Sharing and Automation for
Privacy Preserving Attack Neutralization

(H2020 833418)

D5.2 Global model based on shared anonymized data, final version (M30)

Published by the SAPPAN Consortium

Dissemination Level: Public

H2020-SU-ICT-2018-2020 – Cybersecurity

Document control page

Document file:	Deliverable D5.2
Document version:	1.0
Document owner:	Sebastian Schäfer (RWTH)
Work package:	WP5
Task:	T5.1 Distributed Learning of a Global Model Based on Shared Anonymized Data
Deliverable type:	Report
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	Arthur Drichel (RWTH), Benedikt Holmes (RWTH), Sebastian Schäfer (RWTH)	2021-09-15	First version of outline including first bullet points
0.2	Arthur Drichel (RWTH), Benedikt Holmes (RWTH), Sebastian Schäfer (RWTH)	2021-10-01	Finalised outline and structure
0.3	Arthur Drichel (RWTH), Benedikt Holmes (RWTH), Sebastian Schäfer (RWTH), Alexey Kirichenko (F-Secure), Sivam Pasupathipillai (F-Secure)	2021-10-27	First complete version ready for review
0.4	Arthur Drichel (RWTH), Benedikt Holmes (RWTH), Sebastian Schäfer (RWTH), Alexey Kirichenko (F-Secure), Sivam Pasupathipillai (F-Secure), Lasse Nitz (FIT), Avikarsha Mandal (FIT)	2021-10-29	Updated version with incorporated feedback
1.0	Sebastian Schäfer (RWTH)	2021-10-29	Final version ready to submit

Internal review history:

Reviewed by	Date	Summary of comments
Lasse Nitz (FIT)	2021-10-28	Various wording, spelling and grammar suggestions, and a few comments on technical aspects
Avikarsha Mandal (FIT)	2021-10-29	Mainly editorial comments

Executive Summary

This deliverable is the final version of Deliverable D5.1 which relates to Task T5.1 "Distributed Learning of a global model based on shared anonymised data". Its goal is to build a global model, similar to the tasks T5.2 and T5.3, by directly sharing training data. Therefore, the deliverables D5.2, D5.4, and D5.6 have some overlaps, especially with respect to background, motivation, and the context within SAPPAN. In general, WP5 focuses on sharing and federation for cyber threat detection and response and this task is one of three tasks focusing on collaborative learning. This deliverable focuses on different collaborative machine learning approaches for the Domain Generation Algorithm (DGA) detection methods, as well as rule-based and process mining-based approaches for application profiling, all of which were developed in WP3. Specifically, we define three approaches for building DGA detection models based on shared anonymised data, perform a privacy study on the most promising approach, develop a novel feature-based DGA classifier for DGA multiclass which can be leveraged for data anonymisation for collaborative DGA multiclass classification, and performed experiments to measure the impact on application profiles when sharing data.

Contents

1 Introduction

In D5.2 we report the continuation of our work from the initial version of this Deliverable (D5.1). In D5.1, we described the experiments and approaches to build global models based on shared local data. For some of the experiments we already presented initial results. The final results are now described in this document.

This deliverable for the Task T5.1 has similar goals as the Tasks T5.2 and T5.3, hence, a large part of its motivation and context is similar to the other tasks. In SAPPAN, one of the innovations is to enable privacy-preserving sharing of intrusion detection data, detection models, and response handling information. The objective of sharing this information is to improve the local capabilities of each participating organisation by collaboration. Another flavour of sharing in the scope of SAPPAN is sharing among a cybersecurity service provider or vendor and various user groups of its customer organisations. In WP3, one of the tasks is to develop local detection and response mechanisms. In WP5, we want to utilise these mechanisms on the global level to improve them using different sharing mechanisms.

This deliverable focuses on building global detection models based on shared anonymised data. The idea is to collect data, prepare training data for machine learning models on the local level, anonymise it, and share it to the global level. If this is done by multiple parties, this global dataset can be used to build global detection models with increased accuracy or robustness compared to the local models. Throughout the first three tasks of WP5 we focus on the showcases for which we developed local detection mechanisms, as well as some additional ones. These mechanisms include machine learning models, process mining models, as well as rule-based models.

This document is structured as follows. First, we briefly outline the context of the Task 5.1 in the overall scheme of the SAPPAN project. Next, we describe the general idea of this task in more detail, discuss privacy for machine learning models, and afterwards describe the different showcases including different approaches for building global detection models. First, we present the showcase of DGA detection including three possible approaches for building a global model based on shared anonymised data. For the most promising approach to collaborative DGA binary detection, we perform an extensive privacy study. The results of the study were promising, hence why we developed a novel feature-based DGA classifier for DGA multiclass classification that can be used in the same sharing scenario as investigated for the binary classification case. Next, we present our experiments for three application profiling approaches based on rules and process mining. This is followed by an evaluation and discussion of our model for detecting of anomalous login activity. Finally, we conclude this deliverable with a summary.

2 SAPPAN Context

The context for building a global model in SAPPAN is similar for the first three tasks in WP5. Hence, the following paragraph can also be found in the Deliverables D5.4 and D5.6.

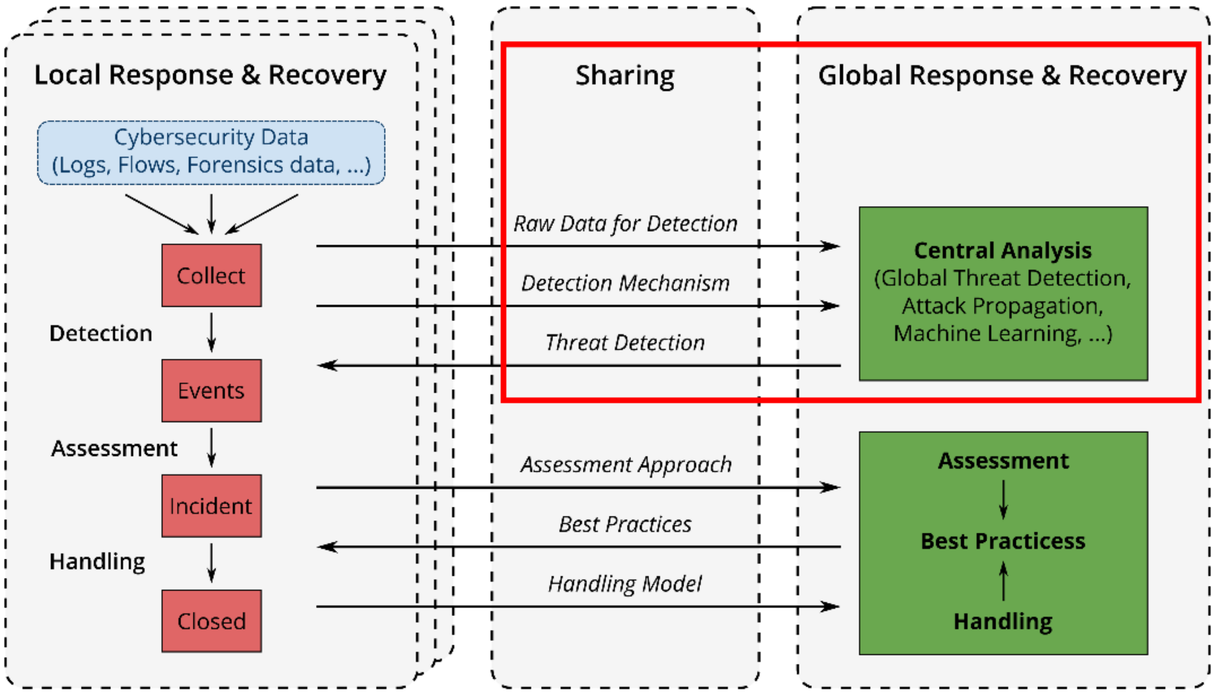

Fig. 1: SAPPAN scheme regarding local and global response and recovery

The overall scheme for sharing, detection, and response in SAPPAN is shown in Fig. 1. The top half of the scheme describes the detection components, and the bottom half the response components, while the left half corresponds to the local level, and the right half to the global level. The goal of WP5 is to implement the global level with respect to sharing of data and models, building global models for detection, sharing of response and recovery information, as well as supporting visualisation. The Tasks T5.1, T5.2 and T5.3 include the development of global models for detection, based on several approaches. The general idea is to utilise the data and models developed in the Task T3.3 on the local level by sharing them among multiple organisations to build global detection models. The goal is to end up with global detection mechanisms that are superior to the local ones. Another flavour of sharing in the scope of SAPPAN is sharing among a cybersecurity service provider or vendor and various user groups of its customer organisations. In such cases, we aggregate data or local attack detection models built in individual endpoints. 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 anonymised data, to sharing of only pre-trained models, to replacing sharing by 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 while providing certain levels of privacy and efficiency, including cost-efficiency. For that, we will use anonymisation techniques developed in Task T3.4 based on the privacy requirements described in the Task T2.2.

For Task T5.1, we develop approaches for building global models based on shared anonymised data. The first challenge is to anonymise the data in a way that eliminates all privacy risks but still preserves enough information to build a useful global model. The second goal is to use the shared data from multiple organisations to create a model superior to the local ones, e.g., with increased accuracy.

3 Distributed Learning of a Global Model Based on Shared Anonymised Data

This deliverable focuses on the creation of a global detection model based on shared anonymised data in order to centralize the detection that is applied at the local level (WP3) to the global level (WP5). To this end, data from several organisations is combined to detect attacks that are not detectable in a single organisation alone, while respecting the privacy requirements of individuals and organisations. Leveraging shared local knowledge enables the identification of common attack patterns and properties on a global level. This gained knowledge enables us to reduce the false positive ratio by creating fitting detection models. In D5.2, we share anonymised data which is the most simple and general form of knowledge distribution compared to the deliverables D5.4 (Global Model based on Shared Local Models) and D5.6 (Global Model without Sharing Local Models) which require trained machine learning models for either sharing the model itself or for sharing intelligence derived from a model (e.g., model predictions in the teacher-student setting or weight updates during federated learning). Thus, in this deliverable we are able to leverage anonymised data sharing in order to increase the performance of various detection types, not just those based on traditional machine learning. In order to investigate the benefit of privacy-preserving sharing of information we make use of the following three use cases which were already defined in D3.4, D3.5, and D5.1 (Algorithms for Analysis of Cybersecurity Data):

- Domain Generation Algorithm (DGA) Detection
- Application Profiling
- Anomalous Login Activity Detection

As any kind of sharing activity may create an attack surface that could allow an attacker to retrieve private information from 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 (D5.1) the landscape of attacks on privacy for machine learning was summarised in a brief but formal overview. We identify the Data Inference class as the relevant attack class for the sharing scenario in this deliverable. In order to avoid further repetition, we refer the reader to Deliverable D5.1 for details about the machine learning privacy attack landscape.

3.1 Domain Generation Algorithm (DGA) Detection

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 anonymised 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 caused by data sharing for this use case in all three deliverables. As a consequence, the three deliverables share the same texts for sections that include general information such as background on DGA detection, state-of-the-art classifiers, or parts of the evaluation setup. However, sections that depend on the different sharing scenarios, such as the actual evaluation and the privacy study, are specific to 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. However, since the following information is essential to understand the evaluation and the privacy analysis, we shortly repeat the most important parts.

Note, 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" [22]. Therefore, parts of the following sections were previously published in and adapted from [22].

3.1.1 Background

Modern botnets rely on DGAs to establish a connection to their command and control (C2) server. In contrast to using individual fixed IP-addresses or fixed domain names, the communication attempt of DGA-based malware is harder to block as such a malware generates a vast amount 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 the 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 allow 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 to not only detect malicious network activities but also to attribute the malicious AGDs to the specific DGAs that generated the domain names. This enables the malware family used to be narrowed down and targeted remediation measures to be taken. 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. [1, 2, 3, 4, 5, 6]), as they entirely rely on information that can be extracted from a single domain name for classification. Thereby, they are less resource intensive and less privacy invasive than context-aware approaches (e.g. [7, 8, 9, 10, 11, 12]) that depend on the extensive tracking of DNS traffic. Even though the classification of the contextless approaches relies solely on the domain name, they are able to compete with the context-aware approaches and achieve state-of-the-art performance [1, 2, 4, 5, 6].

A variety of different types of machine learning techniques have been proposed for the classification of domain names which can be divided into two groups: feature-based classifiers (e.g. [2, 7]) and deep learning (featureless) classifiers (e.g. [1, 4, 5, 6]). While the deep learning classifiers outperform the feature-based approaches in terms of classification performance [4, 5, 13, 14, 15], 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 using domain knowledge. The engineering of these features requires much more effort compared to the usage of deep learning classifiers where all important information has to be encoded and provided to the model. Moreover, after the feature engineering, the best combination of 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 might 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 classifiers which were developed within the SAPPAN project. In detail, we include the two ResNet-based classifiers [16] 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. In order to counteract the explainability problem of deep learning classifiers, we have developed a visual analytics system [AD17] in SAPPAN, which tries to bridge the gap between the predictions of deep neural networks and human understandable features. The results are presented in Deliverables D3.8 and D3.9.

3.1.2 Selected State-of-the-Art Classifiers

In the following, we present several state-of-the-art classifiers which we use in different sharing scenarios 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).

FANCI

Schüppen et al. [2] 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 that is to be classified. Thus, FANCI works completely contextless. FANCI does not incorporate DGA multiclass classification support.

Endgame

Woodbridge et al. [5] proposed two RNN-based classifiers for 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 DGA families are present. We denote the binary classifier by B-Endgame and the multiclass classifier by M-Endgame in the following.

NYU

Yu et al. [6] proposed a DGA binary classifier that is based on two stacked one-dimensional convolutional layers with 128 filters for DGA binary classification. 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 [18] as an optimisation 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.

ResNet

In the context of SAPPAN we developed a binary and a multiclass DGA classifier based on ResNets [16]. We presented all 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 layers unaltered during the training of a classifier and thereby effectively mitigate the vanishing gradient problem [19, 20]. 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.

Class weighting

Tran et al. [4] showed that the model of Woodbridge et al. [5] 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 denotes how much the dataset should be rebalanced. Setting γ = 0 makes the model behave cost-insensitive, 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”.

3.1.3 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 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.

Malicious data

We obtain malicious domains from the open-source intelligence feed of DGArchive [21] which contains more than 126 million unique domains generated by 95 different known DGAs. We make use of all available data up to 2020-09-01.

Benign data

We obtain benign labelled NXDs from four different sources. Three of the sources are networks of project partners, namely CESNET, Masaryk University, and RWTH Aachen University. As real-world benign training data from different 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 far as possible. Additionally, we remove the intersection of all 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 period for data collection overlaps. Note, thereby we are also removing samples from both data sources which would naturally be present in both networks even when they were not interconnected. Such samples could be common typing errors of popular websites. This issue could have an effect on the classification performance of a classifier 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 could only have 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: 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: 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 observed NXDs from the day recording of 2020-06-15. In total, we obtained approximately 362k 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 the USA.

3.1.4 Sharing Approaches

In each deliverable (D5.2, D5.4, D5.6), we investigate different sharing scenarios for the use case of DGA detection. In the initial version of this deliverable, we envisioned three different sharing scenarios:

3.1.4.1 Sharing of anonymised domain names using the URL generalization technique developed in D3.6

As a part of Deliverable D3.6 (Cybersecurity Data Abstractions - Initial Version), a Python commandline tool has been developed to transform URLs into abstracted pseudo-URLs. These resulting abstractions of URLs do not contain privacy-critical information anymore, but are still suitable to be used as training data for machine learning, as shown by the results of experiments that have been presented in D3.6. The scope of D3.6, however, was set to sharing of trained classifiers. For an attacker, this means that there are two complicating factors for the extraction of privacy-critical information: The abstraction of the URLs and the extraction of training data from the classifier. Since this deliverable considers sharing of training data, the second factor does not apply anymore. We consequently solely rely on the abstraction of training data.

3.1.4.2 Sharing of extracted features of feature-based approach FANCI

In this scenario, we utilise FANCI's feature extraction method as a form of data anonymisation. This type of data anonymisation will have no impact on the classification performance of feature-based classifiers compared to the models which were trained using non-anonymised data (i.e., domain names in cleartext). Other anonymisation techniques might have a negative impact on the classification performance. However, it is unclear whether feature extraction is an appropriate approach for data anonymisation. Thus, in this deliverable we perform an extensive study investigating the privacy implications of using this approach.

3.1.4.3 Feature Extractor Sharing (sharing of first-n-layers of deep learning classifiers)

Based on the previous scenario, we designed an approach for sharing extracted features in the context of 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 rear layers are used for the actual classification.

In Fig. 2 we display such a layer partition.

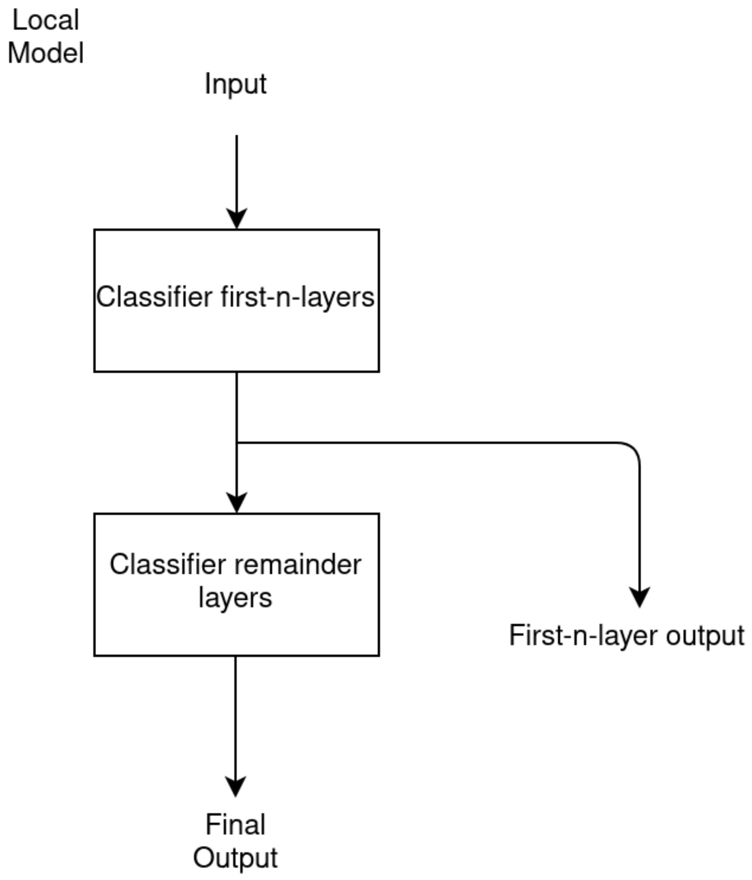

Fig. 2: Partition of a neural network classifier in feature-extractor and classification layers.

