Sharing and Automation for
Privacy Preserving Attack Neutralization

(H2020 833418)

D5.4 Global model based on shared local models, final version (M30)

Published by the SAPPAN Consortium

Dissemination Level: Public

H2020-SU-ICT-2018-2020 – Cybersecurity

Document control page

Document file: Deliverable 5.4
Document version: 1.0
Document owner: Alexey Kirichenko (F-Secure)

Work package: WP5
Task: T5.2
Deliverable type: Other
Delivery month: M30
Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document History:

Version	Author(s)	Date	Summary of changes made
0.1	Andrew Patel (FSC) Dmitriy Komashinskiy (FSC)	2021-10-12	Started writing draft
0.2	Arthur Drichel (RWTH) Sebastian Schäfer (RWTH) David Karpuk (FSC) Alexey Kirichenko (FSC)	2021-10-27	Added DGA, ARW, revisions, structural changes
1.0	Alexey Kirichenko (FSC)	2021-10-31	Final round of checks and revisions

Internal review history:

Reviewed by	Date	Summary of comments
Sebastian Schaefer (RWTH)	2021-10-20	Technical review with minor changes
Franziska Becker (USTUTT)	2021-10-29	Grammar and spelling

Authors

Arthur Drichel, RWTH (drichel@itsec.rwth-aachen.de)
David Karpuk, Senior Data Scientist, F-Secure Corporation (david.karpuk@f-secure.com)
Dmitriy Komashinskiy, Lead Researcher, F-Secure Corporation (dmitriy.komashinskiy@f-secure.com)
Sebastian Schäfer, RWTH (schaefer@itsec.rwth-aachen.de)
Andrew Patel, Researcher, F-Secure Corporation (andrew.patel@f-secure.com)
Alexey Kirichenko, Research Collaboration Manager, F-Secure Corporation (alexey.kirichenko@f-secure.com)

1 Executive Summary

This deliverable presents the results of the second phase of SAPPAN T5.2, continuing and extending the results described in SAPPAN D5.3.

As one of SAPPAN's main objectives is to build a sharing platform that can be used for privacy-preserving sharing of attack detection and response-related data, meta-data and models with a view to improve the local detection capabilities for all participating parties and organizations, we discuss in this deliverable several methods of sharing local models based on ensembling, distributed and federated learning.

In particular, we present in this document:

- The evaluation and the privacy analysis of the proposed DGA detection methods in multiple sharing scenarios
- A method, amenable to federated learning techniques, for deriving low-dimensional embeddings of computer processes, which can be used by and can improve the performance of other cybersecurity-relevant models, in particular, the models presented in D5.3 for detection of anomalous process launch and process access events in endpoints
- Another potential use case for the embeddings approach, where the goal is to detect security-related anomalous Windows Registry write events.

2 Introduction

The cybersecurity community has, for many years, studied and applied machine learning techniques to solve problems in the intrusion detection, threat analysis, malware classification, domain categorization, and other areas. While machine learning techniques were originally used primarily to address an ever-increasing volume of attacks, attack techniques, and malicious samples, they are now also needed to deal with an evolving threat landscape, where more sophisticated tactics and adversaries are present. Businesses and organizations that were once able to rely on firewalls and anti-virus software for cyber defence, now require more modern solutions, such as real-time security monitoring and Endpoint Detection and Response (EDR), to stay ahead of adversaries. In order to detect and respond to attacks and breaches, it is no longer enough to be able to identify a single malicious payload or network anomaly – data and metadata from multiple sources and across multiple endpoints and time-slices must be combined and jointly analysed.

EDR and other similar top-down approaches to cyber defence lend well to the process of combining and distributing machine learning models and their training processes. Multiple methods for combining machine learning inference and training mechanisms exist, including ensembling, distributed machine learning, and federated learning approaches. Each of these have their own pros and cons.

While collaborative learning methods all tend to distribute compute load (in training and/or inference) over multiple separate computers, some methods, such as federated learning, also enable better privacy preservation. A detailed discussion on different approaches for combining machine learning models for use in cybersecurity applications can be found in SAPPAN D5.3.

One of SAPPAN's main objectives is to build a sharing platform that can be used for privacy-preserving sharing of attack detection and response-related data, meta-data and models. This shared platform is expected to improve the local detection capabilities for all participating parties and organizations. Examples of such parties include cybersecurity vendors, their partners (especially Managed Security Service Providers), their customers (especially those having internal security teams or experts), law enforcement agencies, and CERTs.

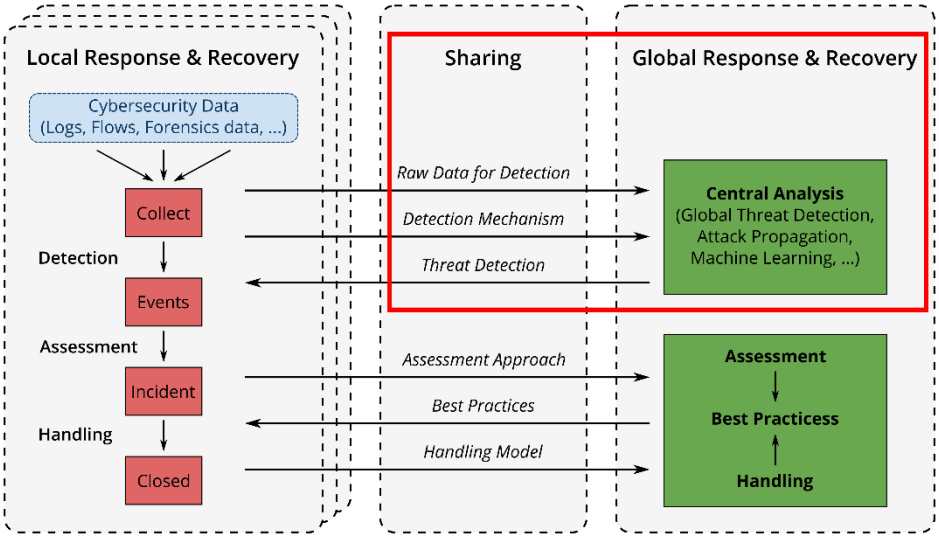

Figure 1: SAPPAN detection and response sharing scheme

Figure 1 illustrates the full sharing, detection, and response scheme covered by SAPPAN. The scheme can be split between top and bottom (detection components and response components) and between left and right (local level versus global level). The scope of WP5 falls in the red-outlined box – that is, the implementation of global-level sharing of data and models, global models for detection, sharing of response and recovery information and visualization tasks. The tasks in WP5 are intended to provide mechanisms to share data and models that were developed in WP3 across multiple organizations, such that global detection models can be built. Ideally these global models will be superior to the local models developed in WP3. Key problems with respect to sharing are, of course, privacy and efficiency, which are tackled by an assortment of approaches investigated in T5.1, T5.2 and T5.3. The approaches range from sharing of anonymized data, to sharing of only pre-trained models, to replacing sharing with other techniques. We apply these techniques to several showcases similar to those described in WP3 in order to build global detection models with adequate recall and precision and providing certain levels of privacy and efficiency, including cost-efficiency.

In Task 5.2 and in this deliverable, our focus is on approaches for building global models based on shared local models (or shared training process, in the federation learning style). In SAPPAN D5.3 (the first version of the deliverable produced by Task 5.2), we presented the initial approaches and experimental results for model sharing in the settings of domain generation algorithms (DGA, used by malware to contact command and control servers) detection, application profiling, and anomalous behaviour detection in endpoints (detection of anomalous process launch and anomalous process access events commonly associated with the execution of malicious software on a system).

In this SAPPAN deliverable, we present the results of the second phase of Task 5.2, structured as follows.

Section 3 continues the DGA detection discussion started in SAPPAN D3.4 (Algorithms for Analysis of Cybersecurity Data) and in Section 4.2 of D5.3. Here, the focus is primarily on the evaluation and the privacy analysis of the proposed DGA detection methods.

Section 4 provides a brief note on application profiling, presented in Section 4.3 of D5.3.

In Section 5, we present a method for deriving low-dimensional embeddings of computer processes. The approach for constructing embeddings, developed originally in the Natural Language Processing (NLP) domain, is amenable to federated learning techniques, so the training of federated models can be carried out in a privacy-preserving manner. Additional benefits of the approach are: (i) efficiency in data storage and transmission; (ii) the produced embeddings can be used by and can improve the performance of other cybersecurity-relevant models, in particular, the models for detection of anomalous process launch and process access operations in endpoints, presented in Section 4.4 of D5.3; and (iii) the produced embeddings can potentially be used for context-specific similarity searches in threat research and analysis systems.

We discuss another potential use case for the embeddings technique in Section 6, where the focus is on detection of anomalous Registry write events.

In Section 7, we briefly conclude this deliverable.

Finally, we want to note that we also experimented with model sharing approaches for the Anomalous Login Activity model, introduced in Section 3.4.2 of SAPPAN D5.1 and evaluated in D5.2. However, since our investigations have not brought significant benefits or improvements, those are just briefly described in D5.2 and are not presenting in this D5.4.

3 Domain Generation Algorithm (DGA) Detection

3.1 Introduction

We use the Domain Generation Algorithm (DGA) detection use case to analyse and compare the benefit of private data sharing within the deliverables D5.2 (global model based on shared anonymized data), D5.4 (global model based on shared local models), and D5.6 (global model without sharing local models). In addition, we investigate the privacy implications of sharing for this use case in all the three deliverables. As a consequence, the three deliverables share the same text for the subsections that include general information, such as the DGA detection background, state-of-the-art classifiers, and parts of the evaluation setup. However, the subsections that depend on specific sharing scenarios, such as the actual evaluation and the privacy study, are presented individually for each deliverable.

We presented DGA detection in D3.4 (Algorithms for Analysis of Cybersecurity Data) in detail. Additionally, we discussed the most important aspects in the initial version of this deliverable (D5.3). However, since the following information is essential to understand the evaluation and the privacy analysis, we shortly repeat the most important parts.

Note that we have published the results of our comprehensive study on collaborative machine learning for DGA detection in the research paper "The More, the Better? A Study on Collaborative Machine Learning for DGA Detection" [1]. Therefore, parts of the following subsections were previously published in and adapted from [1].

3.2 Background

Modern botnets rely on DGAs to establish a connection to their command and control (C2 or C&C) servers. In contrast to using individual fixed IP-addresses or fixed domain names, the communication

activities of DGA-based malware are harder to detect and block as such a malware generates a vast number of algorithmically generated domains (AGDs). The botnet herder is aware of the generation scheme and thus is able to register a small subset of the generated domains in advance. The bots, however, query all generated AGDs, trying to obtain a valid IP-address for their C2 server. As most of the queried domains are not registered, the queries result in non-existent domain (NXD) responses. Only the domains that are registered by the botnet herder in advance resolve successfully to a valid IP-address of the C2 server.

The occurring NXDs within a network that are caused by the non-resolvable queries can be analysed in order to detect DGA activities and thereby to take appropriate countermeasures even before the bots can be commanded to participate in any malicious action. This detection is, however, not trivial, since NXDs can also be the product of typing errors, misconfigured or outdated software, or the intentional misuse of the DNS, e.g., by antivirus software. In the following, we refer to this detection, in which we separate benign from malicious domain names, as the DGA binary classification task.

In addition to this binary classification task, it is useful not to only detect malicious network activities but also to attribute the malicious AGDs to the specific DGAs that generated the domain names. This enables security analysts to get potentially valuable information about the malware used in the attack and to plan response and remediation actions. In the following, we refer to this classification as the DGA multiclass classification task.

In the past, several approaches have been proposed to detect DGA activities within networks. These approaches can be split into two groups: contextless and context-aware approaches. In SAPPAN, we focus on contextless approaches (e.g., [2, 3, 4, 5, 6, 7]), as they fully rely on information that can be extracted from a single domain name to be analysed. Thereby, they are less resource intensive and less privacy invasive than context-aware approaches (e.g., [8, 9, 10, 11, 12, 13]) that depend on the extensive tracking of DNS traffic. Even though the contextless approaches rely solely on domain names, they are able to compete with the context-aware approaches and achieve state-of-the-art performance [2, 3, 5, 6, 7].

A variety of different types of machine learning techniques have been proposed for the classification of domain names. Those techniques can be divided into two groups: feature-based classifiers (e.g., [3, 8]) and deep learning (featureless) classifiers (e.g., [2, 5, 6, 7]). While the deep learning classifiers outperform the feature-based approaches in terms of classification performance [5, 6, 14, 15, 16], their predictions cannot be explained easily. For example, the predictions of a decision tree can easily be traced back to the individual features used to classify a domain name. Such a simple explanation is not possible for the predictions of a deep learning model. However, feature-based approaches rely on specific features that are hand-crafted by utilising domain knowledge. The engineering of these features requires much higher effort compared to the use of deep learning classifiers. Moreover, after the feature engineering, the best combination of the features has to be selected, which is not a trivial task.

While the feature-based and deep learning-based approaches differ in their classification capabilities, they may also provide different privacy guarantees when trained on shared private data. Thus, we evaluate and compare feature-based as well as deep learning-based approaches.

