DOI:10.1145/3643456

To view the accompanying Technical Perspective,
visit doi.acm.org/10.1145/3655635

Pitfalls in Machine Learning for Computer Security

By Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck

Abstract

With the growing processing power of computing systems and the increasing availability of massive datasets, machine-learning (ML) algorithms have led to major breakthroughs in many different areas. This development has influenced computer security, spawning a series of work on learning-based security systems, such as for malware detection, vulnerability discovery, and binary code analysis. Despite great potential, ML in security is prone to subtle pitfalls that undermine its performance and render learning-based systems potentially unsuitable for security tasks and practical deployment.

In this paper, we look at this problem with critical eyes. First, we identify common pitfalls in the design, implementation, and evaluation of learning-based security systems. We conduct a study of 30 papers from top-tier security conferences within the past 10 years, confirming that these pitfalls are widespread in the current security literature. In an empirical analysis, we further demonstrate how individual pitfalls can lead to unrealistic performance and interpretations, obstructing the understanding of the security problem at hand. As a remedy, we propose actionable recommendations to support researchers in avoiding or mitigating the pitfalls where possible. Furthermore, we identify open problems when applying ML in security and provide directions for further research.

1. INTRODUCTION

No day goes by without reading machine-learning success stories. The widespread access to specialized computational resources and large datasets, along with novel concepts and architectures for deep learning, have paved the way for ML breakthroughs in several areas, such as the translation of natural languages[22] and the recognition of image content.[14] This development has naturally influenced security research: Although mostly confined to specific applications in the past, ML has become one of the key enablers to studying and addressing security-relevant problems at large in several application domains, including intrusion detection,[17] malware analysis,[11] vulnerability discovery,[25] and binary code analysis.[20]

The original version of this paper, "Dos and Don'ts of Machine Learning in Computer Security"[3] was published in *Proceedings of The USENIX Security Symposium* (2022).

Machine learning, however, has no clairvoyant abilities and requires reasoning about statistical properties of data across a fairly delicate workflow: Incorrect assumptions and experimental biases may cast doubts on this process to the extent that it becomes unclear whether we can trust scientific discoveries made using learning algorithms at all. Attempts to identify such challenges and limitations in specific security domains, such as network intrusion detection, started two decades ago[5] and were extended more recently to other domains.[12,18] Orthogonal to this line of work, however, we argue that there exist *generic pitfalls* related to machine learning that affect all security domains and have received little attention so far.

These pitfalls can lead to over-optimistic results and, even worse, affect the entire ML workflow, weakening assumptions, conclusions, and lessons learned. As a consequence, a false sense of achievement is felt that hinders the adoption of research advances in academia and industry. A sound scientific methodology is fundamental to support intuitions and draw conclusions. We argue that this need is especially relevant in security, where processes are often undermined by adversaries that actively aim to bypass analysis and break systems.

In this paper, we identify 10 common—yet subtle—pitfalls that pose a threat to validity and hinder interpretation of research results. To support this claim, we analyze the prevalence of these pitfalls in 30 top-tier security papers from the past decade that rely on ML for tackling different problems. To our surprise, each paper suffers from at least three pitfalls; even worse, several pitfalls affect most of the papers, which shows how endemic and subtle the problem is. Although the pitfalls are widespread, it is perhaps more important to understand the extent to which they weaken results and lead to overoptimistic conclusions. To this end, we perform an impact analysis of the pitfalls in four different security fields. The findings support our premise echoing the broader concerns of the community.

In summary, we make the following contributions:

1. **Pitfall Identification.** We identify 10 pitfalls as *don'ts* of ML in security and propose *dos* as actionable recommendations to support researchers in avoiding the pitfalls where possible. Furthermore, we identify open problems that cannot be mitigated easily and require further research effort (§2).
2. **Prevalence Analysis.** We analyze the prevalence of the identified pitfalls in 30 representative top-tier security

papers published in the past decade. Additionally, we perform a broad survey in which we obtain and evaluate the feedback of the authors of these papers regarding the identified pitfalls (§3).

3. Impact Analysis. In four different security domains, we experimentally analyze the extent to which such pitfalls introduce experimental bias, and how we can effectively overcome these problems by applying the proposed recommendations (§4).

REMARK. *This work should not be interpreted as a finger-pointing exercise. On the contrary, it is a reflective effort that shows how subtle pitfalls can have a negative impact on progress of security research, and how we—as a community—can mitigate them adequately.*

2. PITFALLS IN MACHINE LEARNING

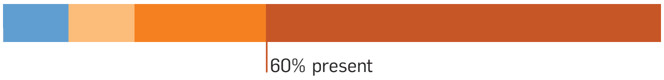
Despite its great success, the application of ML in practice is often non-trivial and prone to several pitfalls, ranging from obvious flaws to minor blemishes. Overlooking these issues may result in experimental bias or incorrect conclusions. In this section, we present 10 common pitfalls that occur frequently in security research. Although some of these pitfalls may seem obvious at first glance, they are rooted in subtle deficiencies that are widespread in security research—even in papers presented at top conferences (see §3 and §4).

We group these pitfalls with respect to the stages of a typical machine-learning workflow, as depicted in Figure 1. For each pitfall, we provide a short description and a discussion of their security implications. To visualize the prevalence of a pitfall, we provide a colored bar depicting the proportion of papers in our analysis that suffer from the pitfall, with warmer colors indicating the presence of the pitfall. Interested readers find *recommendations* on how to avoid these pitfalls in the original publication.[3]

2.1. Data collection and labeling.

The design and development of learning-based systems usually starts with the acquisition of a representative dataset. It is clear that conducting experiments using unrealistic data leads to the misestimation of an approach's capabilities. The following two pitfalls frequently induce this problem and thus require special attention when developing learning-based systems in computer security.

