SHERPA

Responsible Development of Smart Information Systems: Technical Options and Interventions

D3.5 (2nd part): October 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)
Setareh Roshan, Data Scientist, F-Secure Corporation (setareh.roshan@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 (2nd part)
WP/Task Related	WP3, Task 3.5
Delivery Date	October 30, 2020
Dissemination Level	PU
Lead Partner	FSC
Contributors	UCLan Cyprus
Reviewers	Doris Schroeder, Bernd Stahl
Abstract	This report presents the second part of the study of model poisoning attacks on models from a popular class of anomaly detection algorithms, focusing on attack mitigation approaches.
Key Words	Data poisoning attacks, model poisoning attacks, anomaly detection, online distributed learning, federated learning, statistical distribution models with thresholds, backdoor poisoning attacks, poisoned input detection, robust aggregation, elliptic envelope

Table of Contents

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
BDVA	Big Data Value Association (https://www.bdva.eu/)
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
ECSO	European Cyber Security Organisation (https://ecs-org.eu/)
ENISA	The European Union Agency for Cybersecurity (https://www.enisa.europa.eu/)
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

This report is the second and final part of SHERPA D3.5, the main purpose of which 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). In the first part of the deliverable, we introduced three poisoning attacks against a chosen anomaly detection model, and we evaluated their effectiveness with respect to key adversarial goals. While targeted at a specific anomaly detection model, named *Process Launch Distribution* (*PLD*), the three attack techniques generalise to essentially any *statistical distribution model with thresholds* trained in the *online fashion* using *distributed data*. Such models can be applied to many use cases and are easy to understand, which makes it practically relevant to study attacks against them and ways to counter those.

In this report, based on the attack analysis in the first part, we introduce and study an approach to protecting PLD and similar models from poisoning attacks. The approach, which we call *EE-detection*, applies anomaly detection techniques to local models (or data) contributed by the clients in order to identify and discard poisoned inputs before those are aggregated into a global model. We evaluate and compare the performance of our EE-detection algorithm with several existing methods for countering poisoning attacks in the distributed and federated learning settings.

EE-detection applied in the PLD model context exemplifies a general methodology which we propose for constructing defenses against poisoning attacks and which consists of the following four steps:

1. Analyze threats against a model and its training process which we want to protect.
2. Design, implement and analyze concrete attacks realising the identified threats.
3. Define features of the local models (or data) which can help distinguish benign inputs from poisoned ones (produced by the implemented attacks).
4. Design and validate a detection method that uses these features to identify poisoned inputs.

In experiments with the PLD model, we show that the EE-detection algorithm outperforms all the other tested methods in terms of both mitigating poisoning attacks and preserving the utility and high performance of the aggregated global model.

Summarising our learnings, we propose several general recommendations to designers, implementors, integrators, and operators of ML models trained in the distributed online fashion:

- Careful and comprehensive threat enumeration and analysis are a crucial pre-requisite for designing effective model protection methods (different models and attacks against them often require different protection approaches).
- Careful analysis of training data distribution across the clients (what can be realistically assumed) is important.
- Real-time monitoring and analysis of client inputs are often required when attacks against ML models can cause substantial harm (in many scenarios, it is hard to ensure operational resilience only by design and implementation time efforts).
- A managed monitoring service should be considered in cases when the precision or confidence of an automated real-time monitoring system in identifying poisoned inputs are not sufficiently high.

Introduction

This report is a direct continuation of the first part of SHERPA D3.5, available publicly at https://doi.org/10.21253/DMU.12973031, and we refer to that document as D3.5.1 in a number of places throughout the text.

As explained in the Introduction section of D3.5.1, the deliverable 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. In the first part of the deliverable, we introduced three poisoning attacks against a chosen anomaly detection model, and we evaluated their effectiveness with respect to key adversarial goals. While targeted at a specific anomaly detection model, named *Process Launch Distribution* (*PLD*), the three attack techniques generalise to essentially any *statistical distribution model with thresholds* trained in the *online fashion* using *distributed data*. Such models can be applied to many use cases and are easy to understand, which makes it practically relevant to study attacks against them and ways to counter those.

In this report, based on the attack analysis in D3.5.1, we introduce and study an approach to protecting PLD and similar models from poisoning attacks. The approach applies anomaly detection techniques to local models (or data) provided by the clients in order to identify and discard poisoned inputs before those are aggregated into a global model. Methodologically, it consists of four steps:

1. Analyze threats against a model and its training process which we want to protect.
2. Design, implement and analyze concrete attacks realising the identified threats.
3. Define features of the local models (or data) which can help distinguish benign inputs from poisoned ones (produced by the implemented attacks).
4. Design and validate a detection method that uses these features to identify poisoned inputs.

The first two steps of this methodology, in the context of the PLD model, are presented in detail in D3.5.1. The last two steps are explained in Section 2 of this report. We call the resulting algorithm for protecting PLD and similar models through identification of poisoned inputs *EE-detection* (where 'EE' stands for '*Elliptic Envelope*').

To demonstrate the relevance and value of the methodology for designing effective defenses against poisoning, we evaluate and compare the performance of our EE-detection algorithm with several existing methods for countering poisoning attacks in the distributed and federated learning settings. These existing methods fall into two main categories. Approaches in the first category are based on detecting of poisoned local models (or data) to discard them from aggregation into the global model. We call this category *poisoned input detection* methods [CSL+'08, FCJ+'20, FYB'18]. Our EE-detection method and one of the methods we compare it with – AUROR [STS'16] – belong to this category. The second category is called *robust aggregation* methods, which do not discard any local input from aggregation but attempt to make the aggregation process more robust to (potentially malicious) outliers [BGS'17, RNH+'09, AAL'18, DEG+'19]. The underlying idea of such methods is to 'smoothen' differences among local inputs and make it difficult for malicious inputs to significantly influence the global model. We compare EE-detection to three robust aggregation methods proposed earlier:

- *Median aggregation*
- *Trimmed mean aggregation*
- *Weight normalisation*

Through experiments with the PLD model, the comparative analysis shows that our EE-detection algorithm outperforms all the other tested methods in terms of (a) mitigating poisoning attacks and

(b) preserving the utility and high performance of the aggregated global model. Based on the results of the experiments, we observe that the performance of poisoning mitigation methods heavily depends on several properties of the model to protect and the data it is trained with:

- The tested robust aggregation methods dramatically degrade the performance of the aggregated global model if the distribution of the data used to train the local models (or directly provided for aggregation) varies significantly among the clients (more precisely, if the clients' training datasets are far from independent and identically distributed: non-i.i.d.).
- The tested robust aggregation methods are ineffective in preventing poisoning if the features used to train local models (or directly provided for aggregation) are sparse in the sense that typically only few clients observe non-zero values (or presence) of a specific feature. Simple distribution of a poisoning attack among a few compromised clients completely defeats essentially any robust aggregation methods in such a scenario.
- Weight normalisation can actually make poisoning attacks easier and stealthier in non-i.i.d. settings. It reduces the impact of large benign local inputs on the aggregated global model, which in turn increases the impact of poisoned local inputs.
- Existing poisoned input detection methods based on clustering, like AUROR, incorrectly identify many benign models as poisoned, which can be a major problem when there is no poisoning attack taking place or when only a very small fraction of local models (e.g., only one) is poisoned. This problem is, of course, less acute when we deal with massive highly distributed poisoning attacks.

Summarising our learnings, we propose several general recommendations to designers, implementors, integrators, and operators of ML models trained in the distributed online fashion:

- Careful and comprehensive threat enumeration and analysis are a crucial pre-requisite for designing effective model protection methods (different models and attacks against them often require different protection approaches).
- Careful analysis of training data distribution across the clients (what can be realistically assumed) is important.
- Real-time monitoring and analysis of client inputs are often required when attacks against ML models can cause substantial harm (in many scenarios, it is hard to ensure operational resilience only by design and implementation time efforts).
- A managed monitoring service should be considered in cases when the precision or confidence of an automated real-time monitoring system in identifying poisoned inputs are not sufficiently high.

In the recent "Cybersecurity for Artificial Intelligence" online workshop announcement[1], it was stated that "The much needed call for ethics in AI relies on solid cybersecurity, since it is cybersecurity that will guarantee and assure the proper implementation of said ethics." In his workshop talk, Juhan Lepassaar, Executive Director of ENISA, mentioned that "What we currently have is 84 initiatives on ethical principles or guidelines for AI … however, there is no concrete cybersecurity framework, or baseline, for AI and, of course, this is something that the Agency now focuses on …" We believe that the SHERPA work on technical options and interventions for responsible development of Smart Information Systems – in the general context of AI and big data analytics impact on ethics and human rights – is well-aligned with these ongoing ENISA efforts, supported in particular by the Agency's Ad-Hoc Working Group on Artificial Intelligence Cybersecurity[2]. We will approach ENISA and its Ad-Hoc Working Group to contribute to their activities with the SHERPA work outcomes presented in D3.5 and other project deliverables, reports and materials. Furthermore, both European Cyber Security

[1] https://www.enisa.europa.eu/events/cybersecurity-for-ai-c4ai
[2] https://www.enisa.europa.eu/topics/iot-and-smart-infrastructures/artificial_intelligence/adhoc_wg_calls

Organisation (ECSO) and Big Data Value Association (BDVA) are closely following and supporting the ENISA's work on cybersecurity of AI. We have already discussed the SHERPA work with ECSO and BDVA and clearly see opportunities for their help in promoting our results, in particular, in connection with potential contributions to the ENISA activities.

The remaining part of this report is organised as follows. Section 1 provides a reminder on the anomaly detection model that we study (PLD) and on the poisoning attacks designed in D3.5.1. It also presents an overview of the existing defenses against poisoning attacks in distributed and federated learning settings. Section 2 presents our methodology for countering poisoning attacks, explains how to define features helpful for identifying poisoned local models in the PLD context and how to use these features for constructing a poisoned model detector: EE-detection. In Section 3, we extensively evaluate the performance of EE-detection and several other methods for mitigating poisoning attacks in both single-client and distributed attack settings. Section 4 evaluates the impact of each of these methods on the aggregated global model. Sections 3 and 4 show that our EE-detection method is the best for countering poisoning attacks against PLD and draw empirical conclusions about the limitations of the existing defense approaches in the setting of PLD and similar anomaly detection models. The study and the lessons learned are summarised in Section 5.

1 Background

1.1 Process Launch Distribution (PLD) model

In Task 3.5 of the SHERPA project, we study the anomaly detection model called *Process Launch Distribution (PLD)*, introduced in D3.5.1. This model is designed to detect malicious processes launch events. PLD belongs to the class of *statistical distribution with thresholds* models and is trained in the online distributed fashion (with training inputs – local models – coming from multiple clients).

PLD models the distribution of the *PLD_score* values of process pairs (parent process $proc_A$, child process $proc_B$), where *PLD_score* is computed via the following counters:

- $call_{parent}(proc_A)$: the number of times $proc_A$ started a child process
- $call_{child}(proc_B)$: the number of times $proc_B$ was started by another process
- $call_{join}(proc_A, proc_B)$: the number of times $proc_A$ started $proc_B$
- $call_{total}$: the total number of process launch events in the training set

Given several (smoothing and safeguard) constants α_i and β_i, the formula for *PLD_score* is:

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

A *local PLD model* (PLD_i) – essentially a set of $call_{join}$ counters for the observed process pairs – is produced by each client *i* which monitors its process launch events. A *global PLD model* (PLD_{global}) is trained in the federated manner by aggregating all the PLD_i models provided by the clients. The aggregation process first merges all the process pairs observed on the clients (presented in their PLD_i) into a common set. Then, simple *sum* is used for aggregating the local counters into the global counters to be used in PLD_{global}, e.g.:

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

After the aggregation, the global *PLD_scores* can be computed for each process pair. The computed *PLD_score* values are divided into several bins corresponding to quantiles of exponentially decreasing ranges of the values and separated by *thresholds*. Each bin of the *PLD_score* values is assigned an *anomaly category* (*AC*), reflecting how anomalous the process pairs in that bin are. Low *PLD_score* values mean high anomaly categories as illustrated in Table 1. (Compared with D3.5.1, two more categories are added to support finer-grained analysis of very small *PLD_score* values.)

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
]0.000001% - 0.00001%]	1st 100,000,000-quantile	8
]0.0000001% - 0.000001%]	1st 1,000,000,000-quantile	9
]0% - 0.0000001%]	-	10

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

PLD_{global} can then be used to identify anomalous process launch events.

PLD_i models are regularly updated (over a sliding window of recent observations) and submitted by each client to a central aggregator where a new global PLD model PLD_{global}^{t+1} is trained (with the upper index (*t*+1) indicating the model's 'epoch').

The training of PLD_{global} using PLD_i is an example of online model training using distributed data. It was demonstrated in D3.5.1 that PLD with the *sum aggregation* is vulnerable to several poisoning attacks. In this report, we will analyse whether the sum aggregation used in the training of PLD_{global} can be replaced or augmented to make it more resilient to adversarial attacks.

Further details of the PLD model and its training process can be found in Sections 2.1, 2.2 and 2.3 of D3.5.1.

