A policy compliance detection architecture for data exchange infrastructures

Lu Zhang

A policy compliance detection architecture for data exchange infrastructures

Lu Zhang

A policy compliance detection architecture for data exchange infrastructures

Lu Zhang

This work was financially supported by the Dutch NWO Research project “Data Logistics for Logistics Data” (DL4LD, www.dl4ld.net), supported by the Dutch Top consortia for Knowledge and Innovation “Institute for Advanced Logistics” (TKI Dinalog, www.dinalog.nl) of the Ministry of Economy and Environment in The Netherlands and the Dutch Commit-to-Data initiative (https://commit2data.nl/).

Cover designed by Lu Zhang

Printed by IPSKAMP, Enschede

ISBN: 978-94-6421-890-9

A policy compliance detection architecture for data exchange infrastructures

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag 19 oktober 2022, te 10.00 uur

door Lu Zhang
geboren te Shandong

Promotiecommissie

Promotores:	prof. dr. ir. C.T.A.M. de Laat	Universiteit van Amsterdam
	dr. P. Grosso	Universiteit van Amsterdam
Copromotores:	prof. dr. ing. L.H.M. Gommans	Universiteit van Amsterdam
Overige leden:	prof. dr. B. Otto	Technische Universität Dortmund
	prof. dr. P.T. Groth	Universiteit van Amsterdam
	prof. dr. T.M. van Engers	Universiteit van Amsterdam
	dr. Z.A. Mann	Universiteit van Amsterdam
	dr. Z. Zhao	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Contents

Chapter 1

Introduction

In the era of big data, nearly everything in daily life is collected and digitalized, from health care records to airplane engine information. According to [1], there are 2.5 quintillion bytes of data created each day in 2021. Utilising and deriving useful information from this massive scale of data have gained significant attention from both academic and industrial sides. The rapid development of machine learning techniques provides a way for learning from a large amount of data and subsequently using the obtained model to improve their decision-making capabilities. For instance, an airline company may use a machine learning model to predict whether an aircraft engine needs to be maintained or not. Digital collaboration between airline operators can increase maintenance efficiency and lower cost. In the health care domain, a doctor can better diagnose a disease and provide personalised advice with a machine learning model [2].

In principle, better results can be accomplished if more training data is available, improving generalisation and accuracy. It is beneficial for multiple organisations to aggregate their data and take advantage of a better result for a common goal. However, those organisations usually have concerns about such digital collaboration. The organisations that possess the same type of data normally compete with each other. This makes them reluctant to share their data and collaborate digitally. They may worry about the exposure of sensitive information to competing companies may pose risk. It is critical to establish digital infrastructures to facilitate secure data sharing and boost the trust of collaborating parties.

There are other works focused on designing and establishing secure data sharing platforms. Ma et al. established a platform to share digital medical records among patients and hospitals. The data is stored in a centralized manner on a public cloud and accessed by mobile devices. It proposed a cryptographic access control model to achieve fine-grained data sharing management [3]. Lu and Cheng also targeted the secure sharing of medical data. The authors proposed a lightweight data sharing platform. The mechanism protected the data privacy

based on identity-based broadcast encryption and proposed a data integrity verification mechanism to enhance security. However, these platforms do not address computing security. [4].

The SDSBT platform developed by Lei et al. made effort to secure the data processing stage. The platform built the mechanisms to support identity authentication and preserve privacy. It also adopted blockchain technology to further secure the data sharing process. The transactions of data sharing are stored in the smart contract for accounting and auditing. The secure data federation process is conducted with trusted execution environment (TEE) technology, which provides secure containers to guarantee data isolation between trusted and normal environments [6]. However, the SDSBT platform only allows data to be gathered and executed on a trusted third party, e.g. the TEE platform. It can not support other collaborating models that fulfil the requirements, such as ensuring data sovereignty, of participating parties. In addition, the influences of various data federation applications on the performance of platforms are not well addressed.

Different data federation applications have different collaboration models, workflows, and security requirements. The collaboration models and workflows depend highly on the participants' multi-lateral trust relationships and local regulations. Some applications may have strict requirements that the shared data should be in the proximity of the providers or cannot go across the country boundary. Similarly, the security requirement also varies with data federation applications. For instance, privacy protection is normally a critical requirement for digital collaborations in the health care domain. In addition, the performance requirement of a data federation application may also vary due to the data volume or algorithm complexity. It is important to develop secure data sharing infrastructures that take the requirements of a concrete application request into account.

The NWO research project data logistic for logistic data (DL4LD) proposed the concept of a consortium governed Digital Data Marketplace (DDM) to facilitate secure and trustworthy data sharing and federation that complies with a customer-defined policy [1]. A policy is a pre-agreed contract among all collaborating parties regulating how data is stored, accessed, shared and transformed. Firstly, data sharing and federation should comply with various local law requirements. For instance, the European Union General Data Protection Regulation (GDPR) privacy law applies to organizations that collect and process personal information within or outside the European Union. The main principles of GDPR include purpose limitation, data minimization, accuracy, storage limitation, integrity and confidentiality and accountability [7]. Secondly, the digital collaboration should also allow the autonomy of participants to make their own rules.

In a DDM consortium, archetypal patterns are defined to describe the workflows of data access and distribution. An archetypal pattern is used to specify or

[1] www.dl4ld.nl

instantiate a particular DDM infrastructure. Distributed security control mechanisms are designed and implemented in a DDM infrastructure to detect policy compliance and therefore enforce the rules of data exchange and federation.

1.1 Research questions

Considering aspect of a DDM that can be optimized based on a collaboration request to share data in a policy compliant way, we can identify that it is necessary to develop criteria to select the archetypal infrastructure patterns based on a concrete policy of a data federation application. In addition, as the community grows, there might be multiple infrastructures already available. Infrastructure reuse is important to lower the cost. Therefore, it is important to give a systemic description of available DDM digital infrastructures and allow the collaborating parties to compare them. To make the organisations more confident to outsource their data, it is critical to provide the expected or guaranteed risk level for their data federation applications. Also, efforts need to be made to design effective security mechanisms for policy enforcement. When considering the life cycle of a data asset, we can see that there are plenty of investigations and implementations of security controls for data in transit and at rest. However, work to enforce the policy during the execution stage is still limited.

