadgets corresponding to vulnerabilities. In line with the original study, we split the dataset into 80 % training and 20 % testing data, and attain a comparable accuracy.

The four selected security systems provide a diverse view on the current use of deep learning in security. Drebin+ and Mimicus+ are examples of systems that make use of MLPs for detecting malware. However, they differ in the dimensionality of the input: While Mimicus+ works on a small set of engineered features, Drebin+ analyzes inputs with thousands of dimensions. DAMD is an example of a system using a CNN in security and capable of learning from large inputs, whereas VulDeePecker makes use of an RNN, similar to other learning-based approaches analyzing program code [e.g., 10, 42, 51].

TABLE 3: Performance of the re-implemented security systems on the original datasets.

System	Accuracy	Precision	Recall	F1-Score
Drebin+	0.980	0.926	0.924	0.925
Mimicus+	0.994	0.991	0.998	0.994
DAMD	0.949	0.967	0.924	0.953
VulDeePecker	0.908	0.837	0.802	0.819

4. Evaluation Criteria

In light of the broad range of available explanation methods, the practitioner is in need of criteria for selecting the best method for a security task at hand. In this section, we develop these criteria and demonstrate their utility in different examples. Before doing so, however, we address another important question: Do the considered explanation methods provide different results? If the methods generated similar explanations, criteria for their comparison would be less important and any suitable method could be chosen in practice.

To answer this question, we investigate the top-k features of the six explanation methods when explaining predictions of the security systems. That is, we compare the set T_i of the k features with the highest relevance from method i with the set T_j of the k features with the highest relevance from method j. In particular, we compute the *intersection size*

$$\mathrm{IS}(i, j) = \frac{|T_i \cap T_j|}{k},$$

as a measure of similarity between the two methods. The intersection size lies between 0 and 1, where 0 indicates no overlap and 1 corresponds to identical top-k features.

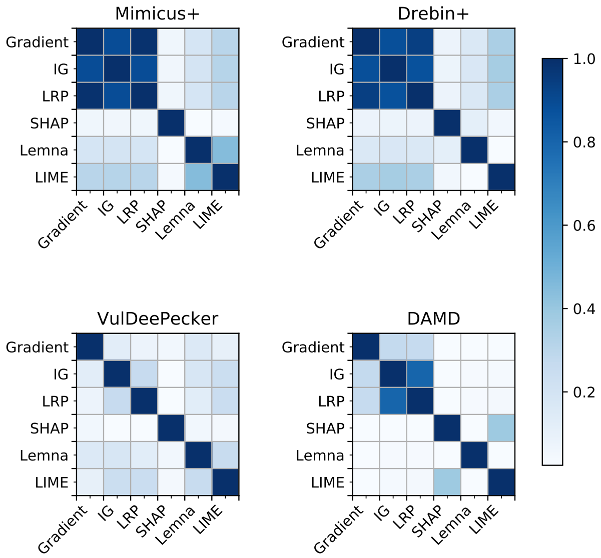


Figure 3: Comparison of the top-10 features for the different explanation methods. An average value of 1 indicates identical top-10 features and a value of 0 indicates no overlap.

A visualization of the intersection size averaged over the samples of the four datasets is shown in Figure 3. We choose $k = 10$ according to a typical use case of explainable learning: An expert investigates the top-10 features to gain insights on a prediction. For DAMD, we use $k = 50$, as the dataset is comprised of long opcode sequences. We observe that the top features of the explanation methods differ considerably. For example, in the case of VulDeePecker, all methods determine different top-10 features. While we notice some similarity between the methods, it becomes clear that the methods cannot be simply interchanged, and there is a need for measurable evaluation criteria.

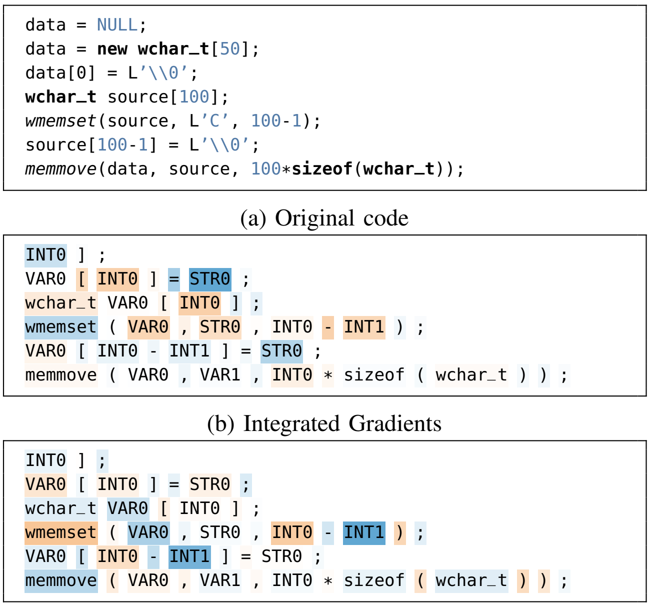
4.1. General Criteria: Descriptive Accuracy

As the first evaluation criteria, we introduce the *descriptive accuracy*. This criterion reflects how accurate an explanation method captures relevant features of a prediction. As it is difficult to assess the relation between features and a prediction directly, we follow an indirect strategy and measure how removing the most relevant features changes the prediction of the neural network.

Definition 2. Given a sample x, the descriptive accuracy (DA) is calculated by removing the k most relevant features $x_1, \ldots, x_k$ from the sample, computing the new prediction using f_N and measuring the score of the original prediction class c without the k features,

$$\mathrm{DA}_k\left(x, f_N\right) = f_N\left(x \mid x_1 = 0, \ldots, x_k = 0\right)_c.$$

If we remove relevant features from a sample, the accuracy should decrease, as the neural network has less information for making a correct prediction. The better the explanation, the quicker the accuracy will drop, as the removed features capture more context of the predictions. Consequently, explanation methods with a steep decline of the descriptive accuracy provide better explanations than methods with a gradual decrease.



```
1 data = NULL;
2 data = new wchar_t[50];
3 data[0] = L'\\0';
4 wchar_t source[100];
5 wmemset(source, L'C', 100-1);
6 source[100-1] = L'\\0';
7 memmove(data, source, 100*sizeof(wchar_t));
```

(a) Original code

```
INT0 ] ;
VAR0 [ INT0 ] = STR0 ;
wchar_t VAR0 [ INT0 ] ;
wmemset ( VAR0 , STR0 , INT0 - INT1 ) ;
VAR0 [ INT0 - INT1 ] = STR0 ;
memmove ( VAR0 , VAR1 , INT0 * sizeof ( wchar_t ) ) ;
```

(b) Integrated Gradients

```
INT0 ] ;
VAR0 [ INT0 ] = STR0 ;
wchar_t VAR0 [ INT0 ] ;
wmemset ( VAR0 , STR0 , INT0 - INT1 ) ;
VAR0 [ INT0 - INT1 ] = STR0 ;
memmove ( VAR0 , VAR1 , INT0 * sizeof ( wchar_t ) ) ;
```

(c) LIME

Figure 4: Explanations for a program slice from the VulDeePecker dataset using (b) Integrated Gradients and (c) LIME.