1.2 Poisoning attacks against PLD training

Since PLD_{global} is trained in a distributed manner (by aggregating local models from multiple clients), it is threatened by *poisoning attacks* that can be mounted by malicious clients. In D3.5.1, we introduced and analysed three *backdoor poisoning* attacks [BVH+'18, BNL'12, CLL+'17] against PLD_{global}. The goal of a backdoor poisoning attack on a PLD model is to make the 'next epoch' model PLD_{global}^{t+1} fail on a specific input: a process launch event $(proc_M, proc_T)$, which the model would normally give a high anomaly category index (e.g., 6), must be – incorrectly – assigned to an anomaly category with a low index (e.g., AC2).

To introduce such a backdoor, the attacker controls one or several compromised or acquired clients and wants to craft poisoned local models PLD_m^t, which – when submitted by the controlled clients and aggregated with the local models of the other clients into PLD_{global}^{t+1} – will lead the latter to output a lower anomaly category for $(proc_M, proc_T)$ than PLD^t_{global}. The attacker has the following capabilities to realise the poisoning attack:

- Control over a number of compromised or acquired clients (e.g., a few dozens) taking part in the distributed learning process.
- Injection of a desired local model PLD^t_m by each of the controlled clients.
- Access to the global model computed at the previous epoch: PLD^t_{global}.

A remark on the number of malicious clients. On the one hand, we want our defense to be effective against powerful attackers controlling many clients. On the other hand, we note that in real life, there would be certain costs for the attacker associated with either compromising existing benign clients or acquiring new client installations through the purchase of licenses. Analysis of the expected resources of relevant attackers should be a part of the threat modeling process.

Considering this adversary model, we aim to develop methods mitigating attacks of the following two types:

1. **Increase target process launch count**: This attack is carried out by adding a single record for the target process launch event $(proc_M, proc_T)$ to the local model(s) PLD^t_m of the controlled client(s). The attack results in an increased value of $PLD\_score(proc_M, proc_T)$, through increasing $call_{join}(proc_M, proc_T)$.
2. **Inject rare process launch events unrelated to the attack:** This attack is carried out by adding several records for process launch events with low *PLD_score*, unrelated to the attacker's target event $(proc_M, proc_T)$, to the local model(s) PLD^t_m of the controlled client(s). The attack results in decreased values of the anomaly category thresholds by filling in the low tail of the *PLD_score* distribution with process pairs unrelated to the planned attack.

The third attack presented in D3.5.1, "**Decrease parent or child process launch count**", can be trivially countered by checking the values in local models to ensure that all the reported process launch counts are positive. Thus, we do not need to consider it in designing our defenses in this report.

When performing a distributed poisoning attack using *n* controlled clients, the attacker naturally sends from all those clients local models poisoned in the same way. Distributing an attack does not change the malicious process pairs contained in the poisoned local models but allows to decrease the counts of those injected process pairs by the factor of *n* (compared to an attack using a single controlled client). This makes the attack harder to detect.

Further details of the poisoning attacks targeting PLD can be found in Sections 2.4 and 3 of D3.5.1.

1.3 Existing defenses against poisoning in distributed training

Several methods have been proposed in recent years to detect and mitigate poisoning attacks in distributed learning settings. In particular, dealing with poisoning of Deep Neural Networks (DNNs) was a popular research topic. The main idea of many of the existing methods is to find anomalies among the models (or data) submitted by clients in training. The techniques based on anomaly detection include, e.g., fine-pruning [LDG'18] and clustering [CCB+'18]. Some of the methods were developed and designed for cases where clients' data and models must be kept confidential [STS'16, FYB'18, CSL+'08]. The class of defenses of this kind, identifying anomalous client inputs and discarding them from aggregation, we will call *poisoned input detection*.

There is another class of defense mechanisms against poisoning attacks called *robust aggregation methods*, which focus on how local models or data are aggregated into a global model. For instance, the originally proposed federated learning aggregation function is based on a linear combination of gradients and vulnerable against distributed attacks, where the adversary controls multiple clients (called sometimes Byzantine adversaries). Applying more robust aggregation functions makes federated and distributed learning more resilient to such attacks. Several methods based on robust aggregation schemes have been proposed [BGS'17, RNH+'09, AAL'18, DEG+'19]. Although most of these techniques cannot fully prevent poisoning, they can mitigate attacks to various extents. A key challenge with robust aggregation techniques is that modifying an aggregation function may significantly reduce the accuracy and other important characteristics of the aggregated global model. In addition, many of the existing techniques assume that the clients' training data is i.i.d. and this assumption does not hold in many real-world distributed and federated learning applications [BVH+'18].

While most of the existing generic defenses against poisoning attacks were developed and tested on public datasets (e.g. MNIST) and in federated learning settings with Deep Neural Networks, their effectiveness for protecting simpler models, such as our statistical distribution with thresholds model, remains unclear. As discussed in D3.5.1, such methods as [KSL'18, PMG+'18, SKL'17], which discard local models with a large distance to the global model, cannot be effective because the assumption that the training data is i.i.d. among the clients often does not hold. In fact, our experiments indicate that local models vary significantly among the clients. Stateful detection techniques, which require to maintain the evolving history of local models for every client over time, are too costly for many scenarios including our PLD model training where PLD_{global} would likely be re-trained frequently and with large numbers of local models.

In this report, we evaluate the effectiveness of several techniques from the two classes of defenses in detecting poisoning attacks against our global PLD model. We select ***AUROR*** [STS'16] as a candidate for anomaly-based ***poisoned input (model) detection*** and three variants of ***robust aggregation methods*** [YCR+'18] called ***median aggregation***, ***trimmed mean***, *and* ***weight normalisation***. These techniques are described briefly in the following subsections.

1.3.1 AUROR

AUROR is a defense mechanism to mitigate the effect of poisoning attacks to the global model in indirect collaborative learning: where clients participate in the training without sharing their local data but by sharing parts of their local models. In this setting, each client trains a model locally and submits a set of parameters, called masked features, based on their local model. According to AUROR, the distribution of specific masked features, called *indicative features*, changes significantly in the models of clients performing poisoning (controlled by the attacker), while remaining the same for benign clients. Therefore, the distribution of indicative features presents an anomalous pattern for malicious clients. Based on this, AUROR distinguishes malicious clients and excludes them from the distributed training process. AUROR assumes that a majority of the clients remain benign throughout the training process. In other words, with *n* being the total number of the clients, the adversary can only control *f* clients where $0 \leq f < n/2$. Based on these assumptions, AUROR takes the following key steps to detect poisoning attacks and exclude the compromised clients:

1. ***Identifying indicative features.*** A feature is *indicative* if it can successfully differentiate the adversaries from the benign clients. In order to find *indicative features*, AUROR looks for anomalous patterns in the distribution of specific masked features by clustering the client inputs with respect to every feature individually when a poisoning attack is ongoing. For each of the masked features, the inputs are split into two clusters. If the distance between the two

cluster centroids is larger than a threshold, the corresponding feature is marked as *indicative*. The threshold values must be adjusted according to the training set.

2. ***Detecting malicious clients.*** After identifying the indicative features, AUROR uses each indicative feature to cluster the clients into two clusters. As a benign majority is assumed, the clients placed in the smaller cluster are considered suspicious. If a client appears in suspicious clusters for at least q times, according to the clustering of each indicative feature, it is marked as malicious and discarded before aggregation. The constant q is upper-bounded by the number of indicative features and can be adjusted to tolerate minor fluctuations in benign clients' training data.

Using this detection and elimination mechanism, AUROR can thwart poisoning attacks. It was developed and tested on two datasets, handwritten digit images (MNIST) and traffic signs (GTSRB), to build a federated DNN model [STS'16]. The masked features in this setting are the gradient values of the weight and bias parameters computed by each client. The experimental results show that for both datasets, AUROR achieves a detection rate of 100% while dealing with the fraction of malicious clients between 10% and 30%. Also, the experiments indicate that the accuracy drop of the global model after discarding the clients identified as malicious (some of which may be false positives) is negligible.

1.3.2 Robust Aggregation Methods

In distributed and federated learning, some clients may behave abnormally due to random failures or the presence of adversaries. The traditional federated gradient descent algorithms, which apply naïve aggregation of gradients, are known to be greatly influenced by outliers. Even a single failing or attacker-controlled client can skew the model toward its desired direction and degrade its performance significantly by injecting random or adversarial outliers [BVH+'18, BCM+'19]. In order to mitigate this issue, two robust distributed gradient descent algorithms were proposed in [YCR+'18], one based on median and the other based on trimmed mean.

Median aggregation aggregates the parameters by taking the coordinate-wise median of the gradient vectors instead of the mean. As a result of this, the contribution of each client on the global model performance can be highly restricted and the attacker can only poison the global model by compromising or acquiring a sufficiently large number of clients. The median aggregation implicitly assumes that the number of benign clients is greater than one half of the total number of clients. Median aggregation for PLD replaces the original *sum aggregation* for count of launches as follows:

$$call_{join}(proc_A, proc_B)_{global} = Median_{i\in[1,N]}(call_{join}(proc_A, proc_B)_i)$$

Trimmed mean aggregation assumes that an α fraction of all the clients are attacker controlled or failing and must be considered outliers. In this technique, the mean aggregation is performed after discarding the smallest and largest α fraction of the gradients in each coordinate. As a result, even injecting of adversarial outliers by compromised clients cannot impact the global model performance significantly. We implement trimming when aggregating the counts of calls for each process pair individually. To implement trimmed mean aggregation for PLD_{global}, we select all the clients that report positive counts for a given process pair and discard the α share of the largest counts. We do not discard low values since most of the process pairs are launched by a minority of the clients, i.e., most of the clients report zero count for most of the process pairs. So, trimming of low values is already implicit in the PLD aggregation.

Weight normalisation normalises each local model in order to ensure the local models contribute equally to the global model prior to perform aggregation. In our problem setting, the aggregation of PLD_i models into PLD_{global} sums together unbounded values of $call_{join}$. A client can submit much higher values than the other clients in order to influence the global model in their interest. One way to address the non-equal contributions issue is to normalise local models prior to aggregation in order to limit their impact. We use the frequency of process launch events rather than their counters. Therefore, the sum of the $call_{join}$ values in each local model is set to 1, and all the clients contribute to the global model equally. Weight normalisation is combined with trimmed mean aggregation using the trimming factor of 10%.

2 Our poisoning detection approach

In this section, we present our approach to countering poisoning attacks against the PLD_{global} model training. It is based on detecting outlier local models to discard them from aggregation into the global model. To design our detection method, we use the results of the poisoning attacks analysis (Section 2.1). We define several features to represent local PLD models and distinguish benign from poisoned models (Section 2.2). Finally, we explain these features are used to train an anomaly detection model designed to detect poisoned models as anomalies (Section 2.3). Once trained, this detection model will be used before any PLD_{global} model aggregation to remove anomalous local PLD models from the set of models to be aggregated.

While the presented design was created for the particular use case of PLD_{global} distributed model training, it follows a general approach to defending models against poisoning attacks and includes the following steps:

1. Analyze threats against a model and its training process which we want to protect.
2. Design, implement and analyze concrete attacks realising the identified threats.
3. Define features of the local models (or data) which can help distinguish benign inputs from poisoned ones (produced by the implemented attacks).
4. Design and validate a detection method that uses these features to identify poisoned inputs.

We discussed and exemplified the first two steps in D3.5.1. This section shows how steps (3) and (4) can be implemented.

2.1 Goal and overview

As we saw in D3.5.1, poisoning attacks against the global PLD model require injecting process launch events with low PLD scores to local PLD models. The *"**Increase target process launch count**"* attack requires injecting one process pair with a sufficiently large counter. The ***"Inject rare unrelated process launches"*** attack requires injecting many process launch events, each having a relatively low counter. Both attacks increase the density of the PLD score distribution in low values (close to 0) for poisoned local models as depicted in Figure 1. The PLD score distribution remains roughly the same when comparing the normal and the poisoned versions of the same local model except for the PLD score values close to 0, where the density in the poisoned model is several orders of magnitude greater than in the normal model. Consequently, we expect a poisoned local model to have a larger-than-usual count / proportion of process launch events with small PLD scores, when compared to a normal model.

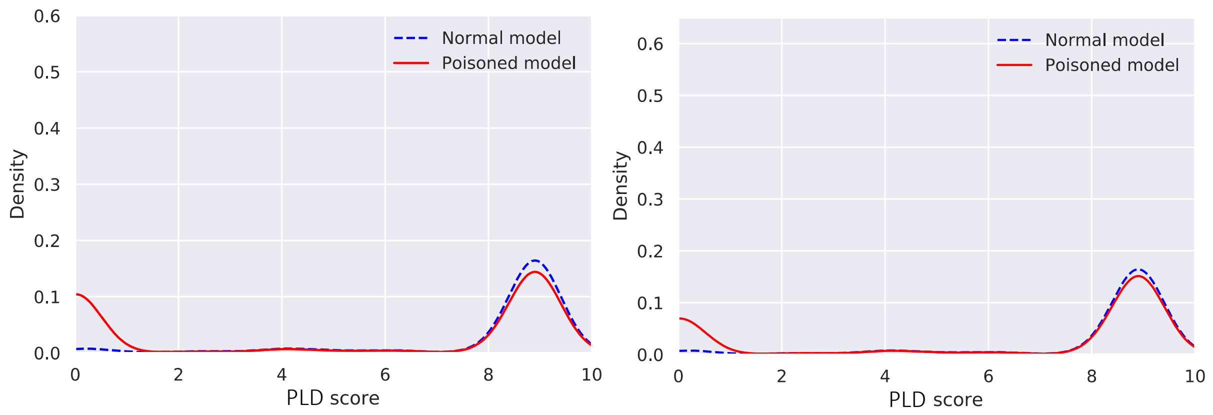

Figure 1: PLD score distributions for normal and poisoned local models. Left: '*increase target process launch count*' attack. Right: '*inject rare unrelated process launches*' attack. Both attacks cause an increased density in low PLD score values.