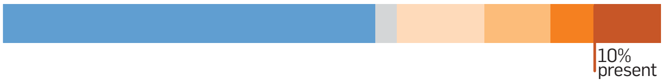
P1 – Sampling Bias.
The collected data does not sufficiently represent the true data distribution of the underlying security problem.



Description. With a few rare exceptions, researchers develop learning-based approaches without exact knowledge of the true underlying distribution of the input space. Instead, they need to rely on a dataset containing a fixed number of samples that aim to resemble the actual distribution. While it is inevitable that some bias exists in most cases, understanding the specific bias inherent to a particular problem is crucial to limiting its impact in practice. Drawing meaningful conclusions from the training data becomes challenging if the data does not effectively represent the input space or even follows a different distribution.

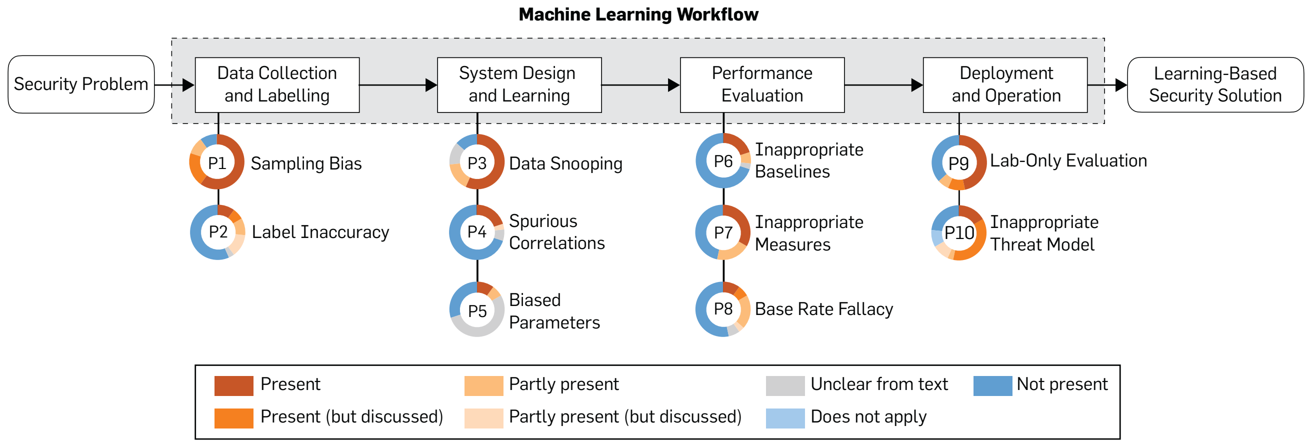
Security implications. Sampling bias is highly relevant to security, as the data acquisition is particularly challenging and often requires using multiple sources of varying quality. As an example, for the collection of suitable datasets for Android malware detection, only a few public sources exist from which to obtain such data.[2,4] As a result, it is common practice to rely on synthetic data or to combine data from different sources, both of which can introduce bias—as we demonstrate in §4 with examples on state-of-the-art methods for intrusion and malware detection.

P2 – Label Inaccuracy.
Ground-truth labels required for classification tasks are inaccurate, unstable, or erroneous, affecting the overall performance of a learning-based system.



Description. Many learning-based security systems are built for classification tasks. To train these systems, a

Figure 1. Common pitfalls of machine learning in computer security.



ground-truth label is required for each observation. Unfortunately, this labeling is rarely perfect and researchers must account for uncertainty and noise to prevent their models from suffering from inherent bias.

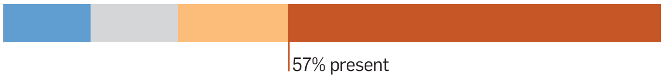
Security implications. For many relevant security problems, such as detecting network attacks or malware, reliable labels are typically not available, resulting in a chicken-and-egg problem. As a remedy, researchers often resort to heuristics, such as using external sources that do not provide a reliable ground-truth. For example, services such as VirusTotal[a] are commonly used for acquiring label information for malware, but these are not always consistent.

2.2. System design and learning.

Once enough data has been collected, a learning-based security system can be trained. This process ranges from data preprocessing to extracting meaningful features and building an effective learning model. Unfortunately, one can introduce flaws and weak spots at each of these steps.

P3 – Data Snooping.

A learning model is trained with data that is typically not available in practice. Data snooping can occur in many ways, some of which are very subtle and hard to identify.

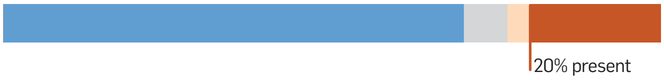


Description. It is common practice to split collected data into separate training and test sets prior to generating a learning model. Although splitting the data seems straightforward, there are many subtle ways in which test data or other background information that is not usually available can affect the training process, leading to data snooping. We broadly distinguish between three data snooping types described in the original paper.[3]

Security implications. In security, data distributions are often non-stationary and continuously changing due to new attacks or technologies. Because of this, snooping on data from the future or from external data sources is a prevalent pitfall that leads to over-optimistic results. For instance, researchers have identified data snooping in learning-based malware detection systems.[18] In this case, the capabilities of the methods are overestimated due to mixing samples from past and present.

P4 – Spurious Correlations.

Artifacts unrelated to the security problem create shortcut patterns for separating classes. Consequently, the learning model adapts to these artifacts instead of solving the actual task.

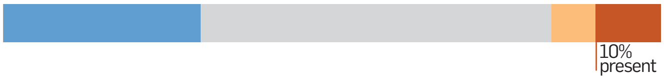


Description. Spurious correlations result from artifacts that correlate with the task to solve but are not actually related to it, leading to false associations. Consider the example of a network intrusion detection system, where a large fraction of the attacks in the dataset originate from a certain network region. The model may learn to detect a specific IP range instead of generic attack patterns.