To demonstrate the utility of the descriptive accuracy, we consider a sample from the VulDeePecker dataset, which is shown in Figure 4(a). The sample corresponds to a program slice and is passed to the neural network as a sequence of tokens. Figures 4(b) and 4(c) depict these tokens overlayed with the explanations of the methods Integrated Gradients (IG) and LIME, respectively. Note that the VulDeePecker system truncates all code snippets to a length of 50 tokens before processing them through the neural network [31].

The example shows a simple buffer overflow which originates from an incorrect calculation of the buffer size in line 7. The two explanation methods significantly differ when explaining the detection of this vulnerability. While IG highlights the `wmemset` call as important, LIME highlights the call to `memmove` and even marks `wmemset` as speaking *against* the detection. Measuring the descriptive accuracy can help to determine which of the two explanations reflects the prediction of the system better.

4.2. General Criteria: Descriptive Sparsity

Assigning high relevance to features which impact a prediction is a necessary prerequisite for good explanations. However, a human analyst can only process a limited number of these features, and thus we define the *descriptive sparsity* as a further criterion for comparing explanations methods as follows:

Definition 3. The descriptive sparsity is measured by scaling the relevance values to the range $[-1, 1]$, computing a normalized histogram h of them and calculating the *mass around zero* (MAZ) defined by

$$\text{MAZ}(r) = \int_{-r}^{r} h(x)\text{d}x \text{ for } r \in [0, 1].$$

TABLE 4: Explanations of LRP and LEMNA for a sample of the GoldDream family from the DAMD dataset.

Id	LRP	LEMNA
0	invoke-virtual	invoke-virtual
1	move-result-object	move-result-object
2	if-eqz	if-eqz
3	const-string	const-string
4	invoke-virtual	invoke-virtual
5	move-result-object	move-result-object
6	check-cast	check-cast
7	array-length	array-length
8	new-array	new-array
9	const/4	const/4
10	array-length	array-length
11	if-ge	if-ge
12	aget-object	aget-object

The MAZ can be thought of as a window which starts at 0 and grows uniformly into the positive and negative direction of the x axis. For each window, the fraction of relevance values that lies in the window is evaluated. Sparse explanations have a steep rise in MAZ close to 0 and are flat around 1, as most of the features are not marked as relevant. By contrast, dense explanations have a notable smaller slope close to 0, indicating a larger set of relevant features. Consequently, explanation methods with a MAZ distribution peaking at 0 should be preferred over methods with less pronounced distributions.

As an example of a sparse and dense explanation, we consider two explanations generated for a malicious Android application of the DAMD dataset. Table 4 shows a snapshot of these explanations, covering opcodes of the `onReceive` method. LRP provides a crisp representation in this setting, whereas LEMNA marks the entire snapshot as relevant. If we normalize the relevance vectors to $[-1, 1]$ and focus on features above 0.2, LRP returns only 14 relevant features for investigation, whereas LEMNA returns 2,048 features, rendering a manual examination tedious.

It is important to note that the descriptive accuracy and the descriptive sparsity are not correlated and must *both* be satisfied by an effective explanation method. A method marking all features as relevant while highlighting a few ones can be accurate but is clearly not sparse. Vice versa, a method assigning high relevance to very few meaningless features is sparse but not accurate.

4.3. Security Criteria: Completeness

After introducing two generic evaluation criteria, we start focusing on aspects unique to the area of security. In a security system, an explanation method must be capable of creating proper results in all possible situations. If some inputs, such as pathological data or corner cases, cannot be processed by an explanation method, an adversary may trick the method into producing *degenerated* results. Consequently, we propose *completeness* as the first security-specific criterion.

Definition 4. An explanation method is complete, if it can generate non-degenerated explanations for all possible input vectors of the prediction function f_N.

TABLE 5: Explanations for the Android malware FakeInstaller generated for Drebin+ using Gradients and SHAP.

Id	Gradients	SHAP
0	feature::android.hardware.touchscreen	feature::android.hardware.touchscreen
1	intent::android.intent.category.LAUNCHER	intent::android.intent.category.LAUNCHER
2	real_permission::android.permission.INTERNET	real_permission::android.permission.INTERNET
3	api_call::android/webkit/WebView	api_call::android/webkit/WebView
4	intent::android.intent.action.MAIN	intent::android.intent.action.MAIN
5	url::translator.worldclockr.com	url::translator.worldclockr.com
6	url::translator.worldclockr.com/android.html	url::translator.worldclockr.com/android.html
7	permission::android.permission.INTERNET	permission::android.permission.INTERNET
8	activity::.Main	activity::.Main

Several white-box methods are complete by definition, as they calculate relevance vectors directly from the weights of the neural network. For black-box methods, however, the situation is different: If a method approximates the prediction function f_N using random perturbations, it may fail to derive a valid estimate of f_N and return degenerated explanations. We investigate this phenomenon in more detail in Section 5.4.

As an example of this problem, Table 5 shows explanations generated by the methods Gradients and SHAP for a benign Android application of the Drebin dataset. The Gradients explanation finds the touchscreen feature in combination with the launcher category and the internet permission as an explanation for the benign classification. SHAP, however, creates an explanation of zeros which provides no insights. The reason for this degenerated explanation is rooted in the random perturbations used by SHAP. By flipping the value of features, these perturbations aim at changing the class label of the input. As there exist far more benign features than malicious ones in the case of Drebin+, the perturbations can fail to switch the label and prevent the linear regression to work resulting in a degenerated explanation.

4.4. Security Criteria: Stability

In addition to complete results, the explanations generated in a security system need to be reliable. That is, relevant features must not be affected by fluctuations and need to remain stable over time in order to be useful for an expert. As a consequence, we define *stability* as another security-specific evaluation criterion.

Definition 5. An explanation methods is stable, if the generated explanations do not vary between multiple runs. That is, for any run i and j of the method, the intersection size of the top features T_i and T_j should be close to 1, that is, $\text{IS}(i, j) > 1 - \epsilon$ for some small threshold ϵ.