We design our poisoning detection method based on this observation. We propose to model the proportion of process launch events with low PLD scores in the local models. Then, we will use this model to identify local models with abnormally high proportions of process launches with low PLD scores (to discard as poisoned). Two questions need to be answered:

1. How low should the PLD score of a process pair be to be considered a potential fake record injected by a poisoning attack?
2. How high should the proportion of process launches with low PLD scores be to consider a local model anomalous / poisoned?

Both these questions will be answered in an 'intelligent' manner, learned from the data that we collect about process launches in our PLD models. First, we will define a number of features to represent the distribution of PLD scores in a local PLD model. Second, we will extract these features from many benign local models in order to build a model of 'normal' PLD scores distribution. Finally, we will use this model to identify abnormal distributions of PLD scores in local models and discard them as likely poisoned.

2.2 Features for detection of poisoned models

2.2.1 Feature definition

Our goal is to define a relatively small number of features to represent the PLD score distribution of a local model. While the PLD score distribution is discrete and bounded, it can have many possible values (tens or even hundreds of thousands). Of course, not all of those are present in a typical local model: values can have 0 as their probability mass because no process launch events with such PLD scores were observed on a given client. In order to find similarities between local PLD models, it is better to define features that would be non-zero for most of the local models. Thus, we define a small number of bins for contiguous ranges of PLD score values. The purpose of these bins is to separate potential fake process launches injected by a poisoning attack from real (benign) ones (answering the first question in the previous subsection). We recall that both studied poisoning attacks involve injection of process launches that belong to a target *anomaly category* in order to succeed. Typically, this anomaly category is quite high (e.g., 5 – 10) because it corresponds to a rare and anomalous target process launch event. Anomaly categories (1, 2, 3, etc.) are bins of PLD score values separated by the global PLD model thresholds. We choose to use anomaly categories as our bins since any poisoning attack should cause a large increase in process launch counts in at least one of the anomaly categories. We discard anomaly category 1 since it contains only common benign process launches, which leaves us with nine bins for nine anomaly categories: 2 – 10. Figure 2 depicts the split of the PLD score distribution into bins.

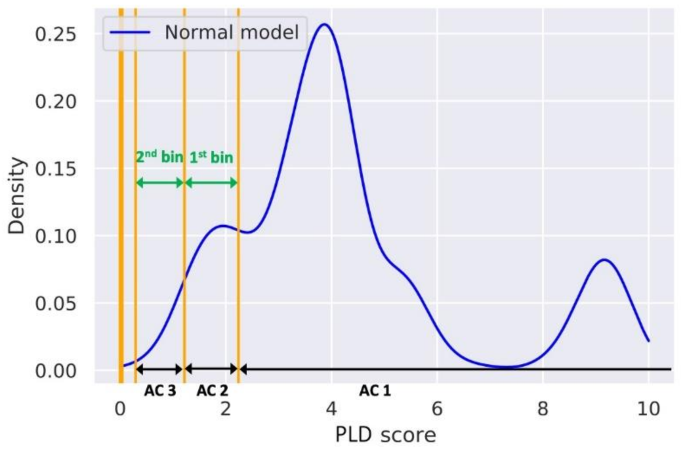

Figure 2: Bin split of PLD score distribution according to anomaly categories and thresholds of the global PLD model: nine bins corresponding to anomaly categories 2 - 10 (AC1 is discarded).

For each bin and each local model, we compute:

- the **sum** of the occurrences of all the process launch pairs with PLD score belonging to the given bin;
- the **maximum** number of launches among all the process launch pairs with PLD score belonging to the given bin.

We divide these numbers by the respective total number of process launch events in the given local PLD model to obtain scaled ratios. Thus, we obtain two features per bin (*sum*, *max*), 18 features in total, defined in Table 2. Values of each of these 18 features belong to [0; 1] and altogether they represent the PLD score distribution of a local PLD model. The *max* features are particularly designed to detect the *"**Increase target process launch count**"* attack, since this attack injects only one process pair with a high number of launches. The *sum* features are designed to detect any attack, but they must be particularly effective in detecting the *"**Inject rare unrelated process launches**"* poisoning attack, since this attack injects many process pairs, each having a small number of launches.

Bins from cumulative distribution	Anomaly category	'Sum' features	'Max' features
]1% - 10%]	2	sum_{ratio}(AC2)	max_{ratio}(AC2)
]0.1% - 1%]	3	sum_{ratio}(AC3)	max_{ratio}(AC3)
]0.01% - 0.1%]	4	sum_{ratio}(AC4)	max_{ratio}(AC4)
]0.001% - 0.01%]	5	sum_{ratio}(AC5)	max_{ratio}(AC5)
]0.0001% - 0.001%]	6	sum_{ratio}(AC6)	max_{ratio}(AC6)
]0.00001% - 0.0001%]	7	sum_{ratio}(AC7)	max_{ratio}(AC7)
]0.000001% - 0.00001%]	8	sum_{ratio}(AC8)	max_{ratio}(AC8)
]0.0000001% - 0.000001%]	9	sum_{ratio}(AC9)	max_{ratio}(AC9)
]0% - 0.0000001%]	10	sum_{ratio}(AC10)	max_{ratio}(AC10)

Table 2: Features representing a local PLD model for poisoning detection. The definition is based on the bins corresponding to the global PLD model's anomaly categories.

2.2.2 Features analysis

In order to check whether our 18 features are able to differentiate normal models from poisoned models, we analysed the differences in the feature values for normal and poisoned local PLD models.

We randomly selected 500 normal models and ran the same poisoning attack against each of these models. For each model, we extracted our 18 features from both original benign version and poisoned version. We plotted the histograms comparing the repartition of the values for some of the features. In figures 3 and 4, the bars represent the counts of the models having one selected feature in a certain range: blue bars represent benign models (500 models in total) while orange bars represent poisoned models (500 models in total). Feature values on the *x*-axis represent the ratios (*max* in fig. 3 and *sum* in fig. 4) for process pairs belonging to a given anomaly category.

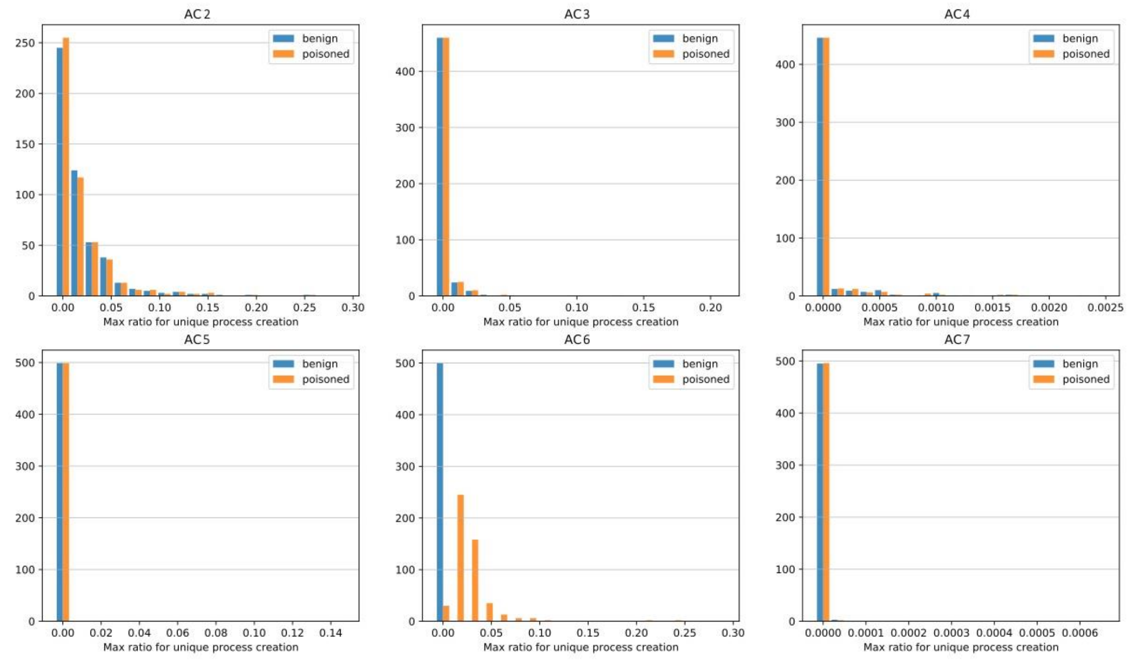

Figure 3: Histograms showing the numbers of local models having *max_{ratio}* (AC2 – AC7) feature values in different ranges. Blue bars: 500 benign models. Orange bars: 500 poisoned models.

Figure 3 compares the 500 benign local models (blue) to the 500 local models (orange) poisoned with a stealthy *"Increase target process launch count"* attack that injects 1630 additional process launch events for one target process pair which originally belongs to Anomaly Category 6 (AC6). The six histograms depict the repartition of the values for the max_{ratio}(AC2 – AC7) features. We see that the benign and the poisoned local models are exactly the same with respect to most of the features. The only difference we notice is for max_{ratio}(AC6), where the poisoned models have significantly larger values (0.01 – 0.1) than the benign models (~0.00001). This is unsurprising since AC6 is the anomaly category of the process pair targeted by the poisoning attack, and the difference is clearly captured by one of the features we defined. Consequently, we can expect that our features can be useful for detecting the *"Increase target process launch count"* attack.

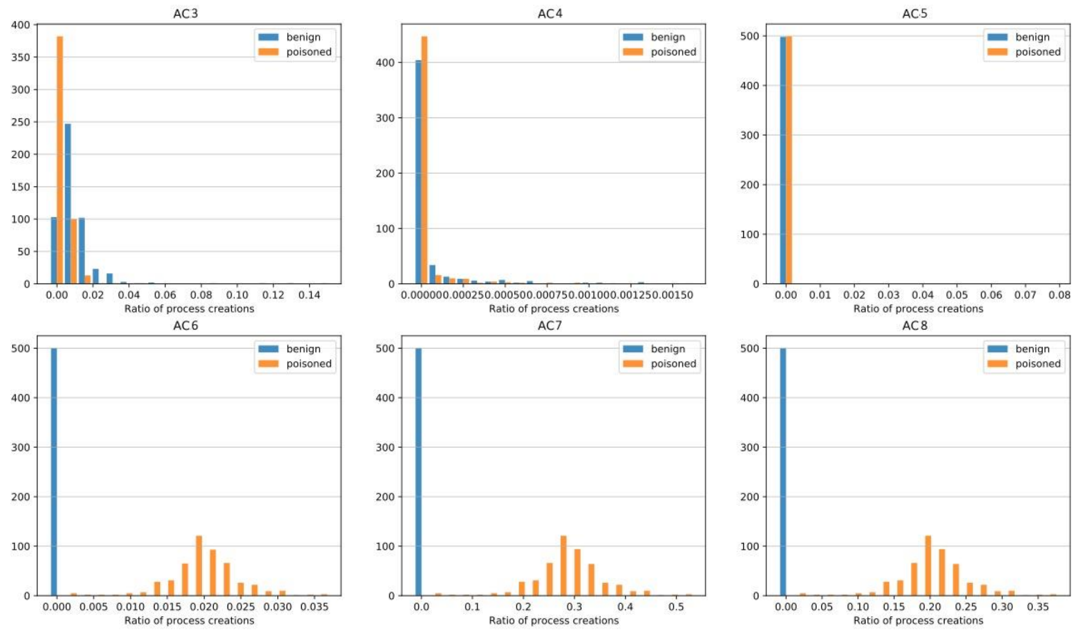

Figure 4: Histograms showing the numbers of local models having *sum_{ratio}* (AC3 – AC8) feature values in different ranges. Blue bars: 500 benign models. Orange bars: 500 poisoned models.

Figure 4 compares the 500 benign local models (blue) and the 500 local models (orange) poisoned with an "*Inject rare unrelated process launches*" attack that injects 60,862 additional process launch events. The target process pair for this attack belongs to Anomaly Category 6 (AC6). The six histograms depict the repartition of the values for the sum_{ratio} (AC3 – AC8) features. We see that the benign and the poisoned local models are very similar with respect to sum_{ratio} (AC3 – AC5) features: the blue and orange bars are distributed in a similar manner. However, the poisoned models have significantly larger values (0.01 – 0.5) than the benign models (~0.0000001 – 0.00001) for the sum_{ratio} (AC6 – AC8) features. The large increase in the numbers of the process launches for processes within and below AC6 is expected, since it is the anomaly category of the process pair targeted by the poisoning attack, and the difference is clearly captured by three of the features we defined. Consequently, we can expect that our features can be useful for detecting the "*Inject rare unrelated process launches*" attack.

