SHERPA

Responsible Development of Smart Information Systems:

Technical Options and Interventions

D3.5 (1st part): April 2020

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme Under Grant Agreement no. 786641

Authors

Samuel Marchal, Senior Data Scientist, F-Secure Corporation (samuel.marchal@f-secure.com)

David Karpuk, Senior Data Scientist, F-Secure Corporation (david.karpuk@f-secure.com)

Alexey Kirichenko, Research Collaboration Manager, F-Secure Corporation (alexey.kirichenko@f-secure.com)

Acknowledgements

We would like to thank Doris Schroeder, Bernd Stahl, the Artificial Intelligence Center of Excellence team at F-Secure, and the SHERPA Stakeholder Board members for comments and helpful discussions.

Document Control

Deliverable	D3.5 (1st part)
WP/Task Related	WP3, Task 3.5
Delivery Date	April 30, 2020
Dissemination Level	PU
Lead Partner	FSC
Contributors	UCLan Cyprus
Reviewers	Doris Schroeder, Bernd Stahl
Abstract	This report presents a study of model poisoning attacks on models from a popular class of anomaly detection algorithms and provides a number of defense recommendation.
Key Words	Data poisoning attacks, model poisoning attacks, anomaly detection, online distributed learning, federated learning, statistical distribution models with thresholds, backdoor poisoning attacks

Table of Contents

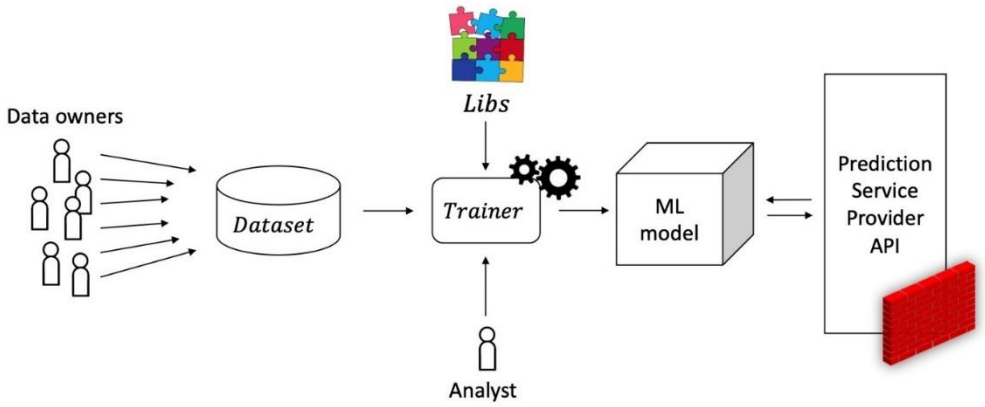
Acronyms and terms 5
Executive Summary 6
Introduction 7
1. Background 10
1.1 Online machine learning 10
1.2 Distributed and federated learning 10
1.3 Model poisoning of online distributed training 12
1.3.1 Adversary's goal 13
1.3.2 Attack surface 14
1.3.3 Adversarial capabilities 14
2. The model to attack 15
2.1 Statistical distribution model with thresholds 15
2.2 Process Launch Distribution (PLD) model 15
2.3 Distributed online training of PLD model 18
2.4 Poisoning PLD model 19
3. Model poisoning approaches 20
3.1 Experimental setup 20
3.2 'Increase target process launch count' approach 21
3.2.1 Attack description and implementation 21
3.2.2 Experimental results 22
3.3 'Decrease parent or child process launch count' approach 25
3.3.1 Attack description and implementation 25
3.3.2 Experimental results 26
3.4 'Inject rare process launch events unrelated to the attack' approach 28
3.4.1 Attack description and implementation 28
3.4.2 Experimental results 29
4. Defense recommendations 33
5. Conclusion 37
References 38

Acronyms and terms

Term	Explanation
AI	Artificial intelligence
Backdoor attack	A model poisoning attack where the goal is to decrease the performance of a target ML model for a set of selected inputs
Denial-of-service attack	A model poisoning attack where the goal is to decrease the performance of a target ML model as a whole
Distributed training	An approach where training data come in multiple parts owned and contributed by multiple parties
i.i.d.	independent and identically distributed (applied to random variables)
ML	Machine Learning
ML model	An artifact created by a training process of an ML algorithm
Normalisation	Adjusting values, often measured on different scales, to bring them into alignment
Online training	An approach where models are periodically and incrementally updated when more training data become available
PLD	Process Launch Distribution, the model this report focuses on
Poisoning attack	Model poisoning, or data poisoning, is a class of training-time attacks on ML-based systems
SIS	Smart Information Systems: The combination of artificial intelligence and big data analytics

Executive Summary

The main purpose of this report is to systematically investigate attack strategies and – based on the analysis – introduce technical defense options and interventions for the responsible use of Artificial Intelligence (AI). The presented study focuses on model poisoning attacks against services powered by Machine Learning (ML) models trained in the online fashion on distributed data from uncontrolled environments (see figure).



Machine learning model training using distributed data owned by multiple clients

To discover and analyse model poisoning attacks (primarily backdoor attacks, but with some denial-of-service effects as well) against online distributed training and to assess their potential impact, we select a popular class of models and consider carefully defined and realistic adversaries targeting such models.

In our analysis and experiments, we show how an adversary can achieve their goals despite lack of knowledge and control over the training contributions of the benign clients. We also show that the poisoning attacks we simulated in the experiments are hard to detect due to their relatively modest scale, which makes it difficult to distinguish them from ordinary variations and concept drift effects.

Learning from the identified vulnerabilities, we provide and illustrate a set of recommendations for detecting and mitigating poisoning attacks against models of the considered class. In particular, the following can help stop attacks, make them costlier for the adversary, or significantly reduce their impact:

1. input validation,
2. normalisation of client contributions (local models or data points),
3. monitoring of contributions of each client over time to detect anomalies, and
4. strong client authentication.

At the same time, we observe that such defence approaches as detecting outliers among contributions of multiple clients in a single round of online training and rejecting local models with a large distance to the global model are often problematic due to their low precision resulting in high false alert rates.

Since the analysed poisoning approaches are applicable to many models successfully used in digital services in multiple domains, our results should be considered a strong warning for organisations developing and operating such services.

Introduction

Attacks on AI systems is a serious concern. Reliability of AI systems in the presence of determined adversaries and resilience to their attacks are of a high importance, since a system controlled, even partially, by an adversary can hardly be considered trustworthy, and one can expect to see violation of multiple values and human rights of the users of such a system. At the same time, with the growing popularity of AI systems and importance of those for our society, they are naturally becoming more attractive targets for attackers.

As in many other domains, Machine Learning (ML) techniques, which power a large share of modern AI systems, were originally designed for benign and controlled environments. That assumption, unfortunately, does not hold in numerous practical applications of ML models today. There are many ways for attacking ML-based systems, both at training and inference time, and their dependence on data, often coming from uncontrolled environments, significantly broadens the attack surface. In particular, *model poisoning*, or *data poisoning*, is an important class of training-time attacks on ML-based systems wherein an attacker injects mislabelled or mis-distributed data into the training process to degrade the performance of ML models. Among multiple types of attacks in the context of ML, data poisoning stands out by deeply exploiting properties specific to ML approaches, it is a threat essentially brought by ML.

This report investigates the threats that data poisoning attacks pose to Machine Learning (ML) and statistical models, considered an important subclass of AI, and their use in various applications.

While injecting data points of their choice is not always possible or affordable for attackers, there are many practically successful systems, the setup of which provides adversaries with rich poisoning opportunities. A good example of such are systems with underlying *ML models trained online and using distributed training data*, which means essentially that (i) their training data come in multiple parts from devices and environments owned by multiple parties, and (ii) their models are periodically and incrementally updated when more training data become available. In this group, one finds, for instance, search engines, systems utilizing sentiment analysis of user opinions and other recommender systems, intrusion and malware detection services, and social media chatbots.

A number of poisoning attacks against such systems have been proposed and studied, and some interesting theoretical results have been reported. Nevertheless, there are almost no flashing media stories about major incidents related to data poisoning[1], and it seems that the level of concern among organisations running services based on ML models susceptible to poisoning attacks, policy-makers, and general public (including users of those services) is quite low. This is perhaps unsurprising, as the experience with cybersecurity showed that unless actual devastating security incidents take place or, at least, fully practical and devastating attacks are demonstrated in certain venues, the industry – and the society in general – are slow and unwilling to invest in fixing vulnerabilities and building better defences.

[1] One rare example is the Tay AI chatbot case (https://www.theverge.com/2016/3/24/11297050/tay-microsoft-chatbot-racist), and it dates back to March 2016.

As one of the top objectives of the SHERPA project is to increase awareness about problems and risks associated with the development and use of AI-based systems, the main goal set for the first stage of the project's technical interventions work in Task 3.5 was, therefore, to emphasize the importance of reliable and attack-resilient AI via designing and analysing practical high-impact poisoning attacks against a popular class of ML models trained in the online distributed fashion. Based on the analysis of the attacks, we also propose defence approaches, which will be further studied, generalised and included in the SHERPA's 'intelligent mix' of recommendations in the second stage of Task 3.5.

To construct practical attacks and assess their impact, one has to select a sufficiently specific attack target, preferably representing well a wider class of systems used in real-life applications. Taking into account the considerations presented above and informed by the SHERPA project work in WP1 (primarily, the work of Task 1.3 on Security Issues, Dangers and Implications of Smart Information Systems and the case study interviews and analysis in Task 1.1) and discussions with the SHERPA Stakeholder Board members, we chose to focus on resilience to poisoning attacks of one ML-based *anomaly detection* system that can be used in the cybersecurity domain.

Anomaly detection is widely used in cybersecurity since it is an effective method to detect unknown and sophisticated attacks [ATK'15, CBK'09, DS'07, MRR'12]. In general, the method is based on modelling the normal, or benign, behavior of computing devices, networks and systems in the absence of attacks and then using learned models to detect attacks as deviations or anomalous behavior. By design, anomaly detection-based security mechanisms do not need any prior knowledge about attacks, they only require examples of normal behavior [CBK'09, SP'10]. The approach is especially useful for countering determined and skillful attackers investing into developing novel attack tactics and techniques.

Online training using distributed data is a common setup for training anomaly detection systems for two reasons. First, it enables leveraging large amounts of diverse data (reported usually by a large number of parties) in order to model comprehensively all possible types of normal behavior. Second, updating models on a regular basis enables adapting to changes and capturing evolving normal behavior.

The system selected as the study target detects anomalous *process launch* events in a computer by modelling the distribution of typical process launches and comparing new events with that distribution. More specifically, the underlying system's model belongs to the class of models named *statistical distribution with thresholds* (explained in Section 2.1). While simple, models of this type can be applied to many use cases and are easy to understand. Consequently, they are widely used in cybersecurity and many other domains.

In security applications, it is prudent to assume that adversaries know the design and parameters of defense mechanisms and will try to compromise the effectiveness of those, in particular, reducing their attack detection power. This assumption naturally applies to ML-based anomaly detection systems: cyber criminals want their attacks to remain undetected and look for ways to bypass anomaly detection mechanisms [BR'18, HJN+'11, PMS+'18], which can be achieved by influencing their underlying models, in particular, via model / data poisoning attacks.

As introduced above, data poisoning [BNL'12, CSL+'08, HJN+'11] is an attack in which an adversary modifies a fraction of training data with the goal of decreasing the performance of ML models trained using the data. In the case of anomaly detection-based security systems, this decrease in performance corresponds to a decrease in attack detection capabilities. Poisoning attacks typically fall in one of the following categories:

- **Denial-of-service attack**, where the goal is to decrease the performance of a target ML model *as a whole*. The predictive accuracy of the ML model will decrease for *any* input (or a majority

of inputs) submitted to it. This means the poisoned anomaly detection system will not be able to distinguish normal / benign inputs from anomalous / malicious ones.

- **Backdoor / Trojan attack**, where the goal is to decrease the performance of a target ML model for a set of selected inputs. The predictive accuracy of the ML model will decrease only for *inputs selected by the attacker*, but must be preserved for any other inputs. This means the poisoned anomaly detection system will not be able to detect a particular attack while remaining effective at distinguishing other anomalous / malicious inputs from normal / benign ones.

Goals and scope of poisoning attacks and techniques used to carry them out can vary widely. Backdoor attacks are typically stealthier, that is, more difficult to detect, than denial-of-service attacks because the former do not manifest in an overall decrease in model performance. They are also more relevant in the security domain where one of the first goals of an adversary is usually to avoid the defensive anomaly detection system alerting on their *specific* cyberattack. Consequently, in the first stage of Task 3.5, we chose to study ***backdoor poisoning attacks***[2].

In this report, we present the study of backdoor poisoning attacks targeting an anomaly detection system practically relevant for the cybersecurity domain.

We introduce three poisoning attack techniques against the chosen anomaly detection model and evaluate their effectiveness and other key properties with respect to several natural adversarial goals. While targeted at the anomalous process launch detection system, our three attack techniques generalize to essentially any statistical distribution model with thresholds. We also present a number of recommendations for detecting and mitigating impact of the considered attacks.

At the project level, our goal for the future work is to communicate the importance of reliable and attack-resilient AI and to draw attention to data / model poisoning attacks and ways of countering those. SHERPA, actively working on AI guidelines and options for standards and regulations in WP3 and closely communicating with the members of its impressively strong project Stakeholder Board (in the framework of WP2), is perfectly positioned for bringing our results and recommendations to the stakeholders. We believe that understanding of dangers and extents of attacks and optimal mitigation strategies and measures is crucial not only for AI developers and providers of AI-powered digital services but also for users of those services, in order to maximize the benefits that AI systems bring and to minimize associated risks.

The remaining part of this report is organized as follows. Section 1 provides background information on online machine learning, distributed training and poisoning attacks. Section 2 presents the ML anomaly detection model which we study and experiment with in the report. It also presents the data poisoning goals and an overview of the three attacks we design and analyze. In Section 3, we describe in detail the three poisoning attacks and assess them as applied against the anomalous process launch detection model. Section 4 provides recommendations for countering poisoning attacks against statistical distribution models. The study and the lessons learned are summarized in the Conclusion (Section 5).