Security implications. Machine learning is typically applied as a black box in security. As a result, spurious correlations often remain unidentified. These correlations pose a problem once results are interpreted and used for drawing general conclusions. Without knowledge of spurious correlations, there is a high risk of overestimating the capabilities of an approach and misjudging its practical limitations. As an example, §4.2 reports our analysis on a vulnerability discovery system indicating the presence of notable spurious correlations in the underlying data.

P5 – Biased Parameter Selection.

The final parameters of a learning-based method are not entirely fixed at training time. Instead, they indirectly depend on the test set.



Description. Throughout the learning procedure, it is common practice to generate different models by varying hyperparameters. The best-performing model is picked and its performance on the test set is presented. While this setup is generally sound, it can still suffer from a biased parameter selection. For example, over-optimistic results can be easily produced by calibrating the detection threshold on the test data instead of the training data.

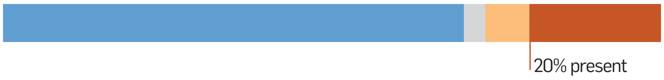
Security implications. A security system whose parameters have not been fully calibrated at training time can perform very differently in a realistic setting. While the detection performance of a network intrusion detection system may be assessed using a receiver operating characteristic (ROC) curve obtained on the test set, it can be hard to select the same operational point in practice due to the diversity of real-world traffic.[21] This may lead to decreased performance of the system in comparison to the original experimental setting. Note that this pitfall is related to data snooping (P3), but should be considered explicitly as it can easily lead to inflated results.

2.3. Performance evaluation.

The next stage in a typical machine-learning workflow is the evaluation of the system's performance. In the following, we show how different pitfalls can lead to unfair comparisons and biased results in the evaluation of such systems.

P6 – Inappropriate Baseline.

The evaluation is conducted without, or with limited, baseline methods. As a result, it is impossible to demonstrate improvements against the state of the art and other security mechanisms.

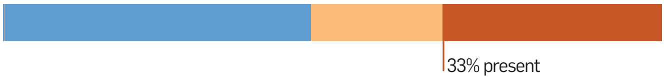


Description. To show to what extent a novel method improves the state of the art, it is vital to compare it with previously proposed methods. When choosing baselines, it is important to remember that there exists no universal learning algorithm that outperforms all other approaches in general.[24] Consequently, providing only results for the proposed approach or a comparison with mostly identical learning models does not give enough context to assess its impact.

Security implications. An overly complex learning method increases the chances of overfitting, and also the run-time overhead, the attack surface, and the time and costs for deployment. To show that ML techniques provide significant improvements compared to traditional methods, it is thus essential to compare these systems side by side.

While these automated methods can certainly not replace experienced data analysts, they can be used to set the lower bar the proposed approach should aim for. Finally, it is critical to check whether non-learning approaches are also suitable for the application scenario. For example, for intrusion and malware detection, there exist a wide range of methods using other detection strategies.

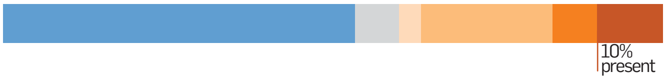
P7 – Inappropriate Performance Measures.
The chosen performance measures do not account for the constraints of the application scenario, such as imbalanced data or the need to keep a low false-positive rate.



Description. A wide range of performance measures are available and not all of them are suitable in the context of security. For example, when evaluating a detection system, it is typically insufficient to report just a single performance value, such as accuracy, because true-positive and false-positive decisions are not observable. However, even more advanced measures may obscure experimental results in some application settings. Therefore, the selection of proper evaluation metrics is a challenging task that requires a thoughtful decision.

Security implications. Inappropriate performance measures are a long-standing problem in security research, particularly in detection tasks. While true and false positives, for instance, provide a more detailed picture of a system's performance, they can also disguise the actual precision when the prevalence of attacks is low.

P8 – Base Rate Fallacy.
A large class imbalance is ignored when interpreting the performance measures, leading to an overestimation of performance.



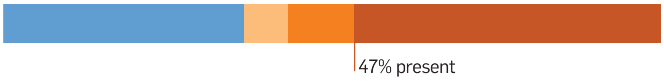
Description. Class imbalance can easily lead to a misinterpretation of performance if the base rate of the negative class is not considered. If this class is predominant, even a very low false-positive rate can result in surprisingly high numbers of false positives. Note the difference to the previous pitfall: while P7 refers to the inappropriate *description* of performance, the base-rate fallacy is about the misleading *interpretation* of results. This special case is easily overlooked in practice (see §3).

Security implications. The base rate fallacy is relevant in a variety of security problems, such as intrusion detection and website fingerprinting.[5,12] As a result, it is challenging to realistically quantify the security and privacy threat posed by attackers. Similarly, the probability of installing malware is usually much lower than is considered in experiments on malware detection.[18]

2.4. Deployment and operation.

In the last stage of a typical machine-learning workflow, the developed system is deployed to tackle the underlying security problem in practice.

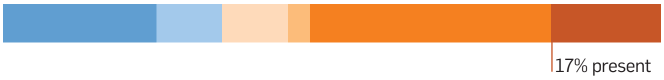
P9 – Lab-Only Evaluation.
A learning-based system is solely evaluated in a laboratory setting, without discussing its practical limitations.



Description. As in all empirical disciplines, it is common to perform experiments under certain assumptions to demonstrate a method's efficacy. While performing controlled experiments is a legitimate way to examine specific aspects of an approach, it should be evaluated in a realistic setting whenever possible to transparently assess its capabilities and showcase the open challenges that will foster further research.