Every party that is 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 is able to share either the feature extractor (i.e., the classifier's first-n-layers) or data that is anonymised by feeding domain names to the feature extractor. Depending on which information is shared, different levels of classification performance and privacy guarantees can be achieved. In this deliverable we will investigate which sharing scenario makes the most sense in terms of classification performance and privacy. A possible approach for combining the feature extractors of different neural networks belonging to different parties can be seen in Fig. 3.

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 organisations freeze the weights of the feature extractors and use local training data to train the classification layer separately.

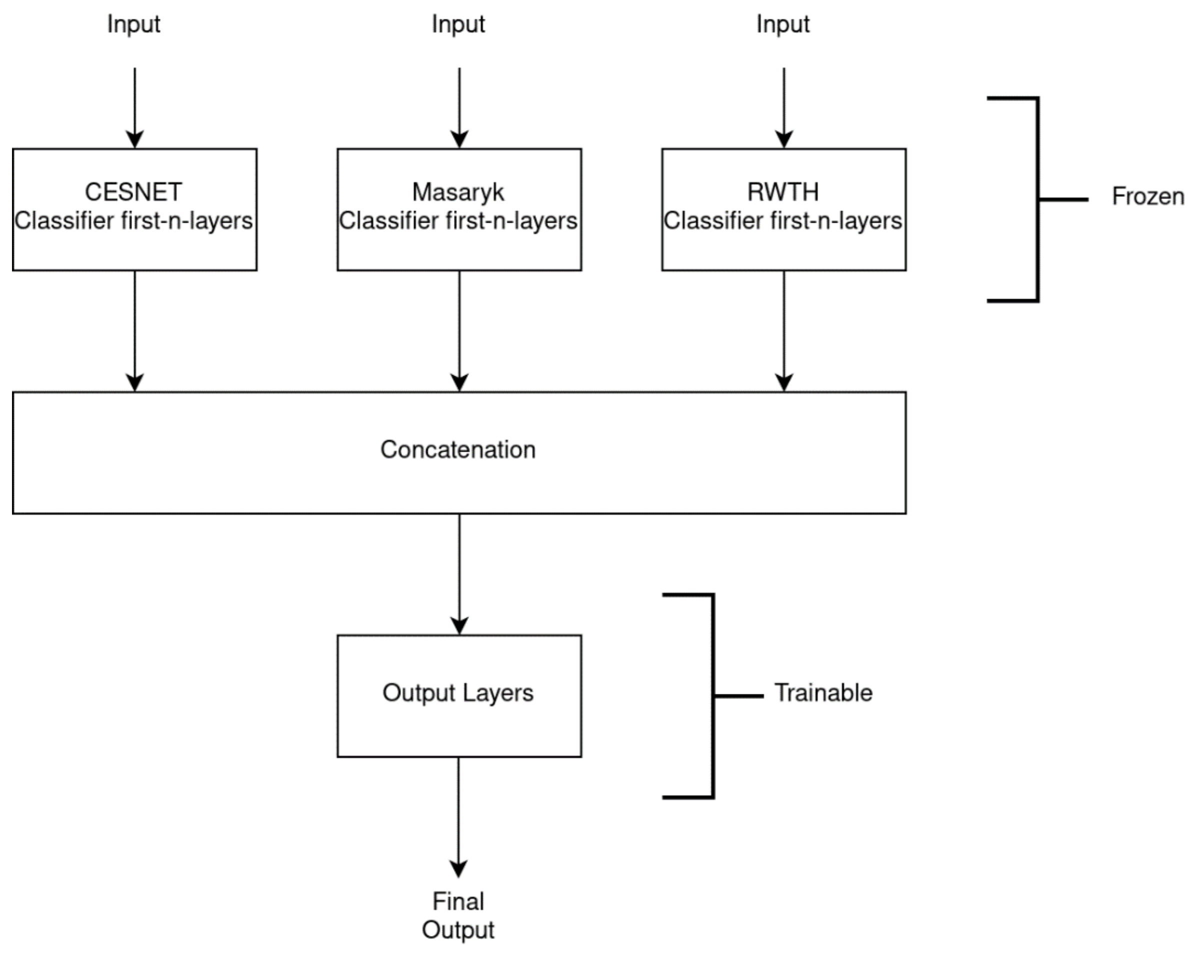

Fig. 3: Approach for combining feature extractors of different neural networks and parties.

3.1.4.4 **Privacy**

In D3.6 (Cybersecurity Data Abstraction), the set of benign training samples has been identified as the main privacy-critical aspect of this use case. Our binary DGA detection models are trained on non-resolving DNS traffic, i.e., NXD samples, labelled either benign or malicious. The benign NXDs (the samples labelled as benign) are privacy sensitive data, as they must be collected locally at some resolver. The benign samples may disclose sensitive information about (1) end-user browsing history or behaviour, and (2) usage of out-dated / misconfigured software. Further, knowledge of frequently occurring benign NXDs originating from user typos may be leveraged for personalized phishing attacks by an adversary. Additionally, the knowledge of NXDs generated by misconfigured or outdated software could be leveraged by an adversary to attack the organisation's network. Due to these privacy issues, it is obviously not possible to directly share benign NXDs in a collaborative ML setting for DGA detection. Thus, we focus in the following on the sharing of private benign labelled data. Preliminary work on privacy assessment has been conducted in the previous deliverable version which is completed in this document.

3.1.5 Analysis and Discussion of Sharing Scenarios

Experiments have shown that classifiers trained on domain names that have been anonymised using the URL generalization technique developed in D3.6 show worse accuracy and detection performance, and yield higher false positive rates than (1) classifiers that are trained on extracted features of the feature-based approach FANCI and (2) collaborative machine learning approaches developed in D5.4 and D5.6. In contrast, using extracted FANCI features for classifier training results in no loss in classification performance compared to using non-anonymised data (i.e., domain names in cleartext). However, it remains to be determined whether feature extraction is an appropriate approach for data anonymisation. Therefore, in the following, we focus on an extensive study which investigates the privacy implications of using this approach.

During the initial version of this deliverable, we envisioned the usage of feature extractors (first-n-layers) derived from deep learning-based classifiers for anonymising domain names. After analysing this approach, we decided to relocate Feature Extractor sharing to Deliverable D5.4 due to the following reasons:

1. Sharing data anonymised by this approach without sharing the actual feature extraction is insufficient for collaborative learning. Experiments have shown that participants cannot make use of data shared by other organisations as their locally trained models are optimised to different optima. Without sharing the actual feature extractors participants cannot recover any useful intelligence out of the anonymised data.
2. Sharing anonymised data along with the feature extractors is a major privacy breach as parties can train an autoencoder which is able to recover domain names out of the anonymised data with high accuracy.
3. Sharing the actual feature extractors for collaborative machine learning fits naturally better in D5.4 in which trained models are exchanged as feature extractors are de facto parts of trained classifiers.

However, in favour of completeness, we still briefly comment on deep learning feature extractors after our extensive study on the privacy of sharing domain names that are anonymised by FANCI's feature extractor.

In summary, for the privacy preservation of domain name data used for DGA detection we present two approaches: The first encodes the domain names with provable privacy-preserving measures. However, here a non-negligible impact on the detection performance is accompanied. Thus, we focus on the second approach which analyses the privacy capability of the feature representation of an existing feature-based DGA classifier. Thereby, the classification performance of the feature-based approach remains untouched while the privacy guarantee must be quantified.

3.1.6 Sharing FANCI Features: A Privacy Analysis of Feature Extraction for DGA Detection

In the following we present our work that was partially presented in Deliverable D5.1. We focus on finalisation of the relevant privacy threats from the Data Inference attack class, namely (Group) Membership Inference and Reconstruction, which were also presented in Deliverable 5.1.

3.1.6.1 **Group Membership Inference**

The threat of Group Membership has already been partially assessed in Deliverable 5.1. To recite, the goal of this attack is to identify clusters in the feature space that correlate with expected origins of the NXD data. Those origins were described as:

1. End-user typing errors while browsing
2. Misconfigured or out-dated software active in the network
3. Benign DGAs active in the network

Random and PC projections were both used to visualise patterns in the data and results are presented in D5.1 We concluded that the visualisation does not form distinctive groups and that other methods could be leveraged to achieve a better distinction.

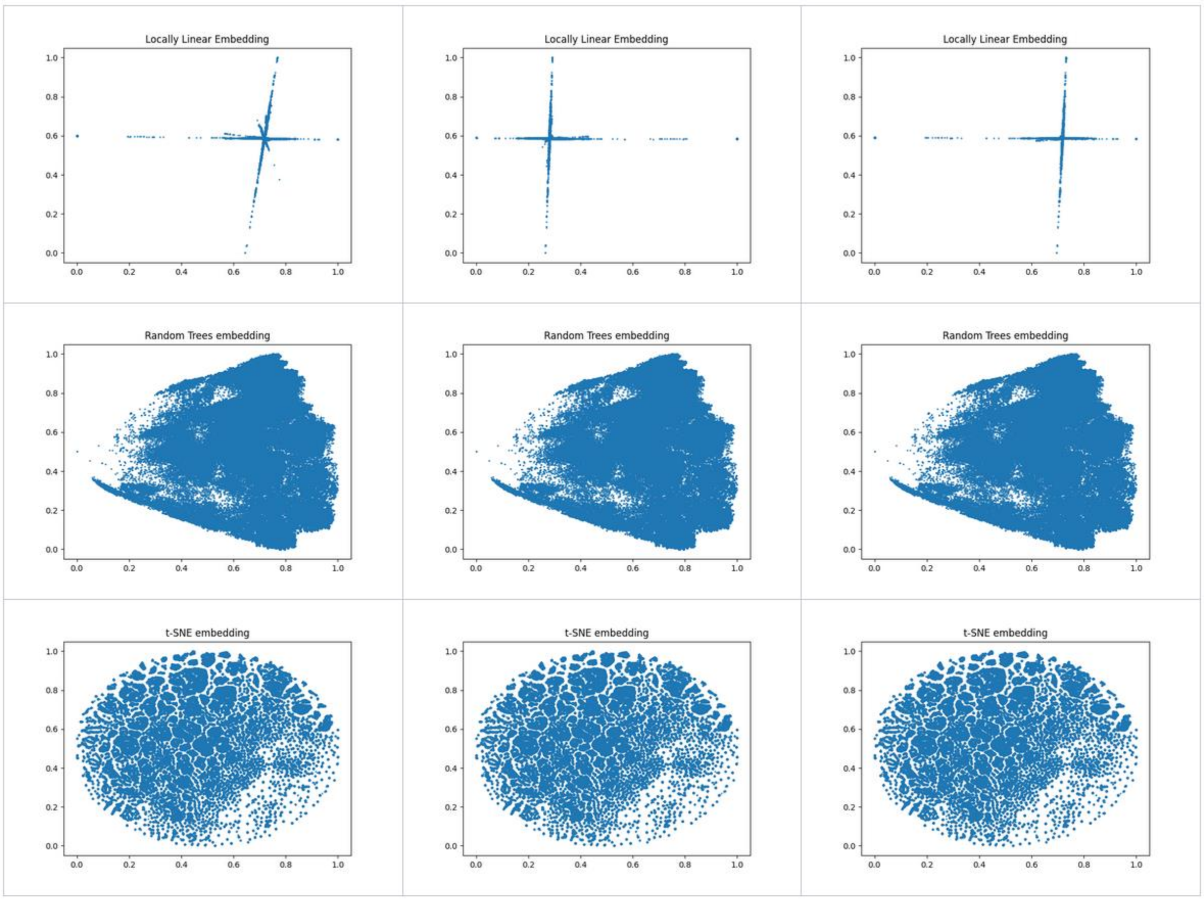

Fig. 4: Embeddings of benign NXD data for different data sets: first column: RWTH, second column: MU, third column: CESNET.

Therefore, we provide results from additional embedding methods, such as locally linear embedding [36], random tree embedding and t-SNE [37] initialised with a PCA embedding, in Fig. 4.

It can be observed that general patterns share similarities independent of the data set on which the embedding was performed. In any case, we cannot identify clear clusters that could correlate the data with the suspected origins listed above. As the data is not labelled by its true origin it is difficult to assess what the semantics of clusters are, e.g., there are many small spherical clusters in the t-SNE embedding. Finally, we can conclude that there is no clear threat against the FANCI feature representation with respect to the inference of the origin group of benign labelled NXD data.

Hence, in this final version of Deliverable 5.2, we focus on concluding the privacy analysis with regard to prevention of Reconstruction attacks.

3.1.6.2 **Privacy Analysis via a Data-Driven Reconstruction Approach**

In a Reconstruction attack, the adversary wishes to infer the original sample from a feature representation, which is why a privacy-preserving representation of the data is required, i.e., a representation that does not allow for a successful reconstruction attack while also retaining high utility for the DGA detection use case.

The aim is to assess the privacy threat associated with a Reconstruction attack with respect to sharing data in the representation of FANCI features. This is performed from two different angles: (1) A manual, handcrafted reconstructions process and (2) a data-driven reconstruction approach that includes training a machine learning model on the reconstruction task. Both assessments are conducted with the same overarching goal: Our main research focus is set on the assessment of whether the data transformation performed by the FANCI feature extractor can also yield a data abstraction that is sufficiently abstract to hide sensitive information. If the analysis results in a positive conclusion, the feature representation of domain names has its practicability towards privacy-preserving intelligence sharing. Concretely, this would allow for risk-free publication of sensitive NXD data in the form of FANCI's feature representation and would thus enable to provide data-privacy in the otherwise privacy-concerning approach of sharing raw data. As research on the feature representation's classification performance is already conducted in the original work [2], we complement it with an investigation on whether FANCI's public feature extractor is prone to malicious inversion. More concretely, we ask whether knowledge of FANCI features threatens the disclosure of the original domain names as these could be reconstructable from feature vectors.

Thereby, we work under the following attack model: We formally refer to the feature extractor as a function $E: S \rightarrow F$ mapping domains from S to the feature space F, where S is the set of strings over the 39-character alphabet with lengths up to 253 (in accordance with RFC 1035 [40]). The context in which the following experiment is conducted is defined by the following aspects:

1. We assume an adversary that is interested in learning the real inputs to the FANCI feature extractor $E: S \rightarrow F$ for a foreign feature set $E(S') = F' \subset F$ of a target's domain name data set S', i.e., for any $E(s) = f \in F$, the adversary aims to find a corresponding $s' \in S$ such that $E(s') = f$ holds and some closeness $s' \approx s$ is satisfied (Note, that finding just any $s' \in S$ with $E(s') = f$ is trivial).
2. The adversary is semi-honest, i.e., he reliably participates in any sharing scenario through which he acquires the foreign feature set.
3. The feature extractor is public knowledge.
4. We only consider the disclosure of benign NXDs as privacy critical.
5. We assume feature sets are shared in the clear, hence no interaction with the target is required.

6. We allow the adversary to be in possession of an arbitrary large data set $S'' \subset S$ of benign NXDs that does not intersect with the target's data, i.e., $S'' \cap S' = \emptyset$.
7. We do not restrict the adversary's computational power that he may apply to his own data. Hence, we allow the adversary to train an ML model.

Aspects 6. and 7. are of course only relevant to the second assessment, the data-driven reconstruction approach.

The first assessment, the manual approach of reconstructing the feature extractor process, has already been presented in the previous version of this deliverable. There, we demonstrated the approach of manually combining feature values to gain more knowledge of a feature vector's pre-image. This approach ultimately fails due to information about the character frequency being lost in the feature extraction process, i.e., the information compression in FANCI's feature extractor is lossy enough to impede a reliable reconstruction on the basis of a single sample. We quantify the compression rate as a quotient of the sizes of the domain and codomain spaces of the feature extraction function. Under the assumption that the elements of the codomain are uniformly distributed among the elements of the function's domain, we concluded that, with $|S|=3.55 \cdot 10^{402}$ and $|F|= 2.71 \cdot 10^{65}$, on average $|S|/|F| \approx 10^{337}$ many pre-images, i.e., domain names, exist for every feature vector [39]. In reality however, elements of codomain (feature vectors) are not uniformly distributed. The distribution is better described by representative data. Subsequent to the manual approach, these insights motivate to analyse the effectiveness of a reconstruction guided by such representative data. Hence, in this deliverable the focus lies on such a data-driven reconstruction approach.

In such an approach, the attacker formally has the goal to train a machine learning model on the task of domain sample reconstruction (i.e., learning an inverse mapping $E^{-1}|_R$: $F \rightarrow S$ on subset R) using an attack set of benign NXDs and their corresponding feature vectors $\{(f_i, s_i)_{i=0,\dots,n}\} \subset F \times S$. This is a realistic scenario in any sharing use case where a party receiving a feature set may also be in possession of an own data set of benign NXDs (exactly as described in the attack model). In principle, the feature extraction process is assumed to be unknown to the model, and we let it learn the inverse mapping without any domain-specific assistance.

For the evaluation of such a data-driven reconstruction, however, we so far only presented an idea for an experimental evaluation framework in the last version, which included a proposition for a quantitative and normalized metric that measures the success of a reconstruction attack. In the meantime, we have already concluded and disseminated our work on the privacy analysis, as proposed in Deliverable 5.1, in an accepted conference paper [39]. Here we would thus like to present results from the paper, while first detailing updates and extensions to the previously described evaluation setup and evaluation methodology in a concise manner. Therefore, we use passages from our disseminated work [39].

Setup & Methodology

Essential aspects of the feature extractor are briefly presented in the following: We utilise the most recent open-source implementation of FANCI's feature extractor [39], which extracts 15 structural, 8 linguistic, and 22 statistical features from domain names. For completeness, a listing of the features is again provided in Table 1. The extraction process recognises 39 unique characters (letters a-z, digits 0-9 and special characters dot, hyphen and underscore).