[2] We note that one of the studied attacks brings some denial-of-service effects as well.

1. Background

1.1 Online machine learning

Online machine learning or online training is a machine learning approach which recognizes that training data become available in portions in a sequential order, continuously or periodically. As new data samples become available, online training is used to incrementally update a machine learning model with new data. This is in contrast to batch training where a training set is collected until a certain point in time and, once the collection is completed, a model is trained using this static dataset and will not change afterwards (but may, of course, be eventually replaced by a new model). Batch training is a one-time operation and the resulting model is used without further updates.

Online machine learning is typically used in two main scenarios:

- When a training dataset is statically defined (i.e. it does not evolve) but too large to be used at once for training a model. The data is then processed in small portions to incrementally update the model in order to make it computationally feasible to use the entire dataset. Such training techniques as Stochastic Gradient Descent (SGD) represent online machine learning approaches addressing the challenges of this scenario.
- When a training dataset evolves over time and a model needs to dynamically adapt to new patterns and other properties of the data. Then the model is updated on a regular basis, sometimes very frequently, as new data samples are available. Model updating methods can use either only the most recent data samples (not used earlier) or add to those some of the earlier collected samples (e.g. in the sliding window fashion).

ML-based anomaly detection methods typically use online training for the reason discussed in the second scenario. The behavior of a system to model usually evolves over time due to various internal and external factors, which is referred to in the ML domain as *concept drift*. Anomaly detection models must capture concept drift changes in order to remain effective. In the cybersecurity context, in particular, the inability to learn new behavior leads to large numbers of false alarms due to detecting novel innocent events as anomalous and potentially malicious, which is often a major problem.

1.2 Distributed and federated learning

Distributed learning refers to a machine learning process which is distributed among multiple data owners, or *clients*, and coordinated by a central entity called *aggregator*. Clients and aggregator collaborate to train a common *global ML model G*, based on all the available training data. There are multiple ways to carry out distributed training, and we present here two approaches which are the most relevant for our study.

The first approach is called *data aggregation* and it considers clients only as sources of data with no processing capabilities. In other words, training data is distributed among multiple clients, and model training operations are fully carried out by an aggregator. Local datasets D_i are submitted by each client and aggregated into a global dataset D by the aggregator. The aggregation process in this scenario comes down simply to merging all the client-specific datasets D_i into the global dataset as follows:

$$D = \bigcup_{i \in C} D_i$$

where C is the set of all the clients participating in the distributed training process. The aggregator then uses the global dataset D to train a global model G using model training techniques selected by data analysts of the aggregator entity, as depicted in Figure 1.

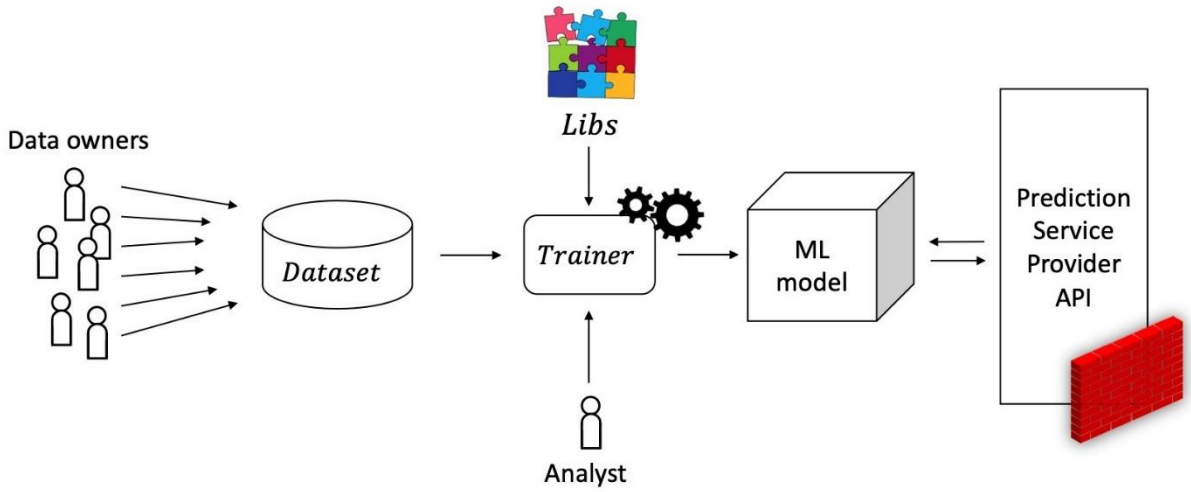


Figure 1: ML model training using distributed data owned by multiple clients.

The second approach is called *federated learning*, and it delegates parts of the training process to clients [BEG+'19, KMY+'16, MMR+'17]. Instead of providing their local datasets for merging into a global centralized dataset, each client uses their D_i to train a local model L_i. These local models are sent to the aggregator instead of the local training datasets. The aggregator then uses a specific procedure in order to aggregate all the local models L_i into the global model G. While federated learning also leverages all the local data D_i owned by the clients in order to train G, in contrast to plain data aggregation, the clients do not have to expose their local data. Thus, federated learning has a number of advantages over data aggregation, including: (i) communication efficiency [KMY+'16, MMR+'17] because local models typically have significantly smaller sizes than the datasets they are trained on; and (ii) privacy-preserving [GKN'17] because potentially sensitive data owned by the clients do not need to be shared with any other parties. Federated learning also distributes computational efforts, offloading a part of the aggregator computations to the clients, which can be desirable when the client devices have unused processing power (which is often the case for modern computers and mobile devices).

More generally, the federated training process can be iterative and composed of several communication rounds performed at successive times t. If all the clients take part in each round, this is essentially about training in the online fashion. Alternatively, the aggregator can select (usually randomly) only a fraction of the clients for each round, and then multiple rounds are required for ensuring sufficiently high representation of the clients in the training. This process is depicted in Figure 2 and it consists of repeating the following four steps:

1. The aggregator sends the global model at time t, G^t, to every (selected) client.
2. Each client updates G^t using their local dataset D_i to obtain a local model at time t, L_i^t.
3. Each client sends their local model L_i^t to the aggregator.
4. The aggregator aggregates all L_i^t into the new global model at time (t+1), G^{t+1}.

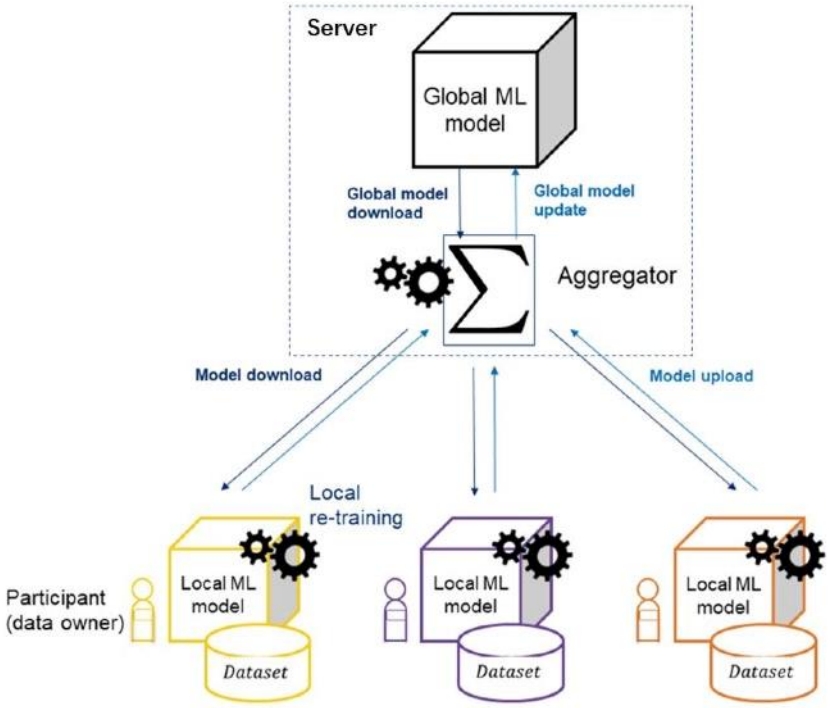


Figure 2: Federated learning of a global ML model using local models from multiple clients.

Several aggregation algorithms exist to combine local models L_i^t into a global model G^t. The most common is called Federated-Averaging [KMY+'16, MMR+'17]. It is defined by the following formula as the weighted sum of local models from each (selected) client:

$$G^{t+1} = \sum_{i \in C} \frac{n_i}{N} \times L_i^t$$

where n_i is the size of D_i, N is the size of D, and L_i^t is the local model of client *i* at time *t*. Each local model is weighted with a coefficient accounting for the number of data samples contained in the local dataset that was used to train it. The coefficients are scaled by dividing the local dataset sizes by the total number of data samples owned by all the clients (or all the selected clients). This approach gives local models trained on large amount of data a larger impact on what the global model will be.

1.3 Model poisoning of online distributed training

To discover and reason about model poisoning attacks targeting online distributed training and to analyse their potential impact, we define here a model of adversary, most importantly, their goals, properties and capabilities. We assume that the aggregator is a trusted party in the distributing training process and an adversary is one or several malicious clients participating in the training of the global model G. The adversary contributes local datasets D_i or local models L_i^t to the online distributed learning process. Contributions from the adversary can be repeated over time due to the online nature of the training setting, where new data are periodically used to update the global model: $G^t \rightarrow G^{t+1}$. This means the adversary has the capability to submit several local datasets or local models over time. We now proceed to define the goal, attack surface and capabilities of the adversary in model poisoning attack on online distributed training.

1.3.1 Adversary's goal

The goal of the adversary is to perform a data poisoning attack, affecting the global model G. This attack can be either a denial-of-service attack or a backdoor attack, which are described and formalized as follows:

- **Denial-of-service attack** sets a goal to decrease the performance of the ML model *as a whole* [BNL'12, CSL+'08, HJN+'11]. The predictive accuracy of the ML model will decrease for *any* input (or a majority of inputs) submitted to it. More specifically, this means that the accuracy of the poisoned model G_p will be significantly lower than that of the non-poisoned model G on a test set randomly sampled from the input space. A denial of service poisoning attack is successful if the following condition is met (with $Acc(G)$ denoting the predictive accuracy of G on a random test set):

$$Acc(G_p) \ll Acc(G), \text{which clearly means } G_p(x) \neq G(x) \text{ for a large share of } x.$$

- **Backdoor / Trojan attack** sets a goal to decrease the performance of the ML model for *a set of selected inputs* [CLL+'17]. The predictive accuracy of the ML model will decrease only for the set of inputs selected by the attacker, which is called a *trigger set*, and we denote it by T. For the inputs from T, the poisoned model G_p must output predictions defined by a backdoor function B. B is defined by the adversary, and we naturally have $B(x) \neq G(x)$. The accuracy of G_p must be preserved for any inputs that are not a part of T. A backdoor poisoning attack characterized by a trigger set T and a backdoor function B is successful if the following condition is met:

$$G_p(x) = \begin{cases} G(x) & \forall\, x \notin T \\ B(x)\ (\neq G(x)) & \forall\, x \in T \end{cases}$$

A secondary, but still important, goal for any poisoning attack is to achieve its primary goal with 'minimal malicious action'. Minimality can be interpreted as either minimal modifications to original local datasets D_i or models L_i^t or a minimum number of local datasets D_i or models L_i^t to poison. The first reason for targeting minimality of the malicious action is that there may be restrictions on the amount of data and the number of local models a single client can contribute to the distributed training process (e.g., one local model per communication round and per client in federated learning). Limiting the 'budget' for a poisoning attack increases its chance of success if such restrictions are in place. The second reason is achieving *stealthiness* of an attack. If a poisoning attack requires significant modifications to local datasets and models, those will likely be very different from datasets and models of other (benign) clients of the distributed training. Then adversarial datasets and models can be identified as anomalous by the aggregator and the poisoning attack can be detected and blocked.

Backdoor poisoning attacks are stealthier and more difficult to detect than denial of service attacks [WYS+'19]. Denial of service attacks can often be easily noticed since the poisoned global model has an overall decrease in performance / accuracy. In contrast, backdoor attacks do not decrease the model performance for most of the inputs to the global model. The aggregator, or the model users, can only notice that the model has a backdoor by submitting inputs from the trigger set to the model. So, in order to detect a backdoor attack, one needs:

a) knowledge and availability of samples from the trigger set;
b) correct labels for those trigger set samples; and
c) knowledge that the non-poisoned model would not make prediction errors on those trigger set samples (i.e., the errors are introduced by the attacker).

We also note that even if we discover that a global model is poisoned, identifying the client(s) responsible for the attack can be challenging.

1.3.2 Attack surface

We consider the adversary to be (controlling) a client in an online distributed training process. Malicious clients have the same access to the training process as any other client. This means that their only interaction with the training process is the submission of at most one local dataset D_i or model L_i^t per communication round and per controlled malicious client. If the adversary controls several clients, they can distribute the poisoning attack among several local models and datasets contributed by those clients. Since G is learned in the online fashion, the adversary can also distribute the poisoning attack over several submissions of local models and datasets.

1.3.3 Adversarial capabilities

Poisoned data injection. We consider fully online adversaries that can only inject poisoned local datasets and models to the regular stream of local datasets and models. The adversary cannot modify datasets and models submitted by other benign clients participating in the training process. The properties, contents and numbers of injected poisoned datasets and models are freely defined by the adversary, within the restrictions of the target online learning method (i.e. one local dataset and model per communication round and per client the adversary controls).