Security implications. Many learning-based systems in security are evaluated solely in laboratory settings, overstating their practical impact. A common example are detection methods evaluated only in a *closed-world setting* with limited diversity and no consideration of non-stationarity. For example, a large number of website fingerprinting attacks are evaluated only in closed-world settings spanning a limited time period.[12] Similarly, several learning-based malware detection systems have been insufficiently examined under realistic settings.[18]

P10 – Inappropriate Threat Model.
The security of machine learning is not considered, exposing the system to a variety of attacks.



Description. Learning-based security systems operate in a hostile environment, which should be considered when designing these systems. Prior work in adversarial learning has revealed a considerable attack surface introduced by machine learning itself, at all stages of the workflow. Their broad attack surface makes these algorithms vulnerable to various types of attacks.[7]

Security implications. Including adversarial influence in the threat model and evaluation is often vital, as systems prone to attacks are not guaranteed to output trustworthy and meaningful results. Aside from traditional security issues, it is therefore essential to also consider ML-related attacks. For instance, an attacker may more easily evade a model that relies on only a few features than a properly regularized model designed with security considerations in mind, although one should also consider domain-specific implications.[19]

3. PREVALENCE ANALYSIS

Once we understand the pitfalls faced by learning-based security systems, it becomes necessary to assess their prevalence and investigate their impact on scientific advances. To this end, we conduct a study on 30 papers published in the last 10 years at the top four conferences for security-related research in our community. The papers have been selected as representative examples for our study, as they address a large variety of security topics and successfully apply ML to the corresponding research problems. A complete list of all selected papers can be found in the original paper.[3]

3.1. Review process.

Each paper is assigned two independent reviewers who assess the article and identify instances of the described pitfalls. The pool of reviewers consists of six researchers who have all previously published work on the topic of machine learning and security in at least one of the considered security conferences. Reviewers do *not* consider any material presented outside the papers under analysis (aside from appendices and associated artifacts, such as datasets or source code). Once both reviewers have completed their assignments, they discuss the paper in the presence of a third reviewer who may resolve any disputes. In case of uncertainty, the authors are given the benefit of the doubt (for example, in case of a dispute between *partly present* and *present*, we assign *partly present*).

Throughout the process, all reviewers meet regularly to discuss their findings and ensure consistency between the pitfalls' criteria. Moreover, these meetings have been used to refine the definitions and scope of pitfalls based on the reviewers' experience. Following any adaptation of the criteria, all completed reviews have been reevaluated by the original reviewers—this occurred twice during our analysis. While cumbersome, this adaptive process of incorporating reviewer feedback ensures the pitfalls are comprehensive in describing the core issues across the state of the art. We note that the inter-rater reliability of reviews prior to dispute resolution is $\alpha = 0.832$ using Krippendorff's alpha, where $\alpha > 0.800$ indicates confidently reliable ratings.[13]

3.2. Assessment criteria.

For each paper, pitfalls are coarsely classified as either *present*, *not present*, *unclear from text*, or *does not apply*. A pitfall may be wholly present throughout the experiments without remediation *(present)*, or it may not *(not present)*. If the authors have corrected any bias or have narrowed down their claims to accommodate the pitfall, this is also counted as *not present*. We also introduce *partly present* as a category to account for experiments that do suffer from a pitfall, but where the impact has been partially addressed. If a pitfall is *present* or *partly present* but acknowledged in the text, we moderate the classification as *discussed*. If the reviewers are unable to rule out the presence of a pitfall due to missing information, we mark the publication as *unclear from text*. Finally, in the special case of P10, if the pitfall *does not apply* to a paper's setting, this is considered as a separate category.

3.3. Observations.

The most prevalent pitfalls are sampling bias (P1) and data snooping (P3), which are at least partly present in 90% and 73% of the papers, respectively. In more than 50% of the papers, we identify inappropriate threat models (P10), lab-only evaluations (P9), and inappropriate performance measures (P7) as at least partly present. *Every* paper is affected by at least three pitfalls, underlining the pervasiveness of such issues in recent computer security research. In particular, we find that dataset collection is still very challenging: Some of the most realistic and expansive open datasets we have developed as a community are still imperfect (see §4.1).

Moreover, the presence of some pitfalls is more likely to be *unclear from the text* than others. We observe this for biased parameter selection (P5) when no description of the hyperparameters or tuning procedure is given; for spurious correlations (P4), when there is no attempt to explain a model's decisions; and for data snooping (P3), when the dataset splitting or normalization procedure is not explicitly described in the text. These issues also indicate that experimental settings are more difficult to reproduce due to a lack of information.

3.4. Feedback from authors.

To foster a discussion within our community, we have contacted the authors of the selected papers and collected feedback on our findings. We conducted a survey with 135 authors for whom contact information has been available. To protect the authors' privacy and encourage an open discussion, all responses have been anonymized.

The survey consists of a series of general and specific questions on the identified pitfalls. First, we ask the authors whether they have read our work and consider it helpful for the community. Second, for each pitfall, we collect feedback on whether they agree that (a) their publication might be affected, (b) the pitfall frequently occurs in security papers, and (c) it is easy to avoid in most cases. To quantitatively assess the responses, for each question we use a five-point Likert scale ranging from *strongly disagree* to *strongly agree*. We also provide an option of *prefer not to answer* and allow the authors to omit questions.

We have received feedback from 49 authors, yielding a response rate of 36%. These authors correspond to 13 of the 30 selected papers and thus represent 43% of the considered research. Regarding the general questions, 46 (95%) of the authors have read our paper and 48 (98%) agree that it helps to raise awareness for the identified pitfalls. For the specific pitfall questions, the overall agreement between the authors and our findings is 63% on average, varying depending on the security area and pitfall. All authors agree that their paper may suffer from at least one of the pitfalls. On average, they indicate that 2.77 pitfalls are present in their work with a standard deviation of 1.53 and covering all 10 pitfalls.