The stability of an explanation method can be empirically determined by running the methods multiple times and computing the average intersection size, as explained in the beginning of this section. White-box methods are deterministic by construction since they perform a fixed sequence of computations for generating an explanation. Most black-box methods, however, require random perturbations to compute their output which can lead to different results for the same input. Table 6, for instance, shows the output of LEMNA for a PDF document from the Mimicus+ dataset over two runs. Some of the most relevant features from the first run receive very little relevance in the second run and vice versa, rendering the explanations unstable. We analyze these instabilities of the explanation methods in Section 5.5.

TABLE 6: Two explanations from LEMNA for the same sample computed in different runs.

Id	LEMNA (Run 1)	LEMNA (Run 2)
0	pos_page_min	pos_page_min
1	count_js	count_js
2	count_javascript	count_javascript
3	pos_acroform_min	pos_acroform_min
4	ratio_size_page	ratio_size_page
5	pos_image_min	pos_image_min
6	count_obj	count_obj
	...	...
27	pos_image_max	pos_image_max
28	count_page	count_page
29	len_stream_avg	len_stream_avg
30	pos_page_avg	pos_page_avg
31	count_stream	count_stream
32	moddate_tz	moddate_tz
33	len_stream_max	len_stream_max
34	count_endstream	count_endstream

4.5. Security Criteria: Efficiency

When operating a security system in practice, explanations need to be available in reasonable time. While low run-time is not a strict requirement in general, time differences between minutes and milliseconds are still significant. For example, when dealing with large amounts of data, it might be desirable for the analyst to create explanations for every sample of an entire class. We thus define *efficiency* as a further criterion for explanation methods in security applications.

Definition 6. We consider a method efficient if it enables providing explanations without delaying the typical workflow of an expert.

As the workflow depends on the particular security task, we do not define concrete run-time numbers, yet we provide a negative example as an illustration. The run-time of the method LEMNA depends on the size of the inputs. For the largest sample of the DAMD dataset with 530,000 features, it requires about one hour for computing an explanation, which obstructs the workflow of inspecting Android malware severely.

4.6. Security Criteria: Robustness

As the last criterion, we consider the *robustness* of explanation methods to attacks. Recently, several attacks [e.g., 14, 45, 53] have shown that explanation methods may suffer from adversarial perturbations and can be tricked into returning incorrect relevance vectors, similarly to adversarial examples [7]. The objective of these attacks is to disconnect the explanation from the underlying prediction, such that arbitrary relevance values can be generated that do not explain the behavior of the model.

Definition 7. An explanation method is robust if the computed relevance vector cannot be decoupled from the prediction by an adversarial perturbation.

Unfortunately, the robustness of explanation methods is still not well understood and, similarly to adversarial examples, guarantees and strong defenses have not been established yet. To this end, we assess the robustness of the explanation methods based on the existing literature.

5. Evaluation

Equipped with evaluation criteria for comparing explanation methods, we proceed to empirically investigate these in different security tasks. To this end, we implement a comparison framework that integrates the six selected explanation methods and four security systems.

5.1. Experimental Setup

White-box Explanations. For our comparison framework, we make use of the `iNNvestigate` toolbox by Alber et al. [1] that provides efficient implementations for LRP, Gradients, and IG. For the security system VulDeePecker, we additionally apply the RNN extension by Arras et al. [4]. In all experiments, we set $\epsilon = 10^{-3}$ for LRP and use $N = 64$ steps for IG. Due to the high dimensional embedding space of VulDeePecker, we choose a step count of $N = 256$ in the corresponding experiments.

Black-box Explanations. We re-implement LEMNA in accordance to Guo et al. [22] and use the Python package `cvxpy` [13] to solve the linear regression problem with Fused Lasso restriction. We set the number of mixture models to $K = 3$ and the number of perturbations to $l = 500$. The parameter S is set to 10^4 for Drebin+ and Mimicus+, as the underlying features are not sequential and to 10^{-3} for the sequences of DAMD and VulDeePecker [see 22]. Furthermore, we implement LIME with $l = 500$ perturbations, use the cosine similarity as proximity measure, and employ the regression solver from the `scipy` package using L_1 regularization. For SHAP we make use of the open-source implementation by Lundberg and Lee [32] including the `KernelSHAP` solver.

5.2. Descriptive Accuracy

We start our evaluation by measuring the descriptive accuracy (DA) of the explanation methods as defined in Section 4. In particular, we successively remove the most relevant features from the samples of the datasets and measure the decrease in the classification score. For Drebin+ and Mimicus+, we remove features by setting the corresponding dimensions to 0. For DAMD, we replace the most relevant instructions with the no-op opcode, and for VulDeePecker we substitute the selected tokens with an embedding-vector of zeros.

TABLE 7: Descriptive accuracy (DA) and sparsity (MAZ) for the different explanation methods.

Method	Drebin+	Mimicus+	DAMD	VulDeePecker
LIME	0.580	0.257	0.919	0.571
LEMNA	0.656	0.405	0.983	0.764
SHAP	0.891	0.565	0.966	0.869
Gradients	0.472	0.213	0.858	0.856
IG	0.446	0.206	0.499	0.574
LRP	0.474	0.213	0.504	0.625

(a) Area under the DA curves from Figure 5.

Method	Drebin+	Mimicus+	DAMD	VulDeePecker
LIME	0.757	0.752	0.833	0.745
LEMNA	0.681	0.727	0.625	0.416
SHAP	0.783	0.716	0.713	0.813
Gradients	0.846	0.856	0.949	0.816
IG	0.847	0.858	0.999	0.839
LRP	0.846	0.856	0.964	0.827

(b) Area under MAZ curves from Figure 5.

The top row in Figure 5 shows the results of this experiment. As the first observation, we find that the DA curves vary significantly between the explanation methods and security systems. However, the methods IG and LRP consistently obtain strong results in all settings and show steep declines of the descriptive accuracy. Only on the VulDeePecker dataset, the black-box method LIME can provide explanations with comparable accuracy. Notably, for the DAMD dataset, IG and LRP are the only methods to generate real impact on the outcome of the classifier. For Mimicus+, IG, LRP and Gradients achieve a perfect accuary decline after only 25 features and thus the white-box explanation methods outperform the black-box methods in this experiment.

Table 7(a) shows the *area under curve* (AUC) for the descriptive accuracy curves from Figure 5. We observe that IG is the best method over all datasets—lower values indicate better explanations—followed by LRP. In comparison to other methods it is up to 48 % better on average. Intuitively, this considerable difference between the white-box and black-box methods makes sense, as white-box approaches can utilize internal information of the neural networks that are not available to black-box methods.