Number of compromised clients. The adversary can control multiple compromised clients. Each controlled client increases the adversarial capability to inject poisoned datasets and models. An increased number of compromised clients can increase either the effectiveness of the attack (by combining the poisoning capabilities of the compromised clients) or the stealthiness of the attack (by distributing poisoning data across the compromised clients). We assume that the adversary cannot compromise a dominant share of all the clients participating in the online training, so there is always a large share of benign clients that contribute benign data and models. In particular, if the total number of clients is *N*, we assume that the adversary cannot control more than *N*/2 clients. Of course, if an aggregation algorithm prioritizes local models trained on larger datasets (e.g. as discussed in the end of Section 1.2), the clients controlled by the adversary can exploit that to 'inflate' their impact. However, if their number is small, their contributions will look anomalously high, which helps detect their activities. So, the upper bound on the number of compromised clients is meaningful even in such scenarios.

Model knowledge. We assume that the adversary knows the type of the machine learning model being trained and its hyper-parameters. This assumption is obvious in federated learning cases where each client trains their own local model and receives evolving global models from the aggregator. To make the adversary stronger, we assume the same knowledge in the data aggregation scenario, even though global models are not necessarily shared with clients in this case.

Then we consider the following two cases of additional model knowledge:

- **Weak adversary** does not know the parameters of the global model G_t at time t. The global models are hosted on the central server and confidential from the client perspective. This is typically the case in the data aggregation mode of distributed training. A weak adversary may have partial access to G_t nevertheless, through a model query interface providing model predictions for submitted inputs.
- **Strong adversary** knows the parameters of the global model G_t at time t. The global model is shared with each client after each communication round and aggregation of the local models.

This is typically the case in federated learning, where local models L_t^i are trained based on the global model G_t.

2. The model to attack

In this section, we present the ML model we chose for the study. We take the problem of anomaly detection as our study case, more precisely, detection of anomalous process launches in a computing system. The model we studied, selected as a realistic attack target, is simple yet effective and similar to models that could be used in real-life defence systems. It models the distribution of process launch events in client machines and detects events anomalous for this distribution. This type of model is referred to as *statistical distribution with thresholds* and it is presented in Section 2.1. Models of this type can be applied to many practical use cases, they are easy to understand and, consequently, they have numerous real-life applications in various domains.

In Section 2.2, we present a concrete instance of a statistical distribution model with thresholds that can be used for the purpose of detecting anomalous process launches in computers. This model is called *Process Launch Distribution (PLD),* and it is trained in a federated and online manner as described in Section 2.3.

Finally, we propose a general attack scenario against the PLD model in Section 2.4 and introduce three poisoning attack techniques which will be analysed in detail in Section 3.

2.1 Statistical distribution model with thresholds

We chose to study one of the simplest but also very popular and effective methods for anomaly detection. This method detects anomalies as deviations from an observed distribution or, more precisely, as rare events with respect to an observed distribution. It requires:

1. To define a score function for quantifying events
2. To model the distribution of the score values computed for the events in the training set and
3. To use the modelled distribution to define threshold(s) that separate common / normal events from rare / anomalous events

When applied to computer security, an underlying assumption of this method is that benign events in a training set are much more frequent than events connected with attacks. The defined score function is computed on the training set events, and the distribution of the obtained score values becomes the model. Using this distribution, one or several threshold values can be assigned, which separate normal events, where the distribution is dense, from anomalous events, where the distribution is sparse. At the inference time, when new events come, we compute their score values and compare them with the assigned threshold values. Based on this comparison, we can see how anomalous a specific event is, depending on how dense the model distribution is in the interval (between two consecutive threshold values) which the event belongs to.

2.2 Process Launch Distribution (PLD) model

A highly useful application of the statistical distribution modelling approach in computer security is a method for detecting anomalous process launch events in a computing system. Operations in computing systems are carried out by so-called processes, instantiating at run-time software programs and containing their code, resources, activities, etc. Processes start each other in various ways, for example, a web browser typically starts a PDF reader to open a PDF file found on the Internet. An

action of a *parent process* starting, or *launching*, a *child process* is called a *process launch* event. Such events can often be used for reliable identification of attempts of cybercriminals to compromise computing systems. On the one hand, many process launch events are observed frequently in nearly all computers and can be considered a part of their normal benign activities. Here are a few examples of popular process launch events represented as (parent process → child process) ordered pairs:

- *java.exe → cmd.exe*
- *cmd.exe → conhost.exe*
- *cmd.exe → find.exe*
- *SearchIndexer.exe → SearchFilterHost.exe*

On the other hand, some process launches are very rare, anomalous and typically a sign of malicious activities. Here are a few examples of suspicious process launch events:

- *winword.exe → cmd.exe*
- *SQLAGENT.EXE → conhost.exe*
- *chrome.exe → rundll32.exe*

Analysis shows that the most reliable signs of attacks are events where common processes are used in anomalous ways. In the context of process launch, this corresponds to a common parent process starting a common child process, but their ordered pair is rare. For instance, *winword.exe* often starts other processes and *cmd.exe* is often started by other processes, but it is very unusual to see *winword.exe* starting *cmd.exe*. Such an event is a strong signal of a spear-phishing attack, and a similar logic can be applied to a number of other process launches related to malicious activities. At the same time, if the parent process or the child process in a given pair is rare itself, using such events for raising security alerts is risky. There are many benign processes which are seen very rarely and, of course, any event including one or two such processes is rare as well but not necessarily connected to any cyberattacks. A good example of such processes is customized software installers. Since false alerts are highly undesirable in security monitoring, a good model should not consider events with rarely seen parent or child processes as anomalous.

Based on the above observations, a statistical distribution model can be designed and trained to detect suspicious process launch events. Summarising the intuition brought by mining process launch datasets, such a model should take into account the following three heuristics:

- How common the *child process* is, i.e. how often it is started by other processes
- How common the *parent process* is, i.e. how often it starts other processes
- How common the *process launch* (*parent, child*) ordered pair is.

A key step in designing a model is to define a score function, which we will call *Process Launch Distribution score* (*PLD score*), combining the three heuristics above in an appropriate way. We chose the PLD score function to be based on the pointwise mutual information (PMI) between two processes, inspired by a generic method for detecting anomalous records in categorical datasets [DS'07]. Given a parent process $proc_A$ starting a child process $proc_B$, the PMI for this process launch event is given by:

$$PMI(proc_A, proc_B) = log\left(\frac{p(proc_A, proc_B)}{p(proc_A) \times p(proc_B)}\right)$$

Of course, the three probabilities in this formula have to be considered empirical and can be estimated via the corresponding frequencies in a training set. This led us to defining the PLD score function in the following way:

$$PLD\_score(proc_A, proc_B) = \frac{\frac{call_{join}(proc_A, proc_B) + \alpha_1}{call_{total} + \beta_1}}{\frac{call_{parent}(proc_A) + \alpha_2}{call_{total} + \beta_2} \times \frac{call_{child}(proc_B) + \alpha_3}{call_{total} + \beta_3}}$$

where:

- $call_{parent}(proc_A)$ is the number of times $proc_A$ started a child process
- $call_{child}(proc_B)$ is the number of times $proc_B$ was started by another process
- $call_{join}(proc_A, proc_B)$ is the number of times $proc_A$ started $proc_B$
- $call_{total}$ is the total number of process launch events in the training set
- $\alpha_1, \alpha_2, \alpha_3$ and $\beta_1, \beta_2, \beta_3$ are smoothing and other constants, often chosen to optimize the model performance

The PLD score is always a non-negative number. The lower the PLD score is, i.e., the closer it is to zero, the more anomalous the process launch event is from the PLD model point of view, thus, the more suspicious it is from our security intuition point of view. We can build a PLD score distribution model by collecting and analysing a training set of process launches observed in selected computing systems, with each specific (*parent, child*) ordered pair represented in the set by as many instances as the number of times the *parent* → *child* event was observed. Since our goal is to identify how anomalous process launch events are with respect to the PLD model, we define several threshold values corresponding to exponentially lower percentiles of the PLD score cumulative distribution. The first threshold separates the 10th percentile of the PLD score cumulative distribution, that is, leaving on the other side the least anomalous process launch events (90% of all the events with the highest *PLD_score* values), which we call *anomaly category* 1. The second threshold separates the 1st percentile of the PLD score cumulative distribution, and process launch events with PLD score above this threshold but below the first threshold are said to belong to anomaly category 2, and so on. The higher the anomaly category index, the more suspicious a process launch event is. The following table shows the threshold definitions and the associated categories:

Range of cumulative distribution	Threshold separates	Anomaly category
]10% - 100%]	10th percentile	1
]1% - 10%]	1st percentile	2
]0.1% - 1%]	1st 1000-quantile	3
]0.01% - 0.1%]	1st 10,000-quantile	4
]0.001% - 0.01%]	1st 100,000-quantile	5
]0.0001% - 0.001%]	1st 1,000,000-quantile	6
]0.00001% - 0.0001%]	1st 10,000,000-quantile	7
]0% - 0.00001%]	-	8

Table 1: Definition of thresholds and anomaly categories for PLD scores based on trained distribution

The following figure depicts an example of a computed PLD score distribution together with the thresholds inferred from it.

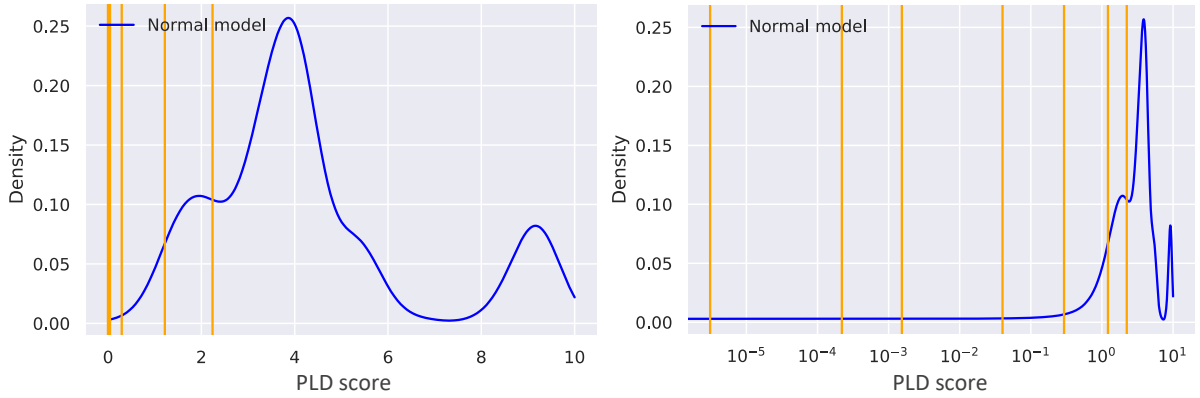


Figure 3: Density distribution of PLD score (blue curve) and thresholds inferred from it (orange lines). Left: PLD score linear scale. Right: PLD score log scale. In both graphs, from right to left: 7 thresholds for categories 1 (10th percentile) to 7 (1st 10,000,000-quantile).

2.3 Distributed online training of PLD model

To compute PLD scores and build their distribution, we need to know observed process launch counts. These statistics are obtained from a number of client machines which monitor their process launch events and contribute their local knowledge to build a global PLD model in a federated manner. More specifically, the PLD model is based on an aggregated table containing counts for all process launches over a chosen time period. Each distinct process launch (a unique pair of parent and child processes) that occurred on a specific client machine is represented by a row in this table. Each row has a counter that records the number of times a specific process launch event was observed over the specific time period. Each client *i* maintains such a table, shown in Table 2, which can be considered the local PLD model PLD_i.

Parent process	Child process	Join call
$proc_{P1}$	$proc_{C1}$	$call_{join}(proc_{P1}, proc_{C1})_i$
$proc_{P2}$	$proc_{C2}$	$call_{join}(proc_{P2}, proc_{C2})_i$
….	….	….
$proc_{Pn}$	$proc_{Cn}$	$call_{join}(proc_{Pn}, proc_{Cn})_i$

Table 2: Local table for PLD model for client *i*

The global PLD model PLD_{global} is built using these clients-specific PLD_i. Every client sends their local model PLD_i^t, representing the 'most recent' process launch behaviour in their machine, at regular time intervals *t* to an aggregator, for example, a security monitoring service provider. As the new local PLD models are received from the clients, the aggregator combines them into the global PLD model PLD_{global}^{t+1} to be used by every client at the next model update point (*t+1*). PLD_{global} contains all the process launch events that occurred on any of the clients contributing to building the global model. Let's call PC_i the set of unique ordered (parent process, child process) pairs contained in PLD_i. The entries in PLD_{global} model are defined by PC_{global} as follows:

$$PC_{global} = \bigcup_{i=1}^{N} PC_i$$

where *N* is the total number of clients participating in the training process.

Then, the count of events in PLD_{global} is computed as the sum over all the clients:

$$call_{join}(proc_A, proc_B)_{global} = \sum_{i=1}^{N} call_{join}(proc_A, proc_B)_i$$

Using PLD_{global} we can obtain all the values required to compute PLD scores:

- $call_{parent}(proc_A) = \sum_{proc_i \in PC_{global}} call_{join}(proc_A, proc_i)_{global}$
- $call_{child}(proc_B) = \sum_{proc_i \in PC_{global}} call_{join}(proc_i, proc_B)_{global}$
- $call_{total} = \sum_{proc_i \in PC_{global}} \sum_{proc_j \in PC_{global}} call_{join}(proc_i, proc_j)_{global}$

It is worth noting that the PLD model training setup is in-between the data aggregation and federated learning scenarios described in Section 2.2. On the one hand, the global model PLD_{global}^{t-1} is not required for computing the local models PLD_i^t, those are recomputed from scratch by each client at each model update point. On the other hand, instead of sending individual process launch events to the aggregator, the clients carry out certain data processing locally and submit only their aggregated PLD tables. We also note that the global models at successive time points *t* and (*t*+1) are aggregated from the local models generated by the same clients, so some indirect dependency and similarities exist between PLD_{global}^t and PLD_{global}^{t+1}, which is important for designing the poisoning attacks, as we will later see.