When assessing the pitfalls in general, the authors especially agree that lab-only evaluations (92%), the base rate fallacy (77%), inappropriate performance measures (69%), and sampling bias (69%) frequently occur in security papers. Moreover, they state that inappropriate performance measures (62%), inappropriate parameter selection (62%), and the base rate fallacy (46%) can be easily avoided in practice, while the other pitfalls require more effort.

In summary, we derive three central observations from this survey. First, most authors agree that there is a lack of awareness for the identified pitfalls in our community. Second, they confirm that the pitfalls are widespread in security literature and there is a need for mitigating them. Third, a consistent understanding of the identified pitfalls is still lacking. As an example, several authors (44%) neither agree nor disagree on whether data snooping is easy to avoid, emphasizing the importance of clear definitions and recommendations.

3.5. Takeaways.

We find that all of the pitfalls introduced in §2 are pervasive in security research, affecting between 17% and 90% of the selected papers. Each paper suffers from at least three of the pitfalls and only 22% of instances are accompanied by a discussion in the text. While authors may have even deliberately omitted a discussion of pitfalls in some cases, the results of our prevalence analysis overall suggest a lack of awareness in our community.

Although these findings point to a serious problem in research, we would like to remark that *all* of the papers analyzed provide excellent contributions and valuable insights. Our objective here is not to blame researchers for stepping into pitfalls but to raise awareness and increase the experimental quality of research on ML in security.

4. IMPACT ANALYSIS

In the previous sections, we have presented pitfalls that are widespread in the computer security literature. However, so far it remains unclear how much the individual pitfalls could affect experimental results and their conclusions. In this section, we estimate the experimental impact of some of these pitfalls in popular applications of machine learning in security. At the same time, we demonstrate how the recommendations discussed in §2 help in identifying and resolving these problems. For our discussion, we consider four popular research topics in computer security:

- §4.1: Mobile malware detection (P1, P4, and P7)
- §4.2: Vulnerability discovery (P2, P4, and P6)
- §4.3: Source code authorship attribution (P1 and P4)
- §4.4: Network intrusion detection (P6 and P9)

Remark. *For this analysis, we consider state-of-the-art approaches for each security domain. We remark that the results within this section do not mean to criticize these approaches specifically; we choose them as they are representative of how pitfalls can impact different domains. Notably, the fact that we have been able to reproduce the approaches speaks highly of their academic standard.*

4.1. Mobile malware detection.

The automatic detection of Android malware using ML is a particularly lively area of research. The design and evaluation of such methods are delicate and may exhibit some of the previously discussed pitfalls. In the following, we discuss the effects of sampling bias (P1), spurious correlations (P4), and inappropriate performance measures (P7) on learning-based detection in this context.

Dataset collection. A common source of recent mobile data is the *AndroZoo* project,[2] which collects Android apps from a large variety of sources, including the official Google Play store and several Chinese markets. At the time of writing, it includes more than 11 million Android applications from 18 different sources. As well as the samples themselves, it includes meta-information, such as the number of antivirus detections. Although AndroZoo is an excellent source for obtaining mobile apps, we demonstrate that experiments may suffer from severe sampling bias (P1) if the peculiarities of the dataset are not taken into account. Please note that the following discussion is not limited to the AndroZoo data, but is relevant for the composition of Android datasets in general.

Dataset analysis. In the first step, we analyze the data distribution of AndroZoo by considering the origin of an app and the number of antivirus detections of an Android app. For our analysis, we broadly divide the individual markets into four different origins: GooglePlay, Chinese markets, VirusShare, and all other markets.

Interestingly, we find that when sampling randomly from the dataset, benign applications come with a probability of around 80% from GooglePlay. In contrast, malicious apps mainly originate from Chinese markets, indicating a sampling bias.

Note, however, that this sampling bias is not limited to AndroZoo. We identify a similar bias for the Drebin dataset,[4] which is commonly used to evaluate the performance of learning-based methods for Android malware detection. The details of the analysis of this dataset can be found in the original publication.[3]

Experimental setup. To get a better understanding of this finding, we conduct experiments using two datasets: For the first dataset (*D* 1), we merge 10,000 benign apps from GooglePlay with 1,000 malicious apps from Chinese markets (*Anzhi* and *AppChina*). We then create a second dataset (*D* 2) using the same 10,000 benign applications, but combine them with 1,000 malware samples exclusively from GooglePlay. All malicious apps are detected by at least 10 virus scanners.

Next, we train a linear support vector machine (SVM) on these datasets using two feature sets taken from state-of-the-art classifiers (Drebin[4] and Opseqs[16]).

Results. The true positive rate for Drebin and Opseqs drops by more than 10% and 15%, respectively, between the

Table 1. Comparison of results for two classifiers when merging benign apps from Google Play with Chinese malware (D_1) vs. sampling solely from Google Play (D_2). For both classifiers, the detection performance drops significantly when considering apps only from Google Play. The standard deviation of the results ranges between 0–3%.