2.3 Models for poisoning detection

2.3.1 Motivation and detection approach

Having defined the 18 features that seem effective at differentiating poisoned local models from benign ones, we want to design an automated algorithm – based on these features – to identify poisoned local models. While we have many examples or benign local models, we do not have any examples of real poisoning attacks and real poisoned models (apart from the ones which we generated using our own poisoning attack implementations). Consequently, we favour an anomaly detection approach to identify poisoned local models. Our anomaly detector will be trained using only benign local models, and any models identified by it as anomalous can be considered potentially poisoned. Such anomalous models will be discarded from the aggregation process creating the global PLD model, thereby mitigating poisoning attacks. It is worth noting that since we chose this anomaly detection-based approach, our poisoning detector is not tied to detection of any specific poisoning attack.

Selecting appropriate features, we can hope that it can be effective in detecting any poisoning attack, including those that we have not seen yet.
To build our poisoning detector, we select three candidate one-class classification methods. One-class classification algorithms are unsupervised algorithms that learn a decision function for novelty detection: classifying new samples as either similar or different to the training set. In our case, the only class we want to model is “benign local models”. Our candidate one-class classification models are the following:

- **One-class SVM** is a method that tries to learn a boundary around the training data [SWS+’00]. Formally, One-class SVM tries to find the smallest hypersphere that includes all the training data points. Any point outside of this hypersphere is considered an anomaly.
- **Elliptic envelope** also tries to learn a boundary around the training data. In contrast to One-class SVM, the elliptic envelope algorithm assumes the data distribution is Gaussian and it learns an ellipse. Modeling each feature individually, it is usually more robust to outliers than one-class SVM (easier to avoid overfitting).
- **Isolation forest** is complementary to the previous methods since it is based on the principle of ‘isolating’ anomalies instead of profiling normal points. It randomly selects a feature and then randomly selects a split value for this feature between the minimum and maximum values for that feature, continuing this tree construction process until each point in the training set can be fully isolated. The algorithm constructs a number of trees in this way. Then a new point is considered anomalous if it can be isolated with the constructed trees in a small number of steps.

2.3.2 Model selection

In order to select the best model for poisoning detection, we performed a small-scale experiment to compare the performance of each model. We built a training set composed of 5174 (benign only) local PLD models collected on April 14, 2020. This training set was used to train the candidate one-class classifiers. We also randomly selected 500 benign local PLD models collected on April 17, 2020. We built a test set of 5000 local models containing those 500 benign models and 4500 poisoned local models generated from the 500 benign ones using nine poisoning attacks. We trained our three candidate models using the training set and tuning several hyperparameter values:

- **One-class SVM** was trained using a *Radial Basis Function (RBF)* and a *Sigmoid* kernel while using several values of the regularisation parameter *nu*. *nu* is used to control the generalisation / over-fitting of the model. The smaller *nu* is, the more the model generalises and the more it tends to accept new anomalous data as normal.
- **Elliptic envelope** was trained using several *contamination* values, which control the share of training data to discard (assumed potentially anomalous or *contaminated*) while learning the ellipse. The lower the *contamination* value is, the more the model tends to accept new anomalous data as normal.
- **Isolation forest** was built using several *contamination* and *estimator* values. *Contamination* plays the same role as for Elliptic envelope. *Estimators* control the number of trees the algorithm can learn.

To compare the performance of these methods under different hyperparameter values, we used two metrics:

- **True positive rate (TPR)** or **recall**: It reflects the ability to mitigate a poisoning attack, the higher it is, the better. It is computed as the ratio of the number of poisoned models correctly identified as poisoned to the total number of poisoned models.
- **False positive rate (FPR)**: It reflects the errors made by incorrectly identifying benign models as poisoned models, the lower it is, the better. It is computed as the ration of the number of benign models incorrectly identified as poisoned to the total number of benign models.

	RBF kernel				Sigmoid kernel			
nu	**TP rate**	**TP (/4500)**	**FP rate**	**FP(/500)**	**TP rate**	**TP (/4500)**	**FP rate**	**FP (/500)**
0.3	1.0	4500	0.858	429	1.0	4500	0.222	111
0.2	1.0	4500	0.404	202	1.0	4500	0.17	85
0.1	1.0	4500	0.064	32	1.0	4500	0.062	31
0.05	1.0	4500	0.036	18	0.99955	4498	0.022	11
0.01	1.0	4500	0.02	10	**0.99955**	**4498**	**0.002**	**1**
0.005	1.0	4500	0.02	10	0.99777	4490	0.002	1
0.001	1.0	4500	0.02	10	0.99577	4481	0.002	1

Table 2: Performance of One-class SVM in detecting poisoned local models. True positives (TP) and false positives (FP) for RBF and sigmoid kernel for different *nu* values. Sigmoid kernel with *nu* = 0.01 gives the best combination of FPR and TPR.

contamination	TP rate	TP (/4500)	FP rate	FP (/500)
0.1	1.0	4500	0.056	28
0.05	1.0	4500	0.028	14
0.01	**1.0**	**4500**	**0.006**	**3**
0.005	**0.99955**	**4498**	**0.006**	**3**
0.001	0.99488	4477	0.002	1
0.0005	0.97844	4403	0.002	1

Table 3: Performance of Elliptic Envelope in detecting poisoned local models. True positives (TP) and false positives (FP) for different *contamination* values. The best combinations of FPR and TPR are marked.

contamination	estimators	TP rate	TP (/4500)	FP rate	FP (/500)
0.05	50	1.0	4500	0.048	24
0.01	50	0.892	4015	0.008	4
0.005	50	0.788	3547	0.004	2
0.05	100	1.0	4500	0.048	24
0.01	**100**	**1.0**	**4500**	**0.008**	**4**
0.005	100	0.825	3712	0.004	2
0.05	200	1.0	4500	0.042	21
0.01	**200**	**1.0**	**4500**	**0.008**	**4**
0.005	200	0.796	3583	0.002	1

Table 4: Performance of Isolation Forest in detecting poisoned local models. True positives (TP) and false positives (FP) for different *contamination* and *estimator* values. The best combinations of FPR and TPR are marked.

Tables 2, 3 and 4 present the performance of One-class SVM, Elliptic envelope and Isolation forest in detecting poisoned local models. We see that under optimally selected hyperparameters, all the three methods achieve good performance, detecting over 99.9% of poisoned models (TPR) while incorrectly identifying less than 1% of benign models as poisoned (FPR). Looking closer at the results, we see that the Isolation forest performance is very sensitive to its *contamination* parameter: the TPR varies significantly with small changes in contamination values. Optimal hyperparameter values usually change over time in non-stationary classification problems, such as poisoning detection, because poisoning attacks evolve with attackers adapting their methods. This requires us to update hyperparameters to avoid degraded performance. Because of the high sensitivity to the *contamination* parameter, we discard Isolation forest as a potential solution.

Elliptic envelope and One-class SVM with a Sigmoid kernel are less sensitive to the hyperparameters values and they provide similar results. While One-class SVM slightly outperforms the Elliptic envelope model, we select the latter as our method to detect poisoned PLD_i models, which is confirmed in large-scale experiments described in Section 4. Another reason is that Elliptic envelope is a simpler model than One-class SVM and, as a rule of thumb in machine learning, simpler models that can provide high TPR and FPR results are preferable because they are more likely to generalise better and avoid over-fitting. We refer to our Elliptic envelope detection method as ***EE-detection***.

3 Evaluation of defense approaches

3.1 Goal

We want to evaluate the performance of our *EE-detection* defense at preventing training time poisoning attacks against PLD_{global}. We compare EE-detection to two main approaches developed to date to protect from poisoning attacks in distributed and federated learning. The first approach is similar to ours; it tries to detect poisoned local models in order to discard them from aggregation. We use the poisoning detection method called *AUROR* [STS'16] (presented in Section 1.3.1) in this class for comparing. The second approach aggregates local models in a robust manner that should be inherently resilient to poisoning attacks without being specifically designed to counter these attacks. We implement three attack-agnostic approaches for secure aggregation (presented in Section 1.3.2):

- *Median aggregation*
- *Trimmed mean aggregation*.
- *Weight normalisation*

These different defenses are evaluated against the two poisoning attacks presented in Section 1.2, namely *"Increase target process launch count"* and *"Inject rare process launch events unrelated to the attack"* that we simply refer to as *targeted* and *unrelated attack* respectively in this section. We test and compare these attacks in both a single attacker setting and a distributed attacker setting (controlling several local models).

In addition to the effectiveness in protecting from poisoning attacks, we evaluate how each defense impacts the global model aggregated when it is deployed. Discarding local models detected as poisoned from aggregation and using robust aggregation methods necessarily modifies the resulting aggregated model even if there is no poisoning attack. This modification can be undesirable if it modifies the anomaly category predicted by PLD_{global} for (some of) the process pairs. It can make usually normal process pairs look more or less anomalous and jeopardise the end goal of the PLD model: detecting malicious process launches.

3.2 Datasets, metrics and experimental setup

3.2.1 Datasets and target process launches

We start by evaluating defenses against poisoning attacks using static datasets of medium size. We selected a fixed set of 3,250 clients that send local PLD models on a regular basis to train the global PLD model (online training). The number of local models sent every day by these clients varies slightly because some clients may not be turned on or may have no network connectivity at certain periods.

Most of our analysis to evaluate the poisoning prevention / detection capability is performed using local models received on September 15 and 16, 2020. We build the model PLD^{t}_{global} from 3,147 local models received on September 15. PLD^{t}_{global} is used by the attacker as a basis to perform its poisoning attack and to compute the process launch events it needs to inject in compromised local model(s) in order to poison PLD^{t+1}_{global}. PLD^{t+1}_{global} is unknown to the attacker at the time of the attack and it is built from 3,149 local models received on September 16th. Some of these models will be considered compromised and modified by the attacker while performing its poisoning attacks.

We collect two additional datasets of local models to evaluate EE-detection and AUROR. The first set is composed of local models received from the same set of clients between September 7 and September 13, it contains 18,291 local models and it is used to train the EE-detection model. We also use this set to find the proper threshold for AUROR. The second set will be used to evaluate the incorrect detections generated by these detection methods. It contains 18,736 local models received between September 14 and September 20. The datasets of local models are summarised in Table 5.

Name	Local models	Date	Purpose
PLD^{t}_{i} set	3,147	15/09	Train PLD^{t}_{global} used to design poisoning attacks
PLD^{t+1}_{i} set	3,149	16/09	Train PLD^{t+1}_{global} targeted by poisoning attacks
PLD_i-*train*	18,291	7-13/09	Train AUROR and our EE-detection model
PLD_i-*test*	18,736	14-20/09	Test false positives for AUROR and EE-detection

Table 5: Datasets of local PLD models used in the experiments

To evaluate the effectiveness of each defense at preventing poisoning attacks, we simulate several poisoning attacks that would be launched by an attacker. We select five process pairs $(proc_M, proc_T)$ having a low anomaly category as our targets for the poisoning attack. These process pairs are presented in Table 6 where we can see that they all belong to anomaly categories 5 to 7 depending on the aggregation method used to produce PLD^{t+1}_{global}. We can see that these process pairs are reported by a small number of clients (1 – 5) and they have a low count of calls when summed over all our clients (2 – 24). The goal of the poisoning attacks will be to decrease the anomaly category of these process launches in PLD^{t+1}_{global} by modifying / poisoning some local models in PLD^{t+1}_{i}.

$proc_M$	$proc_T$	Base PLD^{t+1}_{global}		Base AC	Median AC	Trim. mean AC	Norm. AC
		#clients	**count**				
cmd.exe	*cabarc.exe*	3	24	7	6	6	6
services.exe	*check_mk_agent.exe*	2	3	6	6	6	6
KAPS.exe	*conhost.exe*	4	18	6	5	5	6
Teams.exe	*OUTLOOK.EXE*	5	5	6	6	6	6
ngentask.exe	*WerFault.exe*	1	2	6	6	6	6

Table 6: Target process pairs to use for poisoning and their original anomaly categories (AC) in PLD^{t+1}_{global} using different aggregation methods. Base - original sum aggregation of PLD.

3.2.2 AUROR implementation

As described in section 1.3.1, the AUROR detection mechanism consists of the following steps.

Identifying indicative features. In the first step, AUROR identifies the features showing anomalous pattern in the presence of an attacker. In Section 2.2, we presented that we can model the distribution of each local PLD model using a vector containing 18 features. Each of these features represents a portion of process pairs belonging to a given anomaly category and it can be significantly modified as a result of a poisoning attack. With regards to our designed attack scenarios, we consider all 18 features designed to be used by EE-detection as *indicative features* for AUROR.

Detecting malicious clients. In this step, we should group the clients into two clusters using every indicative feature. We select k-means clustering as the clustering algorithm and apply it on each indicative feature separately. As a result, we will have 18 clustering decisions corresponding to 18 features in the data set. Based on the assumption of AUROR on bounding the number of malicious clients, the cluster with the smallest number of clients is considered as suspicious. If a client is placed in the suspicious cluster by at least *t* indicative features, it will be marked as compromised and excluded from training.