In our evaluation, we include the classifiers which were developed within the SAPPAN project. In detail, we include the two ResNet-based classifiers [17] that we introduced in Deliverable D3.4 (Algorithms for Analysis of Cybersecurity Data). There, we demonstrated that our classifiers achieve better classification scores (f1-score/false positive rate) than the state-of-the-art classifiers described in related work. Note that in order to deal with the explainability problem of deep learning classifiers, we developed in SAPPAN a visual analytics system [18], which tries to bridge the gap between the predictions of deep neural networks and human-understandable features, and reported the results in the Deliverables D3.8 and D3.9. Additionally, in Deliverable D5.2, we reported the feature-based DGA multiclass classifier EXPLAIN [19], developed in SAPPAN, which is explainable by design because its predictions are easier to trace back to features.

3.3 Selected State-of-the-Art Classifiers

In the following, we present several state-of-the-art classifiers which we use in different sharing scenarios in order to: (1) measure the benefit of private data sharing in terms of classification performance and (2) analyse the provided level of privacy of the collaboratively trained classifier.

First, we present the currently best contextless feature-based approach for DGA binary classification. We then continue with different types of deep learning classifiers, including convolutional (CNNs), recurrent (RNNs), and residual neural networks (ResNets).

3.3.1 FANCI

Schüppen et al. [3] proposed a system called Feature-based Automated NXDomain Classification and Intelligence (FANCI). It is capable of separating benign from malicious domain names. FANCI implements an SVM and an RF-based classifier and makes use of 12 structural, 7 linguistic, and 22 statistical features for DGA binary classification. The authors of FANCI state that it uses 21 features, but feature #20 is a vector of 21 values, resulting in 41 values in total. The 41 features are extracted solely from the domain name to be classified. Thus, FANCI is completely contextless. FANCI does not incorporate DGA multiclass classification support.

3.3.2 Endgame

Woodbridge et al. [6] proposed two RNN-based classifiers for the DGA binary and multiclass classification. Both classifiers incorporate an embedding layer, a long short-term memory (LSTM) layer consisting of 128 hidden units with hyperbolic tangent activation, and a final output layer. The last layer of the binary classifier is composed of a single output node with sigmoid activation while the last layer of the multiclass classifier consists of as many nodes as the number of the target DGA families. We denote the binary classifier by B-Endgame and the multiclass classifier by M-Endgame in the following.

3.3.3 NYU

Yu et al. [7] proposed a DGA binary classifier that is based on two stacked one-dimensional convolutional layers with 128 filters. We refer to this model as B-NYU in the following. We additionally adapted the binary model to a multiclass classifier by interchanging the last layer similarly to the M-Endgame model. Additionally, we use Adam [39] as optimization algorithm and the categorical cross-entropy for computing the loss during training. We refer to the multiclass enabled model as M-NYU in the following.

3.3.4 ResNet

In the context of SAPPAN, we developed a binary and a multiclass DGA classifiers based on ResNets [17]. We presented the details as well as a comparative evaluation with the state-of-the-art in Deliverable D3.4 (Algorithms for Analysis of Cybersecurity Data). ResNets make use of so-called skip connections between convolutional layers which build up residual blocks. These blocks allow the gradient to bypass the layers unaltered during the training of a classifier and thereby effectively mitigate the vanishing gradient problem [20, 21]. Our proposed binary classifier, B-ResNet, consists of a single residual block with 128 filters per convolutional layer, while our proposed multiclass classifier M-ResNet has a more complex architecture of eleven residual blocks and 256 filters per layer.

3.4 Class Weighting

Tran et al. [5] showed that the model of Woodbridge et al. [6] is prone to class imbalances, which reduce the overall classification performance of the DGA multiclass classifier. The authors mitigate the effect of class imbalances by using the proposed class weighting:

$$C_i = \left(\frac{total\ number\ of\ samples}{number\ of\ samples\ in\ class\ i} \right)^{\gamma}$$

The class weights control the magnitude of the weight updates during the training of a classifier. The rebalancing parameter γ controls how large part of the dataset should be rebalanced. Setting γ = 0 makes the model behave cost-insensitively, while setting γ = 1 makes the classifier treat every class equally regardless of the actual number of samples per class included in the training set. Tran et al. empirically determined that γ = 0.3 works well for DGA multiclass classification. In all our experiments we thus use γ = 0.3 when working with cost-sensitive models. We denote deep learning models which incorporate class weighting with the suffix ".MI".

3.5 Data Sources

The main goals of our evaluation are: (1) to determine whether we can improve the classification performance by leveraging different approaches for private information sharing, (2) to quantify the level of privacy after enabling privacy-preserving techniques, and (3) to quantify the loss in utility after enabling privacy-preserving techniques.

For the deliverables D5.2, D5.4 and D5.6, we use the same evaluation setup (i.e., the same classifiers and datasets) in order to guarantee comparability of different information sharing scenarios for the use case of DGA detection.

In total, we use five different data sources, four for obtaining benign data and one for malicious data.

3.5.1 Malicious Data

We obtain malicious domains from the open-source intelligence feed of DGArchive [22] which contains more than 126 million unique domains generated by 95 different known DGAs. We make use of all the available data up to 2020-09-01.

3.5.2 Benign Data

We obtain benign labelled NXDs from four different sources. Three of the sources are the networks of project partners, namely, CESNET, Masaryk University and RWTH Aachen University. As real-world benign training data from diverse sources is very difficult to obtain but crucial for a comprehensive study on collaborative machine learning, we tried to get additional data from other parties. Fortunately, Siemens AG, a partner in another research project, provided us with additional data for our analysis. Due to this rich data, we are able to conduct collaborative machine learning experiments that are similar to a real-world setting. Moreover, the different benign data sources enable us to investigate whether collaboratively trained classifiers generalize well to different networks.

For data obtained from each of these sources, we perform a simple pre-processing step in which we remove all duplicates, cast every domain name to lowercase (as the DNS operates case-insensitive), and filter against our malicious data obtained from DGArchive to clean the data as much as possible. Additionally, we remove the intersection of all the obtained samples from two of our benign data sources, namely from CESNET and Masaryk University. The reason for this is that the networks of both parties are interconnected and the recording periods for data collection overlap. Note that thereby we also remove samples from both data sources which would naturally be present in both networks even if they were not interconnected. Such samples could be, e.g., commonly mistyped popular websites. This issue could have an effect on the classification performance of classifiers when samples of these networks are used for training or classification. However, since we record NXDs, we filter significantly fewer samples than if we were to record resolving DNS traffic. Thus, this effect likely has only a negligible influence on the classification performance, but this has yet to be investigated. In the following, we list the recording periods and the number of unique samples obtained from each source for benign data.

RWTH Aachen University (RWTH): We obtained a one-month recording of September 2019 from the central DNS resolver of RWTH Aachen University which is located in Germany. This recording comprises approximately 26 million unique benign NXDs that originate from academic and administrative networks, student residences' networks, and networks of the university hospital of RWTH Aachen.

Masaryk University (MU): We obtained a one-month recording from mid-May 2020 until mid-June 2020 from the networks of Masaryk University which is located in the Czech Republic. This recording contains approximately 8 million unique benign samples.

CESNET: We received additional benign samples from CESNET, an association of universities of the Czech Republic and the Czech Academy of Sciences consisting of 27 members in total. CESNET operates and develops the national e-infrastructure for science, research, and education. From this data source, we obtained a subset of occurred NXDs from the day recording of 2020-06-15. In total, we obtained approximately 362,000 unique samples.

Siemens: We obtained a one-month recording of July 2019 that comprises approximately 21 million unique NXDs from several DNS resolvers of Siemens AG which is a large company that operates in Asia, Europe, and in the USA.

3.6 Sharing Approaches

In each deliverable (D5.2, D5.4, D5.6), we investigate different sharing scenarios for the use case of DGA detection. When preparing the initial version of this deliverable (D5.3), we planned to evaluate only one sharing scenario for distributed learning of a global model based on shared local models: Ensemble Classification. However, in our work on Task T5.1 (Distributed Learning of a global model based on shared anonymized data), an analysis on the envisioned sharing scenarios was conducted, which suggested to relocate First-n-layer sharing (Feature Extractor Sharing) to this deliverable. The reasons for this decision are reported in Deliverable D5.2. Thus, in this deliverable, we examine two sharing approaches:

3.6.1 Sharing Scenario – Ensemble Classification

In this scenario, we build a global model based on shared local models. Each collaborating party trains a local DGA detection model using their own private data. These models are then subsequently shared in such a way that every party is able to create the global model. When a party wants to evaluate new domain names, they feed them into all the local models. The final classification results can then be obtained by two approaches:

Averaging the Local Models' Confidence Scores (Soft Labels)

The final classification result is the average of the local models' confidence scores. For instance, an average score above a certain threshold (e.g., 0.5) would indicate that the input domain is classified as malicious.

Majority Voting (Hard Labels)

The final classification result is the majority vote of the local models. A tie-breaker is needed for an even number of local models. For instance, the tie-breaker could be whether the average of the local models' confidence scores is above a certain threshold.

We also planned to quantify the loss in utility after enabling different types of privacy-enhancing technologies. In this scenario, the overall privacy could be enhanced by using differentially private stochastic gradient descent to train the local models. However, this might have a negative impact on the classification performance.

3.6.2 Feature Extractor Sharing (Sharing of First-n-layers of Deep Learning Classifiers)

We designed an approach for sharing feature extractors as parts of trained deep learning-based classifiers. We assume that the first-n-layers of a deep neural network are used as a sort of feature extractor while the remaining layers are used for the actual classification.

In Figure 2 we display such a layer partition.

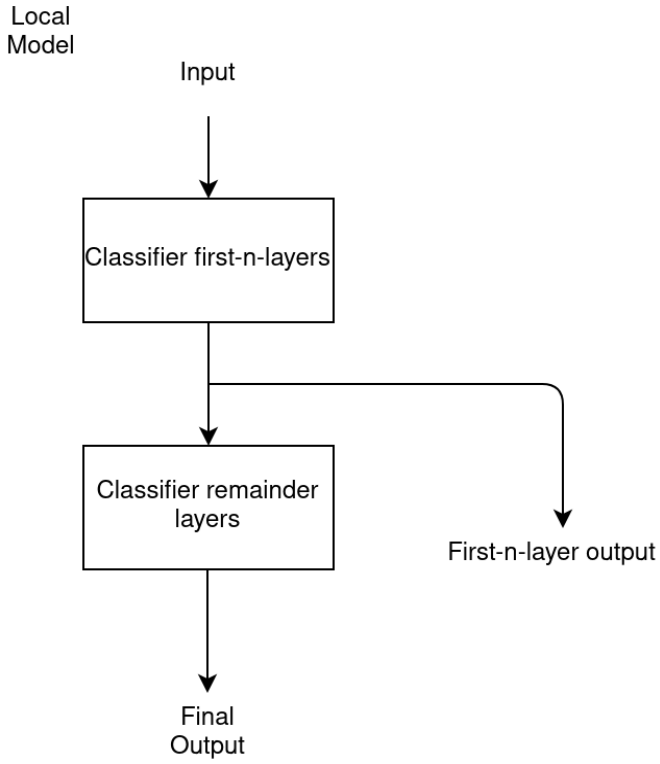

Figure 2: Partition of a neural network classifier into feature-extractor and classification layers.

Every party participating in the collaborative training of a classifier in this scenario trains a local DGA detection model using their own private data. Thereafter, each party can share their feature extractor (i.e., the classifier's first-n-layers). A possible approach for combining the feature extractors of different neural networks belonging to different parties can be seen in Figure 3.