Feature	Feature Name	Data Type	Normalization Factor
0	length	int	253
1	1_part	bool	1
2	2_part	bool	1
3	3_part	bool	1
4	4_part	bool	1
5	vowel_ratio	float	1
6	digit_ratio	float	1
7	contains_ipv4_addr	bool	1
8	contains_digits	bool	1
9	has_valid_tld	bool	1
10	contains_one_char_subdomains	bool	1
11	contains_wwwdot	bool	1
12	subdomain_lengths_mean	float	253
13	prefix_repetition	bool	1
14	char_diversity	float	1
15	contains_subdomain_of_only_digits	bool	1
16	contains_tld_as_infix	bool	1
17	1_gram_std	float	1_gram_max
18	1_gram_median	float	1_gram_max
19	1_gram_mean	float	1_gram_max
20	1_gram_min	float	1_gram_max
21	1_gram_max	float	1_gram_max
22	1_gram_perc_25	float	1_gram_max
23	1_gram_perc_75	float	1_gram_max

24	2_gram_std	float	2_gram_max
25	2_gram_median	float	2_gram_max
26	2_gram_mean	float	2_gram_max
27	2_gram_min	float	2_gram_max
28	2_gram_max	float	2_gram_max
29	2_gram_perc_25	float	2_gram_max
30	2_gram_perc_75	float	2_gram_max
31	3_gram_std	float	3_gram_max
32	3_gram_median	float	3_gram_max
33	3_gram_mean	float	3_gram_max
34	3_gram_min	float	3_gram_max
35	3_gram_max	float	3_gram_max
36	3_gram_perc_25	float	3_gram_max
37	3_gram_perc_75	float	3_gram_max
38	hex_part_ratio	float	1
39	underscore_ratio	float	1
40	alphabet_size	int	38
41	shannon_entropy	float	$\log_2$(alphabet_size)
42	ratio_of_repeated_chars	float	1
43	consecutive_consonant_ratio	float	1
44	consecutive_digits_ratio	float	1

Table 1: Features extracted by the open source implementation of the FANCI feature extractor [41]. The normalisation factor is the value used to normalise the feature value to the range [0, 1].

As noted in Table 1, the implementation of some features deviates from the definition in the original paper [2]: For instance, feature _contains_ipv4_addr_ should also regard IPv6 addresses. Many of FANCI's features are computed on the Dot-free public-Suffix-Free (DSF) part of the domain which excludes both dot characters and the public valid suffix, which is usually the Top-Level-Domain (TLD). The validity of a suffix (valid_tld) is determined by checking against a predefined list that is included in the feature extractor.

Reconstruction Quantification

In Deliverable 5.1 we presented existing members from the family of edit distances on the string space as candidates to compare pairs of original and reconstructed samples. We find that the Damerau-Levenshtein is best suited for this use case, as variable-length samples are compared. Damerau-Levenshtein computes a minimum-change distance via the number of character edit operations (substitutions, insertions, deletions or transposition of adjacent characters) required to transform one input string into the other. Further, a normalised version for the metric is computed by dividing the resulting minimum-change-distance by the length of the longest of both input strings. As the normalised metric is a ratio of edit-operations to string length, it can be interpreted as a lower bound on the percentage of misplaced characters in the reconstruction. Thereby, the normalised metric expresses a reconstruction error (relative to the string length). Note. that this division operation does not threaten the metric's triangle inequality. In fact, all axioms of the original Damerau-Levenshtein metric are retained.

Consequently, if the metric value equals zero, then this indicates equality, while dissimilarity grows in parallel to larger metric values. Attack success can now be measured by an aggregation, e.g., the average, of the per-sample reconstruction errors over an NXD data set used for evaluation. It is important to note that the suggested normalised metric solely operates on a syntactical level. Human readable alternative spellings, as for instance leetspeak, are not considered by this syntactical comparison. However, construction of a quantification that regards semantic differences between samples requires either a data-driven approach which comes with risk of overfitting and other biases, or a sufficient extent of work from the area of psychology. Hence, the syntactical metric will serve as a sufficient indicator of attack success.

Evaluation Loop Setup

In the following experiment, we assess the reconstruction performance of trained Deep Learning reconstruction models using the above-mentioned distance metrics. More concretely, we train a model for each one of the benign data sources and evaluate each of these models against all the data sources including the one on which the individual model was trained on. For each pair of training and evaluation sets we average each metric's scores over all samples (see Fig. 5). Thereby, we assess the models' capability to reconstruct domains from foreign feature sets.

The three real-world benign NXD data sets are used as representative privacy sensitive data for this experiment. We use each set to train an attack model and evaluate each attack model against all three data sets resulting in 9 evaluations.

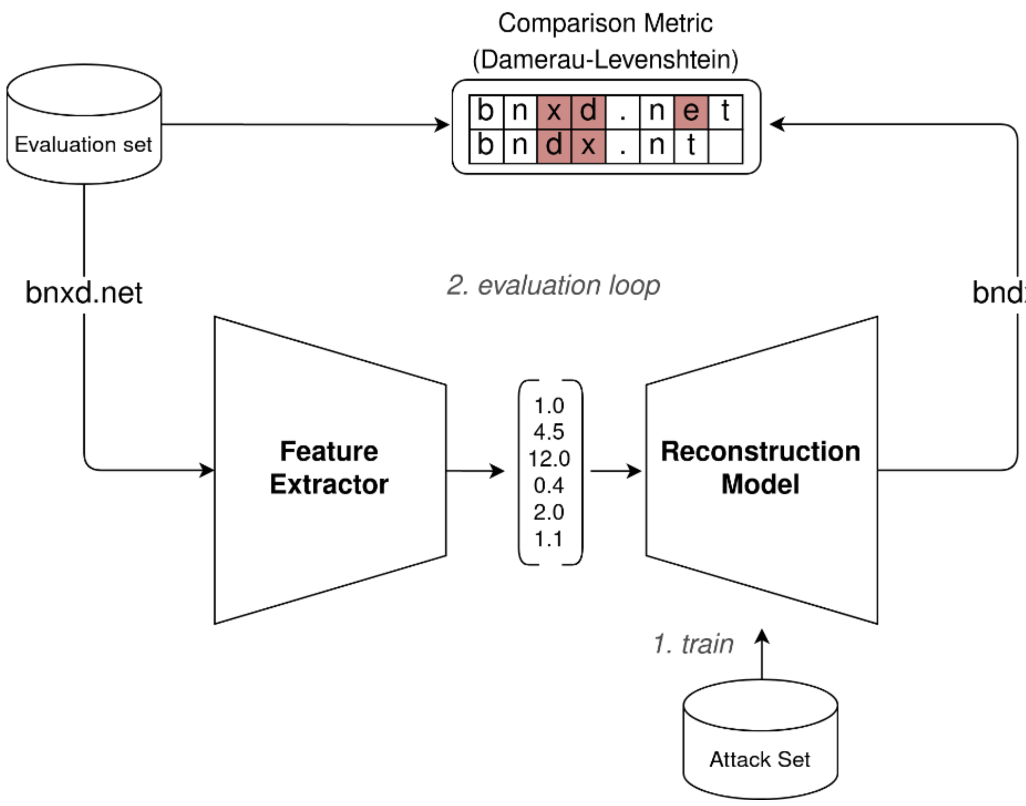

Fig. 5: Evaluation Loop Setup: First a reconstruction model is trained on an attack set, then the reconstruction model is evaluated over an evaluation set. Original and reconstructed domain names are compared via the Damerau-Levenshtein distance.

Benign NX Data Records

True pre-image distributions and domain-feature relations are best described by representative data sets. To provide significance to our results, we make use of three large real-world NXD sets. We leverage these data sets two-fold: (1) We use this data to train a machine learning model on the reconstruction task. (2) The data is used as ground truth to assess the reconstruction capability of the trained reconstruction models. We use the same DNS recordings already utilised in other deliverables so far. Concretely, the benign data sourced by RWTH Aachen University, Masaryk University and CESNET (as presented above) are used. We use the complete record of CESNET and randomly sub-sample a set in the size of that record from each of the other two institutions' records. Note, we do not use any samples from Siemens AG as we obtained the samples after we have started the privacy analysis.

For the evaluation, the data is prepared as follows: We flatten the representation of the feature vector, e.g., for each feature that is represented as vectors (such as small number_of_subdomains or the one-, two-, and three-gram distribution vectors), we view each entry as a single feature. Thereby, our feature count differs slightly from the one presented in the original work [2] and we end up with 45-component feature vectors. All entries in a FANCI feature vector are in some finitely bounded range of the non-negative rationales and are normalised to the range of [0,1] by dividing each entry by the upper bound of its value range (see last column of Table 1). For model training, each data set is prepared as follows: The test set is a random 20% split of the total data. Another random 5% split of the remaining training data is used as validation set.

Domain name strings are encoded to character sequences with start and end markers. This is done to encode the bounds of the target output sequences during training, i.e., the token-wise decoding process begins with a start marker and can be stopped once the end marker is encountered or predicted.

Model Architecture

To realise a model that can fulfil the reconstruction task, we leverage a model architecture from the Sequence-to-Sequence learning (Seq2Seq) area in machine learning. Seq2Seq encompasses encoder-decoder models that solve machine learning tasks related to mapping variable-length input sequences to variable-length output sequences [40, 41]. Both the encoder and decoder of a Seq2Seq architecture utilise recurrent units to process the variable-length sequences and work together as follows: The encoder consumes and compresses the input sequence to a fixed-length state while the decoder is trained to create the target sequence from this state. To use such a model, the domain name strings must be transformed into character sequences with corresponding start and end markers.

At test time a sequence can be sampled from the decoder of a trained Seq2Seq model, i.e., beginning with the start marker, the model iteratively predicts the next character with the currently sampled prefix as prior. This sampling technique is commonly referred to as closed-loop, since the predicted characters are fed back into the model at each step. Seq2Seq approaches have proven to be very successful in use cases such as machine translation. The reconstruction of a variable-length domain sample from a fixed-length feature vector requires a model that performs like a decoder in a Seq2Seq setup. All models share the following architecture (depicted in Fig. 6) whose design follows a related approach [43].

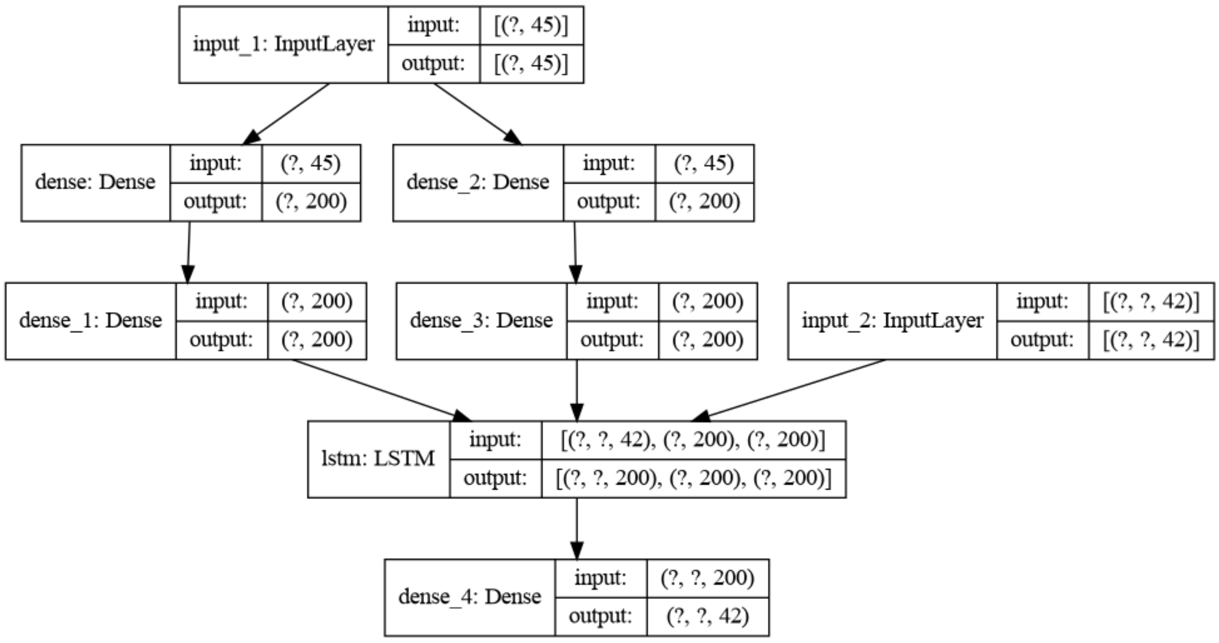
input_1: InputLayer | input: [(?, 45)] | output: [(?, 45)]
dense: Dense | input: (?, 45) | output: (?, 200)
dense_2: Dense | input: (?, 45) | output: (?, 200)
dense_1: Dense | input: (?, 200) | output: (?, 200)
dense_3: Dense | input: (?, 200) | output: (?, 200)
input_2: InputLayer | input: [(?, ?, 42)] | output: [(?, ?, 42)]
lstm: LSTM | input: [(?, ?, 42), (?, 200), (?, 200)] | output: [(?, ?, 200), (?, 200), (?, 200)]
dense_4: Dense | input: (?, ?, 200) | output: (?, ?, 42)

Fig. 6: Model architecture of the Seq2Seq2 style decoder for reconstruction.

The architecture begins with two parallel sequences of two dense layers with 200 units each. This leaves opportunity for the model to manipulate the representation of the input feature vector before the two outputs are used as the initial states for the recurrent unit in the decoder. For the recurrent unit, a single Long Short-Term Memory (LSTM) layer with 200 units is used. The model ends with a dense layer of size 42 and a soft-max activation to output a prediction vector over all relevant characters. The output encodes the 39 recognized domain characters plus start, end and empty markers used internally for the character sequence encoding of domain name strings. Finally, the total architecture comprises 301,642 trainable weights.

Training Setup

A batch size of 64 is fixed in our experiments for a good balance between training time and model performance. Models are trained using the cross-entropy loss to penalize wrong character predictions. A superordinated focal loss, which dynamically down-weighs well-classified samples in the cross-entropy loss during training, is used to counter a potential unbalance-bias [44]. The training process of the decoder model is steered by the RMSprop optimiser with parameters learning_rate=0.001, rho=0.9, momentum=0.0 and epsilon=1e-07. Each model is trained for at most 1000 epochs, while the training data is shuffled after each epoch and training is stopped early whenever 10 consecutive epochs without improvement of the validation loss are exceeded.

We employ Teacher Forcing to train the decoder, which is the common training methodology in Seq2Seq learning [45]. This method essentially sets the input of the decoder to the target sequence shifted by one time step (open loop) instead of feeding the decoder's predictions of previous time steps back into the model (closed loop). Thereby the model predicts each next character based on the correct prior character sequence and not the predicted one.

Reconstruction Results

Network Data Source		**Averaged Reconstruction Performance**	
Training Data	**Evaluation Data**	**Damerau-Levenshtein**	**normalized Damerau-Levenshtein**
RWTH	RWTH	47.85	0.51
	MU	23.22	0.72
	CESNET	20.61	0.66
MU	RWTH	72.94	0.75
	MU	15.00	0.53
	CESNET	18.61	0.61
CESNET	RWTH	62.07	0.67
	MU	20.01	0.65
	CESNET	13.66	0.46

Table 2: Averaged closed-loop reconstruction performance of trained reconstruction models.

Averaged closed-loop reconstruction performance for all combinations of trained models and evaluation sets are given in Table 2.

Baseline Reconstruction Performance

The baseline cases are what we refer to when the evaluation data is equal to all the data used to train, validate and test the model, and is to be interpreted as baseline reconstruction performance of a trained model.
Baseline cases are rows in the table with highlighted data sets.

Although the models achieve a significant reduction in training and test loss during model training, the reconstruction performance is not very good: For RWTH, MU and the CESNET data sets we measure that on average respectively 47.85, 15.00 and 13.66 character edit operations separate each original domain and its reconstruction. The normalised version of the metric measures an average error for RWTH and MU that is just larger than 0.5, i.e., on average at least 50% of characters in each reconstruction are incorrectly predicted. For CESNET, we measure a slightly smaller average score of 0.45. In general, it seems that the models' baseline reconstruction performance is similarly bad on all data sets.

Transferability

The trained models' reconstruction performance on data from foreign networks can be found in the remaining lines in Table 2. These are the transferability cases, i.e., exactly the scenario which we describe in our attack model.
In the transferability cases the reconstruction error is higher than in the baseline cases (score higher than 0.65) with the worst performance (0.75) in the case where the model trained on data from RWTH is evaluated on that of MU.

Discussion on Reconstruction Results

A discussion that reasons about the cause of such bad reconstruction performance is also added at this point. The mathematical review of the feature extraction process in the manual analysis highlighted that it is plausible that the overall reconstruction performance is poor: After all, FANCI's feature extractor considers only very few features and thereby has a very high compression rate. While this might be tolerable for good classification performance, it seems to hinder good reconstruction quality. Arguments towards a quantifiable proof that extractor function E is not injective when restricted to the subspace of real-world benign NXDs is presented in the following.

Feature Space Overlap

While in our experiments the adversary and target NXD data sets are disjunct, the corresponding sets of feature vectors of each respective data set must not be disjunct sets. A large overlap in the feature space most certainly leads to a degraded reconstruction performance, as for the same feature vector the model may learn to reconstruct a domain different from the one the adversary wants to sample at test time. Hence, a quantification of the overlap in feature space for the three data sets used in this study is presented in Table 3.

Network Data Source		Feature Space Overlap				
Training Data	**Evaluation Data**	**#unique Feature Vectors in training data**	**Training ∩ Evaluation**	**% of Evaluation Data**	**#unique Feature Vectors in total data**	**% of total data**
RWTH	RWTH	288118	-	-	3462	10.3
	MU		5789	32.9		
	CESNET		4809	12.1		
MU	RWTH	182786	5789	11.5		30.7
	MU		-	-		
	CESNET		12985	41.8		
CESNET	RWTH	169921	4809	24.6		22.9
	MU		12985	30.9		
	CESNET		-	-		

Table 3: Feature space overlap of the three training data sets. Right hand side lists the unique feature vectors per set or intersection of sets and the corresponding relative share that this intersection makes up of the evaluation or total data.

Although every data set contains approximately 362k unique samples, the amount of unique feature vectors is significantly lower which clearly indicates collisions in the feature space. Additionally, for every combination of two distinct NXD data sets we have an intersection of non-trivial size in the feature space, e.g., 11.5% of MU's data intersects with RWTH's and 41.8% with that of CESNET. Actually, the worst-performing baseline and transferability reconstructions (training data sourced at MU) coincides with the largest feature space overlap with regard to all data sets (see Table 3).