To meet the above-mentioned challenges and facilitate policy-driven data sharing and federation with the concept of DDM, we identify our main research question (RQ) as:

How to select application-tailored infrastructure patterns and enhance policy compliance capabilities?

The optimal infrastructure patterns depend on concrete policy-driven data federation applications. Different applications may have different application requests, such as collaboration relationships, data sharing policies and security requirements. We need to develop mechanisms that select best-fit infrastructure patterns with the input of an application request. First, we need to consider the compatibility between the required collaboration models and the provided infrastructure patterns. To get insight into this procedure, we formulate our first sub-question:

- RQ1: *How to map an application request to a best-fit digital infrastructure pattern based on collaboration models?*

To answer this question, we first model the multi-lateral collaboration relationships with numeric representations with mathematical tools. After modelling, we define similarity measures between collaboration models. Thus we can identify the closeness of two collaboration models even if a perfect match does not exist. We also provide a pre-processing algorithm for more commensurable comparison because the mapping process is at the archetypal level. With an infrastructure

pattern fulfilling the collaboration request, it is essential to let the participating parties know how much security can be guaranteed for the delegated data sharing application and rank the available ones. To solve this problem, we formulate our second sub-question:

- RQ2: *How to select an optimal digital infrastructure with minimum risk?*

We propose a risk assessment system based on the STRIDE/DREAD model from Microsoft. The system quantitatively evaluates the remaining risk of an input application workflow after applying the security controls of a digital infrastructure. It therefore allows us to rank available digital infrastructures and select the best-fit one. We also tackle the unavoidable subjective choices of the STRIDE/DREAD model parameters and prove that our proposed system is robust with our experimental results. We conduct a thorough threat analysis for typical data exchange scenarios and a literature study for the available countermeasures. We notice that there are not many promising security mechanisms to enforce the policy during the execution when the algorithm performs on the data in the data federation application. To improve the security level of a DDM digital infrastructure, we formulate our third sub-question:

- RQ3: *How to develop policy compliance detection components during execution?*

Due to the normally competing relationships among the collaborating parties, it is not feasible to allow the data providers to access the source code of the algorithm. The most promising solution seems to be external monitoring. System call traces serve as an interface between the Linux kernel and the applications and are widely used to model the dynamic behaviours of a running program. We monitor the generated system call traces in real time during the execution. In Chapter 4, we propose a methodology to profile and distinguish running algorithms. The component can detect the policy violation if the data is not accessed by the authorised algorithm in the policy.

In addition, we develop a real time machine learning based host-based intrusion detection system (HIDS) by monitoring the system call traces in Chapter 5. This component monitors whether the data federation process complies with the policy regarding data confidentiality or integrity.

To achieve better accuracy and generalisation of a machine learning based IDS model, it is common to get the training or re-training data from the crowd. This enlarges the attack surface and provides the attacker opportunities to insert malicious samples into the training data. It is crucial to enhance the architecture with a model to defend against poisoning attacks. This leads to our final sub research question:

- RQ4: *How to defend against adversarial machine learning attacks for the monitoring components?*

In chapter 6, we propose a sanitization mechanism based on the DBSCAN clustering algorithms that can filter out malicious samples and improve the performance of the architecture.