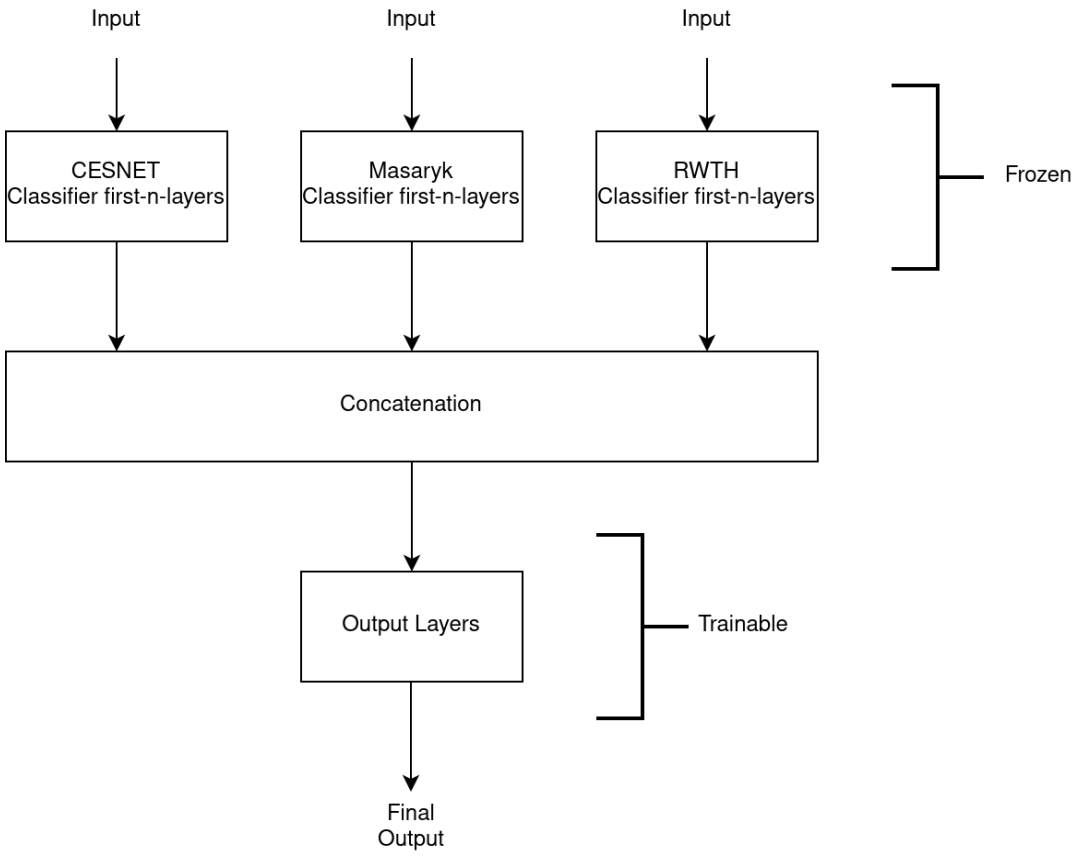

Figure 3: Approach for combining feature extractors of different neural networks and parties.

Here, each participant combines their own and received feature extractors to a new model. To this end, the feature extractors are applied in parallel and their outputs are concatenated and flattened. Additionally, a new dense classification layer is appended to the new model. This classification layer is not trained yet, thus the organizations freeze the weights of the feature extractors and use their local training data to train the classification layer separately.

3.7 Comprehensive Collaborative Machine Learning Study

In the following, we present a comprehensive study on collaborative machine learning approaches for deriving a global model based on shared local models. First, we provide an overview of our evaluation setup, including our dataset generation scheme and the evaluation methodology used. Subsequently, we present different sharing scenarios which are derived from research questions on possible real-world application environments for trained classifiers. These sharing scenarios are used to assess the performance of the different sharing approaches. Finally, we present the results of the study, including a direct comparison to the sharing approaches developed in Deliverable D5.6.

3.7.1 Dataset Generation

We first describe the process of generating suitable datasets for our experiments, using the above data sources. We provide an illustration of this process in Figure 4 for convenience.

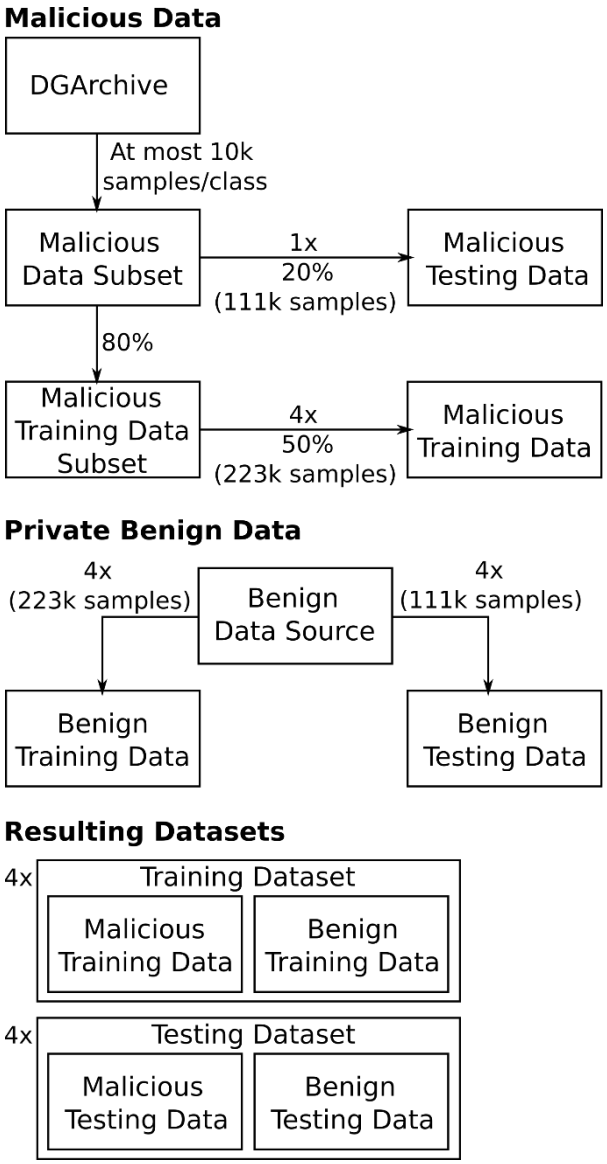

Figure 4: Datasets generation scheme

In order to create diverse datasets and to cope with the large number of available training samples, we first subsample our malicious labelled data into a smaller set that includes at most 10,000 samples per DGA family. We include all the samples for the DGA families for which less than 10,000 samples are available. Thereby, we also include samples from the underrepresented DGA families. We do this because in [23] we showed that by including a few samples to the training set of a classifier, its detection performance for underrepresented DGAs can be increased significantly without reducing its detection rates for well-represented DGAs.

From the selected subset, we then split 20% (approximately 111,000 samples) stratified across all the included DGA families for the test sets. For each of our benign data sources, we select individual malicious training data by subsampling 50% (approximately 223,000 samples) from the remaining malicious labelled samples. By subsampling from a larger common pool of malicious samples, we can create four training sets that contain both duplicate and unique malicious samples. We do this because in a real-world scenario, it is very likely that the collaborating parties use overlapping sets of malicious samples. Note that in contrast to the benign labelled samples, the malicious samples are available in public repositories and are not privacy-sensitive.

The benign samples are not shared between different training sets as they are considered the main privacy-critical asset of the collaborative DGA detection use case. We carry out a similar selection process for the benign training and testing samples. From each of the four benign data sources, we randomly subsample the same number of benign training and testing samples as we did for malicious training and testing samples, respectively.

We use these data selections to create four training and testing dataset pairs, one for each benign data source. To this end, we combine the respective malicious and benign data selections to balanced training and testing datasets. Note that during this dataset generation, we ensured that the training and testing datasets for each party are completely disjoint. Each of the four training and testing datasets include approximately 446,000 and 223,000 samples, respectively.

Additionally, we create two balanced training datasets that include publicly available benign data using the same generation process. These datasets are used to train initial global models for our federated learning experiments in Deliverable D5.6. The public benign data originates from the Tranco list [24] which contains a ranking of the most popular domains that has been hardened against manipulation. Using this data, we create two datasets: one contains the top entries on the list; the benign data of the other dataset is a set of random samples.

3.7.2 Methodology and Sharing Scenarios

Using these datasets, we are able to precisely measure the influence of collaborative machine learning on the classification performance of various classifiers. To obtain reliable results, we repeat the whole dataset generation process five times and thereby create 20 individual training and testing dataset pairs which include malicious labelled samples from DGArchive and benign data from the four benign data sources. In the same way, we also generate ten training datasets used in the federated learning experiments for training an initial global model using publicly available data in Deliverable D5.6. In the following, we repeat every experiment five times and present the averages of the individual experiment results. Note that the datasets are generated similarly to a five-fold cross validation, i.e., the testing datasets are completely disjoint with both the training datasets and the testing datasets within the repetitions.

In the following, we exclude the feature-based approach FANCI from our study and concentrate on the three deep learning-based approaches (B-Endgame, B-NYU, and B-ResNet), as neither Feature Extractor Sharing nor Federated Learning (which is examined in Deliverable D5.6) is possible using a feature-based approach. However, feature-based approaches for collaborative machine learning are analysed in detail in Deliverable D5.2. In this work, we train all the deep learning classifiers using early stopping with a patience of three epochs to avoid overfitting and assess their performance during training on holdout sets that consist of random 5% splits of the used training data.

Additionally, in our comprehensive collaborative machine learning study, we focus on binary DGA detection. The reason for this is that in D3.6 (Cybersecurity Data Abstraction), the set of benign training samples has been identified as the main privacy-critical asset of this use case. The malicious data used in this use case is mostly publicly available and thus has no privacy constraints. The difference between the training samples used in the binary and multiclass classification tasks is that the malicious samples of the multiclass task are additionally labelled with the DGA that generated a specific domain. However, as this information is not privacy-sensitive, we focus on the binary classification task. For simplicity, we always refer to the binary versions of the three investigated deep learning classifiers as Endgame, NYU, and ResNet in the following Sections.

To measure the impact of collaborative machine learning on classification performance, we evaluate all the possible combinations of participants for each examined approach. Overall, our comprehensive study consists of 13,440 evaluation passes; 4,200 evaluations (including the baseline evaluations) are performed for this deliverable. The total number of individual experiments differs across the studied collaborative machine learning approaches. In the following, we provide an overview of the numbers of experiments done per investigated approach.

3.7.3 Baseline

All the sharing approaches are compared against the baseline to evaluate their performance. The baseline evaluations are similar to traditional training and testing of a classifier using training data from a single organization. Each organization trains their own model using their own private benign data and malicious training samples from DGArchive. No training data is shared among any organizations and also no global model is derived.

We train one classifier for each of the four organizations and evaluate them on every available test dataset. To this end, we perform five repetitions of training and testing for four organizations (RWTH, MU, CESNET, Siemens) and three classifier models (Endgame, NYU, ResNet), thus performing $5 \cdot 4 \cdot 4 \cdot 3 = 240$ baseline evaluations in total.

3.7.4 Ensemble Classification

In Ensemble classification, a global classifier is built using the classifiers trained by each organization. Similar to the baseline scenario, each organization first trains a classifier using their own private benign data. These classifiers are then shared with all the participants. Each party now combines the individual classifiers to an ensemble classifier. The combination of the classifiers can be done (1) by using a majority voting system on the binary labels (**Hard Labels**) or (2) by averaging the results of the individual classifiers to a single confidence score (**Soft Labels**). A tie is possible when using the majority voting system with an even number of participants. In such a case, we resort to the soft labels approach, where we average all the predictions and predict a domain name as malicious if the average is greater than 0.5 and as benign otherwise.

In total, there are eleven possible combinations of organizations for building a combined model using four different parties. Here, we also evaluate classifiers on all the four different testing datasets, regardless of the combination of the local classifiers used. Thereby, we are able to measure the generalisation capability of classifiers on benign data from unknown networks. In total, we perform five repetitions using eleven possible organization combinations, two ensemble approaches (Hard and Soft Labels), four test datasets, and three classifier models ($5 \cdot 11 \cdot 2 \cdot 4 \cdot 3 = 1320$ evaluations).

3.7.5 Feature Extractor Sharing

This sharing approach is related to Transfer Learning. All the deep learning classifiers under consideration use a fully connected (dense) layer to output the final classification score. This layer can be viewed as a sort of classification layer that performs a logistic regression for binary classification. The output of this layer is a confidence score that indicates whether an input domain is benign (score < 0.5) or malicious (score ≥ 0.5). All the layers before this classification layer can be treated as a feature extractor, which produces features used for classification by the final output layer. Instead of sharing the complete classifier as in the naive Ensemble approach, we hope here to reduce the model's privacy leakage by sharing fewer layers. Thus, in this approach, each organization trains a model based on their own private training data. Subsequently, the trained feature extractors are shared among all the participants. Each organization now combines the received feature extractors with their own to create a new model. To this end, the feature extractors are applied in parallel and their outputs are concatenated and flattened. Additionally, a new dense classification layer is appended to the new model. This classification layer is not trained yet, thus the organizations freeze the weights of the feature extractors and use their local training data to train the classification layer separately. In the end, each organization obtains a model which incorporates information about the samples from the other organizations through the shared feature extractors. The resulting models are not identical, since the training of the classification layer is performed on private training samples.

For this approach, each organization first trains a classifier using its own private benign data and derives an individual feature extractor. Then we combine the four feature extractors into eleven possible classifiers. In contrast to ensemble classification, here we require additional training to fit the final

classification layer that combines the results of the shared feature extractors. Hence, in total we perform five repetitions using eleven possible organization combinations with four training datasets, four test datasets, and three classifier models ($5 \cdot 11 \cdot 4 \cdot 4 \cdot 3 = 2640$ evaluations).

3.7.6 Sharing Scenarios

The sharing approaches are evaluated in different scenarios which are derived from research questions on possible real-world application environments for trained classifiers.

Best Case: In this scenario, multiple network operators jointly train a classifier and are mostly interested in a good performance in their own networks. This is related to the following research question: is collaborative training beneficial for organizations that mostly classify data from their own distribution? In this best-case scenario, we provide averaged results for the classifiers that are evaluated only on the test datasets containing the samples coming from the organizations involved in the training.

Average Case: The average of all the evaluations is used as a general performance indicator of the trained classifiers for each collaborative machine learning approach. We use this scenario for a comparative evaluation of the different sharing approaches.