	Drebin			Opseqs		
Metric	D_1	D_2	Δ	D_1	D_2	Δ
Accuracy	0.994	0.980	–1.4%	0.972	0.948	–2.5%
Precision	0.968	0.930	–3.9%	0.822	0.713	–13.3%
Recall	0.964	0.846	–12.2%	0.883	0.734	–16.9%
F1-Score	0.970	0.886	–8.7%	0.851	0.722	–15.2%
MCC	0.963	0.876	–9.0%	0.836	0.695	–16.9%

datasets *D* 1 and *D* 2, while the accuracy is only slightly affected (see Table 1). Hence, the choice of the performance measure is crucial (P7). Interestingly, the Web address *play.google.com* turns out to be one of the five most discriminative features for the benign class, suggesting that the classifier has learned to distinguish the origins of Android apps, rather than the difference between malware and benign apps (P4). Although our experimental setup overestimates the classifiers' performance by deliberately ignoring time dependencies (P3), we can still clearly observe the impact of the pitfalls. Note that the effect of other snooping types in this setting has been demonstrated in previous work.[18]

4.2. Vulnerability discovery.

Vulnerabilities in source code are a major threat to the security of computer systems and networks. Since the manual search for vulnerabilities is complex and time consuming, machine learning-based detection approaches have been proposed in recent years.[25] In what follows, we show that a popular dataset for vulnerability detection contains artifacts (P4) used by a state-of-the-art method for vulnerability discovery, VulDeePecker.[15] Surprisingly, we find that it can be outperformed by a simple linear classifier (P6) and discuss how VulDeePecker's preprocessing steps make it impossible to decide whether some snippets contain vulnerabilities or not (P2).

Dataset collection. For our analysis, we use the dataset published by Li et al.[15] We focus on vulnerabilities related to buffers (CWE-119) and obtain 39,757 source code snippets of which 10,444 (26%) are labeled as containing a vulnerability.

Experimental setup. We train VulDeePecker, based on a recurrent neural network (RNN), to classify the code snippets automatically. To this end, we replace variable names with generic identifiers (for example, `INT2`) and truncate the snippets to 50 tokens, as proposed in the paper.[15]

We use a linear SVM with bag-of-words features based on *n*-grams as a baseline for VulDeePecker. To see what VulDeePecker has learned, we use Layerwise Relevance Propagation (LRP)[6] to explain the predictions and assign each token a *relevance* score that indicates its importance for the classification.

Results. To see how the model derives its decisions, we analyze the 10 most important tokens for each code snippet. Following this approach, we notice two things: Firstly, tokens such as '`(`', '`]`', and '`,`' are among the most important features throughout the training data although they occur frequently in code from both classes as part of function calls or array initialization. Secondly, there are many generic `INT*` values which frequently correspond to buffer sizes. From this we conclude that VulDeePecker is relying on combinations of artifacts in the dataset and thus suffers from spurious correlations (P4).

To further support this finding, we show in Table 2 the performance of VulDeePecker compared to an SVM and an ensemble of standard models trained with the *AutoSklearn* library.[10] We find that an SVM with 3-grams yields the best performance with an 18 × smaller model. This is interesting as the SVM uses overlapping but independent substrings (*n*-grams), rather than the true sequential ordering of all tokens as for the RNN. Thus, it is likely that VulDeePecker is not exploiting relations in the sequence, but merely combines special tokens—an insight that could have been obtained by training a linear classifier (P6). Furthermore, it is noteworthy that both baselines provide significantly higher true-positive rates, although the ROC-AUC[9] of all approaches only slightly differs.

Table 2. Performance of support vector machines and VulDeePecker on unseen data. The true-positive rate is determined at 2.9% false positives.

Model#	parameters	AUC	TPR
VulDeePecker	1.2×10^6	0.984	0.818
SVM	6.6×10^4	0.986	0.963
AutoSklearn	8.5×10^5	0.982	0.894

Finally, VulDeePecker discards essential information during its preprocessing steps, making it impossible to detect vulnerabilities in certain cases. For instance, numbers are converted to generic tokens, which removes crucial information for detecting buffer overflows: After the conversion, it is not possible to tell how big the buffer is and whether the content fits into it or not. Depending on the surrounding code, it can become impossible to say whether buffer overflows appear or not, leading to cases of label inaccuracy (P2).

4.3. Source code author attribution.

The task of identifying the developer based on source code is known as authorship attribution.[8] Programming habits are characterized by a variety of stylistic patterns, so that state-of-the-art attribution methods use an expressive set of such features. These range from simple layout properties to more unusual habits in the use of syntax and control flow. In combination with sampling bias (P1), this expressiveness may give rise to spurious correlations (P4) in current attribution methods, leading to an overestimation of accuracy.

Dataset collection. Recent approaches have been tested on data from the Google Code Jam (GCJ) programming competition,[1,8] where participants solve the same challenges in various rounds. An advantage of this dataset is that it ensures a classifier learns to separate stylistic patterns rather than merely overfitting to different challenges. We use the 2017 GCJ dataset, which consists of 1,632 C++ files from 204 authors, solving the same eight challenges.

Dataset analysis. We start with an analysis of the average similarity score between all files of each respective programmer, where the score is computed by *difflib's SequenceMatcher*.[b] We find that most participants copy code across the challenges, that is, they reuse personalized coding *templates*. Understandably, this results from the nature of the competition, where participants are encouraged to solve challenges quickly. These templates are often *not* used to solve the current challenges but are only present in case they might be needed. As this deviates from real-world settings, we identify a sampling bias in the dataset.

Current feature sets for authorship attribution include these templates, such that models are learned that strongly focus on them as highly discriminative patterns. However, this unused duplicate code leads to features that represent artifacts rather than coding style which are spurious correlations.

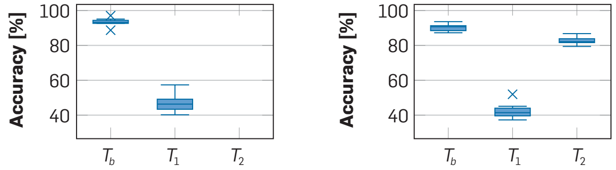
Experimental setup. Our evaluation on the impact of both pitfalls builds on the attribution methods by Abuhamad et al.[1] and Caliskan et al.[8] Both represent the state of the art regarding performance and comprehensiveness of features.