5.3. Descriptive Sparsity

We proceed by investigating the sparsity of the generated explanations with the MAZ score defined in Section 4. The second row in Figure 5 shows the result of this experiment for all datasets and methods. We observe that the methods IG, LRP, and Gradients show the steepest slopes and assign the majority of features little relevance, which indicates a sparse distribution. By contrast, the other explanation methods provide flat slopes of the MAZ close to 0, as they generate relevance values with a broader range and thus are less sparse.

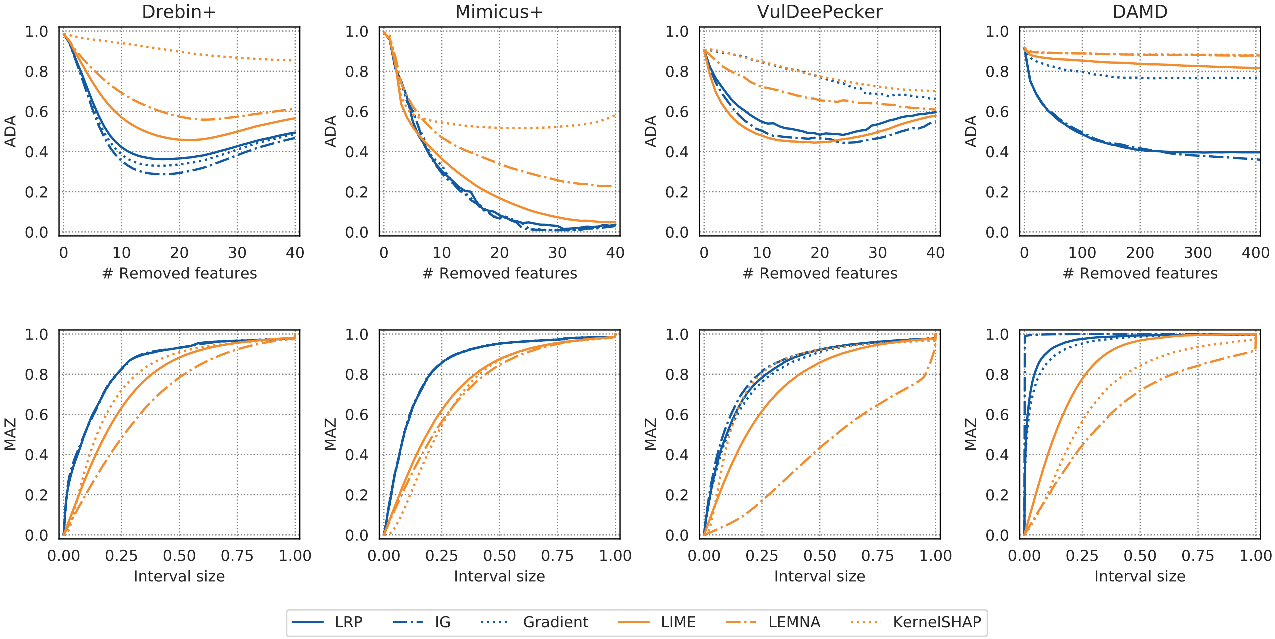


Figure 5: Descriptive accuracy and sparsity for the considered explanation methods. Top row: Average descriptive accuracy (ADA); bottom row: sparsity measured as mass around zero (MAZ).

Interestingly, for VulDeePecker, the MAZ curve of LEMNA shows a strong increase close to 1, indicating that it assigns high relevance to a lot of tokens. While this generally is undesirable, in case of LEMNA, this is founded in the basic design and the use of the Fused Lasso constraint. In case of DAMD, we see a massive peak at 0 for IG, showing that it marks almost all features as irrelevant. According to the previous experiment, however, it simultaneously provides a very good accuracy on this data. The resulting sparse and accurate explanations are particularly advantageous for a human analyst since the DAMD dataset contains up to 520,000 features per sample.

We summarize the performance on the MAZ metric by calculating the *area under curve* and report it in Table 7(b). A high AUC indicates that more features have been assigned a relevance close to 0, that is, the explanation is more sparse. We find that the best methods again are white-box approaches, providing explanations that are up to 50 % sparser compared to the other methods in this experiment.

5.4. Completeness of Explanations

We further examine the completeness of the explanations. As shown in Section 4, some explanation methods can not calculate meaningful relevance values for all inputs. In particular, perturbation-based methods suffer from this problem, since they determine a regression with labels derived from random perturbations. To investigate this problem, we monitor the creation of perturbations and their labels for the different datasets.

When creating perturbations for some sample x it is essential for black-box methods that a fraction p of them is classified as belonging to the opposite class of x. In an optimal case one can achieve $p \approx 0.5$, however during our experiments we find that 5 % can be sufficient to calculate a non-degenerated explanation in some cases. Figure 6 shows for each value of p and all datasets the fraction of samples remaining when enforcing a percentage p of perturbations from the opposite class.

In general, we observe that creating malicious perturbations from benign samples is a hard problem, especially for Drebin+ and DAMD. For example, in the Drebin+ dataset only 31 % of the benign samples can obtain a p value of 5 % which means that more than 65 % of the whole dataset suffer from degenerated explanations. A detailed calculation for all datasets with a p value of 5 % can be found in Table 12 in the Appendix C.

The problem of incomplete explanations is rooted in the imbalance of features characterizing malicious and benign data in the datasets. While only few features make a sample malicious, there exists a large variety of features turning a sample benign. As a consequence, randomly setting malicious features to zero leads to a benign classification, while setting benign features to zero usually does not impact the prediction. As a consequence, it is often not possible to explain predictions for benign applications and the analyst is stuck with an empty explanation.

In summary, we argue that perturbation-based explanation methods should only be used in security settings where incomplete explanations can be compensated by other means. In all other cases, one should refrain from using these black-box methods in the context of security.

5.5. Stability of Explanations

We proceed to evaluate the stability of the explanation methods when processing inputs from the four security systems. To this end, we apply the explanations to the same samples over multiple runs and measure the average intersection size between the runs as explained in Section 4.