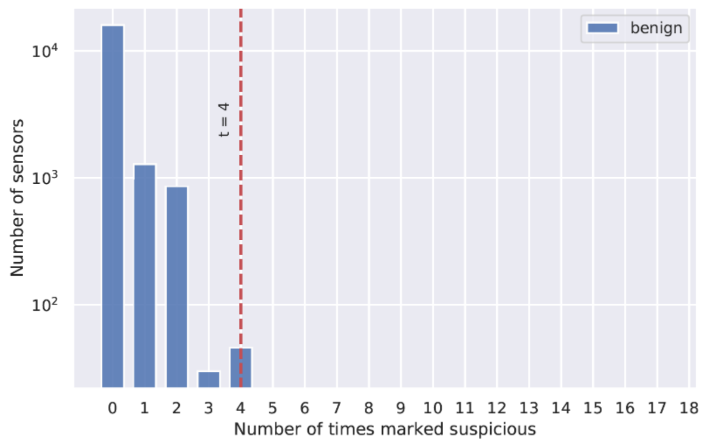

Figure 5: Number of clients vs number of times they were in a suspicious cluster. The red vertical line shows the selected threshold value. A client is marked as malicious if it is in a suspicious cluster at least 4 times.

As mentioned earlier, in order to find the proper threshold value, we use PLD_i-*train*. According to Figure 5, we choose 4 to bound the number of false positives in the predictions of AUROR on the selected data set.

3.2.3 Evaluation metrics

Since we evaluate and compare different defenses against poisoning attacks, the metrics used to evaluate their performance require to be different. Robust aggregation methods defend against poisoning by aggregating local models in a different manner than the sum of process counts used by our original *sum* aggregation of PLD models. This has the potential to partially mitigate poisoning attacks by preventing the process pairs targeted by the poisoning attack $(proc_M, proc_T)$, to reach its targeted low anomaly category. Consequently, each of these defenses requires to build PLD_{global}^{t+1} from benign and poisoned models to evaluate their performance. A sensible metric to measure their success is the change in anomaly category for the target process pair between PLD_{global}^{t+1} not poisoned, PLD_{global}^{t+1} poisoned with no defense and PLD_{global}^{t+1} aggregated with the robust aggregation method under poisoning attack. A robust aggregation method successfully protects against poisoning attack if the anomaly category of $(proc_M, proc_T)$ remains high, ideally as high as for PLD_{global}^{t+1} not poisoned (reported in Table 6).

On the other hand, AUROR and EE-detection detect poisoned local models before they are aggregated, and they do not modify the original aggregation of local PLD models. If a poisoned local model is detected as such, it will be discarded from aggregation and have no impact on PLD_{global}^{t+1}, the poisoning attack will fail. However, if a poisoned local model is missed, the poisoning attack will be fully successful and poisoning detection methods are all-or-nothing type of defenses (strictly speaking, in the "single client" attack case). The success of these methods in preventing poisoning attacks can be measured by poisoning several local models with a given attack and computing the ratio of poisoned models successfully detected. This metric is called Recall or True Positive Rate (TPR). If the TPR is high, close to 1, it means the defense detects most of the poisoned models and it effectively protects against poisoning attacks. An additional measure for the effectiveness of a detection method is the Precision, which is computed as the ratio of poisoned models successfully detected as poisoned among all models detected as poisoned (including benign models incorrectly detected). If the Precision is high, close to 1, most of the poisoned models detected by the method are actually poisoned and the method is precise.

In addition to the success in preventing poisoning attacks, defenses must also be evaluated in terms of impact they have at modifying PLD_{global} by modifying the aggregation method or incorrectly discarding benign models from aggregation. For our defense and AUROR we evaluate this impact by computing the ratio of benign models incorrectly detected as poisoned, also called the False Positive Rate (FPR). The FPR must be low, close to 0, to reduce the impact of the defense on the normal process. For robust aggregation methods, we propose to measure the difference in anomaly categories for reference process pairs, when these categories are computed using the sum of process counts (original PLD_{global}) and when using a robust aggregation method. Ideally, this difference must remain as low as possible and close to 0. The original *sum* aggregation of PLD models was extensively tested and it was confirmed that it currently computes sensible anomaly category ranks that are useful for detecting malicious process launch events.

3.3 Prevention of poisoning attack with single attacker

We start by evaluating the resilience to poisoning attack considering a single attacker, i.e., only one of the 3,149 models in PLD_i^{t+1} will be modified by the attacker. We evaluate both *targeted* poisoning attack and *unrelated* poisoning attack targeting the five process pairs presented in Table 6.

3.3.1 Targeted poisoning attack

We evaluate the targeted poisoning attack first. The attacker injects a record for the target process pair $(proc_M, proc_T)$ with a required count of launches in the compromised local model. The attacker computes this count using PLD_{global}^{t} aggregated with the corresponding aggregation method: when attacking PLD_{global}^{t+1} aggregated using median, the attacker uses PLD_{global}^{t} aggregated using median to design its attack. For each attack we target the anomaly category 1.

We evaluate the robust aggregation methods namely *median, weight normalisation* with trimming ratio of 10%, and *trimmed mean* with trimming ratios of 10%, 30% and 50%. We compute the anomaly category and the count of process launches for each target process pair in PLD_{global}^{t+1} after aggregation of the poisoned local model together with all the benign local models. Table 7 reports the results of this experiment for each aggregation method. We also present results for the original sum aggregation method of the global model. This result provides a baseline for the success of the poisoning attack. We can see that the poisoning attack always succeeds against the baseline aggregation method. Most of the target process pairs belong to the targeted anomaly category 1 while one of them belongs to anomaly category 2. We can see that the attack increases the count of process launches for the target process pair significantly when compared to the original count presented in Table 6.

$proc_M$	$proc_T$	Baseline		Median		Normalisation	
		Count	AC	Count	AC	Count	AC
cmd.exe	*cabarc.exe*	2,684,834	1	32	5	39,699	1
services.exe	*check_mk_agent.exe*	15,216	1	6	5	43,192	1
KAPS.exe	*conhost.exe*	86,354	2	25	5	8,948	3
Teams.exe	*OUTLOOK.EXE*	13,916	1	6	5	14,524	1
ngentask.exe	*WerFault.exe*	34,678	1	4	5	7,460	1

$proc_M$	$proc_T$	Trim. mean (10%)		Trim. mean (30%)		Trim. mean (50%)	
		Count	AC	Count	AC	Count	AC
cmd.exe	*cabarc.exe*	216,198	1	24	5	9	5
services.exe	*check_mk_agent.exe*	11,256	1	11,424	2	3	5
KAPS.exe	*conhost.exe*	18,602	2	18	5	10	5
Teams.exe	*OUTLOOK.EXE*	9,006	1	5	5	5	4
ngentask.exe	*WerFault.exe*	6,893	1	2,395	1	2	4

Table 7: Results for targeted poisoning attacks with single attacker against 6 aggregation methods. We see anomaly category and event count for the target process pair in PLD_{global}^{t+1}. High anomaly categories (green) show when the aggregation method protects from poisoning.

We can see from the results that *weight normalisation* and *trimmed mean* with a low trimming ratio are ineffective in preventing the targeted poisoning attack. This can be explained by the fact that normalisation does not succeed to scale down enough the count of process launch events for the targeted process pair. This means that the addition of process launch events for this pair does not increase the total count of process launches reported by the compromised client to a large enough extent to be mitigated by normalisation. Regarding the ineffectiveness of trimmed mean with a small trimming ratio, this can be explained by the fact that rare / malicious process pairs are by definition reported by very few clients. This has been observed in Table 6. Consequently, even if the count reported by the compromised client is large, it will not be discarded because there are too few other counts for this same process pair. The target process pairs we picked are reported in at most five benign local PLD models and by trimming 10% of the largest counts we never discard the one injected by the attacker. However, when we increase the trimming ratio, we can see that trimmed mean starts to be effective. It successfully protects from every attack when the trimming ratio is 50%, keeping a low count of launches and the anomaly category for target process pairs remains quite high, between 4 and 5. The same effectiveness can be observed for median aggregation, all process pairs remain in anomaly category 5. Considering a single compromised local PLD model, the large count it reports for the target process pair is always discarded from aggregation as it never represents the median value. We can conclude that *median* and *trimmed mean* with a large trimming ratio are the only robust aggregation methods that can prevent the targeted poisoning attack against the global PLD model.

We also note that the aggregation methods obviously modify PLD_{global}. This can be observed by the count of process launches required to be injected to succeed in the targeted poisoning attack. We can see that weight normalisation and trimmed mean reduce the number of launch events required for the target process pair to reach anomaly category 1 when compared to the baseline aggregation method. This means that poisoning attacks against PLD_{global} using these types of aggregation method can be harder to detect.

Now, we evaluate the performance of poisoning detection methods in protecting against the targeted attack. AUROR and EE-detection try to detect 1,000 local models poisoned with the same attack (targeting the same process pair) among 3,149 local models in the PLD_i^{t+1} set. For this experiment, we trained the EE-detection model using PLD_i-*train* and the contamination value of 0.05. The results of the experiments are presented in Table 8. It reports the number of poisoned models detected (TP), its corresponding ratio of detection (TPR) and the Precision for each detection method.

$proc_M$	$proc_T$	AUROR			*EE-detection*		
		TP	TPR	Precision	TP	TPR	Precision
cmd.exe	*cabarc.exe*	1,000	1.00	0.72	1,000	1.0	0.87
services.exe	*check_mk_agent.exe*	875	0.87	0.68	1,000	1.0	0.86
KAPS.exe	*conhost.exe*	943	0.94	0.71	1,000	1.0	0.85
Teams.exe	*OUTLOOK.EXE*	876	0.87	0.69	1,000	1.0	0.86
ngentask.exe	*WerFault.exe*	917	0.92	0.70	1,000	1.0	0.87

Table 8: Results of targeted poisoning attacks detection with single attacker against AUROR and EE-detection. True positives (TP), TP rate (TPR) and Precision in detecting 1000 poisoned models among 3,149 local models in the PLD_i^{t+1} set.

We can see from Table 8 that EE-detection outperforms AUROR. Elliptic envelope detects all 1,000 poisoned models for every process pair while AUROR detects between 87% and 100% of them. AUROR is only able to detect all poisoned model for the *(cmd.exe, carbac.exe)* process pair, which represents the most aggressive poisoning attack: the one requiring the largest injection of process launch events (cf. Table 7). The fact that AUROR does not have 100% detection rate in most cases means that an attacker can find some local PLD models which, when poisoned, would systematically avoid detection. In the poisoned model detection scenario, the poisoning attack is fully successful if only one poisoned model avoids detection because it will be aggregated into PLD_{global}. Thus, 100% detection rate is necessary to effectively defend against an attacker who can choose the local model they want to poison.

AUROR has precision around 70% for all the tested attacks, meaning that only two out of three local models it detects as poisoned are actually poisoned and the rest are benign. In contrast, EE-detection has precision around 85% meaning that it makes roughly twice fewer mistakes compared to AUROR: only one in six models predicted as poisoned by EE-detection is actually benign.

3.3.2 Unrelated poisoning attack

We evaluate the unrelated attack in the same setup as for the targeted attack. The difference is that this attack will inject many process pairs having small launch counts and which are unrelated to the target process pair. The unrelated poisoning attack does not reach the target anomaly category, but it decreases the anomaly category of the target process pair. Our results for different aggregation methods are reported in Table 9. Baseline results correspond to the best anomaly category the attacker can reach using the unrelated attack given the target process pair. We can see that the unrelated attack reduces the anomaly category from 6 to 4 in most cases.

$proc_M$	$proc_T$	Baseline	Median	Trimmed mean			Weight Normalisation
				10%	30%	50%	
cmd.exe	*cabarc.exe*	5	4	5	4	4	5
services.exe	*check_mk_agent.exe*	4	4	4	4	5	4
KAPS.exe	*conhost.exe*	4	4	5	5	5	4

Teams.exe	*OUTLOOK.EXE*	4	4	4	4	4	4
ngentask.exe	*WerFault.exe*	4	4	5	5	4	4

Table 9: Results for unrelated poisoning attacks with single attacker against 6 aggregation methods. We see anomaly category for the target process pair in PLD_{global}^{t+1}. High anomaly categories (green) show when the aggregation method mitigates poisoning.

Looking at the results in Table 9, we can see that none of the robust aggregation methods can completely protect from the unrelated poisoning attack. Median aggregation is ineffective because the unrelated poisoning attack injects rare processes that are for the most part reported only by the compromised local PLD model. Thus, there is only one value to compute the median from. The same explanation holds for trimmed mean aggregation which is in most cases ineffective while sometimes it mitigates the attack to a certain extent, preventing the target process pair to reach anomaly category 4 and keeping it at 5. Finally, weight normalisation is completely ineffective against this attack because it injects only process pairs with very small launch counts. This does not increase the total count of process launches of the compromised local PLD model significantly and it is not much affected by normalisation.

We evaluate the performance of poisoning detection methods at protecting against the unrelated attack. AUROR and EE-detection try to detect 1,000 local models poisoned with the same unrelated attack (targeting the same process pair) among 3,149 local models in the PLD_i^{t+1} set. For this experiment, we trained our *EE-detection* model using PLD_i*-train* and the contamination value of 0.001. The results of the experiments are presented in Table 10. It reports the number of poisoned models detected (TP), its corresponding ratio of detection (TPR) and the Precision for each detection method.

$proc_M$	$proc_T$	AUROR			Our *EE-detection*		
		TP	TPR	Precision	TP	TPR	Precision
cmd.exe	*cabarc.exe*	664	0.66	0.64	1000	1.0	0.998
services.exe	*check_mk_agent.exe*	956	0.96	0.72	1000	1.0	0.999
KAPS.exe	*conhost.exe*	963	0.96	0.72	1000	1.0	0.998
Teams.exe	*OUTLOOK.EXE*	962	0.96	0.62	1000	1.0	0.997
ngentask.exe	*WerFault.exe*	931	0.93	0.72	1000	1.0	0.999

Table 10: Results of unrelated poisoning attacks detection with single attacker against AUROR and our Elliptic Envelope detection method. True positives (TP), TP rate (TPR) and precision at detecting 1000 poisoned models among 3,149 local models in the PLD_i^{t+1} set.