2.4 Poisoning PLD model

We consider an attack scenario in which an adversary has found a vulnerability and can compromise and control a common process $proc_M$ in a victim machine. The adversary wants to perform malicious actions by launching a target child process $proc_T$ from $proc_M$. $proc_M$ is a common parent process, meaning that it often launches various child processes, and $proc_T$ is a common child process, meaning that it is often launched by other processes. However, $proc_M$ usually does not launch $proc_T$. The $PLD\_score(proc_M, proc_T)$ is low and it belongs to the low tail of the PLD score distribution. Consequently, the anomaly category for $(proc_M, proc_T)$ is high (e.g. 5 - 7) according to the PLD model, and this process launch event will be considered suspicious if observed on a machine protected by the PLD model.

The adversary wants to avoid the detection of the $(proc_M, proc_T)$ event as a possible attack by making it look 'less anomalous' with respect to the global PLD model. More specifically, the goal of the adversary is to decrease the anomaly category for the $(proc_M, proc_T)$ event via a poisoning attack against the global PLD model.

We will now elaborate the generic adversary model introduced in Section 1.3. The goal of the poisoning attack is to introduce a backdoor in the global PLD model: process launch event $(proc_M, proc_T)$ must be assigned to an anomaly category with a low index. This backdoor must be integrated in the global PLD model built at the next time interval PLD_{global}^{t+1}. The adversary controls the client *m*. To introduce the backdoor, the adversary wants to craft a poisoned local model PLD_m^t, which – when aggregated with the local models of the other clients $PLD_{i \neq m}^t$ into PLD_{global}^{t+1} – will lead PLD_{global}^{t+1} to output a lower anomaly category for $(proc_M, proc_T)$ than PLD_{global}^t.

The adversary has the following capabilities to realize the poisoning attack:

- Access to one compromised client in the online distributed learning process
- Injection of one local PLD model PLD_m^t that will be used to build PLD_{global}^{t+1}
- Access to the global PLD model computed at the previous time point: PLD_{global}^t (strong adversary described in Section 1.3.3)

The adversary does not know the local PLD models $PLD_{i\neq m}^{t}$ that will be sent by the other clients of the training process.

We propose and will analyse three different approaches to poisoning the PLD model to achieve the adversary's goal. These approaches are described and evaluated in detail in Section 3.

1. **Increase target process launch count**: Increase $PLD\_score(proc_M, proc_T)$ by increasing $call_{join}(proc_M, proc_T)$
2. **Decrease parent or child process launch count**: Increase $PLD\_score(proc_M, proc_T)$ by decreasing $call_{parent}(proc_M)$ or $call_{child}(proc_T)$
3. **Inject rare process launch events unrelated to the attack:** Decrease anomaly category thresholds by filling in the low tail of the *PLD_score* distribution with process launch events unrelated to the planned attack

3. Model poisoning approaches

3.1 Experimental setup

We use a research setup where 247 clients collaborate to train a global PLD model in an online distributed manner. Each client records their process launch events and retrains their local PLD model PLD_i^t once a day using the process launch records. The latest local PLD model PLD_i^t of each client is sent to the aggregator for combining as presented in Section 2.3. The result of this aggregation is the global PLD model PLD_{global}^{t+1}, which is sent back to each of the 247 clients participating in the training.

We took the local PLD models of each client at time (t-1) in order to build PLD_{global}^{t}. This model is based on 22,478,835 process launch events in total, which are divided into 41,975 distinct process launch events (unique ordered pairs of processes). The client, which reported the largest number of events, has 5,610,402 process launches in its local PLD model, accounting for one quarter of all the process launch events used for the global PLD model. On average, one client reported 91,007 events. Computing the PLD score for every process launch and modelling their distribution, we obtained the threshold values shown in Table 3, which define the anomaly categories:

Threshold	PLD score threshold value	Anomaly category
10th percentile	2.237847	1
1st percentile	1.216247	2
1st 1000-quantile	0.293015	3
1st 10,000-quantile	0.040073	4
1st 100,000-quantile	0.001532	5
1st 1,000,000-quantile	0.000219	6
1st 10,000,000-quantile	0.000003	7
-	-	8

Table 3: PLD score thresholds in PLD_{global}^{t} for different anomaly categories.

PLD_{global}^{t}, its thresholds and PLD scores are used by the adversary to construct malicious entries to include into PLD_m^t in order to poison PLD_{global}^{t+1}. To evaluate the three attack approaches introduced in Section 2.4 under different conditions, we randomly selected several pairs representing anomalous process launch events with respect to PLD_{global}^{t}. For each of the selected pairs $(proc_M, proc_T)$, we tried to decrease its anomaly category. Table 4 presents the ten anomalous process launches we randomly selected and their characteristics with respect to PLD_{global}^{t}. These pairs are rarely observed

(168 is the greatest value, but most of the values are single-digit), they have low PLD scores and consequently belong to the high-index anomaly categories from 5 to 7. We will use these pairs for the attack analysis in Sections 3.2 and 3.3.

$proc_M$	$proc_T$	$call_{join}$	$call_{parent}$	all_{child}	PLD_score	An. Cat.
postgres.exe	*conhost.exe*	3	384553	2624273	0,000067	7
fshoster32.exe	*chrome.exe*	1	114103	274529	0,000725	6
Microsoft.Nav.Client.exe	*cmd.exe*	1	3673	3418012	0,001808	5
SearchIndexer.exe	*WerFault.exe*	1	976465	11982	0,001940	5
Dropbox.exe	*sc.exe*	2	8712	1624257	0,003193	5
services.exe	*sc.exe*	168	605335	1624257	0,003841	5
chrome.exe	*rundll32.exe*	3	274355	53447	0,004614	5
svchost.exe	*AcroRd32.exe*	31	2441976	39003	0,007319	5
SQLAGENT.EXE	*conhost.exe*	2	1961	2624273	0,008780	5
svchost.exe	*POWERPNT.EXE*	4	2441976	2426	0,015215	5

Table 4: 10 randomly selected anomalous process launches with their statistics in PLD^t_{global}.

3.2 'Increase target process launch count' approach

The first way to decrease the anomaly category of the $(proc_M, proc_T)$ pair is by increasing its PLD score. Knowing the threshold values of the global PLD model and defining a target anomaly category (e.g., 2), we can infer the minimum PLD score required for our pair to fall in the desired category.

3.2.1 Attack description and implementation

The first proposed approach to increase $PLD\_score(proc_M, proc_T)$ is by increasing $call_{join}(proc_M, proc_T)$. Given that we know PLD^t_{global}, we can compute the target score PLD_{target} to reach in order to fall in the desired anomaly category. Then for the poisoning attack to succeed, we need to satisfy the following inequality:

$$PLD\_score(proc_M, proc_T) > PLD_{target}$$

Replacing $PLD\_score(proc_M, proc_T)$ by its formula, we obtain the following inequality:

$$\frac{\frac{call_{join}(proc_M, proc_T) + \alpha_1}{call_{total}}}{\frac{call_{parent}(proc_M) + \alpha_2}{call_{total}} \times \frac{call_{child}(proc_T) + \alpha_3}{call_{total}}} > PLD_{target}$$

(For simplicity, we ignore the β_i since they are typically negligible compared to real-life large $call_{total}$ values.)

Now we can compute the minimum $call_{join}(proc_M, proc_T)$ required to satisfy this inequality. We set PLD_{target} as the targeted anomaly category threshold to exceed and we take the values for $call_{parent}$, $call_{child}$ and $call_{total}$ from PLD^t_{global}. $call_{join}(proc_M, proc_T)$ will be the unknown value of the inequality that we want to solve. Since $call_{join}(proc_M, proc_T)$ is included in $call_{parent}$, $call_{child}$ and $call_{total}$, we express those values as functions of our unknown. We obtain:

- $call^{t+1}_{parent}(proc_M) = call^t_{parent}(proc_M) + call^{t+1}_{join}(proc_M, proc_T)$
- $call^{t+1}_{child}(proc_T) = call^t_{child}(proc_T) + call^{t+1}_{join}(proc_M, proc_T)$

- $call_{total}^{t+1} = call_{total}^{t} + call_{join}^{t+1}(proc_M, proc_T)$

Then the inequality in the unknown $call_{join}^{t+1}$ to solve is the following:

$$\frac{\frac{call_{join}^{t+1} + \alpha_1}{call_{total}^{t} + call_{join}^{t+1}}}{\frac{call_{prent}^{t}(proc_M) + call_{join}^{t+1} + \alpha_2}{call_{total}^{t} + call_{join}^{t+1}} \times \frac{call_{child}^{t}(proc_T) + call_{join}^{t+1} + \alpha_3}{call_{total}^{t} + call_{join}^{t+1}}} > PLD_{target}$$

With simple transformations, this inequality can be turned into a quadratic inequality in a single variable with 0 as the right-hand side. We can find the solutions of the corresponding quadratic equation, which give us the range(s) of values of $call_{join}^{t+1}$ that satisfy the target inequality. Then in the found range(s), we need to take the minimum positive value as our crafted $call_{join}^{t+1}(proc_M, proc_T)$ value to poison the global PLD model, and the poisoning attack is carried out by adding the following single record to the local PLD model PLD_m^t.

Parent process	Child process	Join call
$proc_M$	$proc_T$	$call_{join}^{t+1}(proc_M, proc_T)$

Three technical remarks:

- We look for the minimum positive value of $call_{join}^{t+1}(proc_M, proc_T)$ in order to keep the adversarial action as modest as possible, the reasons for which were discussed in Section 1.3.1.
- While in principle quadratic equations may not have real-valued roots or the range(s) discussed above may not have positive values, it is easy to see that acceptable values of $call_{join}^{t+1}$ can always be found in our specific case. In particular, if $call_{join}^{t+1}$ is very large and a dominant part of all the other values, the left-hand side of the inequality above will be arbitrarily close to 1, which is a fairly high value for PLD scores and nearly always guarantees a 'non-suspicious' anomaly category. So, acceptable solutions do exist and the adversary just needs to find the smallest among those.
- The formulae for $call_{parent}^{t+1}(proc_M)$, $call_{child}^{t+1}(proc_T)$ and $call_{total}^{t+1}$ above ignore the contributions to the model update from the benign clients, which the adversary, of course, cannot know and control. The same applies to the anomaly category threshold values. This is where the adversary needs to rely on the note in the end of Section 2.3 on similarities between PLD_{global}^{t} and PLD_{global}^{t+1}. We show a practical approach to handling this issue in the next subsection.

3.2.2 Experimental results

We compute our solution to $call_{join}^{t+1}(proc_M, proc_T)$ using the $call_{parent}(proc_M)$, $call_{child}(proc_T)$ and $call_{total}$ values from PLD_{global}^{t}. The anomaly category threshold that we choose for PLD_{target} to reach is also taken from PLD_{global}^{t}, which is the most recent global PLD model available to the adversary. However, our attack targets PLD_{global}^{t+1} and we do not know its parameters. While we invoke the assumption about PLD_{global}^{t} and PLD_{global}^{t+1} similarity, the two models have different $call_{parent}(proc_M)$, $call_{child}(proc_T)$, $call_{total}$, and category threshold values. In our experiments, in order to deal with those differences between two consecutive PLD models and increase the success chance for the attack, we heuristically increase the computed $call_{join}^{t+1}(proc_M, proc_T)$ values by 20%. As a part of the analysis, we compare the performance of the poisoning attacks using the exact

computed values (simply ignoring the differences between consecutive models) and the values heuristically increased by 20%.

In the experiments, we computed the required $call_{join}^{t+1}(proc_M, proc_T)$ values to be added to the local model PLD_m^t in order to move our ten selected process launch pairs $(proc_M, proc_T)$ (see Table 4) to the anomaly categories 1, 2 or 3 in PLD_{global}^{t+1}. The following three tables report the results of those experiments, including the computed $call_{join}(proc_M, proc_T)$ values and their 20% increased versions, the PLD scores and the anomaly categories we obtain in PLD_{global}^{t+1} as the results of our poisoning attacks. The target PLD scores used for the computations are the threshold values to exceed in order to reach the anomaly categories 1, 2 and 3, which are equal to respectively 2.237847, 1.216247 and 0.293015 at time *t*, as presented in Table 2.

$proc_M$	$proc_T$	$call_{join}$		PLD_score		Category	
		Base	+20%	Base	+20%	Base	+20%
postgres.exe	*conhost.exe*	**145080**	**174097**	2,213	2,496	1	1
fshoster32.exe	*chrome.exe*	**3244**	**3893**	2,012	2,397	2	1
Microsoft.Nav.Client.exe	*cmd.exe*	**1895**	**2274**	2,245	2,521	1	1
SearchIndexer.exe	*WerFault.exe*	**1291**	**1550**	2,169	2,553	2	1
Dropbox.exe	*sc.exe*	**1680**	**2017**	2,232	2,595	2	1
services.exe	*sc.exe*	**126728**	**152107**	2,180	2,497	2	1
chrome.exe	*rundll32.exe*	**1507**	**1808**	2,105	2,508	2	1
svchost.exe	*AcroRd32.exe*	**12563**	**15082**	2,146	2,454	2	1
SQLAGENT.EXE	*conhost.exe*	**692**	**830**	2,238	2,550	2	1
svchost.exe	*POWERPNT.EXE*	**775**	**930**	2,184	2,497	2	1

Table 5: $call_{join}$ added to PLD_m^t for *'increase process launch count'* poisoning attack, resulting *PLD scores* and anomaly categories returned by poisoned PLD_{global}^{t+1}. 'Base' results for computed values of $call_{join}$ and '+20%' for increased values. Target anomaly category = 1 (PLD_{target} = 2.237847 from PLD_{global}^t).