Table 8 shows the average intersection size between the top k features for three runs of the methods as defined in Section 4. We use $k = 10$ for all datasets except for DAMD

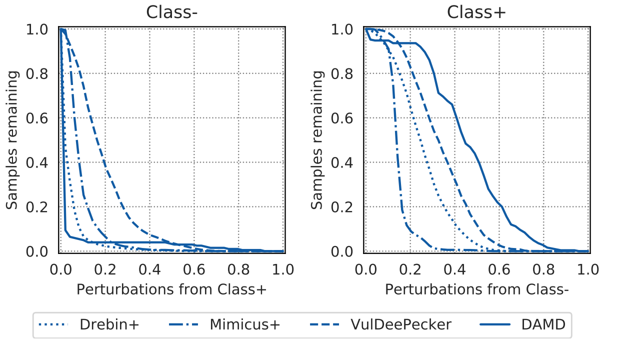


Figure 6: Perturbation label statistics of the datasets. For each percentage of perturbations from the other class the percentage of samples achieving this number is shown.

where we use $k = 50$ due to the larger input space. Since the outputs of Gradients, IG, and LRP are deterministic, they reach the perfect score of 1.0 in all settings and thus do not suffer from limitations concerning stability.

For the perturbation-based methods, however, stability poses a severe problem since none of those methods obtains a intersection size of more than 0.5. This indicates that on average half of the top features do not overlap when computing explanations on the same input. Furthermore, we see that the assumption of *locality* of the perturbation-based methods does not apply for all models under test, since the output is highly dependent on the perturbations used to approximate the decision boundary. Therefore, the best methods for the stability criterion beat the perturbation-based methods by a factor of at least 2.5 on all datasets.

TABLE 8: Average intersection size between top features for multiple runs. Values close to one indicate greater stability.

Method	**Drebin+**	**Mimicus+**	**DAMD**	**VulDeePecker**
LIME	0.391	0.454	0.017	0.411
LEMNA	0.431	0.340	0.047	0.405
SHAP	0.310	0.320	0.148	0.477
Gradients	1.000	1.000	1.000	1.000
IG	1.000	1.000	1.000	1.000
LRP	1.000	1.000	1.000	1.000

5.6. Efficiency of Explanations

We finally examine the efficiency of the different explanation methods. Our experiments are performed on a regular server system with an `Intel Xeon E5 v3` CPU at 2.6 GHz. It is noteworthy that the methods Gradients, IG and LRP can benefit from computations on a graphical processing unit (GPU), yet we consider only CPU results in this study to achieve a fair comparison between all explanation methods.

Table 9 shows the average run-time per input for all explanations methods and security systems. We observe that on all datasets Gradients is at least an order of magnitude faster than the other methods. This advantage arises from the fact that data can be processed *batch-wise* for methods like Gradients, IG, and LRP, that is, explanations can be calculated for a set of samples at the same time. The Mimicus+ dataset, for example, can be processed in one batch resulting in a speed-up factor of more than 400× over the fastest black-box method.

TABLE 9: Run-time per sample. Note the different time units ranging from milliseconds (ms) to seconds (s).

Method	**Drebin+**	**Mimicus+**	**DAMD**	**VulDeePecker**
LIME	30.54 ms	28.32 ms	736.21 ms	30.04 ms
LEMNA	4.59 s	2.56 s	685.90 s	6.14 s
SHAP	9.05 s	424.80 ms	44.55 s	4.95 s
Gradients	4.43 ms	0.07 ms	26.77 ms	1.21 ms
IG	134.00 ms	0.79 ms	26.70 ms	1.01 ms
LRP	4.34 ms	0.08 ms	15.29 ms	254.55 ms

The run-time of the black-box methods increases for high dimensional datasets, especially DAMD, since the regression problems need to be solved in higher dimensions. While the speed-up factors are already enormous, we have not even included the creation of perturbations and their classification, which obviously consume additional run-time as well.

5.7. Robustness of Explanations

Recently, multiple authors showed that adversarial perturbations are also applicable against explanation methods and thus, can manipulate the generated relevance values. Zhang et al. [53] proposed an adversarial attack such that a crafted input $\tilde{x}$ is misclassified by the network but keeps an explanation very close to the one of x. However, they also show that the methods Gradients, IG, and LRP are robust against their attack. Ghorbani et al. [19] show that indistinguishable adversarial samples $\tilde{x}$ can be crafted in such a way that they receive an explanation with a large deviation from the original one and methods like IG and Gradients are vulnerable to this attack. Dombrowski et al. [14] even extend this attack and show that Gradients, LRP, and IG can be tricked such that they produce arbitrary explanations while keeping the input image almost unchanged.

While the aforementioned attacks are constructed for white-box methods only Slack et al. [45] were the first to create an attack against LIME and SHAP. They show that an attacker who controls the perturbations can make the explanations deviate largely from the true behavior of the classifier. Unfortunatly, LEMNA is not considered in [45] and creating an attack for it is beyond the scope of this work however, we conclude that the majority of explanation methods is vulnerable to adversarial attacks and should thus not be used in an unsafe environment.

5.8. Summary

A strong explanation method is expected to achieve good results for each criterion and on each dataset. For example, we have seen that the Gradients method computes sparse results in a decent amount of time. The features, however, are not accurate on the DAMD and VulDeePecker dataset. Equally, the relevance values of SHAP for the Drebin+ dataset are sparser than those from LEMNA but suffer from instability. To provide an overview, we average the performance of all methods over the four datasets and summarize the results in Table 10.

TABLE 10: Results of the evaluated explanation methods. The last column summarizes these metrics in a rating comprising three levels: strong(●), medium (◐), and weak (○).

Explanation Method	Accuracy	Sparsity	Completeness	Stability	Efficiency	Robustness	Overall Rating
LIME	0.582	0.772	–	0.318	2.06×10^{-1} s	○	◐ ◐ ○ ○ ◐ ○
LEMNA	0.702	0.612	–	0.306	1.75×10^{2} s	–	○ ○ ○ ○ ○ –
SHAP	0.823	0.757	–	0.314	1.47×10^{1} s	○	○ ◐ ○ ○ ○ ○
Gradients	0.600	0.867	✓	1.000	1.26×10^{-2} s	○	◐ ● ● ● ● ○
IG	0.431	0.886	✓	1.000	4.06×10^{-2} s	○	● ● ● ● ● ○
LRP	0.454	0.873	✓	1.000	6.92×10^{-2} s	○	● ● ● ● ● ○

For each of the six evaluation criteria, we assign each method one of the following three categories: ●, ◐, and ○. The ● category is given to the best explanation method and other methods that provide a similar performance. The ○ category is assigned to the worst method and methods performing equally bad. Finally, the ◐ category is given to methods that lie between the best and worst methods and thus provide medium results.