1.2 Thesis outline and contributions

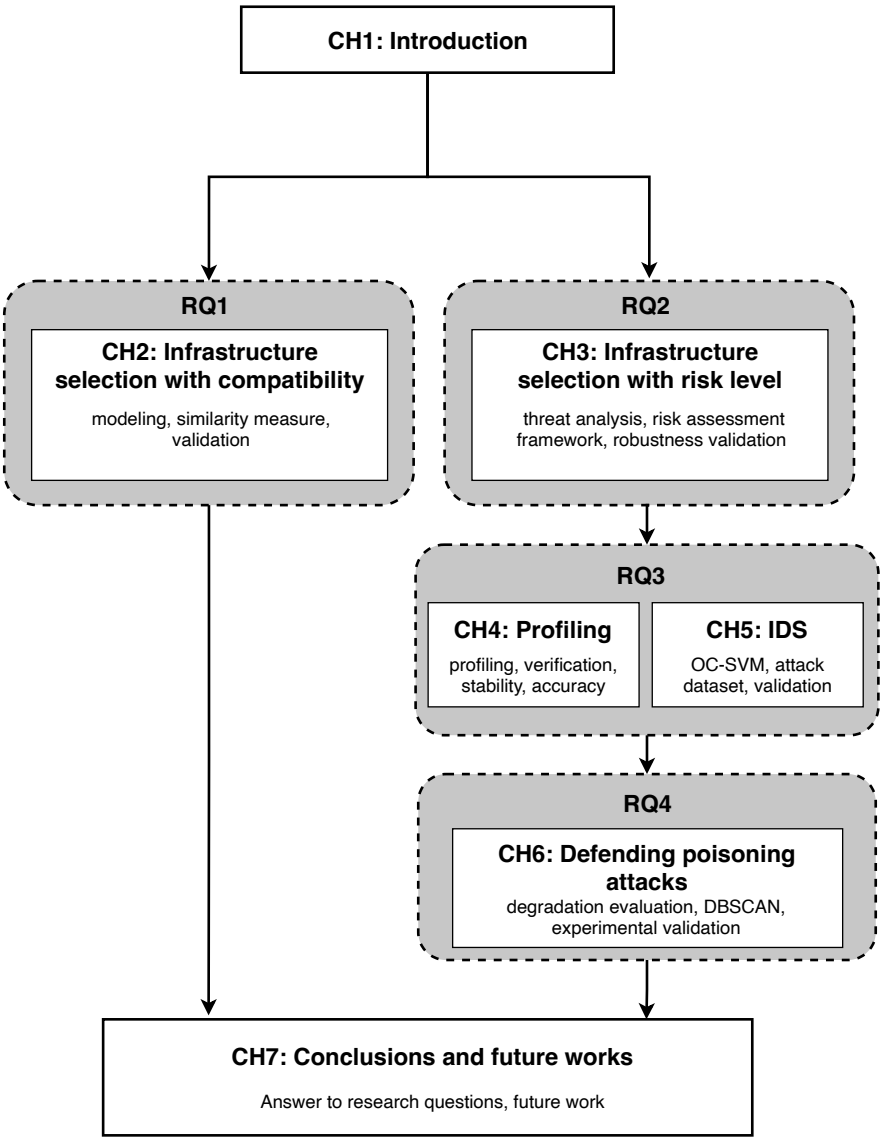


Figure 1.1: Thesis outline

Figure 1.1 visualise the outline of the thesis outline. In Chapter 2 we introduce the DDM concept in more detail and a mechanism that can map an application request to a best-fit infrastructure pattern considering the required

multi-lateral relationships. The mechanisms include mathematical modelling and similarity measure definition. We also validate the approach with a specific logistic use case. In Chapter 3, we present how to select a digital infrastructure with a minimum risk level. An application-dependent threat-analysis driven risk assessment framework is introduced. We also investigate the robustness of the proposed system due to objective choices of input parameters. In Chapter 4, we introduce an architecture that distinguishes algorithms running inside a container only by external monitoring. The architecture comprises source code verification, dynamic profiling in a trusted platform and system call monitoring and profile validation in the execution platform. This can serve as a component in DDM digital infrastructure to monitor policy compliance. The performance of the proposed architecture is also discussed with experimental results. In Chapter 5, we introduce a hybrid IDS by analysing the monitored system calls with OC-SVM as the anomaly detection algorithm. This can be used as an extension of the policy compliance detection architecture mentioned in Chapter 4. In Chapter 6, we address the vulnerabilities of machine learning based IDS to mitigate the threat of poisoning attacks. We investigate the performance degradation of an OC-SVM based anomaly detection model with different poisoning strategies. In addition, we propose a sanitization mechanism and demonstrate its effectiveness with experimental results.

Finally, in Chapter 7, we summarise the answers to our research questions and discuss interesting directions for the future.

1.3 Publications

In this section, we highlight main publications that are highly related to this thesis and describe the authors' contributions. The full list of publications can be found at the end of the thesis.

Lu Zhang "Management of collaborations in digital marketplaces" in proceedings of the 2019 International Conference on High Performance Computing and Simulation (HPCS 2019). L.Zhang proposed the methodology of modeling collaboration relationships.

Lu Zhang, Reginald Cushing, Leon Gommans, Cees De Laat, and Paola Grosso, "Modeling of collaboration archetypes in digital marketplaces" in journal IEEE Access, DOI: 10.1109/ACCESS.2019.2931762. L.Zhang proposed the modeling methodology and performed the experimental evaluation. L.Gommans designed the archetypes for the digital data marketplaces. The remaining authors supervised.

Lu Zhang, Arie Taal, Reginald Cushing, Cees de Laat, Paola Grosso, "A risk level assessment system based on the STRIDE/DREAD model for Digital Data Marketplaces" in journal International Journal of Information Security. L.Zhang developed the risk assessment framework and performed the experimental sim-

ulations. A.Taal provided guidance on the notation and formalization of the equations. The remaining authors supervised.
Lu Zhang, Reginald Cushing, Ralph Koning, Cees de Laat, Paola Grosso, "Profiling and discriminating of containerized ML applications in Digital Data Marketplaces (DDM)" in proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021). L.Zhang proposed the profiling methodology and performed experimental analysis. R.Cushing and R.Koning provided the federated learning use case. The remaining authors supervised.
Lu Zhang, Reginald Cushing, Cees de Laat, Paola Grosso, "A real-time intrusion detection system based on OC-SVM for containerized applications" in proceedings of the 24th IEEE International Conference on Computational Science and Engineering (CSE 2021). L.Zhang designed the structure of the IDS system and conducted the experiments for the performance validation. The remaining authors supervised.
Lu Zhang, Reginald Cushing, Paola Grosso, "Defending OC-SVM based IDS from poisoning attacks" in proceedings of the 2022 5th IEEE Conference on Dependable and Secure Computing/SECSOC workshops. L.Zhang designed the sanitization mechanisms and conducted the experiments for the performance validation. The remaining authors supervised.

1.4 Source code

The scripts and experimental data for each chapter are available at https://github.com/kelsey-1015/phd_thesis_code_data.git

Chapter 2

Select infrastructures with collaboration relationship modelling

DDM infrastructures aim to facilitate secure multi-lateral data exchange applications. On the one hand, a policy is agreed by all collaborating parties, regulating how, where, and what can be done with the traded data for each application. On the other hand, there are a number of available digital infrastructures with different supported collaboration patterns. It is important to allow the participating parties to select a best-fit digital infrastructure for their particular data exchange applications. In this chapter, we answer our RQ1 by presenting an algorithm that identifies the compatibility of any collaboration request and provides digital infrastructures by numerical modelling and similarity measurement. We also propose multiple metrics which allow to evaluate and compare competing DDM infrastructures systemically from a number of dimensions: coverage, extensibility, precision and flexibility. We demonstrate the effectiveness of these metrics in a concrete use case.

This chapter is based on:

- **Lu Zhang**, Reginald Cushing, Leon Gommans, Cees De Laat, and Paola Grosso. "Modeling of collaboration archetypes in digital market places." IEEE Access 7 (2019): 102689-102700.
- **Lu Zhang** "Management of collaborations in digital marketplaces." In 2019 International Conference on High Performance Computing and Simulation (HPCS), pp. 1014-1016. IEEE, 2019.