We implement a linter tool on top of Clang, an open-source C/C++ front end for the LLVM compiler framework, to remove unused code that is mostly present due to the templates. Based on this, we design the following three experiments: First, we train and test a classifier on the unprocessed dataset ($T\,b$) as a baseline. Second, we remove unused code from the respective test sets ($T\,1$), which allows us to test how much the learning methods focus on unused template code. Finally, we remove unused code from the training set and retrain the classifier ($T\,2$).

Results. Figure 2 presents the accuracy for both attribution methods on the different experiments. Artifacts have a substantial impact on the attribution accuracy. If we remove unused code from the test set ($T\,1$), the accuracy drops by 48% for the two approaches. This shows both systems focus considerably on the unused template code. After retraining ($T\,2$), the average accuracy drops by 6% and 7% for the methods of Abuhamad et al.[1] and Caliskan et al.,[8] demonstrating the reliance on artifacts for the attribution performance.

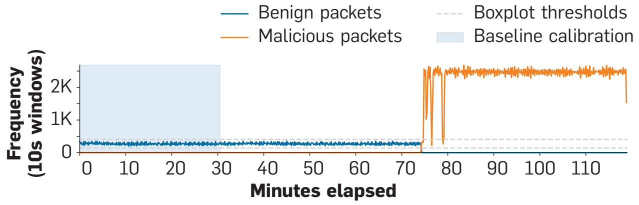
Overall, our experiments show that the impact of sampling bias and spurious correlations has been underestimated and reduces the accuracy considerably. At the same time, our results are encouraging. After accounting for artifacts, both attribution methods select features that allow for a more reliable identification.

Figure 2. Accuracy of authorship attribution after considering artifacts. The accuracy drops by 48% if unused code is removed from the test set (T1); After retraining (T2), the average accuracy still drops by 6% and 7%.



(a) Abuhamad et al. **(b) Caliskan et al.**

Figure 3. Frequency of benign vs. malicious packets in the Mirai dataset.[17] The Gray dashed lines show the thresholds that define normal traffic calculated using the simple baseline (*boxplot method*.[23] The span of clean data used for calibration is highlighted by the light blue shaded area.



4.4. Network intrusion detection.

Detecting network intrusions is one of the oldest problems in security and it comes as no surprise that detection of anomalous network traffic relies heavily on learning-based approaches. However, challenges in collecting real attack data has often led researchers to generate synthetic data for lab-only evaluations (P9). Here, we demonstrate how this data is often insufficient for justifying the use of complex models (for example, neural networks) and how using a simpler model as a baseline would have brought these shortcomings to light (P6).

Dataset collection. We consider the dataset released by Mirsky et al.,[17] which contains a capture of Internet of Things (IoT) network traffic simulating the initial activation and propagation of the Mirai botnet malware. The packet capture covers 119 minutes of traffic on a Wi-Fi network with three PCs and nine IoT devices.

Dataset analysis. First, we analyze the transmission volume of the captured network traffic. Figure 3 shows the frequency of benign and malicious packets across the capture, divided into bins of 10 seconds. This reveals a strong signal in the packet frequency, which is highly indicative of an ongoing attack. Moreover, all benign activity seems to halt as the attack commences, after 74 minutes, despite the number of devices on the network. This suggests that individual observations may have been merged and could further result in the system benefiting from spurious correlations (P4).

Experimental setup. To illustrate how severe these pitfalls are, we consider Kitsune,[17] a state-of-the-art deep learning-based intrusion detector built on an ensemble of autoencoders. For each packet, 115 features are extracted that are input to 12 autoencoders, which themselves feed to another, final autoencoder operating as the anomaly detector.

As a simple baseline to compare against KITSUNE, we choose the *boxplot method*,[23] a common approach for identifying outliers. We process the packets using a 10-second sliding window and use the packet frequency per window as the sole feature. Next, we derive a lower and upper threshold from the clean calibration distribution: $\tau\ low = Q\,1 - 1.5 \cdot \mathrm{IQR}$ and $\tau\ high = Q\,3 + 1.5 \cdot \mathrm{IQR}$. During testing, packets are marked as benign if the sliding window's packet frequency is between $\tau\ low$ and $\tau\ high$, and malicious otherwise. In Figure 3, these thresholds are shown by the dashed gray lines.

Results. The classification performance of the autoencoder ensemble compared to the boxplot method is shown in Table 3. While the two approaches perform similarly in terms of ROC-AUC, the simple boxplot method outperforms the autoencoder ensemble at low false-positive rates (FPR). As well as its superior performance, the boxplot method is exceedingly lightweight compared to the feature extraction and test procedures of the ensemble. This is especially relevant as the ensemble is designed to operate on resource-constrained devices with low latency (for example, IoT devices).

Table 3. Comparing Kitsune,[17] an autoencoder ensemble NIDS, against a simple baseline, boxplot method,[23] for detecting a Mirai infection.

Detector	AUC	TPR	TPR
		(FPR at 0.001)	(FPR at 0)
Kitsune[17]	0.968	0.882	0.873
Baseline[23]	0.998	0.996	0.996

Note this experiment does not intend to show that the boxplot method can detect an instance of Mirai operating in the wild, nor that Kitsune is incapable of detecting other attacks, but to demonstrate that an experiment without an appropriate baseline (P6) *is insufficient to justify the complexity and overhead of the ensemble*. The success of the boxplot method also shows how simple methods can reveal issues with data generated for lab-only evaluations (P9). In the Mirai dataset the infection is overly conspicuous; an attack in the wild would likely be represented by a tiny proportion of network traffic.

4.5. Takeaways.

The four case studies clearly demonstrate the impact of the considered pitfalls across four distinct security scenarios. Our findings show that subtle errors in the design and experimental setup of an approach can result in misleading or erroneous results. Despite the overall valuable contributions of the research, the frequency and severity of pitfalls identified in top papers clearly indicate that significantly more awareness is needed. We also show how pitfalls apply across multiple domains, indicating a general problem that cannot be attributed to only one of the security areas.