Worst Case: The worst-case scenario contrasts the best-case scenario. Here, the classifiers are evaluated on all the test datasets that contain samples from the organizations that did not participate in the classifier training. Using this scenario, we examine the generalization capability of collaboratively trained classifiers (i.e., whether the classifiers improve in their detection performance for samples originating from different networks).

3.7.7 Evaluation Results

In this subsection, we present the results of our comprehensive study. First, we highlight differences between the datasets of the four organizations and provide an overview of the performance in the three sharing scenarios. Subsequently, we present the results of our comparative evaluation. Finally, we analyse the effect of the number of participants in collaborative machine learning.

Network Differences & Sharing Scenarios

To better explain the actual evaluation steps and to detail the computations done for the different sharing scenarios, we present the average scores for the five repetitions of the baseline experiment using the Endgame classifier in Table 1. We provide the results for the Endgame classifier as an example. The results for the NYU and ResNet models are similar.

Train Network	Test Network	ACC	TPR	FPR
RWTH Aachen University	**RWTH Aachen University**	0.99851	0.99968	0.00267
	Masaryk University	0.99304	0.99968	0.01359
	CESNET	0.96892	0.99968	0.06184
	Siemens	0.99834	0.99968	0.00300
Masaryk University	RWTH Aachen University	0.99717	0.99966	0.00532
	Masaryk University	0.99808	0.99966	0.00350
	CESNET	0.97180	0.99966	0.05606
	Siemens	0.99853	0.99966	0.00259
CESNET	RWTH Aachen University	0.99674	0.99739	0.00390
	Masaryk University	0.99444	0.99739	0.00852
	CESNET	0.99645	0.99739	0.00450
	Siemens	0.99771	0.99739	0.00196
Siemens	RWTH Aachen University	0.98923	0.99968	0.02122
	Masaryk University	0.99037	0.99968	0.01894
	CESNET	0.96833	0.99968	0.06302
	Siemens	0.99888	0.99968	0.00192
Best Case		0.99798	0.99910	0.00315
Average Case		0.99103	0.99910	0.01703
Worst Case		0.98872	0.99910	0.02166

Table 1: Averaged baseline results for Endgame classifier

Here we list the average scores for the five repetitions of Endgame classifiers per training dataset used and per test dataset separately. The true positive rates (TPRs) per training network are equal for all the test networks as we use the same malicious samples within all the four test datasets within a repetition. The best false positive rates (FPRs) on the individual test networks are always achieved by the classifiers that were trained using benign samples originating from the same network as the testing samples. This is expected since those classifiers are specifically trained to extract and classify characteristics of the benign domain names from the respective network. For example, benign samples from different networks may miss certain features or exhibit different characteristics. The average of the table entries where the training network is the same as the test network represents the best-case scenario and is presented at the bottom of the table. The classifiers trained using benign samples from distinct networks achieve different results for the individual test datasets. Samples from CESNET are most commonly classified incorrectly. In some cases, the FPR for these evaluations is even greater than 6%. We reckon that this is due to the fact that the samples from this network are the most diverse, as they originate from over 27 different organizations. Moreover, we filtered out the intersection of the samples from the Masaryk University network with the samples from CESNET as both networks are interconnected. Thereby, we likely removed easily recognizable samples that are naturally occurring in both networks. This could be the reason for the larger FPRs for Masaryk University and CESNET compared to those for the other two networks. Similarly to the best case, we provide the results for the average and worst case in the lower part of Table 1. While the average case is calculated using the average of all the table entries, the worst case only contains the results of the entries for which the training network differs from the test network. The results for the different sharing scenarios are not of interest for the baseline evaluation. As could be expected, the best case results are better than the average case results, which are better than the worst case results.

3.7.8 Sharing Approaches

In this section, we compare the different approaches for collaborative machine learning. First, for comparison and to show that training on publicly available data is not enough for DGA detection on private data, we display the averaged results achieved by the two different types of pre-trained models that

we use within our Federated Learning experiments in Deliverable D5.6 for all the three classifier types over all the test datasets in Table 2.

Classifier	Benign Data	ACC	TPR	FPR
Endgame	Tranco Top	0.68329	0.95492	0.58834
	Tranco Random	0.67730	0.95054	0.59594
NYU	Tranco Top	0.68612	0.94876	0.57652
	Tranco Random	0.71336	0.94310	0.51637
ResNet	Tranco Top	0.78667	0.94952	0.37617
	Tranco Random	0.79899	0.93619	0.33821

Table 2: Results of pre-trained models using public data

All the six trained classifiers yield high FPRs between 33% and 59%, indicating that training on publicly available data alone is insufficient for classifying privacy-sensitive domain names.

In Table 3, we present the results for the average case, for all the classifiers, and all the collaborative machine learning approaches examined in this deliverable.

Approach	Endgame			NYU			ResNet		
	ACC	TPR	FPR	ACC	TPR	FPR	ACC	TPR	FPR
Baseline	0.99103	0.99910	0.01703	0.99151	0.99903	0.01600	0.99102	0.99844	0.01640
Ensemble: Soft Labels	0.98812	0.99975	0.02352	0.98870	0.99976	0.02236	0.98834	0.99946	0.02279
Ensemble: Hard Labels	0.98842	0.99981	0.02296	0.98901	0.99976	0.02174	0.98926	0.99909	0.02057
Feature Extractor Sharing	0.99394	0.99921	0.01133	0.99411	0.99913	0.01090	0.99307	0.99880	0.01266

Table 3: Results of the average case including all classifier types for the collaborative machine learning approaches of D5.4

In order to assess whether the collaborative training is beneficial, we additionally provide the baseline results at the top of the table. For convenience, we colour table entries red if the scores are worse than the ones of the baseline and green otherwise. All the approaches achieve better TPRs than the baseline. This is an expected outcome because the training datasets used by the individual organizations contain additional malicious labelled samples from which a collaboratively trained classifier can learn. Thus, due to collaboration, intelligence about additional malicious labelled training samples is combined in the jointly trained classifiers.

The only approach that leads to better classification results regarding the accuracy (ACC) and FPR is Feature Extractor Sharing. Ensemble classification leads to worse results than the baseline. Furthermore, it makes little difference whether soft or hard labels are used.

While the absolute improvement achieved by collaborative machine learning may seem rather small, the relative reduction in the FPR is significant and could be decisive for the real-world application of classifiers. Compared to the baseline classifiers, Feature Extractor Sharing achieves on average a FPR reduction of 33.5%, 31.9%, and 22.8% for Endgame, NYU and ResNet, respectively. In summary, additional malicious samples in collaborative machine learning improve the TPRs for all the sharing approaches. In the average-case scenario, only the Feature Extractor Sharing approach is advantageous for the use case of DGA detection.

3.7.9 Collaboration Analysis

In this subsection, our goal is to determine whether an increasing number of participants has a positive or negative effect on the classification performance of jointly trained classifiers. To this end, we investigate two scenarios.

In the first scenario, we make use of the best-case scenario. Here, organizations want to use jointly trained classifiers to classify samples from their own networks most of the time. Thus, our goal is to

determine whether the classification performance on those samples improves or deteriorates with an increasing number of participants. Thereby, organizations can decide whether or not it makes sense to use a collaboratively trained classifier for their own network.

In the second scenario, the worst case, we want to determine whether an increasing collaboration improves the generalization capabilities of the classifiers and thus the classification performance on samples from external networks.

We use the FPR as a proxy to determine the performance of the classifiers. In Table 4, we present the achieved FPR scores for the different collaborative machine learning approaches and classifier types separated by the number of participants.

Approach	Parties	Best Case			Worst Case		
		Endgame	NYU	ResNet	Endgame	NYU	ResNet
Baseline	-	0.00315	0.00328	0.00326	0.02166	0.02024	0.02078
Ensemble: Soft Labels	2	0.02149	0.02023	0.02081	0.02420	0.02306	0.02376
	3	0.02393	0.02266	0.02316	0.02493	0.02409	0.02387
	4	0.02489	0.02402	0.02366	-	-	-
Ensemble: Hard Labels	2	0.02068	0.01978	0.01887	0.02399	0.02242	0.02127
	3	0.02332	0.02207	0.02077	0.02462	0.02322	0.02186
	4	0.02404	0.02316	0.02167	-	-	-
Feature Extractor Sharing	2	0.00306	0.00303	0.00314	0.01871	0.01756	0.01804
	3	0.00295	0.00302	0.00307	0.01796	0.01722	0.01784
	4	0.00293	0.00298	0.00302	-	-	-

Table 4: FPRs of the best and worst case for all classifier types and collaborative machine learning approaches of D5.4 separated by number of participants

For visibility, we omit the ACC and the TPR metrics, but most of the time a better or worse FPR correlates with a better or worse ACC. In this evaluation, we are not primarily interested in comparing the achieved scores of the different approaches with the performance of the baseline presented at the top of the table. Rather, we are interested in whether an increasing cooperation improves or deteriorates the achieved performance. Thus, our colour code for the entries differs from Table 3. Here, we mark table entries green if they always improve with an increasing number of participants. When the approaches produce 'continuously' worse results, we colour them red. We do not colour any entries for the approaches for which the increases or decreases in classification performance are not consistent. In the following, we present the evaluation results for the best and worst cases in detail.

Best Case

Most of the collaborative approaches achieve (1) worse results compared to the baseline and (2) deteriorate in classification performance with increasing number of participants. This behaviour can be explained by the fact that the baseline's best-case scenario is the ideal training and classification setting. There, the classifiers are assessed on data that comes from the same distribution as the samples used for training. No information of samples from the other organizations are incorporated in those classifiers. Thereby, the baseline classifiers are specialized in classifying samples that originate from the same network as the training samples used. Thus, it is not surprising that the baseline classifiers achieve almost the best results compared to the other approaches. The collaborative machine learning approaches, on the other hand, also incorporate information of samples from the other networks. Thereby, they are less specialized in classifying samples from a single network but are more generalised and, thus, achieve worse results compared to the baseline. The fact that these approaches achieve worse results with an increasing number of the participants can be explained similarly. The more participants, the less the classifiers are specialized on samples of a single network. For example, Ensemble classification uses classifiers that are similar to the baseline classifiers. However, the more classifiers there are in an ensemble, the less influence a single classifier has on the final classification result. Therefore, in the best-case scenario, the classification performance drops.

The only approach that improves with an increasing number of participants is Feature Extractor Sharing. Moreover, for all the three classifier types, Feature Extractor Sharing achieves better FPRs than the baseline. This is because this approach creates models that are similar to the baseline but also incorporates additional information about samples from the other networks via feature extractors that are applied in parallel. Since the shared feature extractors are not retrained, information about samples from individual networks is very well preserved with this approach. In addition, the intelligence incorporated in the shared feature extractors is harnessed by this approach and leads to improvements even beyond the baseline. Note that although the differences in FPRs are rather small, our results are arguably significant because of the large number of evaluations done and the fact that this behaviour is observable for all the three types of classifiers.

From these results, it can be seen that only Feature Extractor Sharing is beneficial in the best-case scenario, where organizations want to use collaborative machine learning classifiers to classify samples from their own network most of the time.

Worst Case

In this scenario, we evaluate whether an increasing number of participants improves the detection performance of jointly trained classifiers for samples that originate from external networks. Since we only have four different sources of benign data, the maximum number of participants in this scenario is three.

The results obtained for the Ensemble classification deteriorate as the number of participants increases for all the three classifiers. For Feature Extractor Sharing, the achieved FPRs improve with an increased number of participants for all the three classifier types. Consequently, in the worst-case scenario, only the Feature Extractor Sharing improves in performance with an increasing number of participants and achieves better scores than the baseline.

3.8 Direct Comparison with Approaches Developed in D5.6 (Global Models without Sharing Local Models)

In this subsection, we compare the sharing approaches developed in this deliverable with the collaborative machine learning approaches investigated in Deliverable D5.6 (Global Models without Sharing Local Models). In Deliverable D5.2 (Global Model based on Shared Anonymized Data), we also proposed different sharing approaches. However, the focus of that deliverable was mainly on the comprehensive privacy study and the development of a context-less and feature-based approach to DGA multiclass classification, which can be used as an anonymizer for domain names in a collaborative machine learning scenario. Hence, we only compare the approaches of this deliverable with the approaches investigated in D5.6. Future work could compare the approaches for intelligence sharing of D5.2 with the collaborative machine learning approaches of D5.4 and D5.6. However, the approaches of D5.2 are less advanced compared to the approaches of D5.4 and D5.6.

In addition to the 4,200 evaluations done for the evaluation of the different sharing approaches of this deliverable, we now present the results of the additional 9,240 evaluations done for the Deliverable D5.6. Our comprehensive study thus comprises a total of 13,440 evaluations.

3.8.1 Collaborative Machine Learning Approaches Developed in D5.6

In the following, we shortly repeat the sharing approaches which are taken from Deliverable D5.6, to which we compare the performance of the collaborative machine learning approaches investigated in this deliverable.