$proc_M$	$proc_T$	$call_{join}$		PLD_score		Category	
		Base	+20%	Base	+20%	Base	+20%
postgres.exe	*conhost.exe*	**65266**	**78320**	1,200	1,394	2	2
fshoster32.exe	*chrome.exe*	**1731**	**2077**	1,093	1,306	3	2
Microsoft.Nav.Client.exe	*cmd.exe*	**833**	**1000**	1,221	1,413	2	2
SearchIndexer.exe	*WerFault.exe*	**668**	**802**	1,179	1,400	2	2
Dropbox.exe	*sc.exe*	**838**	**1006**	1,213	1,431	2	2
services.exe	*sc.exe*	**60345**	**72448**	1,181	1,384	2	2
chrome.exe	*rundll32.exe*	**805**	**967**	1,143	1,368	3	2
svchost.exe	*AcroRd32.exe*	**5917**	**7107**	1,164	1,361	3	2
SQLAGENT.EXE	*conhost.exe*	**323**	**388**	1,217	1,420	2	2
svchost.exe	*POWERPNT.EXE*	**365**	**439**	1,187	1,388	2	2

Table 6: $call_{join}$ added to PLD_m^t for *'increase process launch count'* poisoning attack, resulting *PLD scores* and anomaly categories returned by poisoned PLD_{global}^{t+1}. 'Base' results for computed values of $call_{join}$ and '+20%' for increased values. Target anomaly category = 2 (PLD_{target} = 1.216247 from PLD_{global}^t).

$proc_M$	$proc_T$	$call_{join}$		PLD_score		Category	
		Base	+20%	Base	+20%	Base	+20%
postgres.exe	*conhost.exe*	**13683**	**16420**	0,289	0,344	3	3
fshoster32.exe	*chrome.exe*	**410**	**492**	0,263	0,316	4	3
Microsoft.Nav.Client.exe	*cmd.exe*	**171**	**205**	0,294	0,350	3	3
SearchIndexer.exe	*WerFault.exe*	**154**	**185**	0,285	0,341	3	3
Dropbox.exe	*sc.exe*	**187**	**225**	0,293	0,350	3	3
services.exe	*sc.exe*	**13022**	**15660**	0,284	0,339	3	3
chrome.exe	*rundll32.exe*	**189**	**228**	0,275	0,331	4	3
svchost.exe	*AcroRd32.exe*	**1251**	**1508**	0,280	0,334	4	3
SQLAGENT.EXE	*conhost.exe*	**68**	**82**	0,295	0,352	3	3
svchost.exe	*POWERPNT.EXE*	**76**	**92**	0,287	0,343	3	3

Table 7: $call_{join}$ added to PLD_m^t for *'increase process launch count'* poisoning attack, resulting *PLD scores* and anomaly categories returned by poisoned PLD_{global}^{t+1}. 'Base' results for computed values of $call_{join}$ and '+20%' for increased values. Target anomaly category = 3 (PLD_{target} = 0.293015 from PLD_{global}^t).

We can see that the attack succeeds in around one half of the cases when using the exact $call_{join}$ values computed using PLD_{global}^t, and the success rate is higher when targeting the anomaly categories 2 and 3. The attack always succeeds when taking the increased (20%) values, as we reach the targeted anomaly category in all the 30 cases we tested. So, using the proposed technique, the adversary achieves their goal despite the lack of knowledge and control over the training contributions of the benign clients.

Comparing the obtained PLD scores to the ones we targeted, we observe that the errors – explained by the differences between the two consecutive global PLD models at times *t* and (*t*+1) – do not exceed 10%. While the errors show that the parameters of the global PLD model do change over time, it appears that two consecutive global PLD models are still similar enough for the adversary's purposes (which technically come down to predicting accurately the $call_{join}$ value to inject for poisoning the global model). The 10% bound on the observed errors explains why the attacks using the exact $call_{join}$ values fail sometimes and the attacks using the increased values always succeed.

We also can see that the $call_{join}$ values to inject vary significantly across the studied cases (combination of the ten selected process launch events and the three target anomaly categories). Predictably, reaching a 'less suspicious' anomaly category (e.g., 1 vs. 2) always required higher $call_{join}$ values. It is more interesting to note that for different process launch events, the required $call_{join}$ values to inject can differ by two orders of magnitude. When either the parent process or the child process is not very frequently used and $call_{parent}$ or $call_{child}$ is in the range of tens of thousands, the adversary needs to inject only a few hundreds or a few thousands of process launch events in order to succeed. The attack can be carried out stealthily in this case since a single client reports 80,000 process launches on average and injecting a few thousands more events may not be easily noticed. However, when both parent and child processes are popular, that is, both $call_{parent}$ and $call_{child}$ are high numbers (in the range of millions), the values of $call_{join}$ start exceeding 100,000 (when targeting the anomaly category 1). Such massive injections to local models are, of course, easier to detect.

Takeaways: To summarize the experimental results, we see that the '*increase process launch count*' poisoning attack is very accurate and effective for reaching the adversarial goals. The success rate is 100% when we conservatively increase the computed $call_{join}$ values by 20%. Furthermore, in many

cases the attack requires the adversary to inject relatively small numbers of process launch events, which makes it stealthy.

3.3 'Decrease parent or child process launch count' approach

3.3.1 Attack description and implementation

The second proposed approach to increasing $PLD\_score(proc_M, proc_T)$ is based on decreasing the values of $call_{parent}(proc_M)$ or $call_{child}(proc_T)$. This attack is challenging because it is not possible for the adversary to prevent the benign clients from reporting process launches involving $proc_M$ or $proc_T$. One solution, however, is to insert illegal inputs in the controlled local PLD model PLD_m^t. We can decrease $call_{parent}(proc_M)$ and $call_{child}(proc_T)$ in the global PLD model by inserting to PLD_m^t negative counts for process launch events involving $proc_M$ as a parent and $proc_T$ as a child.

In the analysis, we start in the same way as in Section 3.2. Given that we know PLD_{global}^t and its anomaly category threshold values, we can infer the PLD_{target} value sufficient for the PLD score of $(proc_M, proc_T)$ to exceed in order to fall in the desired anomaly category. The starting inequality is exactly the same as in Section 3.2.1:

$$\frac{\frac{call_{join}(proc_M, proc_T) + \alpha_1}{call_{total}}}{\frac{call_{parent}(proc_M) + \alpha_2}{call_{total}} \times \frac{call_{child}(proc_T) + \alpha_3}{call_{total}}} > PLD_{target}$$

In this attack, we keep $call_{join}(proc_M, proc_T)$ as a fixed value taken from PLD_{global}^t and we consider either $call_{parent}^{t+1}(proc_M)$ or $call_{child}^{t+1}(proc_T)$ as the unknown to solve the inequality for. In both cases, we need to express $call_{total}^{t+1}$ as a function of the unknown (with a caveat similar to the one discussed in the very end of Section 3.2.1):

- $call_{total}^{t+1} = call_{total}^t + call_{parent}^{t+1}(proc_M)$, when $call_{parent}^{t+1}(proc_M)$ is the unknown
- $call_{total}^{t+1} = call_{total}^t + call_{child}^{t+1}(proc_T)$, when $call_{child}^{t+1}(proc_T)$ is the unknown

Considering the case when $call_{parent}^{t+1}(proc_M)$ is the unknown, the above inequality can be transformed into the following inequality in $call_{parent}^{t+1}(proc_M)$:

$$call_{parent}^{t+1}(proc_M) <$$

$$< \frac{call_{total}^t \times (call_{join}^t + \alpha_1) - PLD_{target} \times (call_{parent}^t(proc_M) + \alpha_2) \times (call_{child}^t(proc_T) + \alpha_3)}{PLD_{target} \times (call_{child}^t(proc_T) + \alpha_3) - (call_{join}^t + \alpha_1)}$$

So, assuming similarity of PLD_{global}^t and PLD_{global}^{t+1} if only the benign clients contribute to the model update, we can find the maximum acceptable value of $call_{parent}^{t+1}(proc_M)$, that is, the maximum number of events when $proc_M$ starts other processes as reported by all the clients in the global PLD model update for the poisoning attack to succeed. In the same way, we can find $call_{child}^{t+1}(proc_T)$.

Since the PLD score of $(proc_M, proc_T)$ in PLD_{global}^t is too low, we expect the found values of $call_{parent}^{t+1}(proc_M)$ and $call_{child}^{t+1}(proc_T)$ to be lower than the values of $call_{parent}^t(proc_M)$ and $call_{child}^t(proc_T)$ respectively. Thus, the poisoning attack needs to decrease one of these two values, which can be achieved by creating a fake process $proc_F$ used only in fake process launch events $(proc_F, proc_T)$ and $(proc_M, proc_F)$. Then the adversary needs to satisfy one of the following two conditions where both target values will be negative:

- $call^{t+1}_{join}(proc_M, proc_F) = call^{t+1}_{parent}(proc_M) - call^{t}_{parent}(proc_M)$
- $call^{t+1}_{join}(proc_F, proc_T) = call^{t+1}_{child}(proc_T) - call^{t}_{child}(proc_T)$

So, the poisoning attack can be carried out by adding one of the following records to the local PLD model PLD^t_m. Since the adversary cannot hope for stealthiness in this attack, they are essentially free in choosing one of these two options.

Parent process	Child process	Join call
$proc_F$	prc_T	$call^{t+1}_{join}(proc_F, proc_T)$

Parent process	Child process	Join call
$proc_M$	$proc_F$	$call^{t+1}_{join}(proc_M, proc_F)$

3.3.2 Experimental results

We run the experiments in the way described in Section 3.2.2. To compute the target values of $call^{t+1}_{join}(proc_M, proc_F)$ and $call^{t+1}_{join}(proc_F, proc_T)$, we use $call_{parent}(proc_M)$, $call_{child}(proc_T)$ and $call_{total}$ from PLD^t_{global}. While in the previous subsection we considered injection of *either* fake parent *or* fake child records, in the experiments we inject them both, because that increases the attack effectiveness and because that is what real-life adversaries are likely to do. So, we inject both fake parent and child records with the computed $call^{t+1}_{join}$ values to the local PLD model PLD^t_m and evaluate the success of the attack, which is defined as moving our ten selected $(proc_M, proc_T)$ pairs (see Table 4) to the anomaly categories 1 or 3 in PLD^{t+1}_{global}. Unlike the previous attack, we do not apply any heuristic adjustments to $call^{t+1}_{join}(proc_M, proc_F)$ and $call^{t+1}_{join}(proc_F, proc_T)$ to account for the lack of our knowledge about PLD^{t+1}_{global}. The results show that such adjustments would not be helpful.

The following two tables report the results of the experiments. They show:

- the computed $call_{join}$ values, which tell how many process launch events we need to inject in the local PLD model for both $(\boldsymbol{proc_M}, proc_F)$ and $(proc_F, \boldsymbol{proc_T})$;
- the resulting $call_{parent}(proc_M)$ and $call_{child}(proc_T)$;
- the PLD scores and the anomaly categories we obtain in PLD^{t+1}_{global} as the results of our poisoning attacks.

The target PLD scores used for the computations are the threshold values to exceed in order to reach the anomaly categories 1 and 3, which are equal to respectively 2.237847 and 0.293015 at time *t*, as presented in Table 2.

$proc_M$	$proc_T$	$call_{join}$		PLD_{global}^{t+1}		PLD $score$	Cat.
		$proc_M$	$proc_T$	$call_{parent}$	$call_{child}$		
postgres.exe	*conhost.exe*	**-384542**	**-2624195**	5831	22816	0,739	3
fshoster32.exe	*chrome.exe*	**-114067**	**-274441**	6080	18491	0,200	4
Microsoft.Nav.Client.exe	*cmd.exe*	**-3671**	**-3415250**	-23	33909	-24,922	9
SearchIndexer.exe	*WerFault.exe*	**-975619**	**-11972**	36597	56	10,676	1
Dropbox.exe	*sc.exe*	**-8700**	**-1621940**	42	14643	68,698	1
services.exe	*sc.exe*	**-604296**	**-1621469**	21155	15114	11,246	1
chrome.exe	*rundll32.exe*	**-273790**	**-53337**	18831	422	8,457	1
svchost.exe	*AcroRd32.exe*	**-2433987**	**-38876**	96944	883	8,018	1
SQLAGENT.EXE	*conhost.exe*	**-1954**	**-2613973**	8	33038	152,20	1
svchost.exe	*POWERPNT.EXE*	**-2425361**	**-2410**	105570	7	109,60	1

Table 8: Negative $call_{join}$ added to PLD_m^t for *'decrease parent or child process launch count'* poisoning attack, resulting *PLD_score* and anomaly category returned by poisoned PLD_{global}^{t+1}. Target anomaly category = 1 (PLD_{target} = 2.237847 from PLD_{global}^{t}).

$proc_M$	$proc_T$	$call_{join}$		PLD_{global}^{t+1}		PLD $score$	Cat.
		$proc_M$	$proc_T$	$call_{parent}$	$call_{child}$		
postgres.exe	*conhost.exe*	**-384466**	**-2623673**	5907	23338	0,714	3
fshoster32.exe	*chrome.exe*	**-113821**	**-273850**	6326	19082	0,186	4
Microsoft.Nav.Client.exe	*cmd.exe*	**-3651**	**-3396898**	-3	52261	-124,45	9
SearchIndexer.exe	*WerFault.exe*	**-969997**	**-11903**	42219	125	4,147	1
Dropbox.exe	*sc.exe*	**-8618**	**-1606544**	124	30039	11,352	1
services.exe	*sc.exe*	**-597397**	**-1602945**	28054	33638	3,815	1
chrome.exe	*rundll32.exe*	**-270034**	**-52606**	22587	1153	2,581	1
svchost.exe	*AcroRd32.exe*	**-2380817**	**-38029**	150114	1730	2,650	1
SQLAGENT.EXE	*conhost.exe*	**-1903**	**-2545365**	59	101646	6,738	1
svchost.exe	*POWERPNT.EXE*	**-2314453**	**-2301**	216478	116	3,247	1

Table 9: Negative $call_{join}$ added to PLD_m^t for *'decrease parent or child process launch count'* poisoning attack, resulting *PLD_score* and anomaly category returned by poisoned PLD_{global}^{t+1}. Target anomaly category = 3 (PLD_{target} = 0.293015 from PLD_{global}^{t}).