Top 10% Reconstructions

We would like to estimate what the adversary's theoretical information gain for well-reconstructed domains is. In reality, the adversary has no clear way of estimating the confidence of a single reconstruction without ground truth. In our experimental setting we can however isolate and inspect the well-reconstructed domain names. Hence, we take a closer look at the best 10% of all reconstructions for the transferability cases.

The average reconstruction performance for the top 10% lies at 0.276. Further, approximately 45-55% of the top 10% performers are occupied by IPv4 and IPv6 reverse DNS lookups and 20-35% by spam-related or other DNS-related services, e.g., DNS blacklists. We find the models perform well in reconstructing these types of domains because (1) these domains' contents are well-structured, (2) these domains often share a large suffix, and (3) they stand out by containing a lot of numerical characters. Thus, as they occupy sparse areas of the feature space around features such as a high digit_ratio, or low subdomains_lengths_mean, or a True value for features such as only_digits_subdomains and contains_ipv4_addr. However, these NXDs do not necessarily originate from user typos but rather from misconfigured software, which would also better explain the high occurrence of these types of domains in the data sets.

We claim that knowledge of reverse look-ups and usage of spam-services is not privacy-sensitive information. After all, these domains do not reveal any information about end-user browsing or sensitive tooling usage in the network from which the data was sourced.

Conclusion on Reconstruction Threat Assessment

By emulating the logical approach, a data-rich adversary would pursue for reconstruction, namely training a machine learning model to learn a reconstruction mapping, we facilitate a quantification of the extent that the feature representation of FANCI discloses any sensitive information about the original domain names. Analysis shows that on average at least half of all characters from a reconstructed domain are already misplaced in the baseline cases. Also, the models only perform well on foreign network's data for reverse lookups and other not-sensitive NXDs likely originating from misconfigured software. Consequently, our results suggest that a machine learning model aiding in the attack cannot reliably reconstruct NXDs from foreign networks' FANCI feature vectors which would be, however, the main use case in an attack. Therefore, FANCI's feature representation does not constitute a considerable privacy threat and is well-suited to be used as an alternative representation in sharing benign NXD data.

3.1.7 Analysis of Deep Learning Feature Extractors

In the previous version of this deliverable, we announced that the privacy capabilities of a feature extractor with a deep learning architecture could be analysed in a similar manner. We displace this into the final version of Deliverable D5.4 for the following reason: Sharing an alternative representation of data requires making the data transformation process public. While our previously described analysis highlights privacy-preservation for an extractor of handcrafted features, a deep learning feature extractor must again be trained on data. To share data in the feature representation of a deep learning extractor requires that either (1) one party provides the extractor model or (2) the extractor model must be agreed on with all sharing participants. The former solution corresponds to the setting we have analysed for the feature-based feature extractor, yet in this case a model with a differentiable function is shared that may be misused by an adversary when parties share the data in feature representation later on. Similar privacy threats hold for the latter points in which all parties need to agree on a deep learning extractor. However, this also implies that the deep learning extractor is actually a global model trained without sharing of local data or models, which is exactly the goal of T5.3. Thereby, results of T5.3 would be required to perform preliminary work necessary to accomplish data sharing in T5.1.

For Task T5.2, however, which involves building a global model by sharing local models, we investigate an approach that leverages shared trained deep learning feature extractors. Hence, a privacy analysis or discussion of such feature extractors has more relevance in that scenario and is therefore included in Deliverable D5.4.

3.1.8 EXPLAIN: Feature-based DGA Multiclass Detection Approach

Since our privacy analysis of the feature-based approach FANCI yielded promising results, we developed a contextless and feature-based approach for DGA multiclass classification. To the best of our knowledge this is the first contextless and feature-based multiclass classifier for DGA attribution. The following sections were previously published in and adapted from "First Step Towards EXPLAINable DGA Multiclass Classification" [23].

3.1.8.1 **Introduction**

Previously, for the DGA multiclass classification task, only deep learning based contextless classifiers were published. While these classifiers achieve a very promising performance, deep learning classifiers are often said to lead to less well explainable predictions compared to feature-based classifiers. In the following, we present EXPLAIN [23], the feature-based DGA multiclass classifier developed in SAPPAN. By design our feature-based approach is more explainable compared to the deep learning classifiers proposed in related work as predictions can be traced back to the characteristics of the used features. Thus, beside the usage of the feature extractor for data anonymisation, an additional benefit of using a feature-based approach is explainability.

We describe the engineering of our classifier in detail for the sake of transparency, which is a major requirement for explainability. Without a transparent feature engineering and selection process a feature-based classifier operates similar to a black box. The main target user groups for our classifier are security operation centre (SOC) analysts and model developers. SOC analysts potentially benefit from the use of EXPLAIN over deep learning approaches when analysing predictions in search for potential false positives or false negatives. Here, EXPLAIN enables the analysis of the features that contributed to the classification result. Model developers, on the other hand, heavily profit from the adaptability of our approach. For instance, features can be adjusted or new discriminating features can be engineered in order to enable the correct attribution of newly discovered DGAs.

The source code of the classifier, all specific values for each hyperparameter and for every developed model, as well as the full list of investigated features including a description for each individual feature, are publicly available, in order to support more research in this area in the future [24].

3.1.8.2 **Evaluation Overview**

To be able to assess the performance of the developed classifier afterwards, we perform a comparative evaluation. In the following, we provide an overview of the used classifiers, data sets as well as our experimental setup which includes our used evaluation methodology.

Selected State-of-the-Art Classifiers for Comparative Evaluation

Next to the previously presented deep learning classifiers (Endgame, NYU, and ResNet) we also evaluate the naïve approach of adapting FANCI with its original features developed for the binary task to a multiclass classifier. While the RF-based implementation is inherently capable of multiclass classification, the SVM approach requires modifications. By reducing the problem of multiclass classification to multiple binary classification problems it is possible to enable multiclass classification support for the SVM implementation. Here, we either use multiple one-vs.-one (OvO) or one-vs.-rest (OvR) classifiers. We refer to the multiclass enabled SVM approaches as **M-FANCI-SVM-OvO** and **M-FANCI-SVM-OvR** in the following. Additionally, to the multiclass enabled **M-FANCI-RF** model we also evaluate **M-FANCI-RF-OvO** and **M-FANCI-RF-OvR**. Additionally, we include for every chosen deep learning-based classifier also a cost-sensitive variant for our evaluation.

Data Sets

We use the benign samples obtained from RWTH Aachen University and malicious labelled samples from DGArchive to create two distinct data sets, one for feature engineering and selection ($\mathbf{Set}_{\mathbf{Selection}}$), and one for the final comparative evaluation ($\mathbf{Set}_{\mathbf{Evaluation}}$). In order to obtain meaningful results for our feature engineering, feature selection, as well as for the comparative evaluation, we require that for every included DGA in the data set at least 10 unique samples are available. We thus eliminate the samples of Dnsbenchmark and Randomloader, for which only 5 samples per DGA family are known, from our data. Since we aim for diverse data sets, we randomly draw for every remaining DGA in DGArchive at most 20,000 samples. We take all available domain names for DGAs for which less than 20,000 samples are known. Additionally, we draw 20,000 random samples from our source of benign data. Thereafter, we split the selected data evenly across all class labels into two disjoint data sets. The first set, $\mathbf{Set}_{\mathbf{Selection}}$, is used in the context of feature engineering and selection during the development of our proposed classifier. The second set, $\mathbf{Set}_{\mathbf{Evaluation}}$, is only used for the final comparative evaluation. Each of these data sets comprises approximately 500,000 samples.

Experimental Setup

We use the following software packages for our experiments: Python 3.8.5, scikit-learn 0.23.2, TensorFlow 2.3.0, Keras 2.4.0, CUDA 10.1, and cuDNN 7.6.5. All deep learning models are executed on an NVIDIA Tesla V100 GPU while the feature-based approaches are executed on 48 CPU cores of two Intel Xeon Platinum 8160 processors@2.1GHz. Our evaluation is split into two parts. First, we present evaluations conducted during the development of our classifier. We make use of samples included in $\mathrm{Set}_{\mathrm{Selection}}$ in order to engineer and select well performing feature sets. Here, we additionally perform the hyperparameter optimisation. The results of the evaluations are several promising combinations of feature sets and hyperparameters (configurations) which will be analysed subsequently. Thereafter, we compare our best performing classifier configurations with the various state-of-the-art classifiers proposed in related work using the samples included in $\mathrm{Set}_{\mathrm{Evaluation}}$. Additionally, we measure the classifiers' training and classification speed in order to assess their real-time capability. For every evaluation we present, we perform five repetitions of a five-fold cross validation stratified over the included classes within the respective set. Thus, in every fold, the samples of each class are split into 80% training and 20% testing samples. For the deep learning classifiers, we additionally split 5% from the training samples for a holdout set which is used to assess the performance of the classifiers during training. We train all deep learning models as long as they are improving on the holdout set. After five epochs without improvement we stop the training and evaluate the best model on the test samples. In order to evaluate and compare the different classifiers we primarily use the f1-score which is defined as the harmonic mean of the precision and the recall. The precision measures the fraction of true positives among those samples that are labelled as positive by a classifier. The recall, on the other hand, equals the true positive rate and thus measures the proportion of positives that are correctly identified by a classifier. We calculate these metrics for every class included in our evaluation and afterwards average all class scores to assess the overall performance of a classifier. By using this macro-averaging, we value each class with the same level of importance despite the actual number of available samples per class varying.

3.1.8.3 **EXPLAIN**

The engineering of features requires much more effort compared to the usage of deep learning classifiers where all important information has simply to be encoded and provided to the model, then the model learns the relevant features on its own in an end-to-end fashion. Moreover, after the feature engineering the best combination of features has to be selected. The combination of several engineered features might contain mutual information which could render single features useless for the classification. These features should be removed as their extraction from the raw data might require significant processing time which could have a negative impact on the real-time capability of a classifier. For performing the complex process of feature selection, a multitude of feature filtering and ranking techniques (e.g. [25, 26, 27, 28, 29, 30]) have been suggested in the past. Lastly, in the development process of a feature-based classifier a huge amount of hyperparameters has to be optimised. All in all, for a promising feature-based classifier these three steps (i.e. feature engineering, feature selection, and hyperparameter optimisation) have to be performed which is not a trivial task. During the development of EXPLAIN we investigated RFs and SVMs. In our experiments, our RF variants outperformed our SVM approaches in training time as well as in classification performance. We thus focus on the development of an RF-based implementation of a feature-based multiclass DGA classifier in the following.

Feature Engineering & Selection

We study 136 different features which we gathered or adapted from related work (mainly from [2, 7, 10, 21]) and developed by our own through analysing the samples of the different DGAs and benign samples contained in $\text{Set}_{\text{Selection}}$. As we target a contextless classifier due to privacy considerations we only focus on features that can be extracted from a single domain name. We divide the 136 features into 64 linguistic, 17 structural, and 55 statistical features. Note, we only use samples from $\text{Set}_{\text{Selection}}$ for feature engineering and selection. The final comparative evaluation will be performed on samples from the disjunctive set $\text{Set}_{\text{Evaluation.}}$ We provide a list of the developed features including their total amount and a brief description of their purpose in Table 4.

Feature	#	Type	Goal/Purpose
subdomains-digit-sum	1	linguistic	distinguish families with different character distributions
{...}-character-ratio	4	linguistic	distinguish families with different character distributions
alphabet-{...}	37	linguistic	distinguish families with different character distributions and alphabets
adjacent-duplicates-ratio	1	linguistic	distinguish arithmetic- and wordlist-based families from hex-based families
{...}-max-streak-length	6	linguistic	distinguish arithmetic-, hex- and wordlist-based families
first-character-pair	1	linguistic	detect families with constant prefixes (e.g. *Xxhex* with "xx")
syllable-count	1	linguistic	distinguish families with different levels of readability in their AGDs
weighted-steaks	1	linguistic	distinguish same length domains with different positioning and quantity of consonants and decimal digits
inverse-hamming-distance	1	linguistic	for randomness assessment
{...}-digit-edge-distance	2	linguistic	detect domains with digits at constant relative positions and for randomness assessment
suffix-digit-sum	1	linguistic	distinguish families with different sets of utilized suffixes
suffix-standard-deviation	1	linguistic	distinguish families with different sets of utilized suffixes
suffix-length	1	structural	distinguish families with different sets of utilized suffixes
suffix-dns-level	1	structural	distinguish families with different sets of utilized suffixes
second-level-length	1	structural	detect families using fixed lengths for their subdomains
subdomains-length	1	structural	detect families using fixed lengths for their subdomains
second-level-repeated-prefix	1	structural	detect domains originating from misconfigured software
subdomains-contain-suffix	1	structural	detect domains originating from misconfigured software and typing errors
{1, 2, 3}-gram-{...}	15	statistical	distinguish families with different character distributions and for randomness assessment
zlib-bits-compression-ratio	1	statistical	distinguish families with different character distributions and for randomness assessment
bits-entropy	1	statistical	distinguish families with different character distributions and for randomness assessment
{...}-test[-unicode]	14	statistical	distinguish different underlying pseudo-random number generators

Table 4: Newly developed features.

For instance, simple features attempt to distinguish DGA families based on the suffixes used in their generated domains because different families use different sets of suffixes. Other features try to separate DGA families based on character distributions as, for example, wordlist-based families do have different consonant and vowel distribution in contrast to arithmetic- and hex-based families. Additionally, we introduce novel features to discriminate different underlying pseudo-random number generators used by the DGA families for domain generation. A detailed description for each individual feature can be found in the source code.

After feature engineering we perform feature selection to reduce the computational complexity for training and classification of our classifier and to enhance its overall classification performance. A variety of different feature selection methods have been proposed in the past, all having their advantages and disadvantages. In this work, we thus make use of different filter and wrapper methods to determine valuable features as there is no best technique.

Filtering methods leverage proxy measures to assess the importance of features. The main advantages of filter methods are that they are computationally lightweight, scalable, and independent of the underlying learning algorithm. Common measures are the variance, mutual information, chi-square test, ANOVA F-test, and Relief-based algorithms [27]. The variance as well as the mutual information of a variable with a target label can be used as a proxy to measure the amount of information of a feature. Features that contain little information can be filtered out because they contribute only insignificantly to the classification. The chi-square test measures the dependence between a non-negative categorical feature and the target label which can thus be used to remove features that are independent of a class and therefore irrelevant for the classification. In case of numerical features, the ANOVA F-test should be used instead. As we intent to utilise a mix of different categorical and numerical features we do not make use of these filtering techniques. Relief-based algorithms [27] compute an importance score for every feature based on differences in feature values of nearest neighbour instance pairs. The advantages of Relief-based algorithms is that they run in low-order polynomial time, are not sensitive to feature interactions, and are robust against noise. However, their weaknesses are that they do not discriminate redundant features and that they might be fooled by low numbers of training instances. ReliefF [28] is the most commonly used Relief-based algorithm [30]. Another efficient Relief-based algorithm is MultiSURF [30]. It can be used for the sake of simplicity or when computing resources are limited, since there are no execution parameters to be optimised. For our feature selection, we utilise ReliefF and MultiSURF to select the better than average features according to their computed feature importance.

Wrapper methods are, in contrast to filter methods, more computationally intensive as they are not independent of the underlying learning algorithm. For each feature subset a new model is trained and evaluated which allows for finding the best performing feature set for a particular model and evaluation set. A commonly used wrapper method is recursive feature elimination (RFE) [26]. RFE recursively estimates the feature importance based on a feature ranking method and recursively removes the least important feature, starting with the complete feature set. In each iteration of RFE the classification performance of the current feature set is estimated using a hold-out set. Thus, by using RFE it is possible to determine the best feature set for a given model and evaluation set. In this work, we use the cross-validated variant of RFE included in scikit-learn using the Mean Decrease Impurity (MDI) [29] and the Permutation Importance (PI) [25] as feature ranking methods. For every feature, MDI measures the average gain of purity by splits using the corresponding feature within the trees in the forest. The advantage of MDI is that once the model is trained the MDI for each feature can be calculated without the need of further model executions (e.g., evaluating a hold-out set). However, this property is at the same time a weakness of MDI as the feature importance are only derived from statistics of the training data set and thus this measure does not indicate which features are the most important for good predictions on a hold-out set. Further, MDI might overestimate the importance of high cardinality features. PI, on the other hand, does not suffer from these issues. After a model is trained, the PI can be measured for every feature by calculating the increase of the model's prediction error after permuting the feature values included in a validation set. If the model error increases (compared to the model error measured on the validation set containing the unshuffled feature values), the feature is important as the model uses it for correct predictions. The feature is not important if the model error stays unchanged as the model does not rely on the corresponding feature for classification. To select valuable features, we make use of RFE with MDI as well as RFE with PI.

Results

For our feature selection, we first exclude ill-defined features with zero variance or zero mutual information with the target label since they do not contain any information that a classifier could use to distinguish samples. Thereby we remove three of the 136 investigated features. Thereafter, we derive four different feature sets using an RF classifier with default hyperparameters (set by scikit-learn) and the previously introduced feature selection methods: ReliefF, MultiSURF, RFE-MDI, and RFE-PI. As there is no best feature selection method, we additionally combine all sets into an **Intersection** set (by calculating the intersection of all sets) and a **Union** set (by taking the union of all sets). Additionally, we remove multicollinear features within the Union set. Removing such features could improve the classifier's training and classification time without decreasing its classification performance since the classifier can obtain the same information from correlating features. The removal of such features thus reduces the computational burden of a classifier. In order to achieve this, we perform hierarchical clustering on the features' Spearman rank-order correlation coefficients [31], where the coefficients measure the monotonicity of the relationships between different features. We provide a heatmap of correlating features contained in the Union set in Fig. 7.