Federated Learning

Federated learning (FL) [25] is a technique to train a classifier collaboratively without sharing local data. First, a global model is initialized and shared among all the participants in the collaborative training. This global model can be either randomly initialized using standard initialization methods or pre-trained using non-sensitive public data. In this work, we evaluate FL using three different initial global

models, one that is randomly initialized (**Random Model**), and two pre-trained models. For pre-training a global model, we make use of malicious samples from DGArchive and benign domain names from the Tranco list [24]. This list contains a ranking of the most popular domain names, which is also protected against manipulation. One pre-trained model uses the top entries of the Tranco list for benign domain names, while the other model uses random samples. In the following, we refer to these models as **Tranco Top** and **Tranco Random**, respectively. After the global model is shared among all the participants, an iterative training procedure is performed. The global model is trained locally by each organization, using their own private training data for each federation step. The weight updates to the global model in each federation step are then shared with all the other participants, so that everyone can now average the weight updates of the current step and add them to the global classifier's weights. Thereby, each party obtains the same global model which is then used within the next federation step. This iterative training process continues until the global model converges. The only data shared between the organizations are the model weight updates after each federation step and the initial global model. In this work, we investigate two federation approaches. In the first approach, we federate after each local model epoch (**Federation after Model Epoch**), while in the second approach we only federate once after all the local models have converged (**Federation after Model Convergence**).

Here, we perform five repetitions using eleven organization combinations (when using four benign data sources, there are only eleven possible combinations of organizations for building a combined model), three initial global models (Tranco Top, Tranco Random, Random Model), two possibilities for federation (after Model Epoch, after Model Convergence), four test datasets, and three classifier models ($5 \cdot 11 \cdot 3 \cdot 2 \cdot 4 \cdot 3 = 3{,}960$ evaluation passes).

Teacher-Student

The last examined sharing approach is based on a Teacher-Student (T/S) setup. Here, an organization queries trained classifiers of other organizations (teachers) in order to obtain labels for their own data. This labelled data is then used by the querying organization to train their own classifier (student). Using this approach, the teacher classifiers are not exposed to the organization that is training the student classifier. Thereby, white-box attacks against the privacy of an organization that provides the labelling service are not applicable. Usually more than one teacher is involved in the labelling process of a training sample, thus the individual labels or scores need to be combined. Similar to Ensemble classification, we examine two approaches: (1) majority voting on binary labels (Hard Labels) and (2) soft labelling by averaging confidence scores. When using the majority voting approach, a tie is resolved in the same way as in Ensemble classification. For this approach, no global shared model is trained, instead every party again derives an individual model, similarly to Feature Extractor Sharing.

We train classifiers that are similar to the baseline classifiers as teacher models for this approach. Similarly to Feature Extractor Sharing, here we need a training dataset that is labelled by the teachers and used for training a student classifier. Thus, in total we perform five repetitions using eleven possible organization combinations with four training datasets, two teacher result combination approaches (Hard and Soft labelling), four test datasets, and three classifier models ($5 \cdot 11 \cdot 4 \cdot 2 \cdot 4 \cdot 3 = 5{,}280$ evaluations).

3.8.2 Comparative Evaluation

In the following, we compare the sharing approaches developed in this deliverable with the collaborative machine learning approaches investigated in Deliverable D5.6. For convenience, we display the results of both deliverables in summarized tables.

Average Case

In Table 5, we display the results of the average case, including all the classifier types and all the collaborative machine learning approaches.

Approach	Endgame			NYU			ResNet		
	ACC	TPR	FPR	ACC	TPR	FPR	ACC	TPR	FPR
Baseline	0.99103	0.99910	0.01703	0.99151	0.99903	0.01600	0.99102	0.99844	0.01640
Ensemble: Soft Labels	0.98812	0.99975	0.02352	0.98870	0.99976	0.02236	0.98834	0.99946	0.02279
Ensemble: Hard Labels	0.98842	0.99981	0.02296	0.98901	0.99976	0.02174	0.98926	0.99909	0.02057
FL: Random Model - Model Convergence	0.76755	0.55131	0.01621	0.98300	0.98764	0.02163	0.98671	0.99278	0.01937
FL: Random Model - Model Epoch	0.99396	0.99968	0.01177	0.99392	0.99983	0.01199	0.99511	0.99935	0.00913
FL: Tranco Top - Model Convergence	0.99336	0.99958	0.01286	0.99325	0.99975	0.01326	0.99291	0.99974	0.01393
FL: Tranco Top - Model Epoch	0.99553	0.99956	0.00850	0.99400	0.99981	0.01181	0.99491	0.99922	0.00940
FL: Tranco Random - Model Convergence	0.99365	0.99946	0.01216	0.99360	0.99978	0.01257	0.99268	0.99967	0.01431
FL: Tranco Random - Model Epoch	0.99565	0.99952	0.00823	0.99413	0.99978	0.01153	0.99498	0.99929	0.00934
Feature Extractor Sharing	0.99394	0.99921	0.01133	0.99411	0.99913	0.01090	0.99307	0.99880	0.01266
T/S: Soft Labels	0.98979	0.99963	0.02006	0.99029	0.99965	0.01908	0.99001	0.99949	0.01948
T/S: Hard Labels	0.98966	0.99968	0.02036	0.99017	0.99965	0.01930	0.99026	0.99950	0.01898

Table 5: Results of the average case including all the classifier types and all the collaborative machine learning approaches

Similarly to Ensemble classification and Feature extractor sharing, all the approaches of D5.6 (except for the FL setting Random Model - Model Convergence) achieve better TPRs than the baseline. Again, this is an expected outcome because the training datasets used by the individual organizations contain additional malicious labelled samples from which a collaboratively trained classifier can learn.

The only exception is the FL setting Random Model - Model Convergence. In this setting, we use a randomly initialized model and federate the updates of the local models after they converged. While the NYU and the ResNet models are still functional and achieve only slightly worse classification scores, the TPR of the Endgame classifier falls from over 99.9% (baseline) to 55%. We reckon the reason is that the model updates from a randomly initialized model to a fully converged model vary significantly and the individual organizations optimize their models to different local optima. Averaging and applying all the model updates may result in a non-optimal global model. The Endgame model is far more affected by this compared to the CNN-based NYU and ResNet models. This is due to the fact that RNNs are processing inputs sequentially. Averaging the weight updates of fully converged models that are used to process sequential data can thus result in a non-functional global model. In the following, we exclude the FL setting Random Model - Model Convergence from our study and mainly focus on the FPR for our assessment.

All other FL scenarios lead to an improvement compared to the baseline results. Here, the Endgame model performs significantly better in the scenarios where a pre-trained initial global model was used. We assume that this is due to the fact that when using a pre-trained model, there are significantly fewer gradient updates towards local optima for the participants to optimize their models. Similarly to the FL setting Random Model - Model Convergence, this is an important property, especially for RNN-based classifiers. In contrast, the ResNet model achieves the best results using the randomly initialized model. In all the FL settings, federating after each model epoch achieves better results than federating after model convergence. Compared to the baseline classifiers, the best FL approaches achieve on average a FPR reduction of 51.7%, 27.9%, and 44.3% for Endgame, NYU and ResNet, respectively.

The T/S approach leads to worse results than the baseline. Comparing T/S to Ensemble classification, the T/S approach yields a lower FPR for all the three classifier types. Furthermore, with either approach, it makes little difference whether soft or hard labels are used.

In summary, in the average-case scenario, only the Feature Extractor Sharing and FL approaches are advantageous for the use case of DGA detection. Using federation after model epoch leads to better results than federation after model convergence in FL.

Best Case

In Table 6, we display the FPRs of the best and worst case for all the classifier types and all the collaborative machine learning approaches separated by the number of participants.

Approach	Parties	Best Case			Worst Case		
		Endgame	NYU	ResNet	Endgame	NYU	ResNet
Baseline	-	0.00315	0.00328	0.00326	0.02166	0.02024	0.02078
Ensemble: Soft Labels	2	0.02149	0.02023	0.02081	0.02420	0.02306	0.02376
	3	0.02393	0.02266	0.02316	0.02493	0.02409	0.02387
	4	0.02489	0.02402	0.02366	-	-	-
Ensemble: Hard Labels	2	0.02068	0.01978	0.01887	0.02399	0.02242	0.02127
	3	0.02332	0.02207	0.02077	0.02462	0.02322	0.02186
	4	0.02404	0.02316	0.02167	-	-	-
FL: Random Model - Model Epoch	2	0.00464	0.00620	0.00465	0.02032	0.01886	0.01668
	3	0.00738	0.00922	0.00516	0.02122	0.01786	0.01473
	4	0.01120	0.01120	0.00624	-	-	-
FL: Tranco Top - Model Convergence	2	0.00803	0.00783	0.00856	0.01769	0.01841	0.01924
	3	0.01147	0.01170	0.01230	0.01723	0.01834	0.01877
	4	0.01267	0.01365	0.01415	-	-	-
FL: Tranco Top - Model Epoch	2	0.00361	0.00535	0.00410	0.01571	0.01905	0.01655
	3	0.00491	0.00879	0.00597	0.01492	0.01830	0.01573
	4	0.00591	0.01205	0.00780	-	-	-
FL: Tranco Random - Model Convergence	2	0.00719	0.00745	0.00974	0.01763	0.01799	0.01889
	3	0.01022	0.01062	0.01297	0.01680	0.01773	0.01831
	4	0.01185	0.01235	0.01428	-	-	-
FL: Tranco Random - Model Epoch	2	0.00351	0.00520	0.00394	0.01551	0.01857	0.01593
	3	0.00458	0.00877	0.00599	0.01397	0.01775	0.01557
	4	0.00575	0.01146	0.00958	-	-	-
Feature Extractor Sharing	2	0.00306	0.00303	0.00314	0.01871	0.01756	0.01804
	3	0.00295	0.00302	0.00307	0.01796	0.01722	0.01784
	4	0.00293	0.00298	0.00302	-	-	-
T/S: Soft Labels	2	0.00328	0.00339	0.00339	0.02264	0.02143	0.02198
	3	0.01735	0.01703	0.01646	0.02252	0.02171	0.02247
	4	0.01822	0.01776	0.01744	-	-	-
T/S: Hard Labels	2	0.00329	0.00331	0.00333	0.02386	0.02132	0.02103
	3	0.01725	0.01795	0.01650	0.02208	0.02246	0.02218
	4	0.01766	0.01761	0.01694	-	-	-

Table 6: FPRs of the best and worst case for all the classifier types and all the collaborative machine learning approaches separated by the number of participants

In contrast to Feature Extractor Sharing, all the collaborative machine learning approaches of D5.6 lead to worse results with an increasing number of participants and lead to worse results than the baseline. The reasons for this behaviour are already given in the best case analysis subsection for the collaborative machine learning approaches of this deliverable.

In summary, it can be seen from these results that only Feature Extractor Sharing is beneficial in the best-case scenario, where organizations want to use collaborative machine learning classifiers to classify samples from their own network most of the time.

Worst Case

In contrast to the results obtained for the Ensemble classification, which deteriorate as the number of participants increases for all the three classifiers, the FPRs for the T/S approach improve for the Endgame classifier. However, the achieved rates are worse than those of the baseline.

For FL, the FPRs improve in all the settings and for all the classifiers, except for Endgame, when a randomly initialized model is used. We assume that this is due to the same reasons given in the average case analysis subsection. The ResNet classifier, however, achieves the best results using a randomly initialized model. The achieved FPRs for Endgame and ResNet using federation after model epoch are significantly lower than for the approaches that make use of federation after model convergence. For the NYU classifier, no significant difference can be measured for the various models. Comparing the results with Feature Extractor sharing, the achieved rates by Feature Extractor sharing are significantly worse than the ones achieved by Endgame and ResNet using FL with federation after model epoch. For NYU, the rates are comparable to those seen in the FL settings.

In summary, in the worst-case scenario, only the Feature Extractor Sharing and FL approaches improve in performance with an increasing number of participants and achieve better scores than the baseline.

FL with federation after model epoch achieves the best results for Endgame and ResNet and, thus, generalises best to different networks. When comparing the different types of classifiers, Endgame and ResNet are better suited for FL than NYU. For RNN-based classifiers, pre-trained initial models should be used.

3.9 Conclusion

We performed a comprehensive study of collaborative machine learning for the real-world use case of DGA detection. Thereby, we identified advantageous and disadvantageous approaches for different types of classifiers and showed that collaborative machine learning can lead to a reduction in FPR by up to 51.7%. Additionally, we showed that the usage of publicly available data is insufficient for DGA detection on private data. This shows the need for privacy-preserving collaborative machine learning approaches. In two real-world cases, we have shown that wider participation in collaborative machine learning does not always lead to better classification results. In fact, we only assess Feature Extractor Sharing and FL of the four examined collaborative machine learning approaches as beneficial for DGA detection. Feature Extractor Sharing should be used if a party wants to classify samples that come from their own network most of the time. FL on the other hand generalizes best to unknown networks. The four examined collaborative machine learning variants based on T/S learning and Ensemble classification lead to worse results than the baseline.