Based on Table 10, we can see that white-box explanation methods achieve a better ranking than black-box methods in all evaluation criteria. Due to the direct access to the parameters of the neural network, these methods can better analyze the prediction function and are able to identify relevant features. In particular, IG and LRP are the best methods overall regarding our evaluation criteria. They compute results in less than 70 ms in our benchmark, mark only few features as relevant, and the selected features have great impact on the decision of the classifier. As a result, we generally recommend to use these methods for explaining deep learning in security.

However, if white-box access is not available, we recommend the black-box method LIME as it has higher descriptive accuracy, creates sparser solutions, and is faster in creating solutions than other black-box methods. When working in a black-box environment the user should keep in mind that samples exist that simply cannot be explained well (incompleteness) and possibly multiple iterations are necessary to derive correct results (instability). In case of a commercial product, white-box explanations provided by the vendor are always preferred over external explanations, which require to create a surrogate with black-box models.

6. Insights on the datasets

During the experiments for this paper, we have analyzed various explanations of security systems—not only quantitatively as discussed in Section 5 but also qualitatively from the perspective of a security analyst. In this section, we summarize our observations and discuss insights related to the role of deep learning in security.

6.1. Insights on Mimicus+

When inspecting explanations for the Mimicus+ system, we observe that the features for detecting malware are dominated by `count_javascript` and `count_js`, which both stand for the number of JavaScript elements in the document. The strong impact of these elements is meaningful, as JavaScript is frequently used in malicious PDF documents [28]. However, we also identify features in the explanations that are non-intuitive. For example, features like `count_trailer` that measures the number of trailer sections in the document or `count_box_letter` that counts the number of US letter sized boxes can hardly be related to security and rather constitute artifacts in the dataset captured by the learning process.

To further investigate the impact of JavaScript features on the neural network, we determine the distribution of the top 5 features from the method IG for each class in the entire dataset. It turns out that JavaScript appears in 88 % of the malicious documents, whereas only about 6 % of the benign samples make use of it (see Table 11). This makes JavaScript an extremely discriminating feature for the dataset. From a security perspective, this is an unsatisfying result, as the neural network of Mimicus+ relies on a few indicators for detecting the malicious code in the documents. An attacker could potentially evade Mimicus+ by not using JavaScript or obfuscating the JavaScript elements in the document.

TABLE 11: Top-5 features for the Mimicus+ dataset determined using IG. The right columns show the frequency in benign and malicious PDF documents, respectively.

Class	Top 5 Feature	Benign	Malicious
–	`count_font`	98.4 %	20.8 %
–	`producer_mismatch`	97.5 %	16.6 %
–	`title_num`	68.6 %	4.8 %
–	`pdfid1_num`	81.5 %	2.8 %
–	`title_uc`	68.6 %	4.8 %
–	`pos_eof_min`	100.0 %	93.4 %
+	`count_javascript`	6.0 %	88.0 %
+	`count_js`	5.2 %	83.4 %
+	`count_trailer`	89.3 %	97.7 %
+	`pos_page_avg`	100.0 %	100.0 %
+	`count_endobj`	100.0 %	99.6 %
+	`createdate_tz`	85.5 %	99.9 %
+	`count_action`	16.4 %	73.8 %

6.2. Insights on Drebin+

During the analysis of the Drebin+ dataset, we notice that several benign applications are characterized by the hardware feature `touchscreen`, the intent filter `launcher`, and the permission `INTERNET`. These features frequently occur in benign and malicious applications in the Drebin+ dataset and are not particularly descriptive for benignity. Note that the interpretation of features speaking for benign applications is challenging due to the broader scope and the difficulty in defining benignity. We conclude that the three features together form an artifact in the dataset that provides an indicator for detecting benign applications.

For malicious Android applications, the situation is different: The explanation methods return highly relevant features that can be linked to the functionality of the malware. For instance, the requested permission `SEND_SMS` or features related to accessing sensitive information, such as the permission `READ_PHONE_STATE` and the API call `getSimCountryIso`, receive consistently high scores in our investigartion. These features are well in line with common malware for Android, such as the *FakeInstaller* family [33], which is known to obtain money from victims by secretly sending text messages (SMS) to premium services. Our analysis shows that the MLP network employed in Drebin+ has captured indicative features directly related to the underlying malicious activities.

6.3. Insights on VulDeePecker

In contrast to the datasets considered before, the features processed by VulDeePecker resemble lexical tokens and are strongly interconnected on a syntactical level. This becomes apparent in the explanations of the method Integrated Gradients in Figure 4, where adjacent tokens have mostly equal colors. Moreover, orange and blue colored features in the explanation are often separated by tokens with no color, indicating a gradual separation of positive and negative relevance values.

During our analysis, we notice that it is still difficult for a human analyst to benefit from the highlighted tokens. First, an analyst interprets the source code rather than the extracted tokens and thus maintains a different view on the data. In Figure 4, for example, the interpretation of the highlighted `INT0` and `INT1` tokens as buffer sizes of 50 and 100 wide characters is misleading, since the neural network is not aware of this relation. Second, VulDeePecker truncates essential parts of the code. In Figure 4, during the initialization of the destination buffer, for instance, only the size remains as part of the input. Third, the large amount of highlighted tokens like semicolons, brackets, and equality signs seems to indicate that VulDeePecker overfits to the training data at hand.

Given the truncated program slices and the seemingly unrelated tokens marked as relevant, we conclude that the VulDeePecker system might benefit from extending the learning strategy to longer sequences and cleansing the training data to remove artifacts that are irrelevant for vulnerability discovery.

6.4. Insights on DAMD

As the final subject, we consider Android applications from the DAMD dataset. Due to the difficulty of analyzing raw Dalvik bytecode, we guide our analysis of the dataset by inspecting malicious applications from three popular Android malware families: GoldDream [27], DroidKungFu [26], and DroidDream [18]. These families exfiltrate sensitive data and run exploits to take full control of the device.

In our analysis of the Dalvik bytecode, we benefit from the sparsity of the explanations from LRP and IG as explained in Section 5. Analyzing all relevant features becomes tractable with moderate effort using these methods and we are able to investigate the opcodes with the highest relevance in detail. We observe that the relevant opcode sequences are directly linked to the malicious functionality of the three malware families.

As an example, Table 4 depicts the opcode sequence, that is found in all samples of the GoldDream family[2]. Taking a closer look, this sequence occurs in the `onReceive` method of the `com.GoldDream.zj.zjReceiver` class. In this function, the malware intercepts incoming SMS and phone calls and stores the information in local files before sending them to an external server. Similarly, we can interpret the explanations of the other two malware families, where functionality related to exploits and persistent installation is highlighted in the Dalvik opcode sequences.