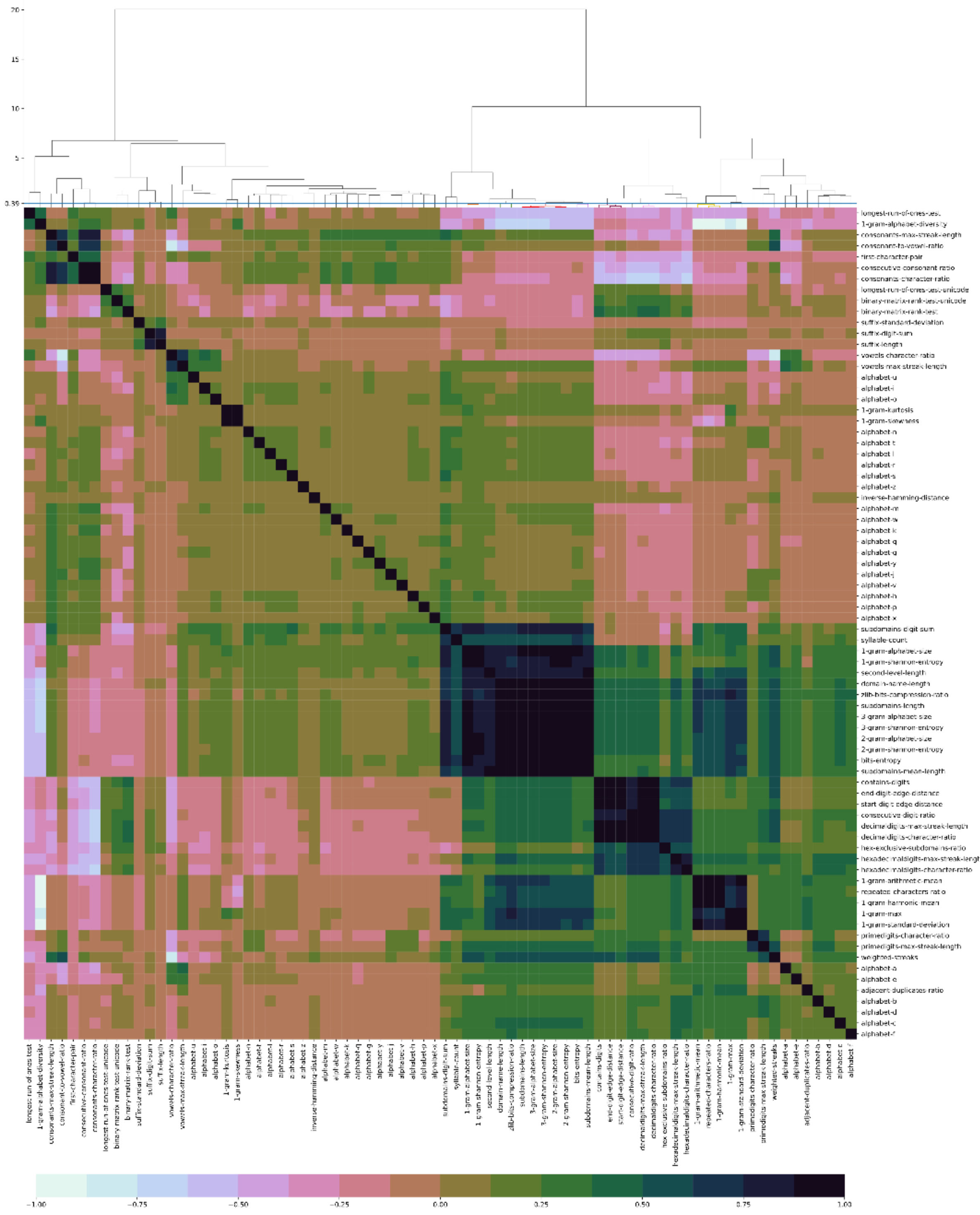

Fig. 7: Heatmap and dendrogram of the correlating features of the Union feature set.

The darker a cell within the figure, the more the features, which are depicted on the x- and y-axis, correlate positively. It can be seen, that while the features in the left upper part of the heatmap are less correlated to each other, features within the right lower part build several clusters. In order to remove the correlating features, we calculate a cut-off threshold targeting the preferred number of remaining clusters. For better understanding, we provide the corresponding dendrogram in Fig. 7 on top of the heatmap including the plot of the cut-off threshold. The y-axis is a measure of closeness of the different clusters. In our case the calculated threshold equals approximately 0.39. The features which are clustered under the threshold line are collapsed by choosing the feature with the highest MDI. Through this process we generate the additional feature set **Union-Spearman**.

The upper part of Table 5 displays the number of selected features per individual selection method as well as evaluation results obtained by classifying the samples of $\text{Set}_{\text{Selection}}$. In the lower part, we additionally include results of the different feature set combinations as well as an evaluation using all 136 features for comparison.

Feature Set	**#**	**F1-score**	**Precision**	**Recall**	**Training Time/Classifier [s]**	**Feature Extraction Time/Sample [μs]**	**Inference Time/Sample [μs]**
RFE-MDI	52	0.74290	0.78390	0.73592	195	156	17
RFE-PI	28	0.75504	0.78528	0.74934	100	106	16
ReliefF	41	0.71707	0.74192	0.71267	184	198	18
Multi-SURF	59	0.72946	0.77833	0.72353	192	241	17
All features	136	0.72806	0.77131	0.72114	320	695	17
Intersection	11	0.64527	0.67936	0.64778	27	85	16
Union	76	0.73352	0.77842	0.72662	264	276	17
Union-Spearman	64	0.74654	0.79204	0.73836	225	239	17

Table 5: Feature Selection Analysis

The best evaluation results on $\text{Set}_{\text{Selection}}$ are achieved with RFE- PI (f1-score of 75.504%). This is remarkable since only 28 of the 136 features are used. The only feature set which contains less features is the Intersection set with eleven features in total. However, the f1-score for this set is with 64.527% the worst. The training and feature extraction speed is the best for RFE-PI when the Intersection set is ignored. All feature selections, except for ReliefF and Intersection, improve the classification performance compared to the classifier that uses all features. The removal of twelve multicollinear features contained in the Union set (yielding Union-Spearman) increases the f1-score by more than 1%. The required inference time per sample does not vary much between all feature sets. Since it cannot be ruled out that some feature sets might perform well on the utilised data set due to overfitting, we consider for our further development the best individual feature set (RFE-PI) as well as the Union and Union-

Spearman feature set combinations. We thus perform individual hyperparameter optimisations for all of these three feature selections.

#	Feature	RFE-MDI	RFE-PI	RELIEFF	MultiSURF	Union-Spearmann	Type	Output	$\mathcal{F}(d_0)$	$\mathcal{F}(d_1)$
1	adjacent-duplicates-ratio	✓			✓	✓	linguistic	rational	0.08333	0.10000
2	alphabet-a	✓	✓	✓	✓	✓	linguistic	integer	0	0
3	alphabet-b		✓		✓	✓	linguistic	integer	0	0
4	alphabet-c				✓	✓	linguistic	integer	1	0
5	alphabet-d			✓	✓	✓	linguistic	integer	0	1
6	alphabet-e			✓	✓	✓	linguistic	integer	3	0
7	alphabet-f				✓	✓	linguistic	integer	0	0
8	alphabet-g				✓	✓	linguistic	integer	0	0
9	alphabet-h				✓	✓	linguistic	integer	0	1
10	alphabet-i				✓	✓	linguistic	integer	2	0
11	alphabet-j	✓	✓		✓	✓	linguistic	integer	0	0
12	alphabet-k		✓		✓	✓	linguistic	integer	0	2
13	alphabet-l	✓			✓	✓	linguistic	integer	0	0
14	alphabet-m	✓	✓		✓	✓	linguistic	integer	0	1
15	alphabet-n	✓	✓		✓	✓	linguistic	integer	0	0
16	alphabet-o			✓	✓	✓	linguistic	integer	0	0
17	alphabet-p				✓	✓	linguistic	integer	0	2
18	alphabet-q	✓			✓	✓	linguistic	integer	0	0
19	alphabet-r	✓			✓	✓	linguistic	integer	1	0
20	alphabet-s	✓	✓		✓	✓	linguistic	integer	1	0
21	alphabet-t	✓			✓	✓	linguistic	integer	1	0
22	alphabet-u				✓	✓	linguistic	integer	1	0
23	alphabet-v				✓	✓	linguistic	integer	0	1
24	alphabet-w	✓			✓	✓	linguistic	integer	0	2
25	alphabet-x	✓			✓	✓	linguistic	integer	0	0
26	alphabet-y	✓	✓		✓	✓	linguistic	integer	1	0
27	alphabet-z	✓	✓			✓	linguistic	integer	0	0
28	consecutive-consonant-ratio	✓	✓	✓		✓	linguistic	rational	0.16667	1.00000
29	consecutive-digit-ratio			✓		✓	linguistic	rational	0.00000	0.00000
30	consonant-to-vowel-ratio	✓		✓		✓	linguistic	rational	0.83333	10.00000
31	consonants-character-ratio	✓	✓	✓		✓	linguistic	rational	0.41667	1.00000
32	consonants-max-streak-length	✓	✓			✓	linguistic	integer	2	10
33	contains-digits			✓			linguistic	binary	0	0
34	decimaldigits-character-ratio	✓	✓	✓		✓	linguistic	rational	0.00000	0.00000
35	decimaldigits-max-streak-length	✓		✓			linguistic	integer	0	0
36	end-digit-edge-distance	✓		✓	✓		linguistic	integer	-1	-1
37	first-character-pair	✓	✓		✓	✓	linguistic	integer	10411	14379
38	hexadecimaldigits-character-ratio	✓	✓	✓		✓	linguistic	rational	0.33333	0.10000
39	hexadecimaldigits-max-streak-length		✓	✓		✓	linguistic	integer	2	1
40	inverse-hamming-distance	✓			✓	✓	linguistic	rational	1.00000	1.00000
41	primedigits-character-ratio	✓			✓	✓	linguistic	rational	0.08333	0.30000
42	primedigits-max-streak-length	✓	✓			✓	linguistic	integer	1	2
43	repeated-characters-ratio			✓	✓		linguistic	rational	0.22222	0.42857
44	start-digit-edge-distance	✓	✓	✓	✓	✓	linguistic	integer	-1	-1
45	subdomain-digit-sum	✓	✓		✓	✓	linguistic	integer	238	237
46	suffix-digit-sum	✓	✓	✓	✓	✓	linguistic	integer	67	80
47	suffix-standard-deviation	✓	✓	✓	✓	✓	linguistic	rational	4.64280	3.67423
48	syllable-count	✓		✓	✓	✓	linguistic	integer	4	2
49	vowels-character-ratio	✓		✓	✓	✓	linguistic	rational	0.50000	0.00000
50	vowels-max-streak-length	✓	✓	✓	✓	✓	linguistic	integer	3	0
51	weighted-streaks	✓				✓	linguistic	rational	0.65278	20.36000
52	domain-name-length	✓	✓	✓	✓	✓	structural	integer	16	15
53	hex-exclusive-subdomains-ratio	✓		✓	✓	✓	structural	rational	0.00000	0.00000
54	second-level-length	✓	✓	✓	✓	✓	structural	integer	12	10
55	subdomains-length	✓	✓	✓	✓	✓	structural	integer	12	10
56	subdomains-mean-length	✓	✓	✓	✓	✓	structural	rational	12.00000	10.00000
57	suffix-length	✓	✓	✓	✓	✓	structural	integer	3	4
58	1-gram-alphabet-diversity	✓		✓	✓	✓	statistical	rational	0.75000	0.70000
59	1-gram-alphabet-size	✓		✓	✓		statistical	integer	9	7
60	1-gram-arithmetic-mean	✓					statistical	rational	1.33333	1.42857
61	1-gram-harmonic-mean	✓		✓		✓	statistical	rational	1.14894	1.27273
62	1-gram-kurtosis				✓	✓	statistical	rational	1.50000	-1.91667
63	1-gram-max			✓	✓	✓	statistical	integer	3	2
64	1-gram-shannon-entropy	✓		✓	✓	✓	statistical	rational	3.02206	2.72193
65	1-gram-skewness				✓	✓	statistical	rational	1.75000	0.28868
66	1-gram-standard-deviation	✓				✓	statistical	rational	0.66667	0.49487
67	2-gram-alphabet-size	✓		✓	✓		statistical	integer	11	9
68	2-gram-shannon-entropy	✓	✓	✓	✓		statistical	rational	3.45943	3.16993
69	3-gram-alphabet-size	✓		✓	✓		statistical	integer	10	8
70	3-gram-shannon-entropy	✓		✓	✓		statistical	rational	3.32193	3.00000
71	binary-matrix-rank-test			✓	✓	✓	statistical	binary	1	0
72	binary-matrix-rank-test-unicode			✓	✓	✓	statistical	binary	0	1
73	bits-entropy	✓		✓	✓		statistical	rational	5.67243	5.45943
74	longest-run-of-ones-test			✓	✓	✓	statistical	binary	1	1
75	longest-run-of-ones-test-unicode				✓	✓	statistical	binary	1	1
76	zlib-bits-compression-ratio	✓		✓			statistical	rational	2.28571	2.42424

Table 6: All selected features: d_0 = iee-security.org, d_1 = mwkwhvkdpp.info

Selected Features

Here, we only present the features we have selected. The full list of investigated features as well as a description for each individual feature can be found in the publicly available source code [24]. All selected features can be separated into three different groups: linguistic, structural, and statistical features. The first category of features analyses the presence or absence of common linguistic patterns. For instance, features of this category evaluate whether a domain name contains digits or compute the vowel ratio. Structural features, on the other hand, investigate structural properties of a domain name such as the domain length. The last category contains features which capture statistical properties such as the frequency distribution of certain n-grams or the entropy. In Table 6, we provide an overview of all features selected by the different selection methods.

For every feature, we mark the membership to a corresponding feature set and present extracted feature values for two sample domains, d_0 and d_1. We define a feature as a function F of a sample d. Thus F(d) denotes the extracted feature. In our example, d_0 = iee-security.org represents a benign NXD caused by a typing error of ieee-security.org while d_1 = mwkwhvkdpp.info is a malicious NXD generated by the Conficker DGA. Note, 60 out of the 76 features from the Union set are newly developed indicating the need of new features for the DGA multiclass classification task.

Hyperparameter Optimisation

The exhaustive grid search is one of the most used hyperparameter optimisation strategies [32]. It generates candidates from a grid of parameter values and evaluates each in order to determine the optimal hyperparameters. As every possible hyperparameter combination for the defined values is evaluated, this brute-force approach is computationally expensive. Random search [32] can be used in order to reduce computational costs by performing a certain number of randomly chosen trials over the hyperparameter space. Obviously, the parameter values that are to be investigated in a grid search can be reduced in order to decrease the computational costs. However, by a coarser grid search it is more probable to miss well performing hyperparameter combinations which might be found by a random search. The number of trials plays a crucial role in finding well performing hyperparameters in a random search. Practically, often 60 trials are used because the maximum of 60 random observations lies within the top 5% of the true maximum with a probability of 95%. However, this only holds true if the close-to-optimal region of the hyperparameters occupies at least 5% of the whole grid surface. Another optimisation strategy to tackle the computational costs is the Bayesian search [33]. The Bayesian search is a sequential process which takes information of the previously evaluated hyperparameter combinations into account in order to choose the next set of hyperparameters for evaluation. The downside of the Bayesian search is that it is not parallelisable since the next search always depends on the results of the previous searches. In this work, we thus choose to utilise random search for hyperparameter optimisation. Thereby, we do not require as many computational resources as for an exhaustive grid search but are able to parallelise the optimisation. As we do not know how much space the close-to-optimal region of the hyperparameters occupies in our case, we double the recommended number of random trials to 120 in order to find well performing hyperparameters. As stated previously, we consider the following feature sets for our hyperparameter optimisation: RFE-PI, Union, and Union-Spearman. For each feature set, we run two hyperparameter optimisations, one using the random forest implementation which is inherently capable of multiclass classification (RF) and one using a one-vs.-rest variant (OvR). We discard all one-vs.one (OvO) variants due to their slow classification speed although their classification performance might be slightly better. In detail, we optimise the following hyperparameters of the random forest implementation of scikit-learn: n_estimators, criterion, max_depth, max_features, bootstrap, and class_weight.

Results: For all three investigated feature sets (RFE-PI, Union, Union-Spearman) we obtain best results on SetSelection using the OvR implementation. We refer to the three different combinations of chosen hyperparameters and feature sets as **EXPLAIN-OvR$_{\text{RFE-PI}}$**, **EXPLAIN-OvR$_{\text{Union}}$**, and **EXPLAIN-OvR$_{\text{Union-Spearman}}$** in the following. We note, that the OvR variants require more training and classification time than the RF variants. Thus, we additionally include the fastest RF implementation to the classifier configurations for comparison. The fastest model makes use of the RFE-PI feature set. We refer to this model as **EXPLAIN-RF$_{\text{RFE-PI}}$** in the following. For reproducibility we provide the specific values for each hyperparameter and for every model in the source code [24].

3.1.8.4 **Evaluation**

We first present the results of our comparative evaluation. Thereafter, we examine the training and classification speed of the various classifiers in order to assess their real-time capability.

Classification Performance

For our comparative evaluation, we compare the selected state-of-the-art classifiers with our developed EXPLAIN classifier configurations using samples of Set$_{\text{Evaluation}}$. We summarise the results of this evaluation in Table 7.

Classifier	F1-score	Precision	Recall
M-FANCI-RF	0.56808	0.58680	0.57805
M-FANCI-RF-OvO	0.57097	0.59210	0.58092
M-FANCI-RF-OvR	0.56907	0.58873	0.57852
M-FANCI-SVM-OvO	0.50320	0.55289	0.51028
M-FANCI-SVM-OvR	0.35113	0.38483	0.37827
M-Endgame	0.74641	0.76731	0.74327
M-Endgame.MI	0.75287	0.77100	0.75351
M-NYU	0.75447	0.79080	0.74648
M-NYU.MI	0.78069	0.80698	0.78038
M-ResNet	0.79574	0.81915	0.79224
M-ResNet.MI	0.80361	0.81435	0.81036
EXPLAIN-RF$_{\text{RFE-PI}}$	0.76114	0.77862	0.75982

EXPLAIN-OvR$_{RFE-PI}$	0.76883	0.79245	0.76624
EXPLAIN-OvR$_{Union}$	0.78554	0.81631	0.77955
EXPLAIN-OvR$_{Union-Spearman}$	0.78046	0.81541	0.77540

Table 7: Multiclass classification results.

The ResNet-based approaches (developed in SAPPAN and reported in D3.4) achieve the best results.