3.10 Privacy Analysis

As any kind of sharing activity may generate an attack surface that threatens the privacy of the shared object, we investigate the privacy implications of the sharing approaches in the DGA detection use case. In the preliminary version of this Deliverable (D5.3), the Machine Learning privacy attack landscape was introduced in a brief but formal overview. We identify the Data Inference class as the relevant attack class for the sharing scenario in this deliverable, with a focus on the Model Inversion and Membership Inference attacks. At this point, a thorough discussion on relevant privacy-threatening inference attacks against the beneficial sharing approaches shall complement our sharing scenario evaluation.

As Feature Extractor Sharing is the only beneficial DGA sharing approach in this Deliverable, we discuss only its privacy aspects. To recapitulate: In this approach, each party shares a partial model (front part / feature extractor), while the final models with global intelligence are later personalized, as each party trains its own decision layers (rear part) locally.

Since partial models are shared openly, an adversary is granted white-box access. With such, they have access to the models' parameters, which are directly influenced by the data in the learning process. Further, the adversary also has access to the gradient computation on the model function. We transferred the privacy discussion about this sharing approach from Deliverable D5.2 to this Deliverable (D5.4), as the sharing approach is better suited here. While in the sharing scenario in D5.2 each party had to share its data after passing it through a public feature extractor, this sharing approach only requires sharing the locally trained feature extraction models, but no data. This shrinks the attack surface and makes the application of deep learning feature extractors more suitable in this scenario than in the scenario described by D5.2. We discuss the threat of both potentially applicable privacy-threatening attacks, Model Inversion and Membership Inference, in the following.

3.10.1 Model Inversion

In a white-box setting, gradient-based Model Inversion attacks [26, 27] may be deployed against the model function. For a given model output, these attacks aim to find an input close to the unknown original input that created the known model output. This is achieved by an iterative process that optimizes the input to the model function to produce the target output. The optimization process is guided by a gradient ascent or descent on the model function's input with respect to some loss comparing target output and model function output. These gradient-based attacks are also applicable to partial models. Due to stochasticity in the gradient descent optimization process, this attack, similar to any

other machine learning task, does not have a closed form solution and is sensitive to initialization. Therefore, the attack will produce a reconstructed input that produces an output close to that of the target, while the closeness of the original input and the reconstruction input is not guaranteed. Among others, the efficacy of the attack is therefore highly dependent on the learning task, the optimization's hyperparameters and the model's input space complexity.

In the following, we provide reasons why we consider Model Inversion to be a minor threat in the Feature Extractor Sharing scenario:

1. The partial models which are disclosed in the sharing process are trained independently from each other in terms of both training data and training party. This does not “fit in” any threat models derived from split learning [28], where consecutive vertical splits of a whole model are trained in succession by different parties, yet on the same data.

2. Classic Model Inversion attacks do not perform well in practice, especially if the targeted model is highly complex (in number of layers or weights) [29]. The non-trivial architecture of our models, given by either convolution or recurrent operations or residual connections, is likely too complex for such an attack to perform well.

3. The embedding layer used as the first layer in all the three model architectures is non-differentiable, and, thus, the gradient optimization may back-propagate up to the output of that embedding layer at most. It is possible to reverse the embedding layer in each step, utilizing an inverse discrete mapping. We view this as an additional obstacle for the attack, as the optimization process cannot traverse the solution space in a continuous manner.

4. The last and most valuable argument is the following: The goal of a Model Inversion attack is to reconstruct an input for a known output. The sharing parties only disclose their feature extractor model but never disclose any intermediate outputs of their model, and, as a result, the attacking party has no starting point for the attack. Reconstructing true inputs requires the knowledge of a set of the targeted party's feature vectors, which are however not shared in this approach.

3.10.2 Membership Inference

The classic Membership Inference attack is defined for complete classifier models, i.e., with a final softmax output. Theoretically, the attack setup, as described in [30], could be redefined for the case of a partial model. However, the semantics of an intermediate layer output are not clearly defined. The original attack builds on the premise that every model will be biased to show a higher confidence for training samples than for samples outside of the training set. Intermediate layer outputs do not necessarily follow such a pattern. Although the layer activations could be analysed with a clustering approach, we do not expect a major privacy threat from such an attack on a partial model because the intermediate layer weights are not directly optimized to separate certain groups of data (class labels). To the best of our knowledge such research has, however, not been conducted yet.

4 Application Profiling

We developed multiple approaches for application profiling, which are described in SAPPAN D3.5. For those, we distinguish between two use cases, namely, identification and classification. For the identification case, the goal is to identify – based on network traffic – which applications run on a host. For this, we developed a rule-based as well as a process mining-based approaches. The goal of the classification case is to determine whether an application behaves as expected, i.e., to detect anomalous behaviour which may be caused by malicious activities. However, for application profiling, we focused on the local use cases considered in WP3. While such classical collaboration approaches as teacher-student or federated learning are applicable to machine learning methods, they are not so relevant for

our rule-based and process mining-based methods. We also experimented with deep learning techniques for application profiling, but the other, simpler, approaches showed more promising results. Hence, when it comes to application profiling in the context of WP5, we only continued with experiments based on shared data, which are covered in D5.2.

5 Deriving Low-Dimensional Embeddings of Computer Processes

5.1 Introduction

This section presents research into the feasibility of using NLP-style embeddings to derive parent-child relationships between executable processes based on recordings of their behaviour within (numerous) computer systems. This work is related to the Process Launch Distribution model (further referred to as PLD) described in D5.3. Since popular methods for constructing embeddings in NLP (such as [37, 35]) can be carried out in the federated learning fashion ([40, 41]), the use of those brings us the benefits of sharing in the training process.

The previously described PLD mechanism is designed to capture the distribution of process launch events in endpoints and to enable the detection of anomalous events via a statistical distribution with thresholds model. The underlying assumptions in PLD are that (i) benign events in a training set are much more frequent than events connected with attacks, and (ii) process launch events where two common (popular) processes are used in anomalous ways are sufficiently reliable signs of attacks. PLD defines a score function that reflects how anomalous a given process launch event is: the closer to zero the PLD score is, the more anomalous the process launch event is, and thus the more suspicious it is from a security intuition point of view.

With respect to the previously studied PLD model, the research described in this Section is designed to determine the following:

1. Whether an alternative way to encode co-occurrence data of processes can be useful for PLD-like approaches for reasons including (1) simplified data storage and transfer routines, (2) mitigated concerns about the need to interchange plain data values and (3) applicability of recent advances in the field of federated learning [31, 32] that could positively address customers' privacy concerns.

2. Whether the use of process embeddings for encoding semantic similarity between their contexts is viable. For the case of PLD, this implies that every process can be described by two embedding vectors representing a process as either a parent or a child (the context for a parent process embedding is defined by its child processes and vice versa). For the sake of simplicity in our initial experiments, we ultimately suggest concatenating embedding vectors to represent each process.

The latter point is very important for further use cases, since intrinsic properties of embeddings contribute to solving a foundational challenge in applying machine learning in the cybersecurity domain: representing unordered categorical feature values in a dense, low-dimensional form instead of a traditional one-hot encoding schema. Additionally, embeddings increase machine learning model efficiency and simplicity (from resource consumption and complexity points of view) and can be used by other cybersecurity-relevant models (where downstream models that rely on features of categorical entities can utilize already pretrained embeddings [33]).

5.2 Global Vectors (a.k.a. GloVe) approach to build word embeddings

This study utilizes the same data that was used in the previous PLD research (detailed in D3.4 and D5.3). The data can be considered a co-occurrence matrix of (parent, child) process pairs. Processes are represented by their file names (e.g., 'cmd.exe', 'svchost.exe', 'iexplore.exe'). This representation

perfectly matches the matrix form needed for building word embeddings in the Natural Language Processing (NLP) domain using popular approaches based on matrix factorization (e.g., Latent Semantic Analysis) and the Global Vectors representation model (known as GloVe [1]). Other shallow window-based methods (skip-gram and continuous bag-of-words methods also known as Word2vec) might also be considered relevant since those are well-documented and straightforward, and existing high-quality implementations are widely available (for example, Python implementations are available in the popular package Gensim [34]).

The main advantage of the GloVe model for our setting is that it captures global corpus statistics directly and can be considered a matrix factorization approach that while trying to approximate co-occurrence matrix also trains semantic-aware embeddings for parent and child processes. This is the key reason why GloVe was selected for this feasibility study.

A detailed description of the method is available in [35]. For the purposes of this report, the GloVe cost function will be described in detail:

$$J = \sum_{i,j}^{V} f(X_{i,j})(w_i{}^T \widetilde{w_j} + b_i + \tilde{b}_j - logX_{i,j})^2$$

V is the vocabulary size (the number of processes in this study is 1,000), function f is a weighting function of the co-occurrence matrix element $X_{i,j}$ that aims – by introducing weights – to avoid treating frequent and rare co-occurrence values equally. As it can be seen, the cost function aims to train model parameters that represent embedding vectors w and bias terms b. For every categorical entity i (word in the original paper or process name in our study), the approach learns a central embedding vector w_i (in our convention – an embedding that represents the parent process with respect to its child processes or, simply put, parent process embedding vector), context embedding vector $\widetilde{w_i}$ (correspondingly, child process embedding vector) and two bias terms b_i and $\tilde{b}_i$.

According to the cost formula's structure, to get the co-occurrence matrix's element $X_{i,j}$ from the model, the following expression is used:

$$X_{i,j} = e^{(w_i{}^T \widetilde{w_j} + b_i + \tilde{b}_j)}$$

To identify entities close to a given entity, we can simply apply a cosine similarity measure to their embeddings. To this end, for every type of embedding vector w (i.e., parent, child, combined) a distance matrix can be computed, which is then used to select a defined number of closest embeddings (5 by default in our experiments). The learned bias values b are not involved in this similarity check.

5.3 Used data

The data collected for the experiments (described in D5.3) represents parent - child interactions of the 1000 most common processes that were observed in the original PLD dataset. The original PLD dataset represents a three-week collection of new process events from a few hundred thousand endpoints running Microsoft Windows. This dataset consists of parent-child process counters that were collected and submitted to the security backend by the sensor software running on each monitored endpoint. Every such a submission is a list of tuples of file names of parent and child processes and a counter of the respective new process events.

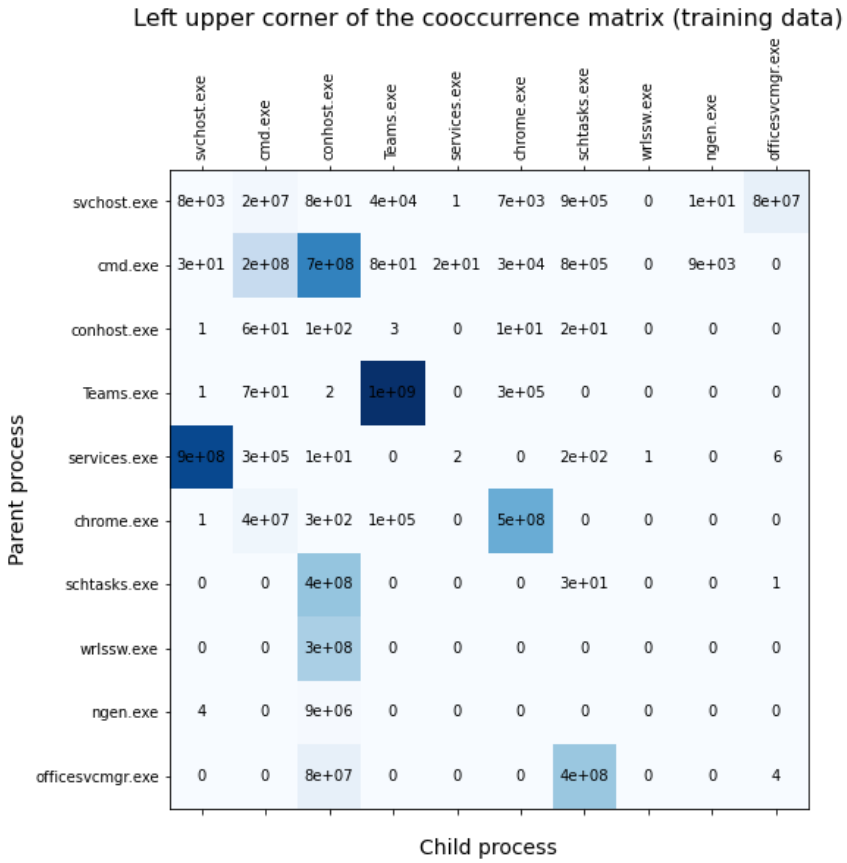

Figure 5. An excerpt of the available PLD data that were used for training

The resulting data consists of 1,000,000 co-occurrence values for the process pairs and is sparse – approximately 1% of all the co-occurrence values are non-zero. Figure 5 illustrates a portion of the data in a co-occurrence matrix form, where the rows and columns represent parent and child process names respectively and the values denote the number of times each parent – child pair was observed. In a process launch event, a launched process is the child process, and a launching process is the parent process.