2.1 Introduction

A potential customer of a DDM normally participates in different data federation applications for different purposes. Both collaborating partners and collaboration requests vary. For instance, airline companies would like to predict the necessity of aircraft maintenance with AI/ML algorithms. They can certainly benefit from a more accurate prediction model by gathering data from the same type of aircraft. However, one company may need to collaborate with a different set of airline partners for different aircraft types. Correspondingly, the policy changes with different trust relationships among involved parties. Each DDM provider may support one or more infrastructure patterns. DDM customers, who would like to collaborate for data sharing and federation for a common goal, may come to a DDM with a concrete application request and seek a best-fitted collaboration archetypal infrastructure pattern. Therefore, it is crucial to develop mechanisms to map any application request into a best-fit infrastructure pattern. It is also important to have a systematic description of those DDMs to allow DDM customers to compare competing ones.

The main contributions of this chapter are:

- We model multi-party collaborations numerically with 3D matrices; We also develop an algorithm to reason on the mathematical representations of collaborations with an effort to match any concrete, complicated collaboration request into the best fit distributed computing archetype from the DDM.

- We define multiple metrics to evaluate a DDM infrastructure from various aspects; namely, we identify *coverage* and *extensibility* as metrics to describe properties and features of a DDM itself; and *precision* and *flexibility* describe the performance associated with a specific user request to the DDM.

The rest of the chapter is organised as follows. Section 2.2 introduces the detailed definition of a DDM infrastructure and collaboration models. Section 2.3 describes the methodology we use to model the multi-lateral collaboration relationships. In Section 2.4, we present an algorithm to compute the mutual similarities between the collaboration models and to select the closest DDM infrastructure. In Section 2.5, we introduce the concrete definitions of 4 DDM evaluation metrics. We further demonstrate the real-world usefulness of our proposed methodologies with a DL4LD use case in Section 2.6. Section 2.7 presents the related work and Section 2.8 concludes the chapter.

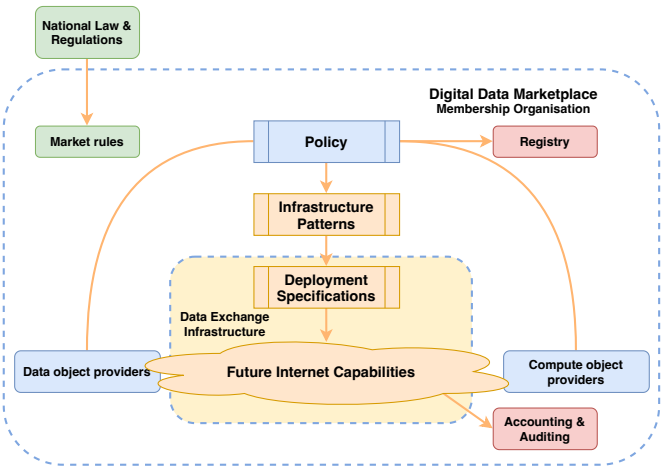


Figure 2.1: A high-level framework of a Digital Data Marketplace.

2.2 Digital Data Marketplace and collaboration models

Figure 2.1 illustrates the high level framework of a DDM. The dotted blue box illustrates a membership organization governing the activities within a DDM. The participating members of a data federation application, including *data object providers* and *compute object providers*, first agree on a *policy*. The *policy*, based on market- and national rules, includes concrete rules such as how the data and algorithm flow, where to perform the computation, the specific use of the outsourced data, and the data security requirement. *Infrastructure patterns* are archetypal patterns defined to describe the workflows of data access and distribution. It can be used to specify or instantiate a particular DDM infrastructure. The *deployment specification* drives the concrete digital infrastructure generation according to the *infrastructure patterns*. Virtual infrastructures can be established with container virtualisation and overlay networks. In a *data exchange infrastructure*, we need to develop and implement distributed security mechanisms to monitor and enforce policy and rights. With the support of the membership organization driven governance model and security mechanisms, the DDM infrastructure can boost the trust of the data providers to collaborate digitally for data sharing and federation.

The main components of a DDM instance are defined as follows:

- **Data object**: A data object is a dataset of a given owner that would be aggregated in the data federation application. Each data object is uniquely identified.

- **Compute object**: A compute object is an algorithm that would execute on the data objects in the data federation application. Each compute object

is uniquely identified and encapsulated in containers for better portability and isolation.