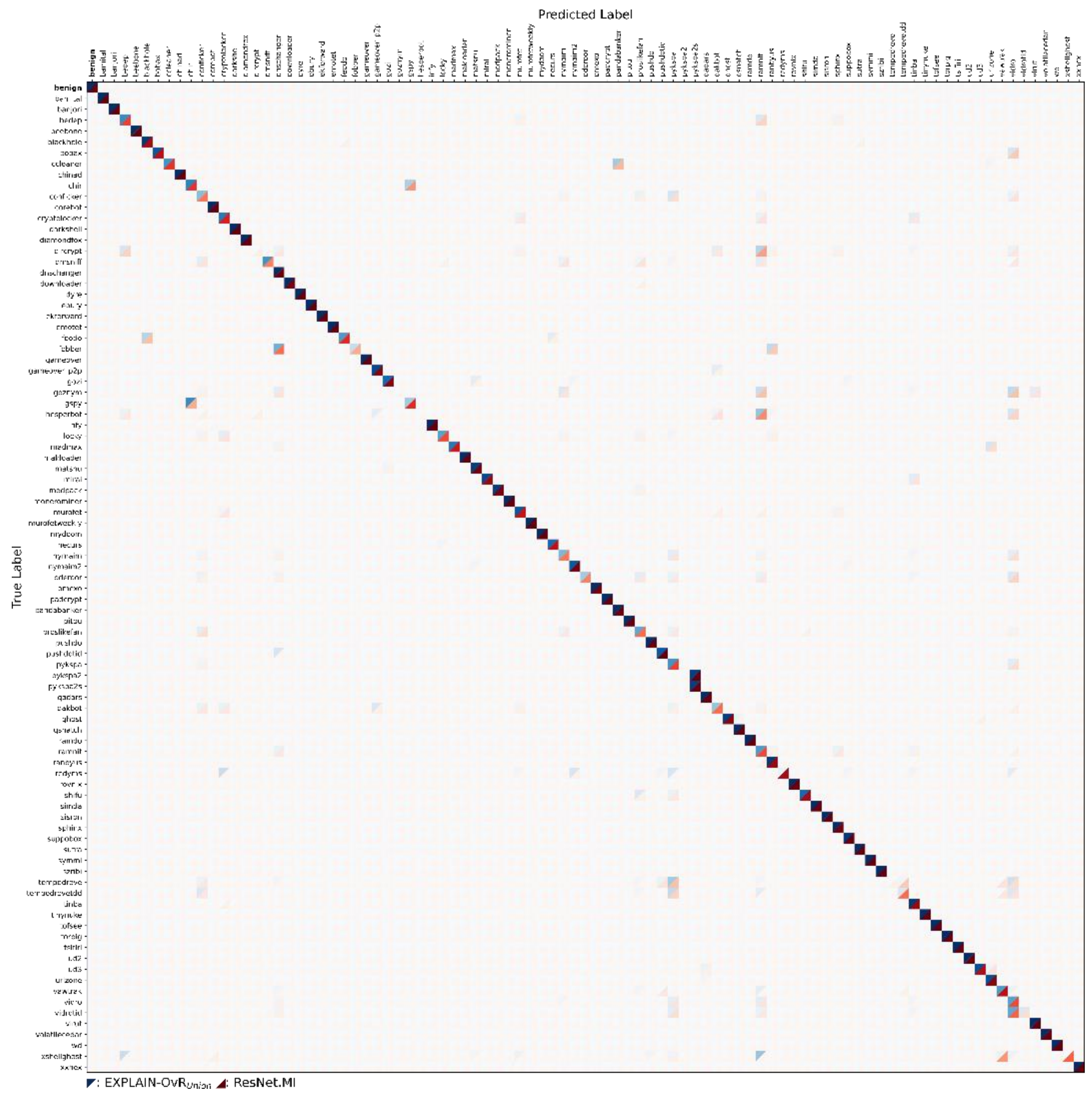

Fig. 8: Combined confusion matrix of EXPLAIN-OvR$_{Union}$ and M-ResNet.MI.

Among our classifiers, our EXPLAIN-OvR$_{Union}$ configuration performs the best and accomplishes an f1-score of 78.554%. By this means, it is the next best classifier after the ResNet-based approaches. The M-NYU.MI model achieves comparable results. An f1-score of 78.554% might appear rather low but devices infected with DGA-based malware will typically generate multiple AGDs per day. Thus, real-world counteractions would not have to be triggered based on a single classification but rather on the fact that multiple AGDs were attributed to the same DGA. All deep learning models benefit from class weighting. The NYU model profits the most and improves its f1-score by over 2.6%. The adapted feature-based approaches of related work perform poorly. Our best EXPLAIN configuration is by over 21.45% in f1-score better than the best FANCI-based approach. EXPLAIN-OvR$_{\text{Union-Spearman}}$ is slightly worse than EXPLAIN-OvR$_{\text{Union}}$. EXPLAIN-OvR$_{\text{RFE-PI}}$ and EXPLAIN-RF$_{\text{RFE-PI}}$ achieve with an f1-score of approximately 76% slightly worse results than the EXPLAIN configurations that are based on the Union feature set. However, both of these classifiers are better than all state-of-the-art classifiers except for M-NYU.MI and the ResNet-based approaches. To better visualise the classification performance and to compare our best classifier configuration (EXPLAIN-OvR$_{\text{Union}}$) with the best classifier (M-ResNet.MI), we present a combined confusion matrix in Fig. 8.

The combined confusion matrix shows for both classifiers the relative number of samples belonging to classes displayed on the vertical axis that are labelled as classes shown on the horizontal axis. For every combination of true and predicted label, space in form of a square is reserved within the figure. Each square is halved into two triangles where the upper left triangle is dedicated to the samples classified by EXPLAIN-OvR$_{\text{Union}}$ and the lower right triangle visualises the classification performance of M-ResNet.MI. The individual achieved scores for both classifiers and every class are encoded within the respective triangles as shades of either blue (EXPLAIN-OvR$_{\text{Union}}$) or red (M-ResNet.MI). An f1-score of 0% is encoded as a transparent triangle and 100% is represented by a fully opaque triangle. A perfect classifier would produce only opaque triangles on the identity matrix diagonal. The benign class is located at the upper left part of the figure. Thereafter are the DGA families listed in alphabetical order. Surprisingly, both classifiers discriminate most DGA families as well as the benign class equally well. Moreover, several DGA families which are almost not recognized by EXPLAIN are also not recognized by M-ResNet.MI (e.g., Dircrypt, Goznym, Hesperbot, Tempedreve). These results indicate that the ResNet-based approach might learn similar features. However, three DGA families (Redyms, Tempedrevetdd, and Xshellghost) are only detected by M-ResNet.MI. It might be possible to engineer new discriminating features by investigating the samples of these three classes in order to enable the correct detection by our classifier. Lastly, both classifiers tend to attribute samples of related DGA families to a single class (e.g., Pykspa-Pykspa2 and Vidro-Vidrotid). All in all, these results show that our EXPLAIN classifiers are able to achieve competitive results while being at the same time, due to the feature-based approach, far more explainable.

Training & Classification Speed

In the following, we compare the training and classification times of the classifiers proposed in related work with our proposed classifiers. Note, we acknowledge that the reported times are difficult to compare as the deep learning classifiers are able to take advantage of GPU processing while the feature-based approaches are evaluated on CPUs. However, the main goal of this study is to determine whether it is realistic to utilise the classifiers for real-time classification. Although, the hardware components may be scaled, the relative time difference in training and classification time allow for a comparison within the group of deep learning classifiers and within the group of feature-based approaches. In Table 8, we display the training time per classifiers and the classification time per samples for all investigated classifiers.

Classifier	**Training Time/Classifier [s]**	**Classification Time/Sample [μs]**
M-FANCI-RF	28	198
M-FANCI-RF-OvO	2528	31711
M-FANCI-RF-OvR	1423	435
M-FANCI-SVM-OvO	114	285851
M-FANCI-SVM-OvR	13042	47239
M-Endgame	1862	151
M-Endgame.MI	1846	149
M-NYU	596	55
M-NYU.MI	577	55
M-ResNet	1106	133
M-ResNet.MI	1028	134
EXPLAIN-RF$_{RFE-PI}$	307	128
EXPLAIN-OvR$_{RFE-PI}$	2533	231
EXPLAIN-OvR$_{Union}$	7036	534
EXPLAIN-OvR$_{Union-Spearman}$	3036	358

Table 8: Performance Analysis

All times are measured during the comparative evaluation. The classification time per sample includes for feature-based approaches the feature extraction time and for deep learning-based approaches the required time for the input pre-processing (i.e. converting domain names to a sequence of integers). We ignore the training and classification times for the FANCI-based approaches as they performed poorly. However, for the sake of completeness we list their times within the table. Regarding the group of deep learning-based approaches, the NYU models are fastest in training and predicting followed by the ResNet-based approaches. The Endgame models are the slowest. Within the group of our proposed classifiers, EXPLAIN-RFR$_{FE-PI}$ is by far the fastest to train and classifies samples similarly fast as the M.ResNet.MI model although it is executed on CPUs. EXPLAIN-OvR$_{RFE-PI}$ is the second fastest EXPLAIN model followed by EXPLAIN-OvR$_{Union-Spearman}$. The twelve additional features included in the Union feature set, compared to Union-Spearman, significantly increase the training as well as the prediction time. In order to approximate the required performance of a classifier for real-time classification, we measure the average amount of occurring NXDs within the university network which was used as source of benign data during our previous experiment. On average, we observed 174 NXD responses per second with a maximum peak of 2325 NXD responses per second. Thus, a classifier has at most 430μs in order to classify a single sample. All classifiers under consideration except for EXPLAIN-

OvR$_{\text{Union}}$ are thus able to perform real-time classification. EXPLAIN-OvR$_{\text{Union}}$ requires 534μs and thus classifies 1872 samples per second. However, during the whole one-month recording, the maximum packets per second exceeded only for four consecutive seconds the mark of 1872. Thus, for the live classification of a large university network, EXPLAIN- OvRUnion would only delay a few packets for a few seconds within a whole month. We thus argue that EXPLAIN-OvRUnion is also well-suited for live classification. Moreover, our proposed EXPLAIN classifiers are highly parallelisable and scale extremely well with the number of CPU cores.

3.1.8.5 **Conclusion and Discussion**

In conclusion, we proposed EXPLAIN, a feature-based and context-less DGA multiclass classifier and compared different EXPLAIN configurations with several state-of-the-art classifiers in a unified setting on the same real-world data. Our best performing model, EXPLAIN-OvR$_{\text{Union}}$ uses 76 features and achieves the best f1-score after the ResNet-based approaches. EXPLAIN-OvR$_{\text{RFE-PI}}$ and EXPLAIN-RF$_{\text{RFE-PI}}$ make use of only 28 features and beat all feature-based approaches proposed in related work by a huge margin. Moreover, they achieve higher f1-scores than the deep learning-based approaches: M-Endgame, M- Endgame.MI, and M-NYU. Surprisingly, the in-detail comparison between EXPLAIN-OvR$_{\text{Union}}$ and M-ResNet.MI indicates that the deep learning classifier might learn very similar features as the ones we have selected for our EXPLAIN classifiers. Finally, we analysed the real-time capability of the different classifiers. All of our proposed classifiers are highly capable of real-time classification. Our fastest proposed model, EXPLAIN-RF$_{\text{RFE-PI}}$, is even able to classify 7812 samples per second. Which EXPLAIN configuration to choose depends on the individual requirements of the classifier. EXPLAIN-OvR$_{\text{Union}}$ achieves the best classification results using 76 features. Selecting a configuration that uses the RFE-PI feature set with only 28 features could make it easier to interpret predictions from a model. EXPLAIN-OvR$_{\text{Union-Spearman}}$ offers a compromise as it uses 64 features but achieves comparable classification results as EXPLAIN-OvR$_{\text{Union}}$. The EXPLAIN-RF$_{\text{RFE-PI}}$ configuration should be chosen over EXPLAIN-OvR$_{\text{RFE-PI}}$ when the computational resources are limited. Otherwise, for a fast, explainable, and slightly better classifier EXPLAIN-OvR$_{\text{RFE-PI}}$ should be chosen. By design our feature-based approach is more explainable compared to the deep learning classifiers proposed in related work as predictions can be traced back to the characteristics of the used of features. In contrast, deep learning classifiers only output a vector of probabilities indicating to which class a particular domain can be attributed to without referring to the actual input.

By proposing a competitive feature-based approach we made a first step towards explainable DGA multiclass classification. Ultimately, a focused comparison of different approaches to DGA multiclass classification with respect to explainability is required. Our work is a necessary prerequisite for a comparative explainability study in which competitively performing feature-based and deep learning-based approaches have to be contrasted. In future work it is required to compare the level of explainability provided by our approach with different techniques, such as Lemna [34] and DMM-MEN [35], which try to explain the predictions of deep neural network classifiers. Moreover, in SAPPAN, we proposed a visual analytics system (reported in D3.8) which strives to provide understandable interpretations for predictions of deep learning based DGA detection classifiers. This system first clusters the activations of a model's neurons and subsequently leverages decision trees in order to explain the constructed clusters.

Additionally, future work could analyse the interpretations of deep learning-based models for correctly classified samples which, however, are incorrectly classified by our approach. It might be possible to extract used features of deep learning models which enable the correct classification of such samples for our classifier. Thereby, it might be possible to further enhance the performance of our approach.

With proposing EXPLAIN we did the first step towards using its feature extractor for sharing anonymised domain names in a collaborative machine learning scenario for DGA multiclass classification. While the privacy analysis of FANCI's feature extractor for sharing anonymised data for DGA binary classification yielded promising results, the results cannot be easily transferred to EXPLAIN for DGA multiclass classification, as EXPLAIN uses different feature sets. Thus, a similar privacy evaluation of the different feature sets has to be conducted in future work.

3.2 Application profiling

In this Section, we discuss the impact of shared datasets on the accuracy of our application profiles. We test, whether application profiles that were generated on one host can be transferred to the application behaviour of another host. Additionally, we discuss, whether application profiles that were created using data of multiple hosts better represent the application's behaviour in general.

3.2.1 Sharing of DNS data

3.2.1.1 Rule-based approach

In Deliverable D3.5, we described our rule-based pipeline for application profiling on DNS data. First, we described the data generation process using our developed tool ATLAS. Based on the generated data, we automatically extract rules which represent the typical behaviour of applications when it comes to DNS traffic. In D3.5, we already presented an evaluation of our approach. We generated a dataset using nine virtual machines, where we installed multiple applications and interacted with them manually. We managed to achieve zero false positives and false negatives on the validation dataset. Only when including applications in the validation dataset that were only active for 60 seconds, we observed a few false negative results.

For the full evaluation of this approach, we refer to Deliverable D3.5. However, to demonstrate the benefit of using data from multiple hosts, we now briefly compare the results of generating rules from only one host with the before mentioned results. We use the same parameters for rule-extraction that we derived as described in D3.5.

We used data from one host for rule extraction. As described in Deliverable D3.5, the Rule-Extractor mostly uses three criteria to pick domain candidates, for required as well as optional labels, that are used in SET-rules. First, whether a domain is queried by different applications or is likely to be queried by user. Second, the fraction of application instances that queried the domain. And third, the median interval between queries of the specific domain for the application. The second criterion requires that multiple instances of the application is present in the dataset. Hence, our dataset contains five individual datasets created using ATLAS, which were captured on different days. Additionally, the applications were restarted multiple times during each day. By doing this, we are still able to make use of the second criterion, even though the dataset contains only application traffic captured on a single host. For more details on the rule extraction, see Deliverable D3.5.

Table 9 shows the number of identified candidates for required and optional labels that are considered in SET rules. As a reminder, required candidates might also be used as optional labels, if the number of required candidates exceeds the maximum number of required labels. We want to highlight two observations from this data. First, the total numbers of required and optional candidates increased by 75% when the data of all hosts are considered. This is the case because first, a larger variety of application instances is present in the data, and second, because the data contains overall more raw capture time. The greater number of candidates allows to use more labels, either required or optional, in each SET rule, making them more robust to reduce false positives. The second observation is that the number of required candidates decreased by around 40%, while the number of optional candidates increased by a factor of seven. This is the case because overall more application instances are in the dataset, and therefore, the second mentioned criterion can be calculated more accurately. While some domains are typically queried by one host, they might not be queried by all other hosts. Hence, some of the domains classified as required candidates in the smaller dataset are either classified as optional candidates or completely discarded in the larger, shared dataset. This makes the generated SET-rules more robust and representative for the applications, independently of individual hosts.

Next, we compare the number of false positives and false negatives of the two created rule sets. For this, we make use of the same validation dataset described in D3.5, containing traffic of all applications spread among four different hosts. Table 10 shows the analysis results of our validation dataset. With the generated rules, we end up with a large number of false positives and false negatives. When using the multi-host dataset for rule generation, we had zero false positives and only false negatives when including applications that only ran for 60 seconds. With the single-host ruleset, we have considerably more false positives and false negatives even without the short running applications. Note, that the one host that produced the data for our rule generation was also used as a host in the validation set. On this host, the ruleset produced only one false positive (MS Edge) and one false negative (Firefox). However, on the other hosts, almost no applications where detected correctly.

Application	**Rule candidates from one host (required/optional)**	**Rule candidates from nine hosts (required/optional)**
Firefox	14/5	12/18
Chrome	9/0	0/4
Windows 10	2/0	1/5
Sophos	0/0	0/1
Skype	8/0	4/10
Onedrive	6/0	2/6
Dropbox	6/1	2/5

Steam	4/0	2/4
Zoom	1/0	6/3
MS Office	2/3	0/11
Teamviewer	3/2	3/6
MS Edge	0/0	0/11
Easyminer	1/1	1/2
Total	56/12	33/86

Table 9: Domains considered as suitable candidates for CF- and SET-rules, based on data from one host vs. nine hosts.

Ruleset	**False Positives (with/without short runtimes)**	**False Negatives (with/without short runtimes)**
single-host	6/11	6/22
multi-host	0/0	0/5

Table 10: False positives and false negatives (with and without including application runtimes of 60 seconds) of the single-host and multi-host rulesets.

This shows that the amount of data used for rule extraction is crucial for the construction of a well performing rule set. We assume that the total amount of traffic in the single-host dataset was simply not enough. We included five days of traffic, however, the multi-host dataset contains more than five times the number of DNS packets. Further, the addition of different hosts allowed the rule-extractor to identify better domain candidates that are more representative of the applications.

3.2.1.2 **Process mining-based approach**

In Deliverable D3.5, we described our process mining approach for application profiling based on DNS data. The general idea is to create process mining models that represent the typical behaviour of applications when it comes to DNS traffic, in order to identify such applications when monitoring a network. Note that the approach was meant as a proof of concept to find out whether process mining is applicable to our use-case. We showed that our models can represent the behaviour of applications well in most cases, especially when using token replay to measure the fitness. However, we encountered problems with browsers in some cases, which was expected due to the dependency on user input for browsers considering DNS traffic.