5.4 Training process

The embeddings construction (training) was performed using CPU-only TensorFlow 2.6.0 on a Dell Precision 5530 running Microsoft Windows 10 Enterprise. The system's specifications were as follows:

- Processor: Intel(R) Core (TM) i9-8950HK CPU @ 2.90GHz
- Installed RAM: 32.0 GB (31.7 GB usable)
- System type: 64-bit operating system, x64-based processor

For the sake of simplicity, we selected the top 1,000 most active processes from the original dataset. For the given training set size, it was enough to limit the number of training iterations to 100, using a batch size of 1024.

5.5 Evaluation scenario 1: From parent and child embeddings to co-occurrence counter values

The model's capability to reconstruct co-occurrence counter values is an inherent property of the Global Vectors model stipulated by the definition of the training objective (minimization of the cost function) it relies on. Our empirical checks of selected examples, relevant in the cybersecurity domain, indicate that the model tends to make correct approximations of co-occurrence information. This is not the case in general though; we believe that this issue can be addressed by a trade-off between two

key parameters that should be defined in a use case-specific manner: embedding size and model accuracy.

(Parent, Child) process pair	Original cooccurrence	Predicted cooccurrence
(powershell.exe, OUTLOOK.EXE)	34166.0	41527.18
(OUTLOOK.EXE, powershell.exe)	1.0	0.61
(Teams.exe, Teams.exe)	981989504.0	24360053.0
(WINWORD.EXE, powershell.exe)	1.0	0.93
(EXCEL.EXE, powershell.exe)	19.0	19.45
(msedge.exe, powershell.exe)	0.0	-0.15

Table 7. Examples of the model's predictions (embedding size is 64).

Table 7 presents some examples of the model's predictions of co-occurrence counters. The first column represents the validated parent-child process pair (per se, it defines the co-occurrence value in the training data matrix). Original occurrences are the values from the training data. Predicted co-occurrences present the model's predictions of the co-occurrence values. Thus, the differences of the original and predicted values can be interpreted as the model's errors for parent-child pairs. It is necessary to emphasize that, for the PLD approach, the larger the original number, the lower the concerns of the significance of prediction errors. In a realistic production setting, we consider additional preprocessing of the training data matrix in order to avoid the overly high counter values; for example, they can be scaled, normalized or substituted by anomaly category values.

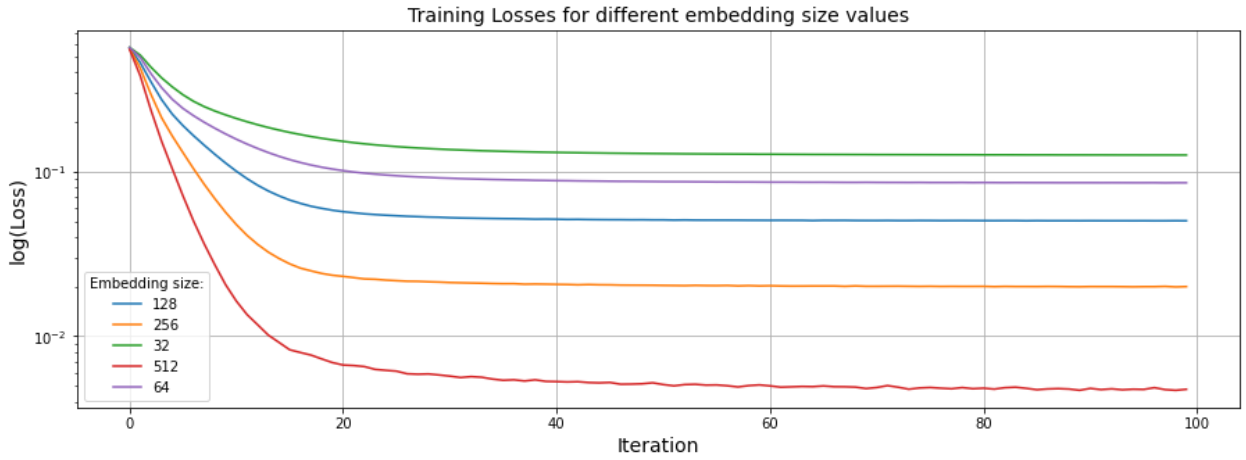

Figure 6. Learning curves for different sizes of embeddings

As can be seen from Figure 6, the loss, and respectively, the model's ability to approximate co-occurrence information, depends on the selected size of embeddings. This is an ordinary fact adjacent with the concept of overfitting, since models with a larger number of parameters memorize better.

5.6 Evaluation scenario 2: From embeddings to similar process names

A trained model can be used to calculate cosine similarity values for any pair of items in the dataset, and thus build a distance matrix that encodes all the information necessary to find closest embeddings and, respectively, closest process names.

Process name	Compared embeddings	Five closest processes (cosine similarity)

OUTLOOK.EXE	Parent	POWERPNT.EXE (0.98), WINWORD.EXE (0.91), EXCEL.EXE (0.88), sihost.exe (0.72), iexplore.exe (0.72)
	Child	EXCEL.EXE (0.96), WINWORD.EXE (0.95), POWERPNT.EXE (0.9), iexplore.exe (0.9), notepad.exe (0.87)
	Combined	WINWORD.EXE (0.94), EXCEL.EXE (0.93), POWERPNT.EXE (0.93), iexplore.exe (0.8), AcroRd32.exe (0.79)
chrome.exe	Parent	msedge.exe (0.89), brave.exe (0.66), rundll32.exe (0.66), iexplore.exe (0.65), explorer.exe (0.65)
	Child	msedge.exe (0.94), iexplore.exe (0.89), firefox.exe (0.85), WerFault.exe (0.83), notepad.exe (0.83)
	Combined	msedge.exe (0.9), iexplore.exe (0.78), EXCEL.EXE (0.72), firefox.exe (0.72), brave.exe (0.72)
WhatsApp.exe	Parent	Skype.exe (0.8), brave.exe (0.75), slack.exe (0.74), Rainbow.exe (0.7), Rainmeter.exe (0.69)
	Child	slack.exe (0.92), Skype.exe (0.83), AppVLP.exe (0.80), RAVCpl64.exe (0.76), wfcrun32.exe (0.75)
	Combined	slack.exe (0.82), Skype.exe (0.8), Atlassian Companion.exe (0.69), sihost.exe (0.65), Teams.exe (0.64)
java.exe	Parent	python.exe (0.84), cmd.exe (0.83), powershell.exe (0.81), Code.exe (0.78), services.exe (0.76)
	Child	node.exe (0.66), perl.exe (0.64), python.exe (0.63), sc.exe (0.63), net.exe (0.61)
	Combined	cmd.exe (0.66), python.exe (0.65), powershell.exe (0.62), javaw.exe (0.61), perl.exe (0.6)

Table 8. Examples of closest process names (embedding size is 64)

Table 8 presents similarity values for the most similar processes for a selection of process names. Sparse context data, such as child and parent process co-occurrence values, help define groups of semantically similar applications (under the assumption that we trust the names of the executable files). Table 8 illustrates that the learned embeddings captured similarities between office applications, browsers, messengers, and scripting / programming language interpreters. This was not the case generally, though, and thus some observations should be manually checked by human experts and data sources. From our perspective, the most promising approach is to rely on a combined embedding representation (concatenation of parent and child embedding vectors).

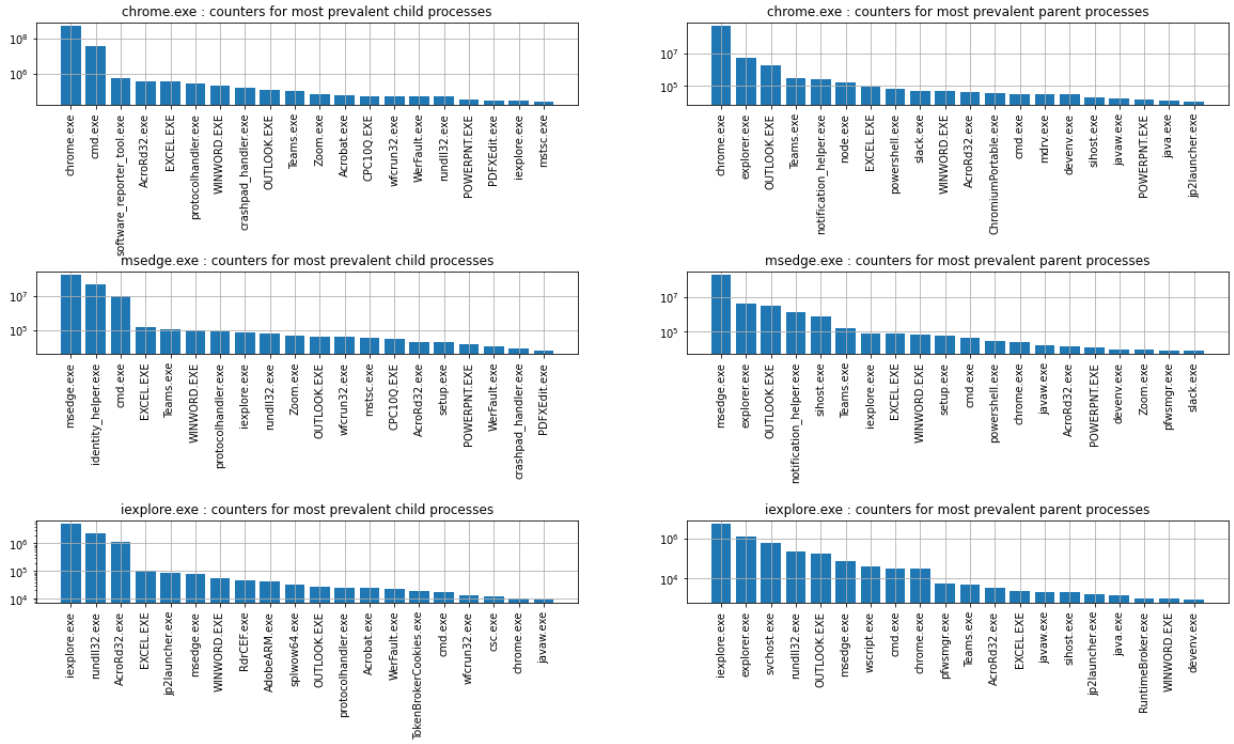

Figure 7. Top 20 parent and child processes for some browser process names

Figure 7 visualizes original training data (co-occurrence counters) for some of the process names that are close from the embedding similarity search's perspective.

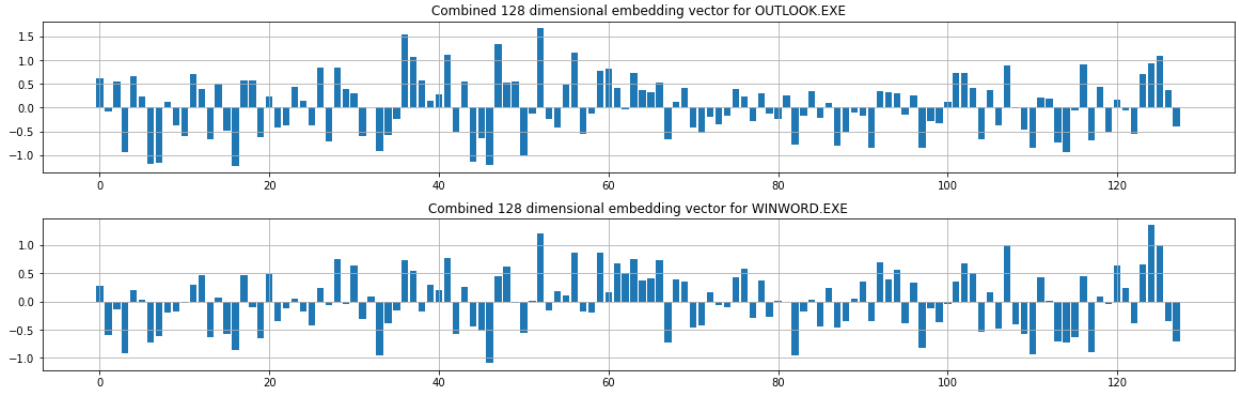

Figure 8. Combined embeddings for two process names that are close to each other

Figure 8 presents an example of two combined embeddings for semantically similar process names. The embeddings size is 64. After concatenating them we get two 128 dimensional vectors where the left side part is the parent embedding vector and the right one is the child embedding vector.

5.7 Conclusions and future directions

The feasibility study described in this section demonstrated that natural language processing approaches for building word embedding models can be utilized in the cybersecurity domain to create embeddings for processes, which traditionally required one-hot encoding transformations. The previously discussed PLD model is well aligned with the GloVe approach, and, in fact, it is possible to substitute co-occurrence counter data with a set of process-name-specific embedding vectors and biases that can be used to encode co-occurrence information on the endpoint in a more private manner (enabling privacy-preserving sharing).

Availability of embeddings makes it possible to consider advanced scenarios of:

- transmitting embedding vectors to endpoints via cloud lookups (instead of via traditional delivery mechanisms such as software or settings updates);
- using embeddings for context-specific similarity searches in other relevant systems;
- using embeddings to present process names and other types of high cardinality categorical entities for solving downstream problems in a machine learning setting.