We see that our detection method outperforms AUROR against the unrelated poisoning attack. EE-detection is again able to detect all 1,000 poisoned models for every process pair. AUROR performs better against this attack for most process pairs though, being able to detects between 93% and 96% of poisoned models. However, its performance degrades against the *(cmd.exe, carbac.exe)* process pair where it was previously able to detect all poisoned models but now detects only 66% of them. It seems given a target process pair to poison, a knowledgeable attacker will be able to choose the targeted or the unrelated poisoning attack in order to best evade AUROR since its performance for the same process pair varies significantly depending on the attack. AUROR always seems vulnerable to at least one of our two poisoning attacks.

The precision of AUROR degrades slightly against the unrelated poisoning attack, down to 62% in some cases, meaning it makes more errors for benign local PLD models. On the other hand, EE-detection has a very high precision, over 99.5% showing that in addition to detecting all poisoned local models, it does it with incorrectly identifying only very few benign models as anomalous.

Takeaways: We see that robust aggregation methods can, in some cases, successfully defend against poisoning attacks involving a single attacker. *Median* and *trimmed mean* with a large trimming ratio successfully protect from the *targeted* poisoning attack against PLD_{global}. On the other hand, none of the robust aggregation methods was able to fully mitigate the *unrelated* poisoning attack. This overall ineffectiveness of robust aggregation methods against the studied poisoning attacks is due to two main factors:

- The nature of the global PLD model, composed of many rare process pairs created only by a very small number of clients. This sparsity of process pairs among clients can be observed in Figure 6 where we see that most of the process pairs are only observed in fewer than 10 out of 3149 clients in the PLD_i^{t+1} set.
- The nature of the studied poisoning attacks, which actually target these rare process pairs, trying to increase their number of launches.

Robust aggregation methods leverage the assumption that many clients provide information about the same features/classes and that this information is consistent among benign clients, i.e., the data is independent and identically distributed (i.i.d.). In many concrete use cases, like PLD, this assumption does not hold. It can be observed in Figure 6 that the vast majority of our features (process pairs) have a value set for only a few clients. Any aggregation method, including robust aggregation, will be unable to infer a representative benign "value" for such rare features and these features become easy to poison. By controlling only a few compromise clients, the value of these features can be biased at the attacker's will. Poisoning attacks against the PLD_{global} model target these rare features (process pairs) for which very few clients (if any) provide information and this is the case of all poisoning attacks against any anomaly detection system: anomalies are the target of the attack and they are rare.

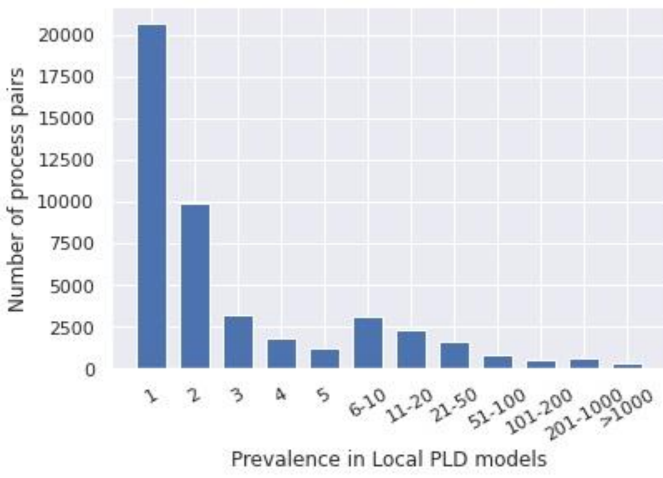

Figure 6: Number of process pairs vs. their prevalence in 3,149 local models for the PLD_i^{t+1} set. Most of the process pairs are sparse and only observed by a few clients.

We observed during our experiments that process pairs that can better be protected from (targeted) poisoning attacks were the ones for which more clients were reporting information (when using *trimmed mean* aggregation). Nevertheless, our experiments show that robust aggregation methods are not effective at defending poisoning attacks against models trained using federated learning where features are sparsely defined among clients and the data is not i.i.d. Most unsupervised anomaly detection systems and recommender systems are examples of such models.

On the other hand, our *EE-detection* detected all poisoned local PLD models and AUROR was able to detect most poisoned local models. These detection methods do not suffer from the feature sparsity. They try to separate local models into two groups: normal/benign and anomalous/poisoned, and totally discard anomalous models rather than trying to use some information from any model given that the model provides information about some rare features (as robust aggregation does). Using a large number of rare process pairs in a poisoned model makes it stand out when compared to benign models. By using a relevant feature representation, both detection methods were able to separate poisoned models from benign ones and effectively prevent poisoning against PLD_{global} federated training. In general, our results show that detection methods are more effective than robust aggregation methods at preventing poisoning attacks in federated learning when features are sparse and the data is not independent and identically distributed among clients.

3.4 Prevention of distributed poisoning attack

We evaluate now the effectiveness of different defenses methods at preventing poisoning attacks in a distributed setting. The attacker will perform the same poisoning attacks but distribute them among several compromised clients and local PLD model. This reduces the count of process launch events in each compromised model and makes the attack more difficult to detect and prevent. We only evaluate the targeted poisoning attack in this distributed setting but the result we obtain generalise to the unrelated attack. We take the same 5 target process pairs used in the previous section and we target once again to reach anomaly category 1.

We start by evaluating the resilience of robust aggregation methods: *median, weight normalisation* and *trimmed mean* with trimming ratio of 10%, 30% and 50% to 5 attackers. We compute the anomaly category and the count of process launches for each target process pair in PLD_{global}^{t+1} after aggregation of the poisoned local PLD models together with the remaining benign local models. Table 11 reports the results of this experiment for each aggregation method. We can see that the distributed poisoning is slightly more effective against the baseline aggregation method since each attack reaches its anomaly category 1 goal. The final counts of process launches for each process pair are similar to the ones we obtained using a single attacker.

$proc_M$	$proc_T$	Baseline		Median		Normalised	
		Count	AC	Count	AC	Count	AC
cmd.exe	*cabarc.exe*	2,796,704	1	153,584	1	64,914	1
services.exe	*check_mk_agent.exe*	15,853	1	10,430	1	11,896	1
KAPS.exe	*conhost.exe*	89,953	1	27,342	1	24,071	2
Teams.exe	*OUTLOOK.EXE*	14,500	1	10	5	19,197	1
ngentask.exe	*WerFault.exe*	36,127	1	7,434	1	13,511	1

$proc_M$	$proc_T$	Trim. mean 10%		Trim. mean 30%		Trim. mean 50%	
		Count	AC	Count	AC	Count	AC
cmd.exe	*cabarc.exe*	180,169	1	35,134	1	35,134	1
services.exe	*check_mk_agent.exe*	9,383	2	10,388	1	10,388	1
KAPS.exe	*conhost.exe*	15,508	2	14,973	1	14,973	1
Teams.exe	*OUTLOOK.EXE*	7,510	1	3,425	1	5	4
ngentask.exe	*WerFault.exe*	5,747	1	2,182	1	2,182	1

Table 11: Results for distributed targeted poisoning attacks (5 attackers) against 6 aggregation methods. We see anomaly category and event count for the target process pair in PLD_{global}^{t+1}. High anomaly categories (green) show when the aggregation method protects from poisoning.

Looking at results in Table 11, we see that the distributed poisoning attack, using only 5 attackers, defeats all robust aggregation methods in almost every case. The attack reaches its goal of anomaly category 1 or 2 and it succeeds to increase the counts of process launches to a large extent. Only attacks are prevented, they both target the process pair *(Teams.exe, OUTLOOK.EXE)* and they are prevented by median aggregation and trimmed mean aggregation with 50% trimming ratio. This attack can be prevented because 5 benign clients report process launches for this process pair. The 5 additional process pairs reported in compromised local PLD models are all discarded by both aggregation method. However, if there would be only one more attacker, the poisoning attack would succeed.

We confirm this observation by launching the same distributed attack against median aggregation and trimmed mean aggregation with 50% trimming ratio using 10 attackers. The results are reported in Table 12 where we see that all attack succeed and reach anomaly category 1.

$proc_M$	$proc_T$	Baseline		Median		Trim. mean 50%	
		Count	AC	Count	AC	Count	AC
cmd.exe	*cabarc.exe*	2,796,704	1	149,747	1	270,244	1
services.exe	*check_mk_agent.exe*	15,853	1	10,728	1	14,073	1
KAPS.exe	*conhost.exe*	89,953	1	25,522	1	23,248	1
Teams.exe	*OUTLOOK.EXE*	14,500	1	16,890	1	11,265	1
ngentask.exe	*WerFault.exe*	36,127	1	8,184	1	8,622	1

Table 12: Results of distributed targeted poisoning attacks (10 attackers) against 3 aggregation methods. We see anomaly category and event count for the target process pair in PLD_{global}^{t+1}. None of the aggregation methods protects from poisoning.

We evaluate the performance of poisoning detection methods at protecting against the targeted attack in a distributed setting. AUROR and our Elliptic Envelope detection method try to detect targeted attacks targeting 5 process pairs and distributed between 5, 10, 20 and 50 local PLD models poisoned with among 3,149 local models in the PLD_i^{t+1} set. For this experiment, we trained the *EE-detection* model using PLD_i*-train* and a contamination value of 0.05. The results of each experiment are presented in Table 13 that reports the number of poisoned models detected (TP) depending on the number of poisoned models / attackers (5 A / 10 A / 20 A / 50 A).

$proc_M$	$proc_T$	AUROR				Our *EE-detection*			
		5 A	10 A	20 A	50 A	5 A	10 A	20 A	50 A
cmd.exe	*cabarc.exe*	5	10	20	47	5	10	20	49
services.exe	*check_mk_agent.exe*	3	5	10	7	5	10	20	50
KAPS.exe	*conhost.exe*	5	8	16	38	5	10	20	50
Teams.exe	*OUTLOOK.EXE*	2	3	5	8	5	10	20	50
ngentask.exe	*WerFault.exe*	5	9	19	14	5	10	20	50

Table 13: Results of distributed targeted poisoning attacks against AUROR and our Elliptic Envelope detection method. Number of poisoned models detected out of 5, 10, 20 and 50 poisoned models. Green = all poisoned models detected / Red = 50% or less poisoned models detected.

Table 13 shows that *EE-detection* is able to all poisoned models in almost every case. The distribution of the attack among up to 50 attackers, which dived the number of process launch events in each poisoned model by the same factor, has almost no effect on its detection capability. The only miss of 1 model among 50 poisoned for the *(cmd.exe, carbac.exe)* process pair will have little impact on PLD_{global} and the poisoning attack will be mitigated. In contrast, AUROR fails to detect many of the poisoned models because of distributing the attack. We see that while it always detects some poisoned models it misses more than 50% of them in two attacks. This means that most poisoned models would be aggregated in PLD_{global} and the poisoning attack would succeed to some extent. It is also worth noting that along these few tens of poisoned models detected, AUROR also incorrectly over 500 benign local PLD models as poisoned. This can be explained by the fact that AUROR uses clustering as base method to form one group of benign models and one group of poisoned models. Forming unbalanced clusters is challenging for many clustering algorithms, which explains that when there are very few poisoned models compared to the number of benign models, many benign models may end up clustered with poisoned models and be discarded from aggregation.

Takeaways: We confirm that robust aggregation methods are ineffective at protecting against poisoning attacks when features are sparse, and the data is non i.i.d. This conclusion is clearer in the distributed attack scenario where an attacker can evade all robust aggregation techniques, including *median* and *trimmed mean (50%)* by controlling only a few compromised clients. This complete evasion is possible because the attacker can control the majority of clients reporting values for the target process pair to poison, given that this process pair is only reported by a few (1-5) benign clients to begin with. The sparsity of features is again the cause for this evasion. It basically strengthens the adversary model, making the adversary able to control the majority (>50%) of clients reporting values for the targeted process pair, since this process pair is not reported by the 1000s of clients contributing to build the global model but only by 1-5 benign clients.

On the other hand, *EE-detection* is very effective at mitigating the distributed poisoning attack while *AUROR* start to show its limitation. The increasing distribution of the attack makes each poisoned local PLD model more similar to benign models and it seems like the difference between these models is not sufficient for AUROR to separate them in different clusters. Despite using the same input features, *EE-detection* is much more effective than *AUROR* against the distributed attack. This shows that *AUROR* is sensitive to large outliers and it is suitable to detect obvious and aggressive poisoning attacks represented by highly compromised local models. However, *AUROR* does not seem to be able to detect smaller outliers and it is vulnerable to stealthy poisoning attacks while *EE-detection* remains effective at detecting them.

3.5 Impact of defenses on global model utility

The effectiveness at preventing poisoning attacks is not the only criterion to evaluate a defense against poisoning attacks. Another criterion being as important is to preserve the utility of the aggregated global model. Assuming the training and aggregation method of the global model are optimised for having the best utility when aggregated from benign local models, the same utility must be achieved when a given defense is deployed. A defense against poisoning attacks must not significantly modify an aggregated global model when no poisoning attack is ongoing.