We can see that the attack succeeds in 70% of the cases when targeting the anomaly category 1 and in 80% of the cases for the anomaly category 3. More importantly, we note that the returned categories are exactly the same, regardless of what our target category is, so the attack accuracy is not high.

The main challenge of this attack is that in order to succeed it must decrease $call_{parent}(proc_M)$ and $call_{child}(proc_T)$ of frequently used processes dramatically, by several orders of magnitude, e.g., from 1,000,000 to 1,000 or from 10,000 to 100. If the differences between $call_{parent}(proc_M)$ (respectively $call_{child}(proc_T)$) in PLD_{global}^{t} and PLD_{global}^{t+1} are large, it is difficult to compute the records to inject accurately, and then the attack either fails by not increasing the PLD score sufficiently or over-performs by increasing it too much. Even when we succeed in targeting the anomaly category 1, the PLD score we obtain is often much larger than the one we targeted (2.237847). For the (*Microsoft.Nav.Client.exe, cmd.exe)* pair, we decrease $call_{parent}(proc_M)$ so much that it becomes negative, which results in the negative PLD score and the highest anomaly category (9) assigned. Besides, the need for the malicious client to use large negative values in the injected records can make the sum of all the process launch events, which the client reports from its local model, negative.

Takeaways: To summarize the experimental results, we see that the '*decrease parent or child process launch count'* poisoning attack can be effective in some cases, but can also fail even if we inject both fake parent and fake child records at the same time. It succeeds in reaching the anomaly category 1 in many cases, but it is quite inaccurate for anomaly categories other than 1. Finally, the attack requires to inject large negative numbers of process launch events (in the range of millions sometimes) to reach its goal, which makes it easy to detect and prevent via input validation.

3.4 'Inject rare process launch events unrelated to the attack' approach

3.4.1 Attack description and implementation

Compared with the previous two poisoning attacks in Sections 3.2 and 3.3, the last attack we evaluate takes a different approach. Instead of modifying the PLD score of $(proc_M, proc_T)$ pairs, its goal is to decrease the threshold values for several anomaly categories. By moving an appropriate number of the threshold values below the value of $PLD\_score(proc_M, proc_T)$, we can 'lift' the $(proc_M, proc_T)$ pair to a less suspicious anomaly category without modifying the pair's PLD score. In order to decrease anomaly category threshold values, we need to drift and densify the PLD scores distribution towards low values.

For this attack, we look for processes $proc_P \neq proc_M$ and $proc_C \neq proc_T$ with high $call_{parent}(proc_P)$ and $call_{child}(proc_C)$ values respectively. Such processes combined in a process launch event will have a low PLD score, if $call_{join}(proc_P, proc_C)$ is low. Then if the PLD score of the event is lower than $PLD\_score(proc_M, proc_T)$, the event is a candidate for injection into PLD_m^t since it densifies the PLD score distribution in values lower than $PLD\_score(proc_M, proc_T)$ and will contribute to shifting the anomaly category threshold values below that score. So, we need to find as many such $(proc_P, proc_C)$ process pairs as we can and compute the maximum $call_{join}(proc_P, proc_C)$ value for each pair to maximize its 'threshold shifting' contribution while keeping its PLD score in PLD_{global}^t below $PLD\_score(proc_M, proc_T)$[3]. The maximum acceptable $call_{join}$ value is computed from the inequality defined in Section 3.2.1, we just need to reverse the inequality sign:

$$PLD\_score(proc_P, proc_C) < PLD\_score(proc_M, proc_T)$$

The computation is carried out essentially in the same way as in 3.2.1, by reducing the task to solving a quadratic equation.

All the found candidate process launch pairs, together with their computed maximum $call_{join}(proc_P, proc_C)$ values, are added to the local PLD model PLD_m^t. It is worth noting that none of the injected process launch pairs contain $proc_M$ as a parent process or $proc_T$ as a child process. This was done in order to avoid impacting $PLD\_score(proc_M, proc_T)$, increasing the chances that as many as possible anomaly category threshold values will be shifted below that score. This also hides the final goal of the poisoning attack because none of the process launch events we are injecting into the local (and then the global) model involve the malicious process launch operation that the adversary eventually wants to carry out and avoid detection.

Parent process	Child process	Join call
$proc_{P1}$	$proc_{C1}$	$call_{join}(proc_{P1}, proc_{C1})$
$proc_{P2}$	$proc_{C2}$	$call_{join}(proc_{P2}, proc_{C2})$

[3] For each process pair $(proc_P, proc_C)$, our attack injects $call_{join}$ process launch events $(proc_P \rightarrow proc_C)$, as recorded in Table 10.

....	....	....
$proc_{Pn}$	$proc_{Cn}$	$call_{join}(proc_{Pn}, proc_{Cn})$

Table 10: The local table of PLD_m^t.

In this model poisoning scenario, we do not aim to reach a specific anomaly category for our target process launch event since it may not be possible to fill the PLD score distribution with sufficiently many events having sufficiently low PLD score values. We rather want to evaluate what the lowest anomaly category that we can reach using this poisoning attack is, given that our target pair $(proc_M, proc_T)$ belongs initially to a specific anomaly category. We also want to investigate how many process launch events overall we need to inject to succeed with the attack. For the experiments, we selected two process pairs, highly relevant in the cybersecurity context, which belong to the categories 6 and 5. The cases of these two target pairs will be analysed separately, and the pairs are as follows:

$proc_M$	$proc_T$	$call_{join}$	$call_{parent}$	$call_{child}$	PLD_score	An. Cat.
fshoster32.exe	*chrome.exe*	1	114103	274529	0,00072	6
OfficeClickToRun.exe	*rundll32.exe*	1	18619	53447	0.02281	5

Table 11: Process pairs selected for experimenting with 'Inject rare process launch events unrelated to the attack' approach and their parameters in PLD_{global}^t.

3.4.2 Experimental results

We selected 100 processes with the highest $call_{parent}$ values and 100 processes with the highest $call_{child}$ values in PLD_{global}^t. We built all the possible 10,000 pairwise combinations and computed their PLD scores from PLD_{global}^t. As explained above, we kept only the pairs with PLD score lower than the PLD score of the target process launch pair and discarded the others. Then we computed the maximum acceptable $call_{join}$ values for each of the remaining process pairs in order to fill in the PLD score distribution of PLD_{global}^{t+1} with as many low PLD score values as possible. We note that those maximum $call_{join}$ values are computed from the parameters of PLD_{global}^t but with the goal of poisoning PLD_{global}^{t+1}. Since there is some difference between the two consecutive global models (see, e.g. the end of Section 3.2.1), we chose to be conservative and reduced the value of $call_{join}$ for each process launch pair that we injected by 10% to help ensure that the PLD scores of the injected pairs remain lower than the PLD score of the target process launch pair in PLD_{global}^{t+1}.

First, we took (*fshoster32.exe*, *chrome.exe*) as a target process launch pair. In PLD_{global}^t, it belongs to the anomaly category 6. We found 616 candidate process launch pairs for injection out of the 10,000 pairs tested. The sum of the $call_{join}$ values for all these 616 pairs was equal to 9,047 process launch events that we could use for increasing the density in the low value end of the PLD score distribution. The average of the $call_{join}$ values for all those pairs was 15. By injecting the process launch events to the local PLD model PLD_m^t, we succeeded in reducing the anomaly category of (*fshoster32.exe*, *chrome.exe*) from 6 to 4. The following table shows the statistics of the target process launch pair in the initial and poisoned PLD models:

PLD model	Poisoning events	$call_{join}$	$call_{parent}$	$call_{child}$	PLD_score	An. Cat.
PLD_{global}^t	-	1	114103	274529	0,00072	6
PLD_{global}^{t+1} (poisoned)	+9047 (616 unique pairs)	1	120147	292932	0.00065	4

Table 12: Comparison of PLD_{global}^t and PLD_{global}^{t+1} models for (*fshoster32.exe*, *chrome.exe*)

For the (*OfficeClickToRun.exe*, *rundll32.exe*) pair from the anomaly category 5, we found 3,258 candidate process launch pairs for injection (the same 10,000 pairs were tested). The sum of the

$call_{join}$ values for all these pairs was equal to 345,466 process launch events. The average of the $call_{join}$ values for all the found pairs was 106. By injecting the process launch events to the local PLD model PLD_m^t, we succeeded in reducing the anomaly category of (*OfficeClickToRun.exe*, *rundll32.exe*) from 5 to 2. The following table shows the statistics of the target process launch pair in the initial and poisoned PLD models:

$PLD\ model$	Poisoning events	$call_{join}$	$call_{parent}$	$call_{child}$	PLD_score	An. Cat.
PLD_{global}^t	-	1	18619	53447	0.02281	5
PLD_{global}^{t+1} (poisoned)	+345466 (3258 unique)	1	18740	53759	0.02305	2

Table 13: Comparison of PLD_{global}^t and PLD_{global}^{t+1} models for (*OfficeClickToRun.exe, rundll32.exe*)

As we can see, going from the anomaly category 6 to the anomaly category 4 was achieved by injecting some hundreds of process launch records, with a very modest average value of $call_{join}$. Since the clients report around 80,000 process launch events on average in one round of training, injecting 9,047 process launch events does not look as a serious anomaly. On the other hand, the effort to go from the anomaly category 5 to the anomaly category 2 required the injection of over 300,000 process launch events, which is significantly larger than the average of 80,000. We have to remember, however, that some benign clients report sometimes up to 5,000,000 process launches in one local model. Using such a client to perform the poisoning attack would certainly make the injection less noticeable.

The anomaly category thresholds selection for the PLD model is based on quantiles exponentially decreasing in size, as presented in Table 1 above. This means that for a target process pair to move to the anomaly category 1, we need roughly to inject 10% of the total number of process launch events in the global PLD model, injecting around 1% is sufficient for moving to the category 2, and so on. In our experimental setup, with over 22,000,000 process launch events in PLD_{global}^t, we must inject around 2,200,000 new events with low PLD scores to reach the anomaly category 1 and around 220,000 to reach the category 2.

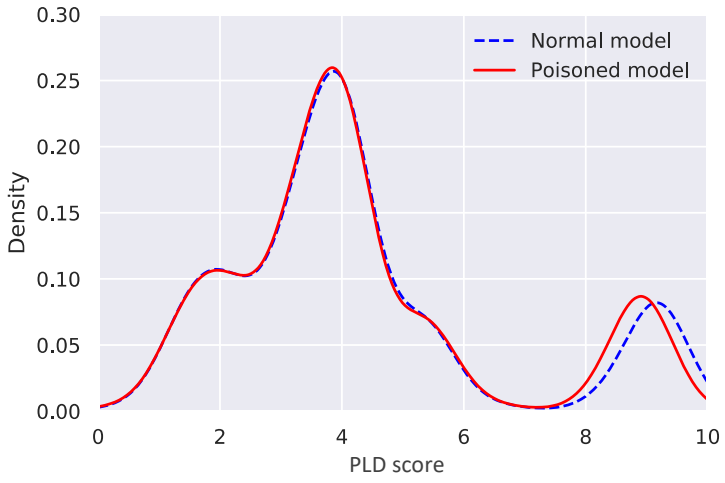


Figure 4: PLD score distribution for initial and poisoned PLD models in the experiment with (*fshoster32.exe*, *chrome.exe*). Target category is 4. 9047 process launch events injected in the poisoned model. Distributions are highly similar.

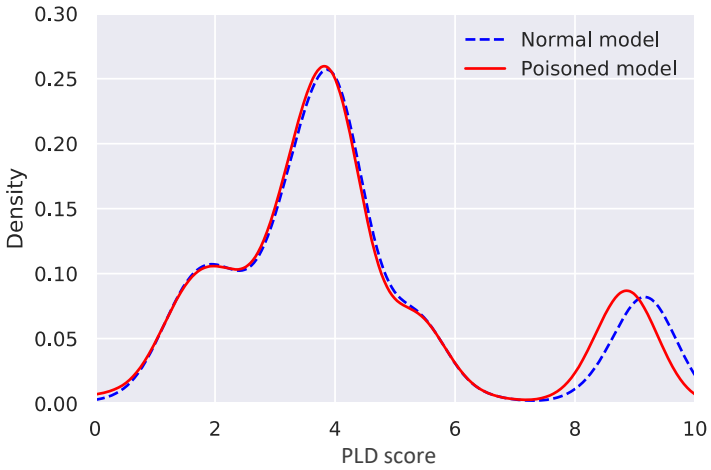


Figure 5: PLD score distribution for initial and poisoned PLD models in the experiment with (*OfficeClickToRun.exe*, *rundll32.exe*). Target category is 2. 345,466 process launch events injected in the poisoned model. Distributions are highly similar.

It is important to analyse the impact of our attacks on the PLD score distribution, comparing the distributions of the poisoned model PLD_{global}^{t+1} and the initial model PLD_{global}^{t}. Figures 4 and 5 depict these distributions for the two studied attacks. Looking at the place where we poison the model (PLD score values close to 0), we do not see any difference between the initial and poisoned models for the first attack, and the discrepancy is very modest for the second attack. The differences are much more noticeable in the range of PLD scores between 8 and 10, but those are certainly not due to the attacks and explained by concept drift processes captured by the retraining. This shows that our poisoning attacks are hard to detect by superficially comparing the PLD score distributions and the concept drift effects are more significant.

Taking into account that the numbers of injected events are quite small compared with the total training set size (over 22 million), this is not very surprising. The effects of the poisoning attacks can be better observed in Figures 6 and 7, where we zoom in the range of low PLD score values. We see that the first poisoning attack actually led to the increased density of the PLD score distribution by a factor of 2 in the interval close to 0 (Figure 6), and a greater change can be observed for the second attack, where the density increased by a factor of 30. While the former effect can still be explained by concept drift, the latter increase should raise an alarm. So, monitoring closely how the model changes in sensitive places certainly makes good sense.