For all members of each malware family, the opcode sequences identified using the explanation methods LRP and IG are identical, which demonstrates that the CNN in the DAMD system has learned an discriminative pattern from the underlying opcode representation.

7. Conclusion

The increasing application of deep learning in security renders means for explaining their decisions vitally important. While there exist a wide range of explanation methods from the area of computer vision and machine learning, it has been unclear which of these methods are suitable for security systems. We have addressed this problem and propose evaluation criteria that enable a practitioner to compare and select explanation methods in the context of security. While the importance of these criteria depends on the particular security task, we find that the methods Integrated Gradients and LRP comply best with all requirements. Hence, we generally recommend these methods for explaining predictions in security systems.

Aside from our evaluation of explanation methods, we reveal problems in the general application of deep learning in security. For all considered systems under test, we identify artifacts that substantially contribute to the overall prediction, but are unrelated to the security task. Several of these artifacts are rooted in peculiarities of the data. It is likely that the employed neural networks overfit the data rather than solving the underlying task. We thus conclude that explanations need to become an integral part of any deep learning system to identify artifacts in the training data and to keep the learning focused on the targeted security problem.

Our study is a first step for integrating explainable learning in security systems. We hope to foster a series of research that applies and extends explanation methods, such that deep learning becomes more transparent in computer security. To support this development, we make all our implementations and datasets publicly available. This includes the four re-implemented security systems, the six explanation methods, and all generated explanations.

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

[2] e.g., MD5: `a4184a7fcaca52696f3d1c6cf8ce3785`

networks! Technical Report abs/1808.04260, Computing Research Repository (CoRR), 2018.

[2] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *International Conference on Learning Representations, ICLR*, 2018.

[3] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck. Drebin: Efficient and explainable detection of Android malware in your pocket. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, Feb. 2014.

[4] L. Arras, F. Horn, G. Montavon, K.-R. Müller, and W. Samek. "what is relevant in a text document?": An interpretable machine learning approach. *PLoS ONE*, 12(8), Aug. 2017.

[5] 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. *PLoS ONE*, 10(7), July 2015.

[6] N. Carlini. Is AmI (attacks meet interpretability) robust to adversarial examples? Technical Report abs/1902.02322, Computing Research Repository (CoRR), 2019.

[7] N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 39–57, 2017.

[8] A. Chattopadhyay, A. Sarkar, P. Howlader, and V. N. Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018, Lake Tahoe, NV, USA, March 12-15, 2018*, pages 839–847, 2018.

[9] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. Technical Report abs/1606.04435, Computing Research Repository (CoRR), 2014.

[10] Z. L. Chua, S. Shen, P. Saxena, and Z. Liang. Neural nets can learn function type signatures from binaries. In *Proc. of the USENIX Security Symposium*, pages 99–116, 2017.

[11] P. Dabkowski and Y. Gal. Real time image saliency for black box classifiers. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 6967–6976. 2017.

[12] A. Datta, S. Sen, and Y. Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE Symposium on Security and Privacy*, pages 598–617, 2016.

[13] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 2016.

[14] A.-K. Dombrowski, M. Alber, C. J. Anders, M. Ackermann, K.-R. Müller, and P. Kessel. Explanations can be manipulated and geometry is to blame. In *Advances in Neural Information Proccessing Systems (NIPS)*, 2019.

[15] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley & Sons, second edition, 2000.

[16] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[17] R. C. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *IEEE International Conference on Computer Vision*, pages 3449–3457, 2017.

[18] J. Foremost. DroidDream mobile malware. https://www.virusbulletin.com/virusbulletin/2012/03/droiddream-mobile-malware, 2012. (Online; accessed 14-February-2019).

[19] A. Ghorbani, A. Abid, and J. Y. Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[21] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel. Adversarial examples for malware detection. In *Proc. of the European Symposium on Research in Computer Security (ESORICS)*, pages 62–79, 2017.

[22] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing. LEMNA: Explaining deep learning based security applications. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 364–379, 2018.

[23] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[24] W. Huang and J. W. Stokes. MtNet: A multi-task neural network for dynamic malware classification. In *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 399–418, 2016.

[25] P. jan Kindermans, K. T. Schütt, M. Alber, K.-R. Müller, D. Erhan, B. Kim, and S. Dähne. Learning how to explain neural networks: Patternnet and patternattribution. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2018.

[26] X. Jiang. Security Alert: New sophisticated Android malware DroidKungFu found in alternative chinese App markets. https://www.csc2.ncsu.edu/faculty/xjiang4/DroidKungFu.html, 2011. (Online; accessed 14-February-2019).

[27] X. Jiang. Security Alert: New Android malware GoldDream found in alternative app markets. https://www.csc2.ncsu.edu/faculty/xjiang4/GoldDream/, 2011. (Online; accessed 14-February-2019).

[28] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna. Revolver: An automated approach to the detection of evasive web-based malware. In *Proc. of the USENIX Security Symposium*, pages 637–651, Aug. 2013.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Proccessing Systems (NIPS)*. Curran Associates, Inc., 2012.

[30] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In *The Handbook of Brain Theory and Neural Networks*. MIT, 1995.

[31] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong. Vuldeepecker: A deep learning-based system for vulnerability detection. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[32] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 4765–4774. 2017.

[33] McAfee. Android/FakeInstaller.L. https://home.mcafee.com/virusinfo/, 2012. (Online; accessed 1-August-2018).

[34] N. McLaughlin, J. M. del RincÃşn, B. Kang, S. Y. Yerima, P. C. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. DoupÃľ, and G.-J. Ahn. Deep android malware detection. In *Proc. of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 301–308, 2017.

[35] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *Proc. of the International Conference on Learning Representations (ICLR Workshop)*, 2013.

[36] N. Papernot, P. D. McDaniel, A. Sinha, and M. P. Wellman. Sok: Security and privacy in machine learning. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414, 2018.

[37] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proc. of the ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, 2016.

[38] R. Rojas. *Neural Networks: A Systematic Approach*. Springer-Verlag, Berlin, Deutschland, 1996. ISBN 3-450-60505-3.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(Foundation), 1986.

[40] 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 *The IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, Oct 2017.

[41] L. Shapley. A value for n-person games. 1953.

[42] E. C. R. Shin, D. Song, and R. Moazzezi. Recognizing functions in binaries with neural networks. In *Proc. of the USENIX Security Symposium*, pages 611–626, 2015.

[43] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 3145–3153, 2017.

[44] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2014.

[45] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *AAAI/ACM Conference on Artificial Intelligence , Ethics, and Society (AIES)*, 2019.