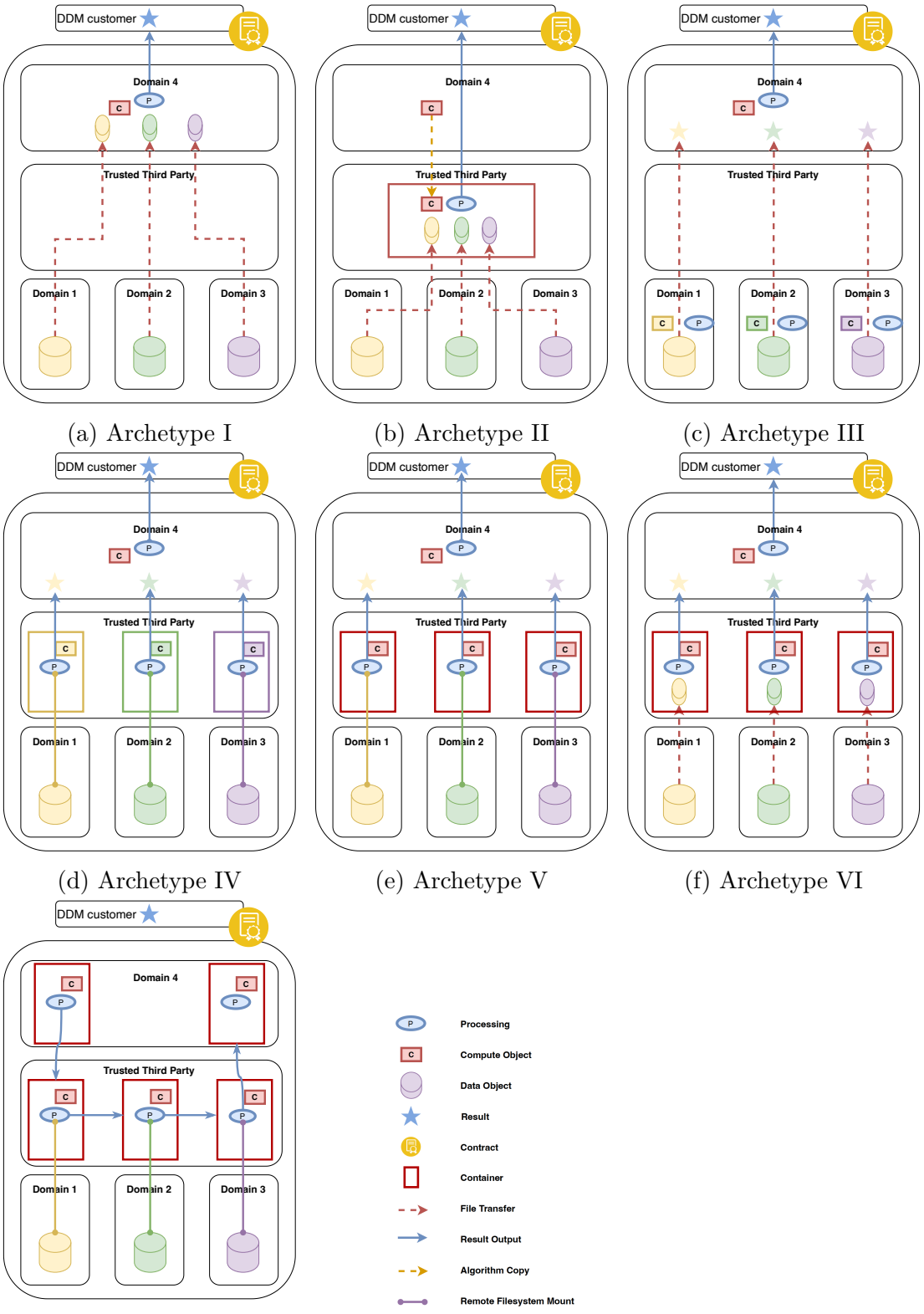
- **Data provider**: A data provider is a party who provides data objects in the data federation application.
- **Compute object provider**: A compute object provider is a party who provides compute objects in the data federation application.
- **DDM customer**: DDM customers are collaborating parties who delegate their data federation applications to a DDM infrastructure. DDM customers can be either data providers or compute object providers.
- **DDM provider**: A DDM provider offers DDM digital infrastructures with security countermeasures with policy enhancement capabilities. There might be a number of available DDM digital infrastructures.

Collaboration models describe multi-lateral collaborating relationships. Normally, collaboration models are defined and described from both the DDM provider perspective and the DDM customer perspective. Here we clarify some terminologies for better explanations. From the DDM provider side, we call those collaboration model archetypes. From a DDM customer side, we call those collaboration models application requests.

2.2.1 Archetypes

According to the definition in [8], archetypes are defined as an original model or type based on which similar things are patterned. We call these collaboration models archetypes because they only capture the main features but are not specific to some details. Those details include the concrete participating parties and the total number of parties for the collaboration. Figure 2.2 illustrates seven collaboration archetypes defined in project DL4LD. All the collaboration archetypes describe the scenarios in that multiple parties, Domain 1, Domain 2, Domain 3 and Domain 4, aggregate their data and compute objects for a result to achieve a common goal. The execution can be performed on the data object provider side, on the compute object provider side of the trusted third party.

In Archetype I illustrated in Figure 2.2a, all the data objects from Domain 1 to 3 are transferred to the compute object provider side (Domain 4). The compute object provider aggregates the data with the compute object and outputs the result. In Archetype II as shown in Figure 2.2b, data objects and compute objects are aggregated in an isolated container in a trusted third party and the final result is sent to the customer directly. In Archetype III as shown in Figure 2.2c, compute objects come to the execution platforms of the data object providers. The data objects are processed locally in separate containers. Intermediate results, illustrated as stars, are then transmitted to compute object provider and



(a) Archetype I (b) Archetype II (c) Archetype III

(d) Archetype IV (e) Archetype V (f) Archetype VI

(g) Archetype VII

Figure 2.2: The collaboration archetypes in DL4LD.

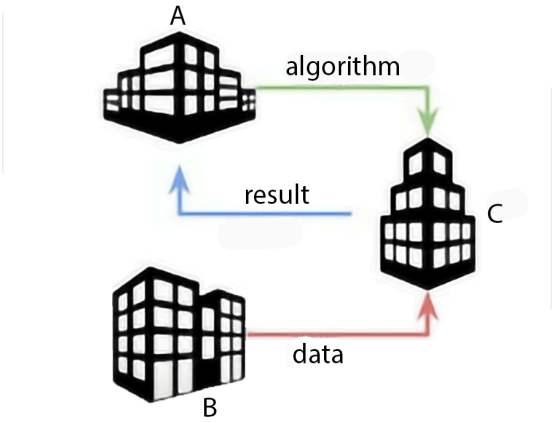


Figure 2.3: An example application request from a potential DDM customer.

aggregated for a final result. Archetype IV, V and VI have similar patterns as shown in Figure 2.2d, 2.2e and 2.2f, in which data objects and compute objects are aggregated in a trusted third party execution platform. The data object from each data object provider is processed in separate containers for an intermediate result. The intermediate results are transferred to and merged at compute object provider side for the final result. Archetype IV uses different compute objects to process data objects from different data providers and the data objects are accessed via remote mounting. Archetype V executes the same compute object for all data objects via remote mounting. Archetype VI also uses the same compute object but the data objects are accessed via direct transfer. For archetype VII illustrated in Figure 2.2g, the data objects and compute objects also meet in a trusted third party and are merged separately. However, the intermediate results are aggregated in a cascaded manner.