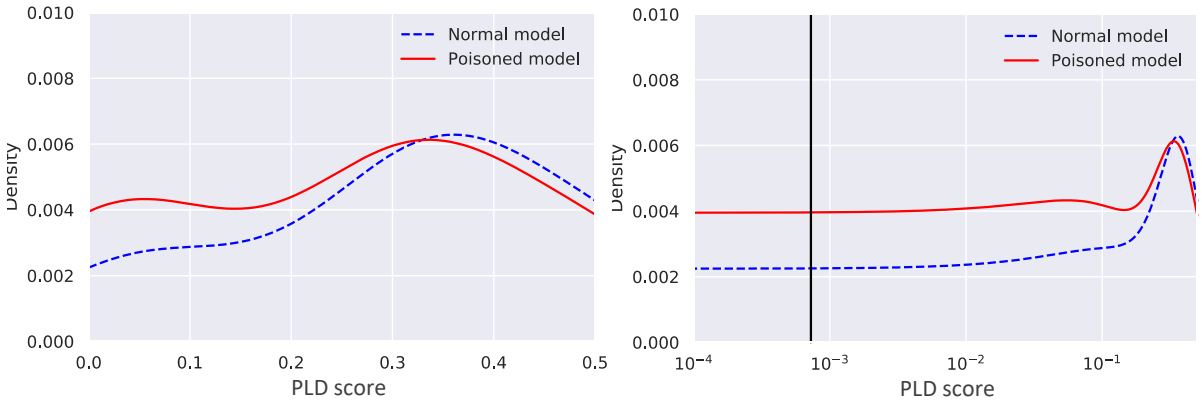


Figure 6: PLD score distribution for initial and poisoned PLD models in the interval close to 0 in the experiment with (*fshoster32.exe*, *chrome.exe*).

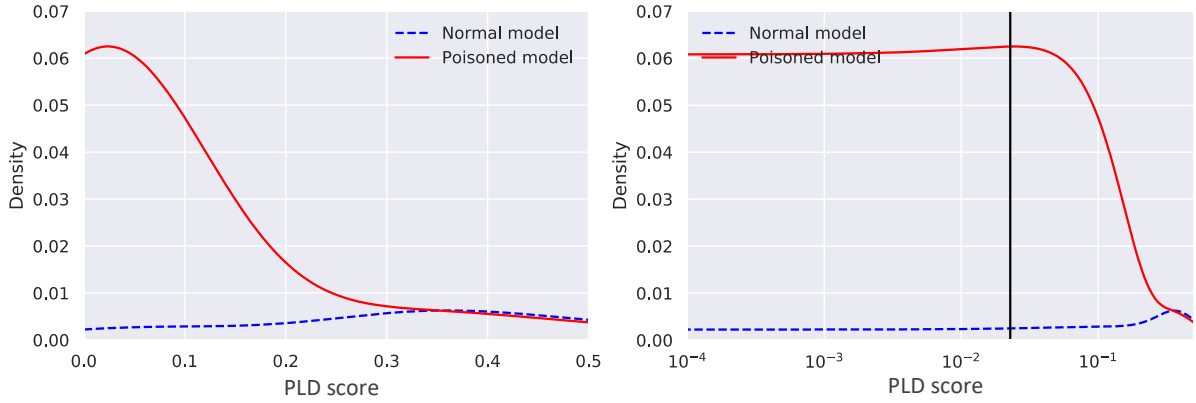


Figure 7: PLD score distribution for initial and poisoned PLD models in the interval close to 0 in the experiment with (*OfficeClickToRun.exe*, *rundll32.exe*).

Finally, we review the poisoning attacks effect in terms of the anomaly category threshold values in Figures 8 and 9. In both cases, we see that while the initial model PLD_{global}^{t} has well distributed anomaly category thresholds around PLD_{target}, the poisoned model PLD_{global}^{t+1} has several 'stacked' thresholds all below PLD_{target}. This essentially illustrates that the attacks successfully reached their goal and were near-optimal in poisoning the distribution-based model. All the thresholds moved below PLD_{target} are very close to that value and to each other, which shows that the attacks are effective at maximizing $call_{join}$ of the injected process launch pairs while keeping their PLD score below PLD_{target}. The anomaly category thresholds are close to each other because in the low value tail of the distribution there are many (injected) events close to PLD_{target}.

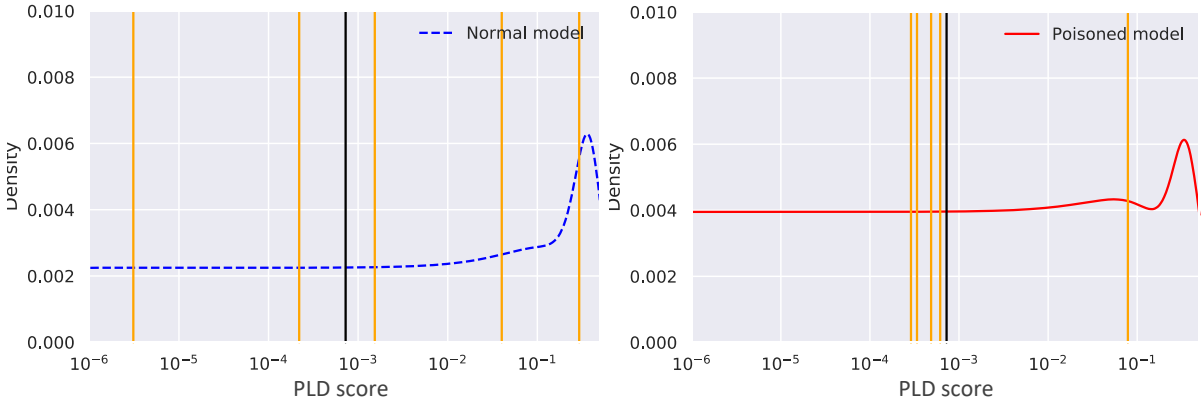


Figure 8: Anomaly category thresholds (orange lines) in initial and poisoned PLD models in the experiment with (*fshoster32.exe*, *chrome.exe*). Black line = PLD_{target}. <u>Left</u>: initial model has 2 thresholds (categories 7 and 6) below PLD_{target}. <u>Right:</u> poisoned model has 4 thresholds (categories 7, 6, 5 and 4) below PLD_{target}.

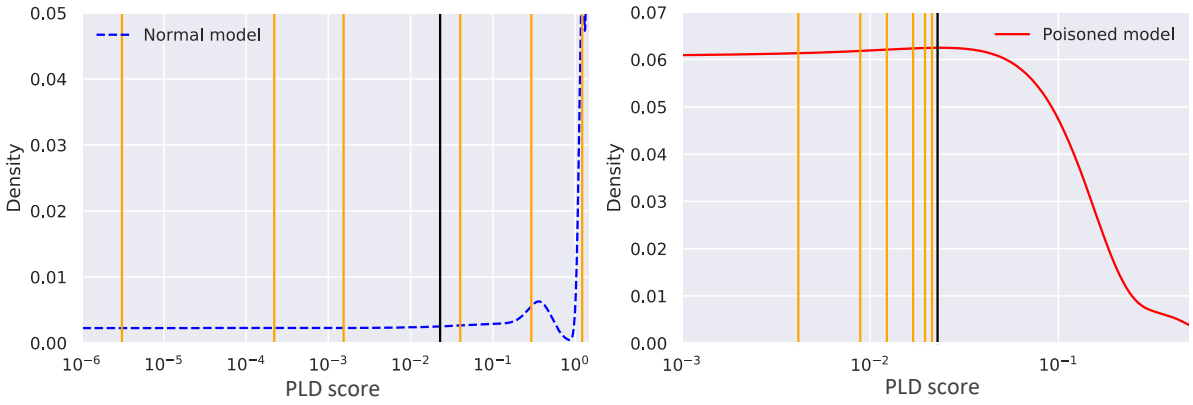


Figure 9: Anomaly category thresholds (orange lines) in initial and poisoned PLD models in the experiment with (*OfficeClickToRun.exe*, *rundll32.exe*). Black line = PLD_{target}. Left: initial model has 3 thresholds (categories 7, 6 and 5) below PLD_{target}. Right: poisoned model has 6 thresholds (categories 7 to 2) below PLD_{target}.

Takeaways: To summarize the experimental results, we see that the *'Inject rare process launch events unrelated to the attack'* poisoning approach is effective and accurate at reaching the adversarial goal of decreasing the anomaly category of a target process launch. At the same time, such an attack may not always be able to reach any anomaly category of the attacker's choice. Required volumes of injected process launch events are quite modest for this approach. For instance, tens of thousands of injected events should normally be sufficient if the target anomaly category is not 1 or 2. Such attacks are stealthy since they inject many process launch pairs with relatively low $call_{join}$ values (in the range of 10 - 100) and do not inject any pairs involving the target pair processes planned by the adversary for the actual cyberattack.

4. Defense recommendations

Learning from the poisoning approaches that we introduced and analysed, we present in this section a set of recommendations to counter model poisoning attacks that target AI-based systems using online distributed training and to mitigate their negative impact.

Checking input format and validity: Attacks following the '*decrease parent or child process launch count*' approach can be easily defeated by rejecting, prior to aggregation, any local records or models including negative $call_{join}$ values. When designing an online distributed training process, it is important to define formats and validity conditions for local models and data, listing all their parameters, properties, fields, etc. and defining acceptable ranges for those and other more complex conditions as appropriate. Each local model or data record must be checked against the defined conditions and rejected if any of the checks fail. One example of this procedure is sometimes referred to as 'bounded norm distance' validation, where any inputs to aggregation are verified to not exceed a pre-defined norm.

Normalisation of local models and data before aggregation: The aggregation of local models into the global model in PLD sums together unbounded values of $call_{join}$. While this is a typical aggregation approach in distributed and federated learning, it allows compromised or dishonest clients to have greater impact on the global model. A client can submit much higher values than the other clients in order to influence the global model in their interest [BVH+'18, BCM+'19]. One way to address the unequal contributions issue is to normalise local models (or data points) prior to aggregation in order to control their impact. For instance, in the PLD training process, we could require clients to submit

local frequencies of events instead of their counters. Then the sum of the $call_{join}$ values in each local model would be equal to 1, and all the clients would contribute to the global model equally. When such 'full normalisation' is not desirable, one should look for other, softer, ways to bound impact of individual contributions.

Outlier detection for poisoned local models and data: One typical approach to detection of poisoning attacks is based on identifying local models or data points that deviate too much from the majority of local models or data points submitted for aggregation. For example, a clustering algorithm can be applied to local models to identify outliers, that is, models which are far from all the obtained clusters [STS'16]. This defence approach is effective when local data points, which client contributions are based on, are independent and identically distributed (i.i.d.), causing local models to be similar. However, if the i.i.d. assumption does not apply to local data points across the clients, outlier detection methods will not work well, producing numerous mistakes due to the high variance of the client contributions.

For the PLD model training, we analysed the PLD score distribution of different local models. The PLD score distribution for the local models from four different clients is depicted in Figure 10. We clearly see that the local models vary widely, and the i.i.d. assumption is likely not valid in the case of the PLD distributed training. So, outlier detection-based methods are unlikely to be effective in this case.

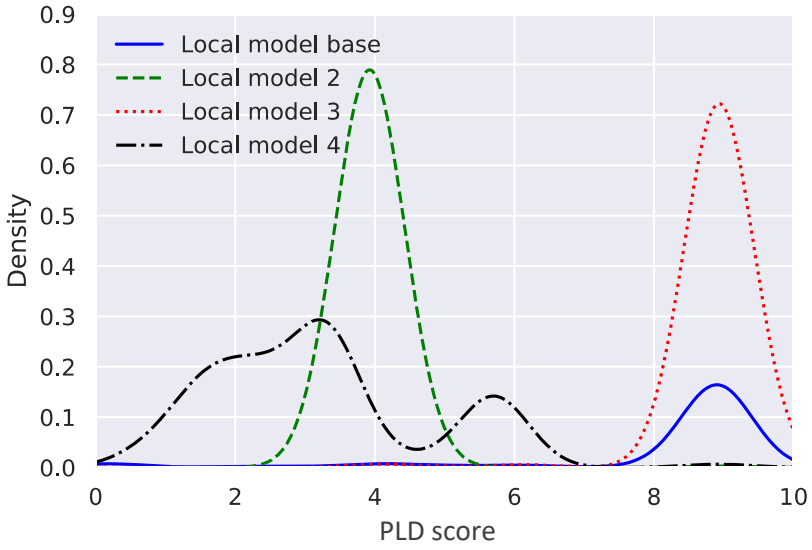


Figure 10: PLD score distribution for local models of 4 different clients.

Rejecting local models with a large distance to the global model: Another approach to detecting and removing poisoned local models is based on computing their distances to the global model [KSL'18, PMG+'18, SKL'17]. If the distance from a local model to the global model is too high, the local model is rejected. One challenge of this approach is with defining an appropriate distance function. A potentially greater limitation is that such techniques assume that all benign local models are close to the global model. Thus, similarly to the case of outlier detection-based methods, the i.i.d. assumption seems to be required here. If we compare the PLD score distribution of the four local models in Figure 10 and the PLD score distribution of the global model in Figure 3, we can see that all the local models are significantly different from the global model.

Detecting abnormal evolution of local models or data points over time: Instead of comparing local models or data points from different clients among each other or with the global model, usually ineffective if the i.i.d. assumption does not hold, we can monitor the evolution of individual local

models or data over time. The underlying assumption for such approaches is that the local models or data points produced by the same benign client over time are likely to be similar. So, a poisoned local model would likely differ significantly from the local models submitted by that client in the past. We can leverage the online training aspect of the PLD method for monitoring evolution of the local PLD models.