5. CONCLUSION

We identify and systematically assess ten subtle pitfalls in the use of machine learning in security. These issues can affect the validity of research and lead to overestimating the performance of security systems. We find that these pitfalls are prevalent in security research, and demonstrate the impact of these pitfalls in different security applications. To support researchers in avoiding them, we provide recommendations that are applicable to all security domains, from intrusion and malware detection to vulnerability discovery.[3]

Ultimately, we strive to improve the scientific quality of empirical work on ML in security. A decade after the seminal study of Sommer and Paxson,[21] we again encourage the community to reach *outside the closed world* and explore the challenges and chances of embedding ML in real-world security systems.

ACKNOWLEDGMENTS

The authors gratefully acknowledge funding from the German Federal Ministry of Education and Research (BMBF) as BIFOLD—Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref 01IS18037A), by the Helmholtz Association (HGF) within topic “46.23 Engineering Secure Systems”, and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy EXC 2092 CASA-390781972, and the projects 456292433; 456292463; 393063728. Moreover, we acknowledge that this research has been partially sponsored by the U.K. EP/P009301/1 EPSRC research grant.

References

1. Abuhamad, M., AbuHmed, T., Mohaisen, A., and Nyang, D. Large-scale and language-oblivious code authorship identification. In *Proc. of ACM Conf. on Computer and Communications Security*, (2018).
2. Allix, K., Bissyandé, T.F., Klein, J., and Le Traon, Y. Androzoo: Collecting millions of android apps for the research community. In *Proc. of the Int. Conf. on Mining Software Repositories* (2016).
3. Arp, D. et al. Dos and don’ts of machine learning in computer security. In *Proc. of USENIX Security Symp.* (2022), 3971–3988.
4. Arp, D. et al. Drebin: Efficient and explainable detection of android malware in your pocket. In *Proc. of Network and Distributed System Security Symp.* (2014).
5. Axelsson, S. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security* (Aug. 2000).
6. Bach, S. et al. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, (July 2015).
7. Biggio, B. et al. Evasion attacks against machine learning at test time. In *Joint European Conf. on Machine Learning and Knowledge Discovery in Databases*, Springer (2013).
8. Caliskan, A. et al. De-anonymizing programmers via code stylometry. In *Proc. of USENIX Security Symp.* (2015).
9. Fawcett, T. An introduction to ROC analysis. *Pattern Recognition Letters 27*, 8 (2006), 861–874.
10. Feurer, M. et al. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems* (2015).
11. Jang, J., Brumley, D., and Venkataraman, S. Bitshred: Feature hashing malware for scalable triage and semantic analysis. In *Proc. of ACM Conf. on Computer and Communications Security* (2011).
12. Juarez, M. et al. A critical evaluation of website fingerprinting attacks. In *Proc. of ACM Conf. on Computer and Communications Security* (2014).
13. Krippendorff, K. *Content Analysis: An Introduction to Its Methodology*. Sage publications (2018).
14. Krizhevsky, A., Sutskever, I., and Hinton, G.E. Imagenet: Classification with deep convolutional neural networks. In *Advances in Neural Information Proccessing Systems (NIPS)*, 2012.
15. Li, Z. et al. Vuldeepecker: A deep learning-based system for vulnerability detection. In *Proc. of Network and Distributed System Security Symp.* (2018).
16. McLaughlin, N. et al. Deep android malware detection. In *Proc. of ACM Conf. on Data and Applications Security and Privacy* (2017).
17. Mirsky, Y., Doitshman, T., Elovici, Y., and Shabtai, A. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Proc. of Network and Distributed System Security Symp.* (2018).
18. Pendlebury, F. et al. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *Proc. of USENIX Security Symp.* (2019)
19. Pierazzi, F., Pendlebury, F., Cortellazzi, J., and Cavallaro, L. Intriguing properties of adversarial ML attacks in the problem space. In *Proc. of IEEE Symp. on Security and Privacy* (2020).
20. Shin, E.C.R., Song, D., and Moazzezi, R. Recognizing functions in binaries with neural networks. In *Proc. of USENIX Security Symp.* (2015).
21. Sommer, R. and Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *Proc. of IEEE Symp. on Security and Privacy* (2010).
22. Sutskever, I., Vinyals, O., and Le, Q.V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems* (2014).
23. Tukey, J.W. Addison-wesley series in behavioral science: Quantitative methods. *Exploratory Data Analysis*. Addison-Wesley (1977).
24. Wolpert, D.H. The lack of a priori distinctions between learning algorithms. *Neural Computation* (1996).
25. Yamaguchi, F., Maier, A., Gascon, H., and Rieck, K. Automatic inference of search patterns for taint-style vulnerabilities. In *Proc. of IEEE Symp. on Security and Privacy* (2015).

Daniel Arp (d.arp@tu-berlin.de), Technische Universität Berlin, Germany.

Erwin Quiring (erwin.quiring@rub.de), ICSI, Ruhr University Bochum, Germany.

Feargus Pendlebury (feargus@trustypatch.es), University College London, U.K.

Alexander Warnecke (a.warnecke@tu-berlin.de), Technische Universität Berlin, Germany.

Fabio Pierazzi (fabio.pierazzi@kcl.ac.uk), King’s College London, U.K.

Christian Wressnegger (c.wressegger@kit.edu), Karlsruhe Institute of Technology, Germany.

Lorenzo Cavallaro (l.cavallaro@ucl.ac.uk), University College London, U.K.

Konrad Rieck (rieck@tu-berlin.de), Technische Universität Berlin, Germany.