Each DDM may support one or more collaboration archetypes to allow potential DDM customers to choose from.

2.2.2 Application request

A potential DDM customer may come with a concrete collaboration request and seek a best-fitted collaboration archetype. We call such collaboration models application requests. Application requests describe how the involving members would like to share their assets in the specific data federation application. Normally application requests are included in the policy and highly dependent on the trust relationships among involving members.

Figure 2.3 describes a concrete application request. Party A would like to perform its algorithm on the data from Party B. However, Party A and B do not trust each other, so they employ a trusted third party C and send their

compute and data to it. Party C executes A's algorithm on B's data and sends the result back to A. A customer-defined application request may comprise both hard requests and soft requests. Hard requests are not negotiable and must be fulfilled in the collaboration process. However, soft requests could be adjusted to fit any existing collaboration archetype better.

2.3 Modelling of multi-party collaborations

To manage and manipulate multi-party collaborations among participating members, we should, in the first place, model them properly. In this chapter, we model them with numeric representations because this would give us standard mathematical tools to further reason about them. For example, we can measure the similarity between an archetype and an application request by computing mutual distance with those mathematical representations.

Firstly, a bilateral collaboration relationship can be fully described by four attributes:

- *Source* is the resource provider;
- *Target* is the resource consumer;
- *Collaboration level* represents the concrete approach of resources exchange;
- *Collaboration scope* describes which resource could be shared between specific parties[9].

Collaborations among participating members may take place in multiple scopes, *data scope*, *algorithm scope* and *intermediate result scope*. More scopes can be added when necessary, e.g. geographical locations.

Also, the collaboration level captures important information about concrete collaborating approaches of each scope between parties. Those features may influence the implementation and performance of underlying digital infrastructures. Table 2.1 explains the concrete collaborating approaches represented by collaboration levels under each scope. These values are ordered and larger numbers indicate a stronger collaboration, which implies more trust between *source* and *target* parties.

In *data scope*, the collaboration levels indicate whether the data is accessed by the *target* with directly data transfer or remote file system mounting. In *algorithm scope*, *partial algorithm* means that *source* only shares the necessary part of its algorithm, dedicated to individual partners, to reduce information exposure. *Entire algorithm* means that the total algorithm is shared for all distributed partners and this certainly requires more trust from *source* to *target*. In *intermediate result scope*, collaboration levels represent whether the intermediate result is aggregated in a parallel or a cascaded manner.

Table 2.1: Collaboration levels under individual scopes.

Collaboration levels	Data Scope	Algorithm Scope	Intermediate Result Scope
1	Remote Mounting	Partial Algorithm	Cascaded Aggregation
2	Direct transfer	Entire Algorithm	Parallel Aggregation

A bilateral collaboration relationship is represented as $\{source, target, scope_1 : level_1 \cdots, scope_n : level_n\}$.

A multi-party collaboration relationship can be modeled as a labeled weighted graph for each scope and represented as its corresponding adjacent matrix.

We denote the graph as $G(V, E, W)$. The set of nodes V represent participating members. The edges set E represent bilateral collaboration relationships and weights W represent corresponding collaboration levels. For example, w_{ij} is the *collaboration level* from member i to member j. We also use labels to indicate whether a bilateral collaboration relationship belongs to hard or soft requests when modeling an application request.

As illustrated in Figure 2.4, a multi-party collaboration relationship among multiple members is effectively modeled as a 3D matrix. Each 2D matrix along the scope-axis is the adjacent matrix of a graph under a specific scope.

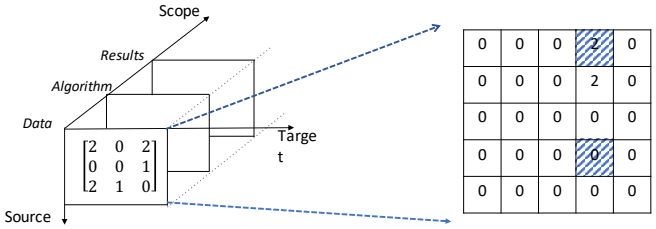


Figure 2.4: Modeling of a multi-party collaboration relationship. On the left we see the relations between sources and targets for the three scopes; on the right, we zoom in on one specific scope, where the crossed-out cells represent hard requests.

2.4 Selection of collaboration archetypes in a DDM

Each DDM may support multiple collaboration archetypes to meet individual application requests. Furthermore, the requests may vary over applications and even vary in time. Therefore it is highly beneficial to develop an algorithm to perform the matching procedure from any incoming application request to a collaboration archetype supported by DDM.

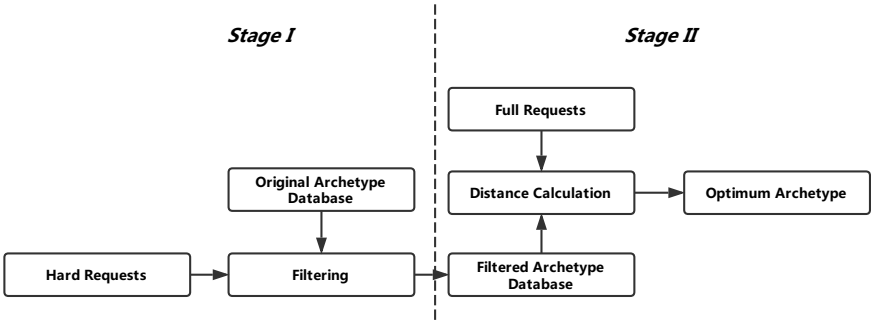


Figure 2.5: Flow chart of the archetype selection algorithm. Stage I is concerned with filtering the archetypes based on hard requests, and Stage II calculating distances to identify the optimal archetype.

We define similarity measures between collaboration models, effectively quantified as a distance metric. A collaboration archetype or an application request can be mapped as a point in a discrete space by calculating their mutual distances.