To validate this approach, we randomly selected a client. In its 'last good' (or normal) local model, the client reported 556 process launch pairs and 93,423 process launch events. We injected the poisoned process launch events required for carrying out the first and the third poisoning approaches into the client's local model. We then compared the PLD score distributions of the 'last good' model and the poisoned version in order to see how easy it is to detect the poisoning operations. We tested the following four attack scenarios:

- '*increase target process launch count*' for the (*postgres.exe, conhost.exe*) pair
- '*increase target process launch count*' for the (*fshoster32.exe, chrome.exe*) pair
- '*inject rare process launch events*' for the (*fshoster32.exe, chrome.exe*) pair
- '*inject rare process launch events*' for the (*OfficeClickToRun.exe, rundll32.exe*) pair

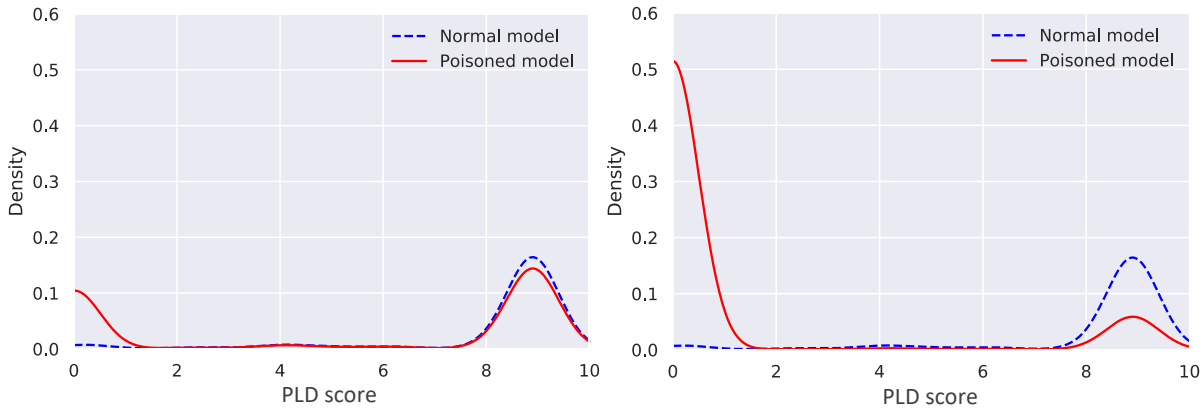


Figure 11: PLD score distributions for normal and poisoned local models in '*increase target process launch count*' attack for (*postgres.exe, conhost.exe*). Left: target anomaly category = 3 (13,683 process launch events injected). Right: target anomaly category = 1 (145,920 process launch events injected).

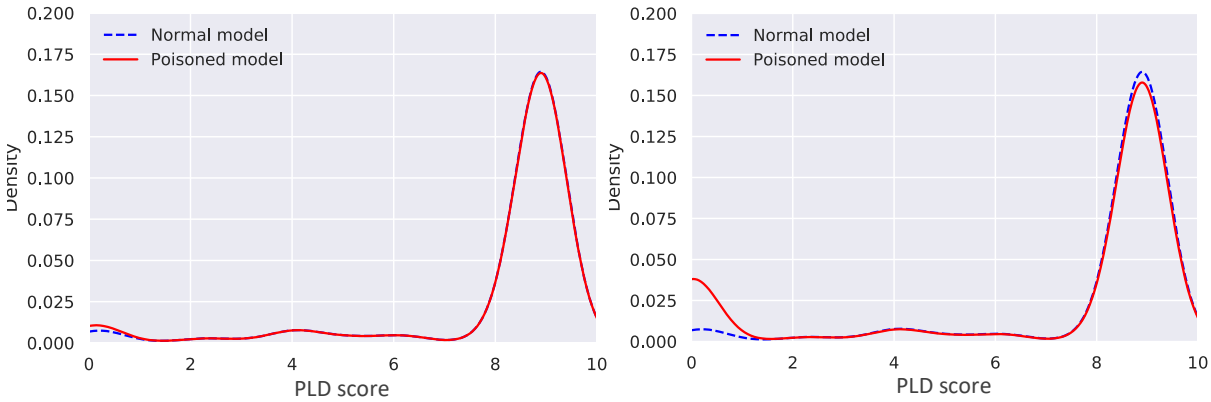


Figure 12: PLD score distributions for normal and poisoned local models in '*increase target process launch count*' attack for (*fshoster32.exe, chrome.exe*). Left: target anomaly category = 3 (410 process launch events injected). Right: target anomaly category = 1 (3,244 process launch events injected).

Comparing the PLD score distributions of the normal and poisoned local models for the '*increase target process launch count*' attack in Figures 11 and 12, we clearly see differences in low PLD score areas. The poisoned models have higher densities in those areas, especially when targeting the anomaly category 1, when the attack requires to inject high numbers of process launch events. This indicates that poisoned local PLD models can often be detected by comparing those with earlier local models from the same clients. However, we also see in the left part of Figure 12 that some poisoning attacks that require to inject only small numbers of process launch events (hundreds in the considered case) are difficult to detect with model evolution monitoring techniques.

Similar observations can be made in the 'Inject rare process launch events unrelated to the attack' case, presented in Figure 13. These poisoning attacks also resulted in significantly increased distribution densities in the area of low PLD scores.

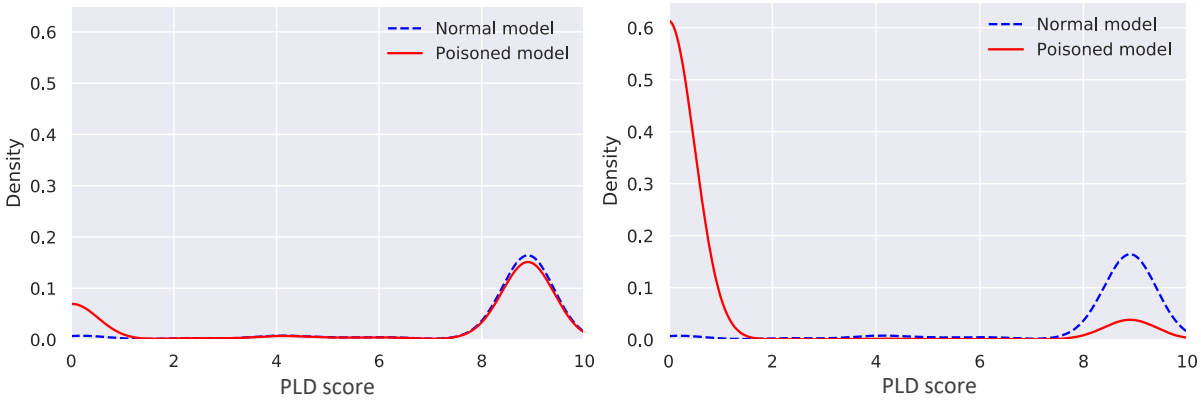


Figure 13: PLD score distributions for normal and poisoned local models in 'Inject rare process launch events' attack. Left: experiment with (*OfficeClickToRun.exe, rundll32.exe*), 9,047 events injected. Right: experiment with (*fshoster32.exe, chrome.exe*), 345,466 events injected.

As shown in these experiments, leveraging the retraining aspect of online distributed learning is a promising avenue for detecting poisoned local models (or data). Assuming that a client was a benign participant of the training in the past (e.g., prior to getting compromised by an adversary), we can likely detect its poisoned local model by comparing it with earlier local models.

Strong client authentication (countering Sybil attacks): In this study, our focus was on the case of a single compromised client operating in a single communication (model update) round. Poisoning attacks can be mitigated by normalising client contributions or can be detected by comparing poisoned contributions with benign ones (from other clients or from the past). The adversary can make poisoning attacks stealthier and / or increase their effectiveness by distributing them among multiple compromised clients. The adversary can actually create new fully controlled fake clients to increase their poisoning capability by contributing more local models or data to the training process. This is referred to as Sybil attacks. To avoid such attacks, it is important to implement strong client authentication, preventing the adversary from creating fake clients. While some defences have been developed to detect Sybil attacks in federated learning [FYB'18], they are based on strong assumptions of the similarity of local models contributed by every Sybil client and can be circumvented. Enforcing strong client authentication better mitigates this threat and increases the attack cost for the adversary (in particular, if obtaining authentication credentials requires payments). Unfortunately, there exist scenarios when client authentication is undesirable or impossible, for instance, due to anonymity requirements.

5. Conclusion

In this report, we presented the SHERPA project Task 3.5 work on vulnerability of ML models to poisoning attacks. Since attack – and corresponding defense – approaches are usually highly case-specific, we decided to focus on analysing a simple but realistic anomaly detection system similar to ones that could be used in the computer security domain and based on a model called *Process Launch Distribution* (PLD). This made it possible to comprehensively review potential model poisoning tactics of the adversary, consider ways to implement and optimize specific attacks following those tactics, and analyse the extents and impact of the attacks. The choice of the anomaly detection system based on the PLD model as the study target was mainly based on:

i. its practical relevance to the cybersecurity domain;
ii. its relative simplicity allowing for comprehensive analysis;
iii. its training method, which is widely used by digital service providers; and
iv. the fact that the system is a good representative of a practically popular class of anomaly detection systems.

Points (iii) and (iv) are important arguments in favour of the relevance and generalizability of the results that we obtained.

We designed and demonstrated in experiments the effectiveness of three model poisoning approaches targeting the PLD-based system. The three implemented attacks were able to effectively poison the studied global model for detecting anomalous process launch events. Each attack required adversarial control over only a single client participating in the online distributed training process (out of several hundreds of clients in the experimental setup) and modifying its local model in a minimal manner in most cases. Since the three analysed poisoning approaches are applicable to similar anomaly detection systems in multiple domains, our results should be considered a strong warning for organisations developing and operating such systems.

Learning from the vulnerabilities identified in our experiments, we provided and illustrated a set of recommendations for detecting and mitigating poisoning attacks, which we are planning to extend and translate to other use cases in the future project work. These recommendations can be summarised as follows:

- Defining and checking the format and validity of local contributions to model training
- Normalisation of local contributions prior to aggregating those to the global model
- Detecting and discarding outliers in local contributions
- Discarding local models having large distances to the global model
- Detecting abnormal evolution of local contributions over time
- Using strong client authentication to mitigate the risks of distributed poisoning attacks

We note that some of these methods for countering poisoning attacks rely on certain assumptions, in particular, on the distribution of training data among clients contributing to model training and on evolution of training data over time. It is important to carefully check limitations and assumptions of defence techniques and their applicability to specific use cases.

References

[ATK'15] Akoglu, L., Tong, H., & Koutra, D. (2015). Graph based anomaly detection and description: a survey. Data mining and knowledge discovery, 29(3), 626-688.

[BVH+'18] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., & Shmatikov, V. (2018). How to backdoor federated learning. arXiv preprint arXiv:1807.00459.

[BNL'12] Biggio, B., Nelson, B., & Laskov, P. (2012). Poisoning attacks against support vector machines. In Proceedings of the 29th International Conference on International Conference on Machine Learning (pp. 1467-1474).

[BR'18] Biggio, B., & Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition, 84, 317-331.

[BCM+'19] Bhagoji, A. N., Chakraborty, S., Mittal, P., & Calo, S. (2019). Analyzing Federated Learning through an Adversarial Lens. In International Conference on Machine Learning (pp. 634-643).

[BEG+'19] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., & Van Overveldt, T. (2019). Towards federated learning at scale: System design. arXiv preprint arXiv:1902.01046.

[CBK'09] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), 1-58.

[CLL+'17] Chen, X., Liu, C., Li, B., Lu, K., & Song, D. (2017). Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526.

[CSL+'08] Cretu, G. F., Stavrou, A., Locasto, M. E., Stolfo, S. J., & Keromytis, A. D. (2008). Casting out demons: Sanitizing training data for anomaly sensors. In 2008 IEEE Symposium on Security and Privacy (S&P 2008) (pp. 81-95).

[DS'07] Das, K., & Schneider, J. (2007, August). Detecting anomalous records in categorical datasets. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 220-229).

[FYB'18] Fung, C., Yoon, C. J., & Beschastnikh, I. (2018). Mitigating sybils in federated learning poisoning. arXiv preprint arXiv:1808.04866.

[GKN'17] Geyer, R. C., Klein, T., & Nabi, M. (2017). Differentially private federated learning: A client level perspective. arXiv preprint arXiv:1712.07557.

[HJN+'11] Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., & Tygar, J. D. (2011, October). Adversarial machine learning. In Proceedings of the 4th ACM workshop on Security and artificial intelligence (pp. 43-58).

[KSL'18] Koh, P. W., Steinhardt, J., & Liang, P. (2018). Stronger data poisoning attacks break data sanitization defenses. arXiv preprint arXiv:1811.00741.

[KMY+'16] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492.

[MMR+'17] McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017, April). Communication-Efficient Learning of Deep Networks from Decentralized Data. In Artificial Intelligence and Statistics (pp. 1273-1282).

[MRR'12] Muniyandi, A. P., Rajeswari, R., & Rajaram, R. (2012). Network anomaly detection by cascading k-Means clustering and C4.5 decision tree algorithm. Procedia Engineering, 30, 174-182.

[PMS+'18] Papernot, N., McDaniel, P., Sinha, A., & Wellman, M. P. (2018, April). SoK: Security and privacy in machine learning. In 2018 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 399-414). IEEE.

[PMG+'18] Paudice, A., Muñoz-González, L., Gyorgy, A., & Lupu, E. C. (2018). Detection of adversarial training examples in poisoning attacks through anomaly detection. arXiv preprint arXiv:1802.03041.

[STS'16] Shen, S., Tople, S., & Saxena, P. (2016, December). AUROR: Defending against poisoning attacks in collaborative deep learning systems. In Proceedings of the 32nd Annual Conference on Computer Security Applications (pp. 508-519).

[SP'10] Sommer, R., & Paxson, V. (2010, May). Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE symposium on security and privacy (pp. 305-316). IEEE.

[SKL'17] Steinhardt, J., Koh, P. W. W., & Liang, P. S. (2017). Certified defenses for data poisoning attacks. In Advances in neural information processing systems (pp. 3517-3529).

[WYS+'19] Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., & Zhao, B. Y. (2019, May). Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In 2019 IEEE Symposium on Security and Privacy (SP) (pp. 707-723). IEEE.

[WC'18] Wang, Y., & Chaudhuri, K. (2018). Data poisoning attacks against online learning. arXiv preprint arXiv:1808.08994.