[46] C. Smutz and A. Stavrou. Malicious PDF detection using metadata and structural features. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, pages 239–248, 2012.

[47] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015.

[48] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3319–3328, 2017.

[49] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 3104–3112, 2014.

[50] N. Šrndić and P. Laskov. Practical evasion of a learning-based classifier: A case study. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 197–211, 2014.

[51] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 363–376, 2017.

[52] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014*, pages 818–833. Springer International Publishing, 2014.

[53] X. Zhang, N. Wang, H. Shen, S. Ji, X. Luo, and T. Wang. Interpretable deep learning under fire. In *Proc. of USENIX Security Symposium*, 2019.

[54] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, 2016.

[55] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 95–109, 2012.

Appendix

1. Related Concepts

Some of the considered explanations methods share similarities with techniques of adversarial examples and feature selection. While these similarities result from an analogous analysis of the prediction function f_N, the underlying objectives are fundamentally different from explainable learning and cannot be transferred easily. In the following, we briefly highlight these different objectives:

Adversarial examples. Adversarial examples are constructed by determining a minimal perturbation δ such that $f_N(x + \delta) \neq f_N(x)$ for a given neural network N and an input vector x [7, 36]. The perturbation δ encodes which features need to be *modified* to change the prediction. However, the perturbation does not explain *why* x was given the label y by the neural network. The Gradients explanation method described in Section 3 shares similarities with some attacks generating adversarial examples, as the gradient $\partial f_N / \partial x_i$ is used to quantify the difference of f_N when changing a feature x_i slightly. Still, algorithms for determining adversarial examples are insufficient for computing reasonable explanations.

Note that we deliberately do not study adversarial examples in this paper. Techniques for attacking and defending learning algorithms are orthogonal to our work. These techniques can be augmented using explanations, yet it is completely open how this can be done in a secure manner. Recent defenses for adversarial examples based on explanations have proven to be totally ineffective [6].

Feature selection. This concept aims at reducing the dimensionality of a learning problem by selecting a subset of discriminative features [15]. At a first glance, features determined through feature selection seem like a good fit for explanation. While the selected features can be investigated and often capture characteristics of the underlying data, they are determined independent from a particular learning model. As a result, feature selection methods cannot be direclty applied for explaining the decision of a neural network.

2. Incompatible Explanation Methods

Several explanation methods are not suitable for general application in security, as they do not support common architectures of neural networks used in this area (see Table 1). We do not consider these methods in our evaluation, yet for completeness we provide a short overview of these methods in the following.

PatternNet and PatternAttribution. These white-box methods are inspired by the explanation of linear models. While PatternNet determines gradients and replaces neural network weights by so-called *informative directions*, PatternAttribution builds on the LRP framework and computes explanations relative to so-called root points whose output are 0. Both approaches are restricted to feed-forward and convolutional networks. Recurrent neural networks are not supported.

DeConvNet and GuidedBackProp. These methods aim at reconstructing an input x given output y, that is, mapping y back to the input space. To this end, the authors present an approach to revert the computations of a convolutional layer followed by a rectified linear unit (ReLu) and max-pooling, which is the essential sequence of layers in neural networks for image classification. Similar to LRP and DeepLift, both methods perform a backwards pass through the network. The major drawback of these methods is again the restriction to convolutional neural networks.

CAM, GradCAM, and GradCAM++. These three white-box methods compute relevance scores by accessing the output of the last convolutional layer in a CNN and performing global average pooling. Given the activations a_{ki} of the k-th channel at unit i, GradCam learn weights w_k such that

$$y \approx \sum_i \sum_k w_k a_{ki}.$$

That is, the classification is modeled as a linear combination of the activations of the last layer of all channels and finally $r_i = \sum_k w_k a_{ki}$. GradCam and GradCam++ extend this approach by including specific gradients in this calculation. All three methods are only applicable if the neural network uses a convolutional layer as the final layer. While this setting is common in image recognition, it is rarely used in security applications and thus we do not analyze these methods.

RTIS and MASK. These methods compute relevance scores by solving an optimization problem for a *mask* m. A mask m is applied to x as $m \circ x$ in order to affect x, for example by setting features to zero. To this end, Fong and Vedaldi [17] propose the optimization problem

$$m^* = \underset{m \in [0,1]^d}{\arg\min} \lambda \|\mathbf{1} - m\|_1 + f_N(m \circ x),$$

which determines a sparse mask, that identifies relevant features of x. This can be solved using gradient descent, which thus makes these white-box approaches. However, solving the equation above often leads to noisy results which is why RTIS and MASK add additional terms to achieve smooth solutions using regularization and blurring. These concepts, however, are only applicable for images and cannot be transferred to other types of features.

Quantitative Input Influence. This method is another black-box approach which calculates relevances by changing input features and calculating the difference between the outcomes. Let $X_{-i}U_i$ be the random variable with the ith input of X being replaced by a random value that is drawn from the distribution of feature x_i. Then the relevance of feature i for a classification to class c is given by

$$r_i = \mathbb{E}\big[f_N(X_{-i}U_i) \neq c | X = x\big].$$

However, when the features of X are binary like in some of our datasets this equation becomes

$$r_i = \begin{cases} 1 & f_N(x_{\neg i}) \neq c \\ 0 & \text{else} \end{cases}$$

As noted by Datta et al. [12] this results in many features receiving a relevance of zero which has no meaning. We notice that even the extension to sets proposed in [12] does not solve this problem since it is highly related to degenerated explanations as discussed in Section 5.

3. Completeness of Datasets: Example calculation

In Section 5 we discussed the problem of incomplete or degenerated explanations from black-box methods that can occur when there are not enough labels from the opposite class in the perturbations. Here we give an concrete example when enforcing 5 % of the labels to be from the opposite class.

Table 12 shows the results of this experiment. On average, 29 % of the samples cannot be explained well, as the computed perturbations contain too few instances from the opposite class. In particular, we observe that creating malicious perturbations from benign samples is a hard problem in the case of Drebin+ and DAMD, where only 32.6 % and 2.8 % of the benign samples achieve sufficient perturbations from the opposite class.

TABLE 12: Incomplete explanations of black-box methods. First two columns: Samples remaining when enforcing at least 5 % perturbations of opposite class.

System	**Class-**	**Class+**	**Incomplete**
Drebin+	24.2 %	97.1 %	66.3 %
Mimicus+	73.5 %	98.9 %	14.2 %
VulDeePecker	90.5 %	99.8 %	7.1 %
DAMD	5.9 %	94.8 %	44.9 %
Average	48.3 %	97.7 %	33.15 %