In D3.5, we only looked at models created from data of one host tested with data from the same host. In general, the approach already showed promising results when only considering one data source. Here, we want to test whether models created on one host work on data from a different host. Additionally, we test whether the fitness improves if we take data from multiple hosts into account for model creation. As data source, we use data of four different hosts where the same applications are installed. Note that all of the hosts are virtual machines and we interacted with the applications manually. Hence, this experiment only serves as an indication whether the models are transferable to different hosts in a real environment.

Application	Average trace fitness (token replay)	Log fitness (token replay)	Fitness (footprints)
Firefox	0.999	0.999	0.993
MS Edge	0.857	0.857	0.714
Steam	0.992	0.987	0.926
Nicehash	1.0	1.0	1.0
Windows 10	0.989	0.989	0.596
Chrome	0.700	0.963	0.910
Dropbox	0.998	0.998	0.978
Sophos	1.0	1.0	1.0
MS Office	1.0	1.0	1.0

Table 11: Fitness of application models when tested against test data of a different host, to test transferability of the models.

Application	Average trace fitness (token replay)	Log fitness (token replay)	Fitness (footprints)
Firefox	1.0	1.0	1.0
MS Edge	0.997	0.997	0.913
Steam	0.993	0.993	0.941
Nicehash	0.931	0.931	0.712
Windows 10	0.971	0.971	0.648
Chrome	0.997	0.997	0.964
Dropbox	1.0	1.0	1.0
Sophos	0.786	0.786	0.667
MS Office	1.0	1.0	1.0

Table 12: Fitness of application models generated using data from multiple hosts, to test the impact on model fitness.

Table 11 shows the results when testing the model generated on data from one host on data from another host. For many applications, the fitness values are high, with the only exceptions of Chrome and Edge. As described in D3.5, it is harder to create profiles for browsers that are not too general (and always yield a fitness of 1.0), or too imprecise. In Table 12, we can see the fitness values for the experiment, where the models were generated using data from three hosts, and tested with data from a forth host. In general, the fitness values are high for token replay. However, the fitness for Nicehash and Sophos are worse compared to the other experiments.

In summary, we conclude that models created on data of one host can be transferred to other hosts in most cases. Especially for non-browser applications, the fitness stays quite high. However, we could not measure a reproduceable benefit of combining data from different hosts. One reason for that is most likely the similar nature of our virtual machines that we used for data generation. At time of writing this deliverable, we had no access to real-world datasets to repeat these experiments, which would be necessary to further explore the benefits of sharing in this use-case.

3.2.2 Sharing of System Event Data

In the following, we briefly discuss the impact of merging multiple datasets for our application profiles based on system events. In Deliverable D3.5, we described our pipeline for the creation of application profiles using process mining on system event data, captured by the F-Secure sensors. In an experiment, we created a model of the applications installed on a host based on 50 days of benign data. Afterwards, we used the Trace-Checker with data captured during our red-teaming experiment to see, whether the model outputs anomalies.

Now we want to discuss how our models behave when data from different hosts is considered. First, we want to test, whether a model created on one host accurately represents benign behaviour of different host. Next, we create a model based on data from multiple hosts and use the Trace-checker again on the attack data (similarly to the experiment described in Deliverable D3.5) to test whether the results are considerably different compared to using data from only a single host.

We used similar datasets as described in Deliverable D3.5. This time, we consider attack and benign data from the red-teaming host, as well as an additional host that was set up similarly to the red teaming machine, but without any malicious activity. The datasets are listed in Table 13.

Dataset	Type	Time (days)	System Events
RED-benign	benign	51	8.035.194
RED-attack	malicious	1	17.089
RED-train	benign	50	7.845.569
RED-test	benign	1	189.625
normal-test	benign	31	61.623
normal-train	benign	249	186.525

Table 13: Datasets used for the anomaly detection experiment, adapted from [45].

Table 14 shows the results of experiments with different hosts. The Trace-checker outputs are described in Deliverable D3.5 in more detail. It shows, that the RED-benign dataset is not fitting well to the models created by the normal host. Also, the RED-attack dataset has only slightly more errors in comparison. However, when combining the data from the RED-train and normal-train datasets for model creation, the RED-test dataset only produces a very low amount of errors. The RED-attack dataset, however, still shows 23.5% errors, indicating an anomaly. It is worth noting that the benign test of different hosts shows way more errors, even without malicious activity. This shows that models created on one host cannot simply be transferred to another hosts, even if both are similar.

Trace-checker output	**RED-benign vs. normal-train**	**RED-attack vs. normal-train**	**RED-test vs. RED-train + normal-train**	**RED-attack vs. RED-benign + normal-train**
true	51.22%	49.54%	97.03%	76.49%
true, but insertions	1.96%	1.52%	0%	2.7%
multiple fingerprints not part of the fingerprint matrix	4.16%	0.91%	0%	3.51%
successor is not correlated to the fingerprint	23.5%	6.23%	0%	10%
no startpoint found	17.51%	40.43%	2.97%	5.14%
no endpoint found	0.69%	0.3%	0%	0%
startpoint not valid	0.07%	0%	0%	0%
endpoint not valid	0.1%	0%	0%	0.54%
startpoint not in the fingerprint matrix	0.14%	0.76%	0%	1.08%
endpoint not in the fingerprint-matrix	0.48%	0.15%	0%	0.54%

Table 14: Trace-checker errors for the test and attack datasets compared to the benign model computed from one or two datasets, adapted from [45].

Our focus for the process mining approach on system events was to use the created models on the same systems as they were generated on, as described in Deliverable D3.5. Because this first experiment regarding transferability of models between different hosts did not show promising results, we did not continue its development, and rather focused on the other approaches presented in this deliverable.

3.3 Anomalous Login Activity model

The Anomalous Login Activity model was introduced in Section 3.4.2 of SAPPAN D5.1. It is a supervised learning model that can be used to detect anomalous login events on organization's endpoints. The model is trained on successful logins within an organization and predicts whether a given login is expected on a given endpoint. Unexpected logins are detected as anomalies and flagged for further analysis. In this section, we briefly recollect the model, present the evaluation methods and results, and discuss the efforts of combining organization-specific models from multiple organizations for reducing false positives and for increasing the confidence of detected anomalies. The mentioned efforts connect the Anomalous Login Activity work to the research in Task 5.2 (though, unfortunately, our model combining experiments did not result in visible improvements).

3.3.1 Motivation

Detecting signs of anomalous login events is a time-consuming task when performed manually. To automate this task, a rule-based system that receives information about login events on an endpoint can be configured to catch suspicious activities of certain types, for example, multiple failed login attempts indicative of brute force attacks. Such attacks recently gained attention (see, e.g., [46] with the US and UK intelligence claiming that Russian military hackers are running a cyber-campaign to steal emails and other information, including from parliaments, targeting primarily the United States and Europe. (The same hacker group is believed to steal and leak Democrat emails during the US 2016 presidential election and to target the Norwegian parliament in the summer of 2020.) The campaign has mainly been directed at organisations using Microsoft Office 365 cloud services, with the hackers using multiple attempts to log in with different passwords to try to access systems, employing specialist software to scale their activities and using Virtual Private Networks and Tor, an anonymising system, to try to hide their traces. As reported by Microsoft, the targeted organisations typically saw more than 300 log-in attempts per hour for each targeted account, over the course of several hours or days.

From the detection point of view, however, it is very difficult to manually write rules capable of, for instance, detecting a successful login from an intruder who has already gained access to credentials on one of the targeted endpoints. Such a ruleset would require a broad context, including data from a wide time window.

A machine learning model trained to detect anomalous login events is well-suited to catch phenomena that would be difficult to spot manually. One example of this would be a user, that has never logged into a system using Remote Desktop Protocol (RDP), suddenly doing so. A machine learning model can also be able to identify anomalies from other variables present in its input data, such as source IP address and time of day, and can hence be well-suited to identify tactics employed by adversaries for lateral movement purposes.

3.3.2 Approach

The ground truth on confirmed intrusions is rare and costly to find or produce. So, we follow a different approach which does not rely on labeled data. We consider a relatively large training period (e.g., 2 months) and we assume that malicious intrusions are very rare in this period, i.e., the absolute majority of logins are "clean".

Each training login is paired with its host / endpoint identifier, and a multi-class classification model is trained using the host identifier as a label, i.e., a model is trained to predict to which host in the organization a given login belongs. This training is carried out for each organization, generating multiple organization-specific models (which opens opportunities for combining those models in the style of SAPPAN Task 5.2, see Sections 3.4.2 and 3.4.3 below).

In the testing phase, new logins are fed to the model. If the model correctly predicts the host identifier for a given testing login, this login is considered normal. Otherwise, i.e., if the predicted host identifier does not match the actual host identifier, the login is considered as anomalous.

3.3.3 Data sources

We use a software sensor which collects relevant information from each endpoint in a privacy-preserving fashion (F-Secure's endpoint sensors were used in the research). This information is used by various detection models, rules and security analysts to detect anomalous or malicious behavior. The data for the Anomalous Login Activity model were a subset of that sensor data. In particular, it consists of events representing user logins on endpoints.

Each login record includes the following features:

- username
- source IP
- authentication protocol
- timestamp

This information is gathered from both Windows and Linux endpoints.

3.3.4 Data preprocessing

3.3.4.1 Removal of common logins

To discriminate between endpoints, we remove common (popular) logins appearing on multiple endpoints within an organization. These usually represent system-level users or maintenance accounts, which may cause noise in the model output.

3.3.4.2 **Vectorization**

The login records are vectorized using a bag-of-words approach. First, a document is created from all the successful login events in a given endpoint. Each word in this document corresponds to one of the features. Then, a count vectorizer is used to convert the document into a numerical vector.

3.3.5 Implementation

Most of the implementation uses pandas [47], scikit-learn [48] and PySpark [49] APIs. The large number of organizations and the large training periods considered make training of each model sequentially infeasible. Instead, we implemented a distributed approach by leveraging the PySpark partition mapping functionality. We partition the training set so that each partition contains the training data from a single organization. Then, we use PySpark API to train each organization's model in parallel by taking advantage of each thread in the PySpark cluster. This greatly reduces the execution time required for the training, significantly reducing the analysis costs.

3.3.6 Evaluation

To validate our approach, we devised an ad-hoc evaluation scheme. Traditional classification metrics are, unfortunately, difficult to compute due to the absence of the ground truth on the intrusions and login events associated with those. Instead, we define a set of approximated quality metrics, based on the assumption that malicious login events are very rare. Note that since we have a different model instance for each organization, all the metrics are averaged over all the model instances.

3.3.6.1 False Positive Rate

Assuming the training set contains very few or no intrusions (so very few or no malicious login events), we can estimate the false positive rate of the model as the number of anomalous logins the model generates on a portion of the training set, referred to in the text that follows as the validation set.

3.3.6.2 Precision/recall on poisoned data

To produce accuracy metrics, we can artificially generate a labelled data set by "poisoning" a testing set. By poisoning, we refer to replacing real login features with unusual values. An accurate model should pick up on these changes, and it should flag the "poisoned" logins as anomalous. By considering poisoned logins classified as anomalous by the model as true positives, we can compute accuracy metrics, such as precision and recall. Note that this approach assumes that the testing set contains only a few non-poisoned malicious login events, or none at all. This assumption should reasonably hold in most real-world cases.

3.3.6.3 Experimental setting

We experimented with several multi-class classification models. All the implementations were taken out-of-the-box from the scikit-learn library. The evaluated models are:

- naive bayes classifier with multinomial prior (multi-nb)
- naive bayes classifier with Gaussian prior (gauss-nb)
- decision tree classifier (decision-tree)
- SVM classifier (svm)

The random forest classifier was also considered but discarded due to its excessive memory usage. The training set is taken to be increasingly large from 5 days to 25 days. The validation set is taken to be 2 hours (10:00-12:00) in the last day of the training period. The testing set is taken to be 2 hours (10:00-12:00), 5 days after the last day of the training period.

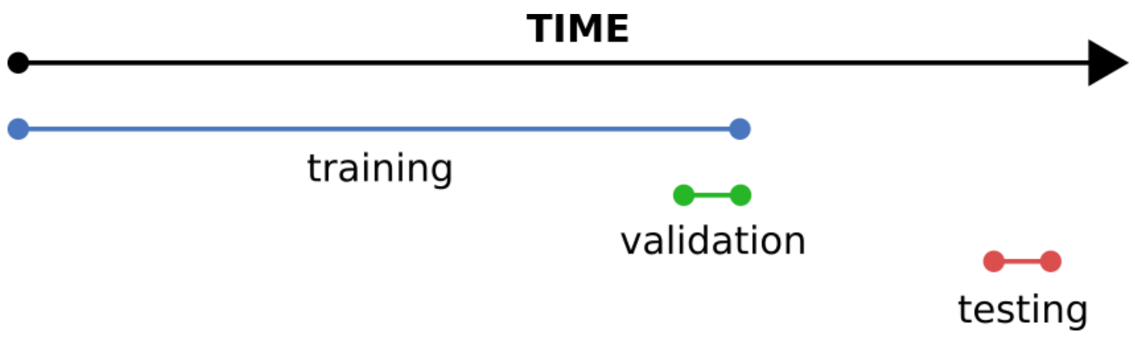

Fig. 9: Timeline of experimental setting

All the data were generated by the F-Secure's production systems during the months of July and August 2021.

3.3.6.4 **Results**

False positive rate

Fig. 10 shows the estimated false positive rate of the considered models.

The Gaussian naive Bayes and SVM classifiers did not meet the memory requirements for the largest training set (25 days), so the results stop at the period of 20 days.

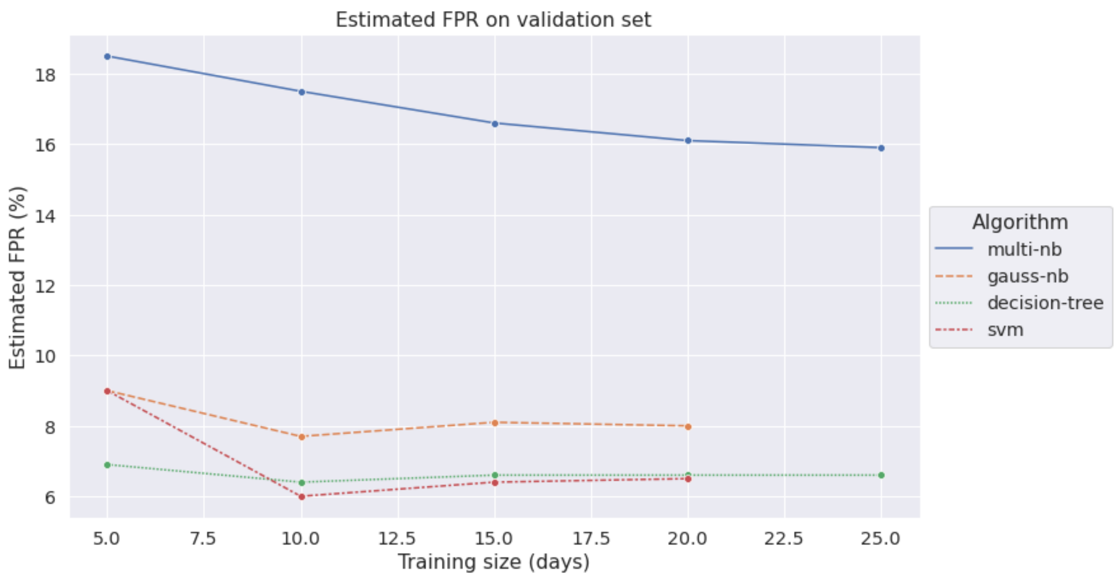

Fig. 10: Estimated FPR on validation set

Precision on poisoned data

Fig. 11 shows the estimated precision on poisoned data of the considered models. The Gaussian naive Bayes and SVM classifiers did not meet the memory requirements for the largest training set (25 days), so the results for those stop at the period of 20 days.

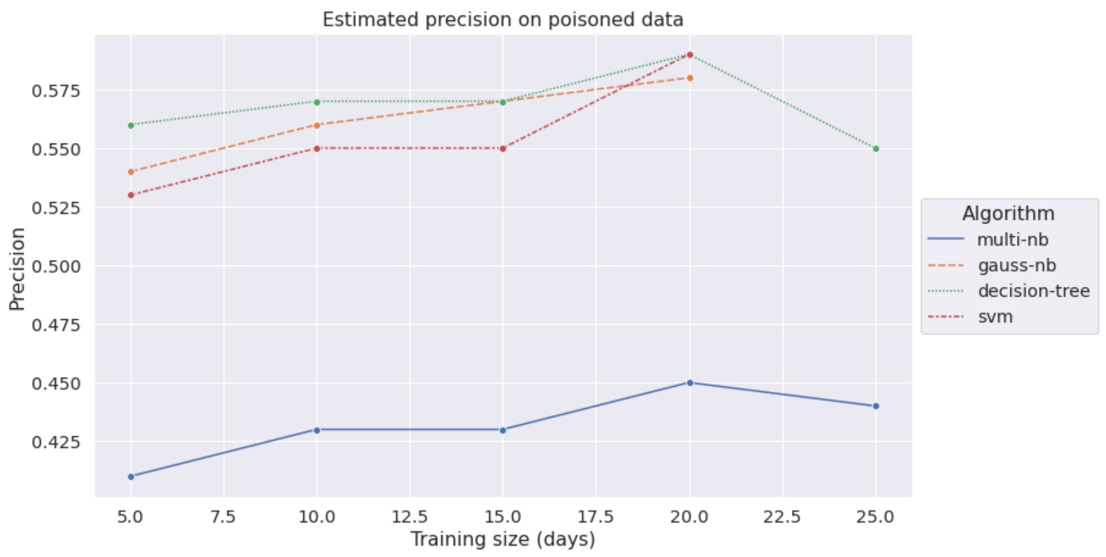

Fig. 11: Estimated precision on poisoned data

Recall on poisoned data

Fig. 12 shows the estimated recall on poisoned data of the considered models. The Gaussian naive Bayes and SVM classifiers did not meet the memory requirements for the largest training set (25 days), so the results for those stop at the period of 20 days.