The algorithm aims to select a collaboration archetype which fully satisfies hard requests from customers and best fits the soft requests. Here "best fit" means the highest similarity, described by the minimum distance to the input application request.

2.4.1 Algorithm overview

The matching algorithm consists of filtering (Stage I) and archetype selection (Stage II). Figure 2.5 describes the algorithm flowchart.

At Stage I, all collaboration archetypes from *Original Archetype Database* are filtered with *Hard Requests* given by a potential customer. After *Filtering*, a subset of archetypes is kept in *Filtered Archetype Database* for further processing and the corresponding searching space shrinks. All the remaining archetypes are acceptable by potential customers for compliance with *Hard Requests*.

At Stage II, we first calculate the distances between *Full Application Request* and the remaining archetypes in *Filtered Archetype Database*. Then select the *optimal archetype* as the one with minimum distance towards *Full Application Request*.

The operational details of each stage are described in the remaining part of this section.

2.4.2 Stage I: filtering with hard requests

An application request includes three scopes, as discussed in Section 2.3, and we perform the filtering stage scope-wise. Suitability under one specific scope does not necessarily mean a completely identical adjacent matrix. For example,

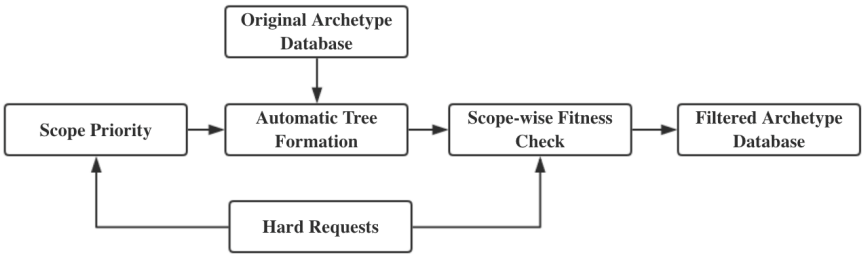


Figure 2.6: Stage I components performing the filtering.

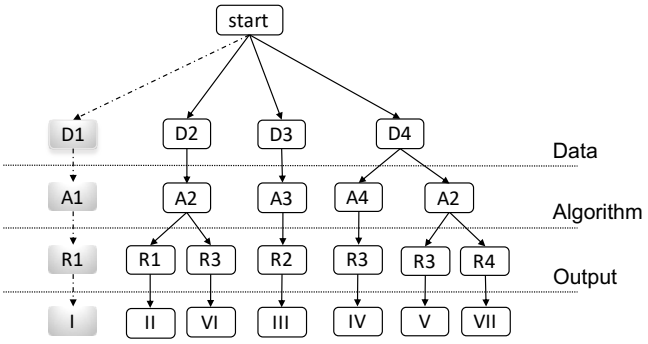


Figure 2.7: An example tree structure formed by the filtering mechanism.

if an application requires no third party, any matrix with all-zero entries in the corresponding positions is qualified. The mechanism is illustrated in Figure 2.6.

Scope Priority depends on the ratio of hard request entries in each scope. Higher priority is achieved for more non-negotiable request entries. A tree structure is automatically generated with inputs of *Scope Priority* and *Original Archetype Database*.

An example tree structure with *Scope Priority* [data, algorithm, output] is shown in Figure 2.7. The path from *start* to a concrete collaboration archetype consists of matrices under each scope and different archetypes may share the same scope-level matrix. If the data scope matrix D1 does not satisfy the hard request, all its children nodes are excluded from the search space.

2.4.3 Stage II: distance calculation and archetype selection

We should define a distance calculation method that effectively measures the dissimilarities among collaboration models. For example, a smaller distance is expected for two collaboration models that are intuitively more similar.

What do we mean by similarities among collaboration models? Firstly, multi-

party collaborations are more similar if more bilateral collaboration relationships are equivalent. Secondly, two bilateral collaboration relationships are more similar if they are identical in more scopes. Thirdly, the existence of a collaboration between parties weights more in our similarity assessment than the level to which they collaborate. The distance calculation method is illustrated in Figure 2.8.

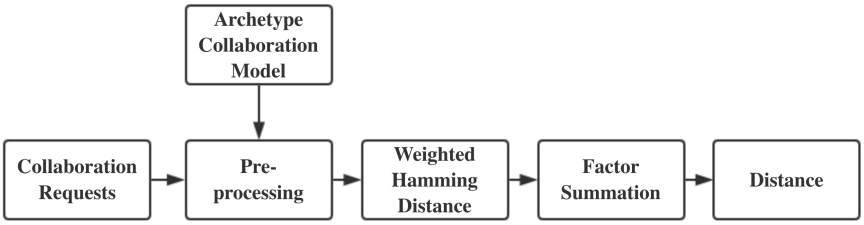


Figure 2.8: Stage II components performing the distance calculation for individual collaboration archetypes.

Firstly, we pre-process both *Application Request* and *Collaboration Archetype* for more commensurate comparison. In *Pre-processing* module, we adjust the dimension of collaboration archetypes in the database to the dimension of the input application request, which is equal to the number of involved parties. Also, we extract all non-zero vectors along the scope axis, each of which represents a bilateral collaboration relationship. We call such vectors *bilateral relationship vector* and each vector can be denoted as $\{source, target, (level_1, level_2, level_3)\}$. Also, *source* and *target* in the *bilateral relationship vector* are represented as the roles of involved parties instead of concrete matrix indexes. The purpose is to eliminate the influences of how those members are positioned into a collaboration matrix to represent their application requests. Those *bilateral relationship vectors* from both application request and collaboration archetype are passed from *Pre-processing* module to the next.

In the *Weighted Hamming Distance* module, we calculate weighted Hamming distances between pairs of *bilateral relationship vectors* with equivalent $\{source, target\}$ [10].