PLD_{global} aggregated using *sum aggregation* has been tuned, tested and shown to be effective in detecting malicious process launch events. We use this model as reference baseline for a global model achieving best utility. We want to assess how defenses to poisoning attacks modify and degrade the results of this baseline model.

We evaluate this property by building PLD_{global} models using the same base dataset PLD_i^{t+1} while different defenses are deployed. Robust aggregation methods *median, weight normalisation* and *trimmed mean* use the PLD_i^{t+1} set as it is to train PLD_{global}. We only evaluate *trimmed mean* with 30% and 50% trimming ratio because a 10% trimming ratio was ineffective at preventing poisoning attacks. *AUROR* and *EE-detection* are first ran against the PLD_i^{t+1}set and every local model detected as anomalous/poisoned is removed from the PLD_i^{t+1} set. The remaining set of undetected models is used to train PLD_{global} using the original *sum aggregation.* Out of 3,149 local models in the PLD_i^{t+1} set, 568 were detected as anomalous by *AUROR* and 241 by *EE-detection.* We compare the resulting PLD_{global} models to the baseline PLD_{global} aggregated using *sum aggregation.* The comparison consists in collecting all process pairs having an anomaly category lower than or equal to 2 in the baseline PLD_{global} or in the PLD_{global} to compare to. For each of these pairs we compute the absolute difference in anomaly category score, e.g., AC2 – AC2 = 0, AC4 – AC2 = 2. We aggregate these score differences among all process pairs compared to quantify the difference between two PLD_{global} models.

Table 14 shows the average difference in anomaly category score between the baseline PLD_{global} model and PLD_{global} models aggregated using each of the poisoning defenses we evaluated so far. Figure 7 shows the exact distribution of process pairs based on their difference in anomaly scores. We see that robust aggregation methods modify the anomaly category score significantly, increasing/decreasing it by 1 on average. This mean that on average, every process pair change anomaly category when using robust aggregation method instead of *sum aggregation.* This can be observed in Figure 7 where we see that most process pairs have their anomaly category modified, with a pic on the value 1 for each robust aggregation method. Also, a significant number of process pairs (100s) change their anomaly category by 2 or more. Normalised aggregation if the most resilient to this difference in anomaly category score, it is 0.75 on average with a maximum of 4. It is also worth noting that on the flip side, normalised aggregation was the least effective defense against poisoning attacks. On the other hand, median and trimmed mean aggregation (50%) were the most effective defense to poisoning attacks but they also cause the largest difference in anomaly category scores, up to 5. This is concerning because this mean that some highly anomalous and rare process pairs may move from AC6 to the most common category AC1 by using one of these aggregation methods. The effect of this change is the same as the goal of a poisoning attack where the attacker wants to dramatically decrease the anomaly category of a target process pair. Thus, we can conclude that robust aggregation methods modify the PLD_{global} model to large extent and that this change decreases significantly the utility of PLD_{global}.

AC difference	Median	Weight normalisatio n	Tmean 30%	Tmean 50%	AUROR	EE-detection
Average (± std)	0.92 (±0.82)	0.75 (±0.72)	0.96 (±0.76)	1.13 (±0.81)	0.26 (±0.46)	0.14 (±0.37)

Table 14: Average and standard deviation of the difference in anomaly category score between baseline PLD_{global} vs. PLD_{global} aggregated using *median, weight normalisation*, *trimmed mean (30% and 50%)* and *sum aggregation* when AUROR and EE-detection are deployed.

Despite incorrectly identifying a few hundred local models as poisoned, *AUROR* and *EE-detection* are able to produce a PLD_{global} model being close to baseline. The average difference in anomaly category score is 0.26 and 0.14 for *AUROR* and *EE-detection* respectively, which is 4 to 6 times less than the difference produced by robust aggregation methods. We can also observe in Figure 7 that most process pair do not change anomaly category when using these detection methods. Our *EE-detection* is the defense causing the smallest change when compared to the baseline PLD_{global}. It causes a few process pairs to change anomaly category score by 1 or 2, while only one process pair changes it by a value of 3. We can conclude that *EE-detection* is the defense degrading the least the utility of the global PLD_{global} model and the one which preserves its optimal performance. We also computed the false positive rate (FPR) for each of these methods on the larger PLD_i-*test* dataset and found that *AUROR* produces 0.3% FPR, while *EE-detection* produces 6.1% FPR with a contamination of 0.05 and 0.1% FPR with a contamination of 0.001. The contamination must be set as a trade-off between detection of poisoned models and generation of false positives.

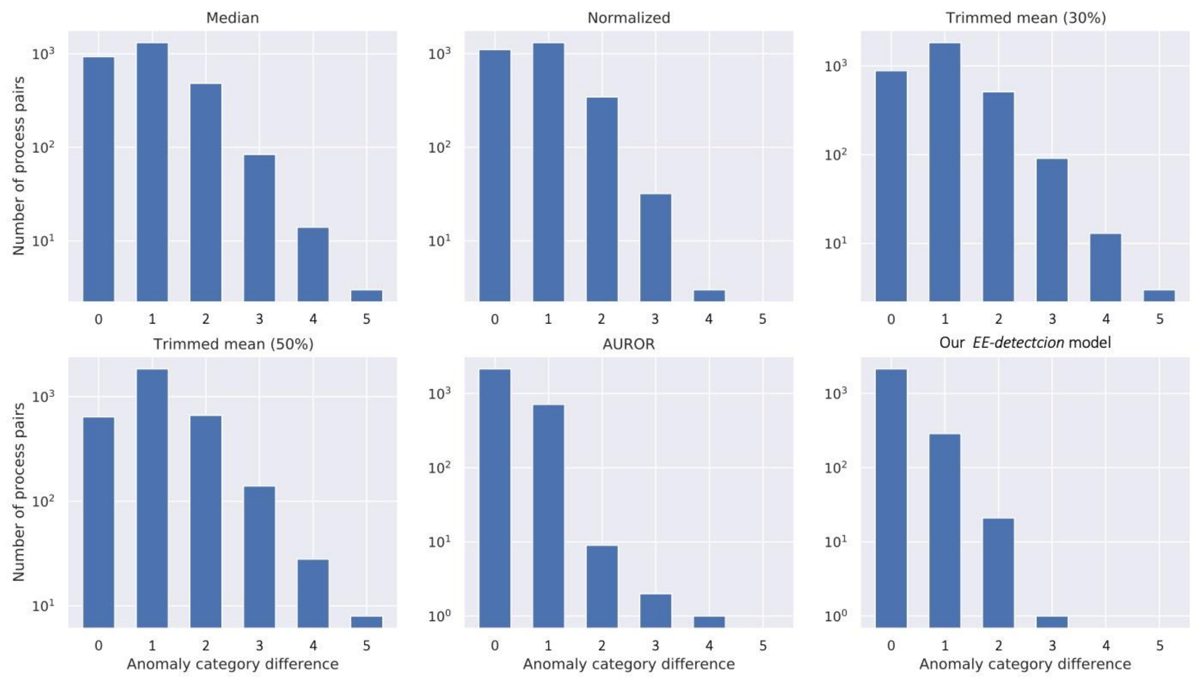

Figure 6: Number of process pairs having 0, 1, 2, 3, 4, and 5 differences in anomaly category score between baseline PLD_{global} vs. PLD_{global} aggregated using median, weight normalisation, trimmed mean (30% and 50%) and sum aggregation when AUROR and EE-detection are deployed.

Takeaways: Robust aggregation methods have the potential to produce similar models given that the data they aggregate meets some assumptions about distribution, e.g., it follows a normal distribution where median and mean are close, and outliers are in minority. However, considering our non-i.i.d. anomalous process detection scenario, we have observed that modifying the method for aggregating a global model can change this model to a large extent and modify its predictions for most inputs. If we have evaluated that one aggregation method provides the best performance when fed with benign data, other aggregation methods are not anymore options to be used for defending against poisoning since they would degrade the utility of the model when there is no attack. Thus, robust aggregation methods cannot be chosen as defense mechanism in federated learning setting with non-i.i.d. data.

In contrast, anomaly detection solutions can be deployed on top of any aggregation method, sanitising the set of local models before aggregation. We have seen that for PLD_{global} aggregation, these methods have little impact on the aggregated global model despite discarding a significant ratio of local models. *EE-detection* was the best defense technique, the one causing the smallest difference compared to the baseline PLD_{global} model, which is likely due to its lower number of incorrect anomalous predictions for benign models. We can conclude that an anomaly detection method able to provide a low false positive rate will effectively defend against poisoning attacks and insignificantly degrade the utility of an aggregated global model.

4 Large-scale evaluation of EE-detection

In this section, we evaluate the performance of our solution to detect poisoned local models, EE-detection, in a large-scale setting. We deployed EE-detection to protect the experimental implementation of the distributed learning process of PLD_{global}. In this scenario, PLD_{global}^{t} is aggregated daily from over 40,000 local PLD models. EE-detection is applied before the aggregation step. It reads all the local models received during the previous day t (and supposed to be aggregated into the new PLD_{global}^{t+1}) and it extracts the 18 features defined in Section 2.2.1 from each local model. Each feature vector represents one local model and all the feature vectors are fed to our EE-detection model to identify anomalous local models. Anomalous local models are then discarded and the remaining 'normal' ones are aggregated into PLD_{global}^{t+1}.

4.1 Longitudinal study and stability of EE-detection over time

We evaluate how the performance of EE-detection evolves over time. EE-detection is trained using a static training set of local PLD models collected at a certain time point and it models "how a normal local model is supposed to look like" at that point. However, we know that many anomaly detection problems are non-stationary, meaning that their underlying data distributions change over time, e.g., due to concept drift [QSS+'09]. This is the case with process launch events, evolving due to new software installations, software and firmware updates, changes in users' habits and device use patterns, and so on. We want to study how the EE-detection model adapts to such changes in order to understand how often it needs to be re-trained in order to preserve its performance over time.

We trained the Elliptic Envelope anomaly detection model with a contamination value of 0.001 using the local models received in our experimental setup between June 7 and June 14. This training set is composed of over 250,000 local models. We deployed this model on July 5, using it to discard the detected anomalous local models from PLD_{global} aggregation, monitoring its performance and analysing the results of the poisoned local models detection by our model every day for a period of three months from July 5 to October 5. We focus on two performance metrics for this experiment:

a. The share of the local PLD models identified as anomalous among all the local models received on a specific day. Assuming there is no poisoning attack taking place, this ratio must be low, meaning that most of the local models are retained and aggregated into PLD_{global}.
b. The ratio of the process launch events that these anomalous local models contain to the total count of the process launches in all the local models received on a specific day. This ratio provides an estimate of how large / important the local models that we detect as anomalous and discard are. Assuming there is no poisoning attack taking place, this ratio must also be low, meaning that most of the process launch events contained in the local models are aggregated into PLD_{global}. If this ratio is lower than the ratio of the local models predicted as anomalous, it means that the local models detected by EE-detection as anomalous are smaller than average and they contain fewer process launch events than the average local PLD model.

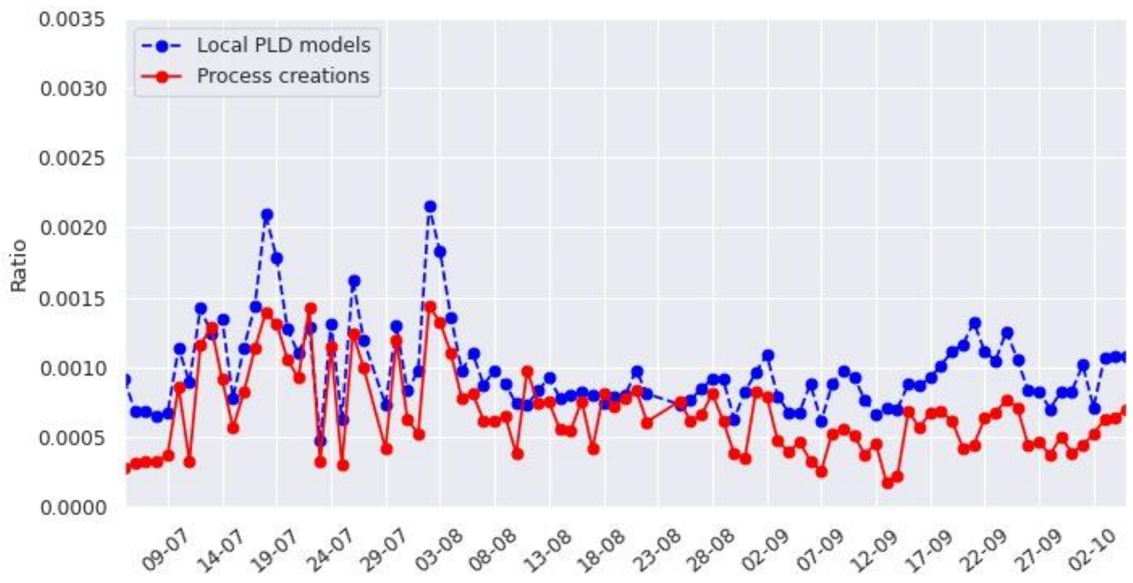

Figure 7: Ratio of local PLD models detected as anomalous by EE-detection and corresponding ratio of process launches discarded from aggregation. Daily results over 3 months of monitoring.