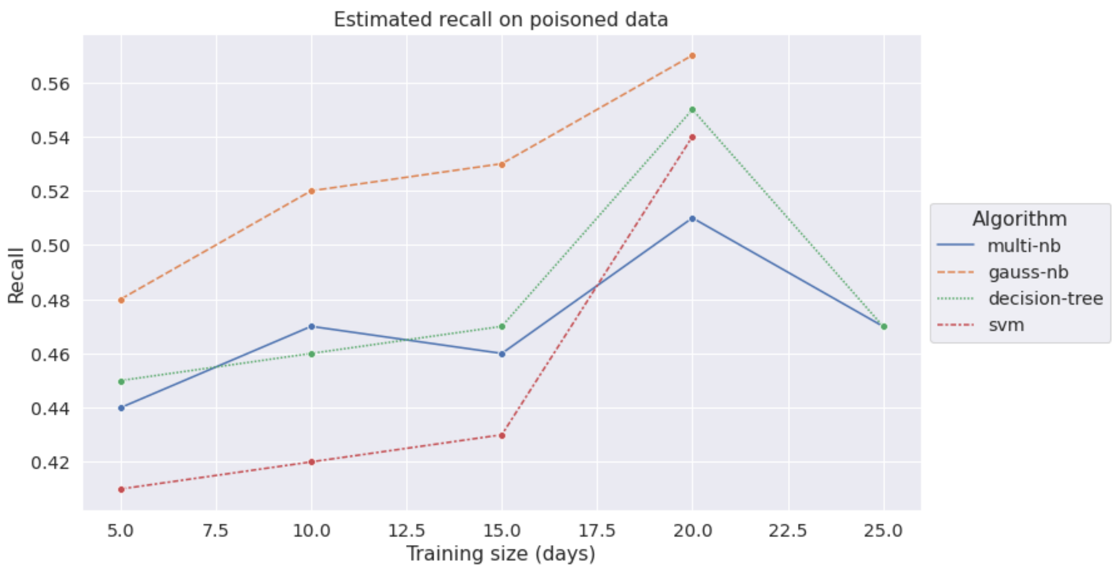

Fig. 12: Estimated recall on poisoned data

Model comparison

Fig. 13 shows a visual comparison of the models, with the Gaussian naive Bayes classifier and decision tree models showing the best accuracy. However, the naive Bayes classifier did not meet the memory requirements for the largest data set considered.

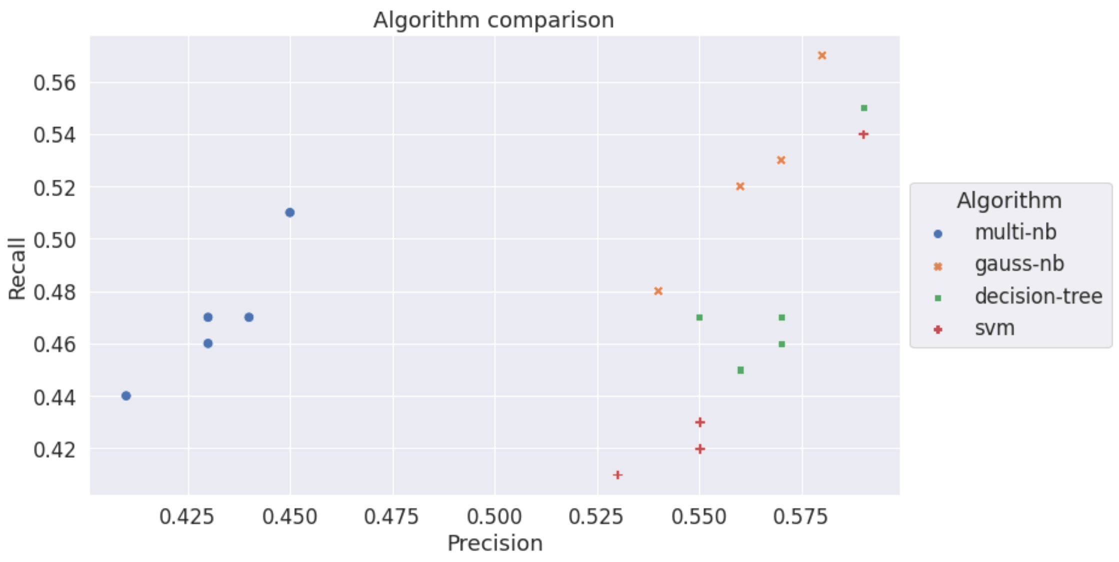

Fig. 13: Algorithm comparison

3.3.6.5 **Performance across organizations**

The results presented above were aggregated over all the organizations. As mentioned, each organization is analyzed and ‘modelled’ independently to produce a classification model. The data produced by each organization can vary widely depending on several factors including: the number of active endpoints, the configuration of the installed sensors, the number of active users. Therefore, the models from different organizations might have different characteristics. To investigate this, we analyzed the accuracy on the validation set across the organizations. The following histogram summarizes our results. These were produced using the decision tree classifier.

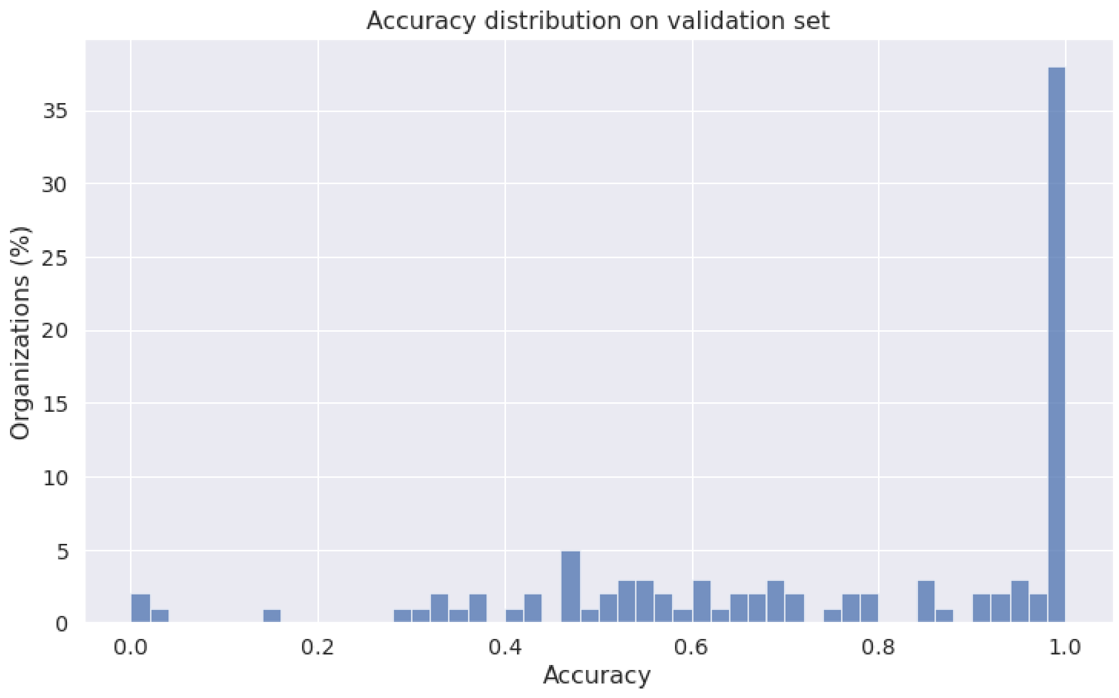

Fig. 14: Accuracy distribution on validation set

Fig. 14 shows that over 35 percent of the organizations produced extremely accurate models (with the accuracy above 95 percent). The rest of the sample is more evenly distributed, with the accuracy ranging from 0 to 90 percent. This analysis shows that for many organizations the data collected may not produce a reliable model. This also explains the low precision and recall scores obtained by averaging over all the models. To improve the quality of the results produced in production, and to avoid large numbers of false positives produced by inaccurate models, we decided to deploy in production only the models with the validation accuracy above a given quality threshold. This choice reduces the model's coverage, i.e., the percentage of customer organizations covered by the model, to around 35%. However, this drawback is offset by the low number of false positives the most accurate models produce.

3.3.7 Combining models for improved accuracy

As already mentioned, data from different organizations were analyzed independently to train multiple organization-specific classification models, opening opportunities for using those models jointly in the style of SAPPAN Task 5.2. In this section, we investigate how multiple Anomalous Login Activity models could be combined for increased accuracy.

An anomalous login event is characterized by a user name, a login protocol and a source IP. IP addresses depend on the network configuration and are therefore organization-specific. On the other hand, the same user names and protocols can be present in multiple organizations. As an example, think of application processes that need to be run by a particular system-level user. We expect to find these processes across multiple organizations, especially if they relate to widely adopted software frameworks.

We can use this information to reduce the false alarm rate, by considering anomalies reported by multiple models and filtering out those that are common across similar organizations.

To talk about similar organizations, we need to define a similarity metric. We chose to compare organizations using their sensor distribution as features. These include: the total number of endpoints; the average number of login events per endpoint; the number of Linux and Windows sensors respectively. This set of features is privacy-preserving and it provides a similarity metric to compare organizations based on their size and endpoint types distribution. While naïve, this notion of similarity is in line with our analysis, since we expect similar-sized organizations to have a higher probability of having common software frameworks, as they can be expected to have similar technological problems to solve based on their size.

With this similarity metric, we can compute nearest neighbour organizations for each given organization. We can then compare the login anomalies produced by one organization's classification model with those produced by its nearest neighbours. Anomalies that appear identical across multiple similar organizations would plausibly represent false positives. We conducted an experimental evaluation to validate this idea. However, we could not find common anomalies across organizations in any of the conducted experiments. A natural next step will be to experiment with other similarity metrics and larger sets of organizations.

3.3.8 Clustering similar alarms

We also investigated how anomalous login events cluster together. Each login event is characterized by a source IP, an authentication protocol, a username, and a target endpoint. Clustering together similar login events can provide additional information to the security analyst with respect to the severity of the alarms.

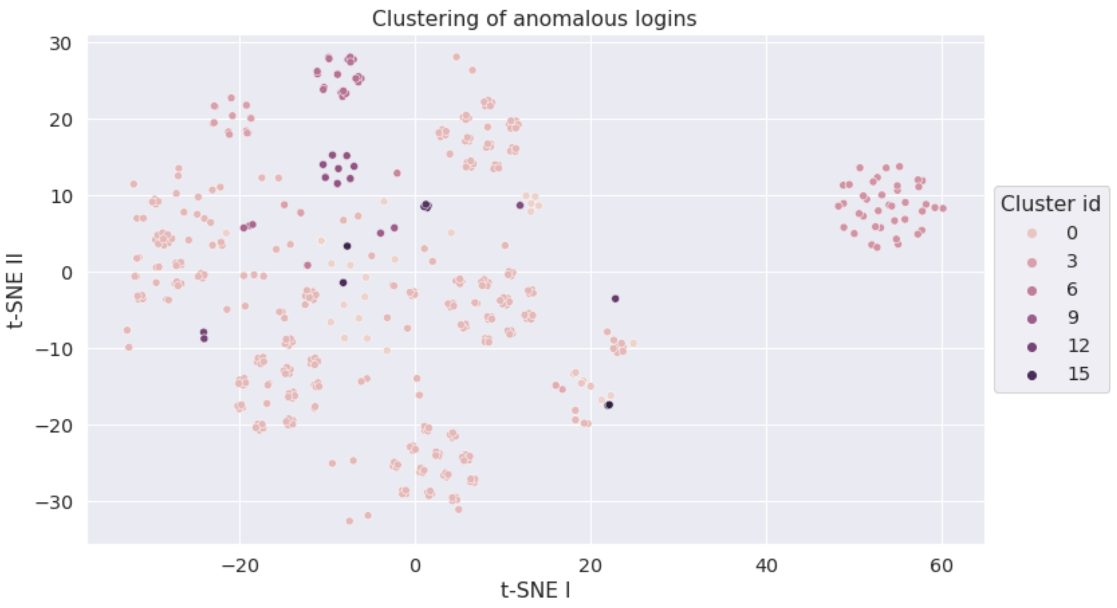

Fig. 15: Clustering of anomalous logins

To investigate this, we experimented with the DBSCAN algorithm on the anomalous login events generated by the models. We used the same data preprocessing techniques on the login data to produce vector representations and then we applied density-based clustering on the resulting vectors. Fig. 15 shows a representation of the clustering obtained using t-SNE dimensionality reduction.

The clustering results seem promising at first. However, a deeper analysis shows that anomalous login events tend to cluster around unreliable features, such as the source IP, which depend on the organizational network configuration, and thus cannot be used to cluster alarms across different organizations. The same is also true for user names and host identifiers. These are specific to each organization, and thus should not be used in looking for similarities across organizations.

With respect to the authentication protocols, the majority of login events utilize the same authentication protocols. Thus, these do not add much information to the clustering. In conclusion, although the idea of clustering similar alarms together for a deeper understanding seems viable in theory, our results show that it does not provide significant benefits in practice. We believe this is mainly due to the very limited feature set that we use to detect anomalous login events. Further experiments are required to explore other potential features.

4 Conclusion

In this Deliverable, we presented the results of Task T5.1 "Distributed Learning of a global model based on shared anonymised data". We briefly discussed the context of this task within the overall scope of the SAPPAN project and explained its general concept. The developed approaches build on the use-cases DGA detection and application profiling. In contrast to the tasks in WP3, where only local data is used, we focused on approaches based on shared data.

For DGA detection, we initially defined three different scenarios for creating a global model based on shared anonymised data for DGA detection. An analysis of the three scenarios showed that sharing extracted features of a feature-based approach is the most promising scenario. Using this anonymisation technique has no impact on classification performance, but the privacy guarantees provided were still to be determined. Hence, we performed an extensive privacy evaluation of FANCI's feature extractor. There we showed that sets of feature vectors are resilient against a reconstruction attack and can therefore be used as a way of anonymisation in this sharing scenario. Since this analysis yielded promising results, we developed EXPLAIN, the first context-less and feature-based DGA multiclass classifier. This classifier achieves competitive results, is real-time capable, and its predictions are easier to trace back to features than the predictions made by the DGA multiclass classifiers proposed in related work. We intend to use EXPLAIN's feature extractor in the same sharing scenario as explained for FANCI's feature extractor. Here, we solve the problem of DGA multiclass classification in contrast to the DGA binary detection task. However, as EXPLAIN uses different feature sets, a similar privacy evaluation of the different feature sets has to be conducted in future work.

For Application profiling, we conducted experiments to determine whether application profiles that were generated based on one host can also model the behaviour of the same application running on a different host. Additionally, we explored whether application profiles become more robust when using data from different hosts or application instances for model generation. For the process mining models based on DNS data, we found that a model that was created from data of one host also works well for other hosts. However, we could not measure a meaningful difference when using data from multiple hosts during model generation. For the process mining models based on system events, the results were similar. For our rule-based approach, we found that creating rules based on traffic of only one host is not sufficient to detect the applications running on other hosts. However, when including data from multiple machines, the number of false positives and false negatives decreased considerably.

For the Anomalous Login Activity Model, we recollected the approach, presented the model evaluation methods and results, and discussed the efforts of combining organization-specific models from multiple organizations for reducing false positives and for increasing the confidence of detected anomalies. The last part connects the work to the efforts of Task 5.2, though, unfortunately, our model combining experiments did not result in visible improvements.

5 References

[1] 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.

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

[3] 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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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

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

[12] 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.

[13] 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

[14] 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.

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

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

[17] 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.

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

[19] 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.

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

[21] 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.

[22] 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.

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

[24] A. Drichel, N. Faerber, and U. Meyer. 2021. Source Code of EXPLAIN: Feature-based DGA Multiclass Classifier. https://gitlab.com/rwth-itsec/explain

[25] L. Breiman. 2001. Random Forests. Machine learning 45, 1 (2001).

[26] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. 2002. Gene Selection for Cancer Classification using Support Vector Machines. Machine learning 46 (2002).

[27] K. Kira and L. A. Rendell. 1992. The Feature Selection Problem: Traditional Methods and a New Algorithm. In National Conference on Artificial Intelligence. AAAI Press / The MIT Press.

[28] I. Kononenko, E. Šimec, and M. Robnik-Šikonja. 1997. Overcoming the Myopia of Inductive Learning Algorithms with RELIEFF. Applied Intelligence 7, 1 (1997).

[29] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts. 2013. Under- standing variable importances in forests of randomized trees. (2013).

[30] R. J. Urbanowicz, R. S. Olson, P. Schmitt, M. Meeker, and J. H. Moore. 2018. Benchmarking relief-based feature selection methods for bioinformatics data mining. Journal of biomedical informatics 85 (2018).

[31] C. Spearman. 1961. The proof and measurement of association between two things. (1961).

[32] J. Bergstra and Y. Bengio. 2012. Random search for hyper-parameter optimization. The Journal of Machine Learning Research 13, 1 (2012).

[33] J. Mockus, V. Tiesis, and A. Zilinskas. 1978. The application of Bayesian methods for seeking the extremum. Towards global optimization 2, 117-129 (1978).

[34] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing. 2018. LEMNA: Explaining Deep Learning based Security Applications. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.

[35] W. Guo, S. Huang, Y. Tao, X. Xing, and L. Lin. 2018. Explaining Deep Learning Models – A Bayesian Non-parametric Approach. In Advances in Neural Information Processing Systems.

[36] S. T. Roweis, and L. K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. American Association for the Advancement of Science. 2000.

[37] L. van der Maaten, and G. Hinton. Visualizing data using t-SNE. Journal of Machine Learning Research. 2008.

[38] B. Holmes, A. Drichel, and U. Meyer. Sharing FANCI Features: A Privacy Analysis of Feature Extraction for DGA Detection. In The Sixth International Conference on Cyber-Technologies and Cyber-Systems (CYBER 2021). 2021.

[39] P. Mockapetris. Domain Names - Implementation and Specification. Technical Report. RFC1035. 1987. https://tools.ietf.org/html/rfc1035

[40] FANCI: Feature-based Automated NXDomain Classification Intelligence. Implementation. 2018. https://github.com/fanci-dga-detection/fanci/tree/d6c7d08

[41] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014.

[42] I. Sutskever , O. Vinyals, and Q. V. Le. Sequence to Sequence Learning with Neural Networks. In Twenty-eighth Conference on Neural Information Processing Systems. 2014.

[43] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal Loss for Dense Object Detection. In Transactions on Pattern Analysis and Machine Intelligence. 2020.

[44] R. J. Williams, and D. Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. In Neural Computation. 1989.

[45] N. Heinen, Master thesis. Application Fingerprinting based on System Events using Process Mining, 2021

[46] https://www.bbc.co.uk/news/technology-57658656

[47] https://pandas.pydata.org/

[48] https://scikit-learn.org/stable/

[49] http://spark.apache.org/docs/latest/api/python/