Future research directions include additional studies addressing (1) ways of representing the data in a counter-unaware form (for example with categories), (2) possible approaches to take detailed contextual data for PLD, for example from process trees, (3) other relevant categorical entity types like settings (Registry keys), folders (file locations), network port numbers; (4) other relevant contexts for process entities that define their operations with file system, network, users, etc.; (5) efficient construction of relevant embeddings in the federated learning manner.

6 Towards Detecting Anomalous Registry Writes

6.1 Motivation and Introduction

A telltale sign of a suspicious Windows executable is its anomalous behaviour with respect to the Windows Registry. For example, malicious executables in many cases try to access or even overwrite critical passwords stored in the Windows Registry. Additionally, a typical Windows Registry contains an enormous number of keys, so a rule-driven approach is likely to miss many edge events or produce a lot of noise.

A Windows Registry write event contains three main attributes:

1. The key being accessed
2. The executable accessing the key
3. The value being written into the key

If any one of these three features is sufficiently suspicious, we have a good reason for raising an alert. However, for example, a (key, executable) pair may be suspicious without either individual component of the pair raising concerns (or just known to be malicious). As such, subtle ML methods may be able to detect suspicious Registry write events.

Aside from the lone reference [36], there is little academic literature about detecting anomalous behavior with respect to Windows Registry writes or reads. Moreover, the anomaly detector in [36] essentially one-hot encodes Registry keys, causing an explosion in dimensionality. The authors deal with the large dimensionality of the problem by using a simple Dirichlet-estimator based anomaly detector, which predicts when a (key, executable) pair is anomalous. Given that [36] is currently nearly two decades old and that the Machine Learning domain has seen rapid advances in the recent past, we believe that the topic is worth revisiting and amenable to the application of the embeddings techniques introduced in Section 5.

6.2 Normalizing Registry Keys

To construct an anomaly detector on Registry write events, we must first do a fair bit of pre-processing to the Registry key data. To illustrate the need for normalizing the Registry keys, consider the following example registry key:

```
\\REGISTRY\\USER\\S-1-5-21-304820375-743328050-817656539-1131\\Soft-
ware\\Microsoft\\Windows\\CurrentVersion\\Explorer\\UserAssist\\{F4E57C4B-
2036-45F0-A9AB-443BCFE33D9F}
```

The SID beginning with `S-1-5-21` and the UUID beginning with `F4E57C4B` contain many random characters but very little information, and there can be millions of such identifiers throughout the database. As such, our task is made easier by simply replacing all SID's with some pre-determined string, say `sid_string`, and similarly for the UUID's.

More pre-processing is done to the key to remove additional sources of randomness, punctuation is eliminated, and the key is split along the backslashes into an array of strings. After normalization, the above example key becomes:

```
[registry, user, sid_string, software, microsoft, windows, currentversion, explorer, userassist, uid]
```

6.3 Vectorizing Registry Keys

Our goal is to vectorize such arrays in order to obtain a numerical representation of Registry keys. It is possible to use a traditional CountVectorizer or TfIDFVectorizer to do this, but there are two key problems with such an approach:

1. The dimensionality of the vectors is still enormous, even after key normalization.

2. The context of each token is lost. That is, such simple models do not take into account which tokens typically appear next to each other, which is valuable information.

Using a Word2vec model [37, 38] fixes both of these problems, as it allows us to set the dimension to whatever we wish, and it takes word order and context into account when constructing embedding vectors.

To give more details about the embedding vectors construction, we train a Word2vec model on the 'sentences' given by the normalized Registry keys. This yields embedding vectors for each individual token, which are then summed to get an embedding vector for the entire registry key. This particular model was trained with one day's worth of data from one organization, which contained 700,000,000 Registry write events and 780,000 unique normalized Registry keys.

We can validate the Word2vec embedding vectors of Registry keys by selecting a few keys at random and computing the most similar keys with respect to cosine similarity. With the embedding dimension set to $d = 100$, we obtain the results as depicted in Figures 9 and 10. In both examples, the top row is the query Registry key selected, and the rows below it reflect the most similar non-identical keys. In both examples, we see that cosine similarity between vectorized Registry keys reflects real semantic similarity between the keys.

normalized_registry_key	score
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, speech, headsetbuttonpress]	1.000000
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, speech, speechmodeldownloadtask]	0.999884
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, timezone, synchronizetimezone]	0.987516
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, pushtoinstall, logincheck]	0.985948
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, workfolders, workfolderslogonsynchronization]	0.985767
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, filehistory, filehistorymaintenancemode]	0.985692
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, workfolders, workfoldersmaintenancework]	0.985567
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, cuassistant, culauncher]	0.985497
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, cuassistant]	0.985216
[registry, machine, software, microsoft, windowsnt, currentversion, schedule, taskcache, tree, microsoft, windows, deviceinformation, deviceuser]	0.984840

Figure 9: Most similar Registry keys to a given random key

normalized_registry_key	score
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, 0a0b0d00, uuid]	1.000000
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, 0cb0e0d0, uuid]	0.999786
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, d0c0a00, uuid]	0.999762
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, a0ae00, uuid]	0.999757
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, ac0d00, uuid]	0.999752
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, 0cd0f00, uuid]	0.999751
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, 0aa0e00, uuid]	0.999750
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, 0fe0ea0, uuid]	0.999746
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, c0fb0c00, uuid]	0.999746
[registry, user, weirds, software, microsoft, internetexplorer, lowregistry, audio, policyconfig, propertystore, b0f0a00, uuid]	0.999745

Figure 10: Most similar registry keys to a given random key

6.4 Discussion and Future Work

The work we have done towards detecting anomalous Registry writes is still in progress, and we hope to build on what we have so far to deliver a working anomaly detector. The success of using Word2vec to embed Registry keys in a low-dimensional Euclidean space is a promising start, and the next step will be to link such embedding vectors to the write events as a whole.

7 Conclusion

This deliverable continues and extends the discussion started in SAPPAN D5.3 (the first version of the deliverable produced by SAPPAN Task 5.2), where we presented the initial approaches and experimental results for model sharing in the settings of domain generation algorithms (DGA) detection, application profiling, and anomalous behaviour detection in endpoints.

The focus in this document was on: the evaluation and the privacy analysis of the proposed DGA detection methods; a method, amenable to federated learning techniques, for deriving low-dimensional embeddings of computer processes, which can be used by and can improve the performance of other cybersecurity-relevant models, in particular, the models presented in D5.3 for detection of anomalous process launch and process access events in endpoints; another potential use case for the embeddings approach, where the goal is to detect security-related anomalous Windows Registry write events.

As we find the presented methods and techniques promising for the practical use in real-world attack detection solutions, a number of further experiments with those are in the plans and we expect to report on the results in the SAPPAN WP6 deliverables.

8 References

[1] A. Drichel, B. Holmes, J. von Brandt, and U. Meyer. The More, the Better? A Study on Collaborative Machine Learning for DGA Detection. In Workshop on Cyber-Security Arms Race. 2021.

[2] J. Saxe and K. Berlin. eXpose: A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys. arXiv:1702.08568. 2017.

[3] S. Schüppen, D. Teubert, P. Herrmann, and U. Meyer. FANCI: Feature-Based Automated NXDomain Classification and Intelligence. In USENIX Security Symposium. 2018.

[4] R. Sivaguru, C. Choudhary, B. Yu, V. Tymchenko, A. Nascimento, and M. De Cock. An Evaluation of DGA Classifiers. In IEEE International Conference on Big Data. 2018

[5] D. Tran, H. Mac, V. Tong, H. A. Tran, and L. G. Nguyen. A LSTM Based Framework for Handling Multiclass Imbalance in DGA Botnet Detection. Neurocomputing 275. 2018.

[6] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant. Predicting Domain Generation Algorithms with Long Short-Term Memory Networks. arXiv:1611.00791. 2016

[7] B. Yu, J. Pan, J. Hu, A. Nascimento, and M. De Cock. Character Level Based Detection of DGA Domain Names. In International Joint Conference on Neural Networks. IEEE. 2018.

[8] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S.Abu-Nimeh, W. Lee, and D. Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In USENIX Security Symposium. 2012.

[9] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel. Exposure: A Passive DNS Analysis Service to Detect and Report Malicious Domains. ACM Transactions on Information and System Security 16, 4. 2014.

[10] M. Grill, I. Nikolaev, V. Valeros, and M. Rehak. Detecting DGA Malware Using NetFlow. In IFIP/IEEE International Symposium on Integrated Network Management. 2015.

[11] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero. Phoenix: DGA-Based Botnet Tracking and Intelligence. In Detection of Intrusions and Malware, and Vulnerability Assessment. Springer. 2014

[12] Y. Shi, G. Chen, and J. Li. Malicious Domain Name Detection Based on Extreme Machine Learning. Neural Processing Letters 48, 3. 2018.

[13] S. Yadav and A. L. N. Reddy. Winning with DNS Failures: Strategies for Faster Botnet Detection. In International Conference on Security and Privacy in Communication Systems. Springer. 2011.

[14] J. Peck, C. Nie, R. Sivaguru, C. Grumer, F. Olumofin, B. Yu, A. Nascimento, and M. De Cock. CharBot: A Simple and Effective Method for Evading DGA Classifiers. ArXiv:1905.01078. 2019

[15] J. Spooren, D. Preuveneers, L. Desmet, P. Janssen, and W. Joosen. Detection of algorithmically generated domain names used by botnets: A dual arms race. In Proceedings of the 34rd ACM/SIGAPP Symposium On Applied Computing. ACM. 2019.

[16] P. Lison and V. Mavroeidis. Automatic Detection of Malware-Generated Domains with Recurrent Neural Models. In Norwegian Information Security Conference. 2017.

[17] A. Drichel, U. Meyer, S. Schüppen, and D. Teubert. Analyzing the real-world applicability of DGA classifiers. In International Conference on Availability, Reliability and Security. ACM, 2020.

[18] F. Becker, A. Drichel, C. Müller, and T. Ertl. Interpretable visualizations of deep neural networks for domain generation algorithm detection. In Symposium on Visualization for Cyber Security. IEEE, 2020.

[19] A. Drichel, N. Faerber, and U. Meyer. First Step Towards EXPLAINable DGA Multiclass Classification. In International Conference on Availability, Reliability and Security. ACM, 2021.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Identity Mappings in Deep Residual Networks. In Computer Vision – ECCV. Springer. 2016.

[22] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla. A Comprehensive Measurement Study of Domain Generating Malware. In USENIX Security Symposium. 2016.

[23] A. Drichel, U. Meyer, S. Schüppen, and D. Teubert. 2020. Making Use of NXt to Nothing: Effect of Class Imbalances on DGA Detection Classifiers. In Conference on Availability, Reliability and Security. ACM.

[24] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In Network and Distributed System Security Symposium. Internet Society.

[25] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Artificial Intelligence and Statistics. PMLR.

[26] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart. Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing. In USENIX Security Symposium. 2014.

[27] M. Fredrikson, S. Jha, and T. Ristenpart. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In Computer and Communications Security. 2015.

[28] D. Pasquini, G. Ateniese, and M. Bernaschi. Unleashing the Tiger: Inference Attacks on Split Learning. In ACM Conference on Computer and Communications Security. 2021

[29] B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning. In Computer and Communications Security. 2017.

[30] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership Inference Attacks Against Machine Learning Models. In IEEE Security and Privacy. 2017.

[31] WIPO IP portal, WO2021080577: "Online federated learning of embeddings", link: https://patentscope.wipo.int/search/en/detail.jsf?docId=WO2021080577&tab=PCTDESCRIPTION

[32] Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R. and D'Oliveira, R.G., 2019. Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977.

[33] Burr, B., Wang, S., Salmon, G. and Soliman, H., 2020, April. On the detection of persistent attacks using alert graphs and event feature embeddings. In NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium (pp. 1-4). IEEE.

[34] GENSIM: topic modeling for humans. Link: https://radimrehurek.com/gensim/

[35] Pennington, J., Socher, R. and Manning, C.D., 2014, October. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543). Link: https://nlp.stanford.edu/pubs/glove.pdf

[36] F. Apap, A. Honig, S. Hershkop, E. Eskin, and S. Stolfo. Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses. Proceedings of the 5th international conference on recent advances in intrusion detection, October 2002.

[37] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in Proceedings of the 26th International Conference on Neural Information Processing Systems, vol. 2, pp. 3111-3119, Dec. 2013.

[38] Yoav Goldberg, Omer Levy, word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method, https://arxiv.org/abs/1402.3722

[39] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In International Conference on Learning Representations. 2015.

[40] Zhiqi Huang, Fenglin Liu, Yuexian Zou, Federated Learning for Spoken Language Understanding. Proceedings of the 28th International Conference on Computational Linguistics, pages 3467–3478, Barcelona, Spain (Online), December 8-13, 2020. https://aclanthology.org/2020.coling-main.310.pdf

[41] Wei Huang, Tianrui Li, Dexian Wang, Shengdong Du, Junbo Zhang, Fairness and Accuracy in Federated Learning. https://arxiv.org/pdf/2012.10069.pdf