Figure 8 presents the results we obtained in this experiment. We can see that the ratio of the local models identified as anomalous remains low over these three months: between 0.05% and 0.2% of the local models are discarded per day, which means 20 to 80 local models out of over 40,000 received on average daily. A similar observation can be made for the ratio of the discarded process launch events: between 0.02% and 0.15%. We see that this ratio is always slightly lower than the ratio of the anomalous local models. This means that the anomalous local models are smaller than average. So, after the poisoned local model detection process, around 99.9% of the local models and 99.95% of the process launch events are kept for aggregation into PLD_{global}. This means that our EE-detection mechanism has a very small impact on the PLD_{global} model training in the absence of poisoning attacks, which is certainly a desirable property.

Regarding the detection performance over time, we can see that both metrics we study are rather stable and there is no global trend in increasing or decreasing the two ratios. This means that the local model features we chose for EE-detection provide a robust representation of the models. Consequently, our detection model seems to be 'immune' by design to concept drift in process launch events, another desirable property reducing operational costs, confirmed by the consistent performance over the three months of running without re-training (while PLD itself is re-trained online).

Looking closer at Figure 8, we can see that there are higher variations in the monitored metrics between the second week of July and the first week of August compared to the remaining two months of the monitoring period. These fluctuations can probably be explained by the changes brought by the holiday period, when the use of client machines and the work patterns are less predictable, influencing the distribution of the PLD scores of process launch events.

Takeaways: We can see that the carefully engineered features of EE-detection help the anomaly detection model show consistently good performance over time. A key challenge in designing an effective poisoned model detection method is to find features that meet the following two criteria:

- Reflecting differences between benign and poisoned local models
- Being stable over time, avoiding significant sensitivity to concept drift

The first criterion can be met through careful threat modeling and designing of poisoning attacks against the model to protect, as we did in D3.5.1. The second criterion is more difficult to meet, since it often requires large-scale collecting of local models over an extended period of time in order to reliably infer features that are not sensitive to concept drift and covariate shift [QSS+'09]. However, this is of high importance to ensure that our defense method has low maintenance costs and is suitable for real-world deployment. It is also worth noting that low re-training needs increase the resilience to evasion attempts by attackers which can try to poison the defense method itself. The opportunities for poisoning the training set are much smaller when training takes place, e.g., every six months rather than every day.

4.2 Analysis of local PLD models identified as anomalous

We do not expect to have any poisoning attacks in our large-scale experiment. Yet EE-detection identifies around 40 local models as anomalous on average every day. We analysed those false positive cases to better understand why the – presumably benign – local models were detected as anomalous and potentially poisoned. The analysis can shed light on actual anomalous behaviour of client machines, failures of the monitoring sensors in those, or help improve EE-detection to produce fewer false positives and, thereby, aggregate more benign models into PLD_{global}.

We started by analysing how the local models detected as anomalous were distributed over the clients. If multiple anomalous local models came from the same client, this can indicate a consistent abnormal behaviour of the client, making its process launch events significantly different from the other clients. The results of this analysis are depicted in Figure 9. We can see that a major part of the anomalous local models are unique submissions of distinct clients. These are likely due to seldom abnormal behaviours, not connected to attempts to poison PLD_{global}. These behaviours are not significant enough to last over time in their local models, and the subsequent submissions from the same clients are accepted by EE-detection. These can be considered false positives that we would like to avoid and to aggregate these models to PLD_{global}.

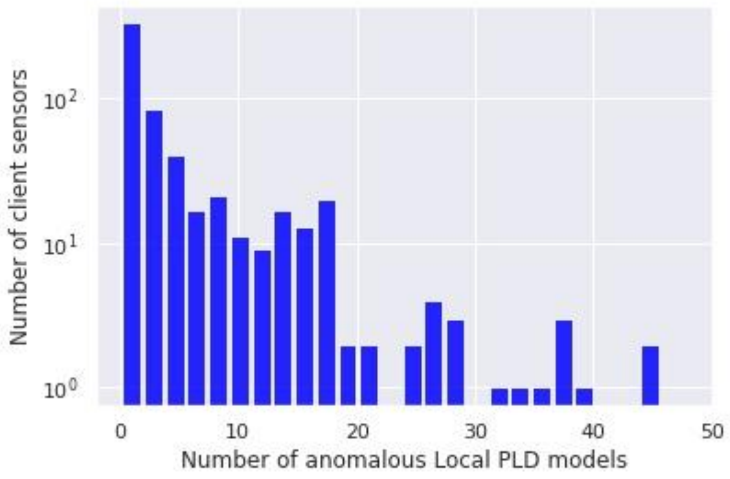

Figure 8: Number of clients vs. number of anomalous local models they sent over 93 days

We also see that some tens of clients submit local PLD models that are consistently (10 to 45 times) detected as anomalous. This shows that the process pairs in those clients have something significantly different from the 'general' case. Although those clients are not compromised, it may be beneficial to discard their local models from aggregation. Including their local models in PLD_{global} might make it more susceptible to evasion attacks in actual operation [BCM+'13] by introducing rare process launch events not observed in other clients we want to protect, which can be used by an attacker as an attack vector.

We analysed the process launch events for 20 of the clients which got their local models regularly detected as anomalous by EE-detection. We summarise our findings about these as follows:

- 5 servers running repeating maintenance tasks
- 4 clients running rare benign software that launches many processes
- 3 clients with newly installed PLD software (no known behaviour yet)
- 3 clients installing new software, making software updates and troubleshooting
- 1 malfunctioning client launching the same process over and over again
- 1 client where the user was doing atypical work (e.g., malware analysis and sandboxing)
- 2 clients that were under penetration testing
- 1 client with nothing abnormal to note

We can see that most of these clients, even though not malicious, had something special to notice in their behaviour. Servers typically behave differently from personal machines, they run different software and processes, and they are used more intensively than end-user machines. The number of processes they launch and the repetitive launches with the same process pairs make their models stand out from the rest of local PLD models, which explains their 'anomalous' rating. The same logic explains why the four clients running rare software launching many processes got detected. Finally, the three clients with the newly installed PLD software and the three making software updates and troubleshooting are identified as anomalous because they launch rarely used installation-related process pairs. All these clients run fully legitimate processes that either (a) are not run by many other clients (for the five servers and the four running rare legitimate software) or (b) are seldom launched (the three 'newly installed' and the three running software updates). One may argue that not capturing these different types of benign seldom behaviour in PLD_{global} can be beneficial, even though it may trigger some occasional false alarms for a few clients, because it makes PLD_{global} more restrictive and better capable of detecting stealthy attacks. One solution to cope with such false alarms can be to implement explicit whitelisting rules or have different PLD_{global} models for clients known to have highly specific behaviour, e.g., servers.

Considering the remaining four clients where we observed unusual behaviour, we can argue that it is beneficial to discard their local PLD models from aggregation into PLD_{global}. One of them was malfunctioning, and it is actually good that PLD_{global} is able to detect such cases in addition to malicious activities and prompt for fixing the problems. The remaining three clients were clearly running malicious processes even though this was in a controlled way and for testing purposes. Processes of this kind must never be included in PLD_{global} because that opens clear attacks vectors for attackers to exploit. PLD_{global} is not aware of user intents and it only tries to detect malicious process launch events. So, the three local models representing malicious behaviour must not be used for training PLD_{global}.

Takeaways: While being designed for detecting poisoned models, EE-detection also rates some legitimate local models as anomalous. On the one hand, we showed that low numbers of discarded "false positive" models would not significantly impact the quality of the global model (Section 3.5). On the other hand, we noted that discarding such local models could actually be beneficial for improving the performance of the global model. In the case of PLD_{global}, EE-detection prevents the model from:

- generalising rare benign behaviours (process launch events);
- unintentionally learning malicious behaviours.

This increases our ability to detect compromised and malfunctioning clients. So, in addition to effectively preventing poisoning attacks, methods identifying anomalous local models can also improve the performance of the global model by 'sanitising' the set of local models, removing 'noise' before the aggregation.

5. Conclusion

Concerns related to the security of machine learning models have emerged only recently in the industry. With ML-powered systems becoming increasingly important in business, public services and other domains, the topic has gathered a great deal of attention in the academia over the past 5 - 10 years. Many attacks (such as model evasion [BCM+'13], model poisoning and model stealing) have been demonstrated in experimental setups and a number of defense approaches have been proposed in the same theoretical setting [BR'18, PMS+'18]. Nevertheless, our knowledge of practical attacks targeting real widely deployed ML-systems remains close to non-existent. In Task 3.5 of the SHERPA project, we tried to partially fill in this gap by systematically studying poisoning attacks against a practically important and popular class of anomaly detection models trained in a distributed online manner, designing defense approaches and evaluating their effectiveness. Our focus was on two defense paradigms that claim certain level of "*general applicability*" and can in principle be used for countering any poisoning attacks against a wide range of model types: *robust aggregation* and *poisoned input detection* methods. Two main criteria were used in the evaluation: (i) effectiveness in mitigating poisoning attacks and (ii) preserving the target global model's utility.

We showed that two characteristics of distributed training data dramatically influence the performance of existing defenses against poisoning attacks and must be taken into account:

- **The training data distribution among clients**. If the training data is far from independent and identically distributed among the clients, robust aggregation methods will likely (a) degrade the performance of the aggregated global model (even in the absence of poisoning attacks) and (b) make poisoning attacks easier for the attacker. In the same non-i.i.d. scenario, poisoned input detection methods based on clustering will incorrectly detect many legitimate local models as compromised, degrading the aggregated model's performance.
- **The sparsity of features used as input to models**. If the features used to train local models (or directly provided for aggregation) are sparse in the sense that typically only few clients observe non-zero values (or presence) of a specific feature, robust aggregation will be ineffective in preventing poisoning attacks, especially distributed ones.

We hope these findings will help practitioners design better defenses against poisoning attacks. We also introduced a four-step systematic approach to mitigating poisoning attacks. One critical step of this approach is to define features capable of distinguishing benign local models from poisoned ones. We demonstrated our approach through designing the *EE-detection* method for identifying poisoned local models in the distributed training of the PLD model. We showed that EE-detection outperforms several robust aggregation methods and a popular poisoned model detection method in terms of attack mitigation and preserving the global model utility.

While proposing a general process for designing mechanisms countering poisoning attacks, we believe that effective defense of a specific ML model requires careful threat enumeration and analysis of the model and its training process and data.

Together with providing an effective method for countering poisoning attacks against the chosen class of models (exemplified by PLD), the SHERPA Task 3.5 work indicates that more studies are required for understanding attacks on real-world ML models. The *representation* and *distribution* of the training data are likely of high importance for mitigating attacks of other types, such as model evasion and stealing. While most of the research on adversarial machine learning has been focusing on Deep Neural Network models, simpler models are very popular in many industrial and public service domains. These simpler models must be analysed for potential threats in order to protect citizens and organisations relying on the model verdicts.

As noted in a recent ENISA event: “Cybersecurity is … of paramount importance for the reliable and trustworthy deployment of AI.” Fully sharing this view, we hope to see increased efforts of the cybersecurity industry in protecting practical AI-powered systems.

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.

[AAL'18] Alistarh, D., Allen-Zhu, Z., & Li, J. (2018). Byzantine stochastic gradient descent. In *Advances in Neural Information Processing Systems* (pp. 4613-4623)

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

[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).

[BCM+'13] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., ... & Roli, F. (2013, September). Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases* (pp. 387-402). Springer, Berlin, Heidelberg.

[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.

[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.

[BGS'17] Blanchard, P., Guerraoui, R., & Stainer, J. (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems* (pp. 119-129).

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

[CCB+'18] Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., ... & Srivastava, B. (2018). Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*.

[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).

[DEG+'19] Damaskinos, G., El Mhamdi, E. M., Guerraoui, R., Guirguis, A. H. A., & Rouault, S. L. A. (2019, April). Aggregathor: Byzantine machine learning via robust gradient aggregation. In *The Conference on Systems and Machine Learning (SysML), 2019* (No. CONF).

[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).

[FCJ+'20] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *USENIX Security*, 2020.

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

[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.

[LDG’18] Liu, K., Dolan-Gavitt, B., & Garg, S. (2018, September). Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses* (pp. 273-294). Springer, Cham.

[MKK’20] Marchal S., Karpuk D., & Kirichenko A. (2020) D3.5 Technical Options and Interventions (Interim report). De Montfort. Online resource: https://doi.org/10.21253/DMU.12973031.v1

[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.

[QSS+’09] Quionero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND. Dataset shift in machine learning. The MIT Press; 2009 Feb 27.

[RNH+’09] Rubinstein, B. I., Nelson, B., Huang, L., Joseph, A. D., Lau, S. H., Rao, S., ... & Tygar, J. D. (2009, November). Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement* (pp. 1-14).

[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).

[SWS+’00] Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., & Platt, J. C. (2000). Support vector method for novelty detection. In *Advances in neural information processing systems* (pp. 582-588).

[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.

[XHC’19] Xie, C., Huang, K., Chen, P. Y., & Li, B. (2019, September). DBA: Distributed Backdoor Attacks against Federated Learning. In *International Conference on Learning Representations*.

[YCR+‘18] Yin, D., Chen, Y., Ramchandran, K., & Bartlett, P. (2018). Byzantine-robust distributed learning: Towards optimal statistical rates. *arXiv preprint arXiv:1803.01498*