The distance between two collaboration models is achieved by summing up all the individual Hamming distances generated from *Weighted Hamming Distance* module. This is performed in the module *Factor Summation* and the mathematics equation 2.1 is

$$D(\mathrm{CM}_i, \mathrm{CM}_j) = \sum_{s=0}^{P-1} \sum_{t=0}^{P-1} \sum_{k=0}^{S-1} w_k[\mathrm{level}(i)_{s,t,k} \neq \mathrm{level}(j)_{s,t,k}] \tag{2.1}$$

, where CM_i denotes ith collaboration model. It can be either a customer-defined application request or a collaboration archetype supported by a DDM. P, S denote the number of involved parties and number of defined scopes respectively.

$\text{level}(i)_{s,t,k}$ denotes the collaboration level from source s to target t at kth scope in collaboration model i. w_k is the weight of Hamming distance, which is jointly decided by scope priority and collaboration entries.

As discussed previously, the *source* or *target* are represented as roles of members rather than index. So there may be multiple *bilateral relationship vectors* with same $\{source, target\}$. The distance is the minimum value of all results computed from all *bilateral relationship vector* combinations between two collaboration models. We aim to find an optimum archetype for a concrete application request by considering all possible arrangements of members when they put themselves into the matrix to represent their application request.

2.5 Evaluation metrics of a DDM

As discussed in the previous sections, application requests can be matched into most similar collaboration archetypes in a DDM.

For DDM customers, it is interesting to know a-priori how easily one of their application requests can be fulfilled by a particular DDM; for DDM providers it is essential to assess how well they can serve their user base generally.

Suppose that two DDMs all support an equal number of archetypes. They may performance differently according to particular customer-defined application requests or mutual distances among archetypes in the discrete space. For example, if all archetypes of a DDM are concentrated in a small area, it might have less capability to fulfill overall application requests than a DDM whose archetypes are sparsely distributed. We propose multiple metrics that allow a more nearly complete evaluation of a DDM:

- *Coverage*: How well the overall application requests can be satisfied by a DDM with a certain mismatch.
- *DDM Extensibility*: What is the potential richness of a DDM by decomposing and composing collaboration archetypes.
- *Application Extensibility*: How elastic an application request is for achieving a perfect match with a given DDM.
- *Precision*: How well the supported collaboration archetypes of a DDM fit an application request.
- *Flexibility*: How easily an application request can be satisfied generally.

Metrics like *coverage* and *DDM extensibility* are not related to individual requests but represent a general feature of a DDM. However, *precision*, *flexibility* and *application extensibility* depend on both concrete customer-defined application requests and DDM itself.

Besides conceptual definitions, we also define quantization methods for each metric, which we will introduce in detail.

2.5.1 Coverage

With metric *coverage*, we can assess how well the overall application requests can be satisfied by the archetypes of a given DDM. It is intuitively clear that *coverage* highly depends on how we define customer satisfaction. In our work, a DDM customer is considered satisfied if the distance between her application request and the optimum archetype, is not larger than a pre-defined value. We call the parameter *affordable distance* and denote it as D_A.

First, we try to identify the number of overall application requests. Suppose a DDM supports collaboration archetypes $A = \{A_1, A_2, \cdots, A_n\}$. Let P, S, and C denote the number of participating parties, number of defined scopes, and number of collaboration levels respectively. Since the diagonal elements are invalid in a collaboration matrix, the number of entries containing effective collaboration information N_E is

$$N_E = (P^2 - P) * S \tag{2.2}$$

Theoretically, the total number of possible collaboration models with fixed P, S and C is

$$N_T = C^{N_E} \tag{2.3}$$

In reality, this number of feasible collaboration models is much smaller. On the one hand, not all collaboration matrices describe a valid collaboration model. On the other hand, multiple mathematically different collaboration matrices might represent the same collaboration model due to symmetry. We will develop a feasibility validation model in future work.

As illustrated in Figure 2.9, the covered area of ith archetype A_i is modeled as a sphere with radius of the affordable distance D_{A}. The total covered area of multiple collaboration archetypes is the union of individual covered areas.

Ultimately, *coverage* is quantified as the percentage of the application requests that fall into the covered area of supported archetypes, over the total number of overall collaboration models.

$$coverage = \frac{N_{\mathrm{covered}}}{N_T} \tag{2.4}$$

, where N_{covered} denotes the number of application requests that fall into total covered area of the DDM.

Coverage is calculated by computing the distances between all possible application requests and supported archetypes. However, this leads to a heavy computational burden and the complexity grows exponentially with larger C and P.

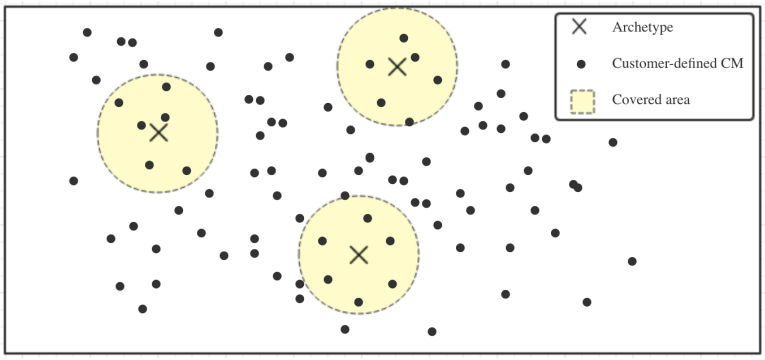


Figure 2.9: Illustration of coverage in discrete space, with archetypes identified as crosses, application requests as dots, and covered areas represented by the yellow circles.

We develop an optimization algorithm to reduce computation complexity. The general principle is to exclude those application requests that fall outside the covered areas before simulation.

Described in algorithm 2.1, $N_{nz,A}$ is the number of non-zero entries in the collaboration matrix of a supported archetype and w_h is the maximum weight of Hamming distance in equation 2.1. We sort overall application requests with the number of non-zero entries in their collaboration matrices and AR_i is the set of application requests with i non-zero entries. $AR_{covered,i}$ denotes number of covered application requests in AR_i.

For instance, if there are four and seven non-zero entries in an archetype matrix and an application request matrix respectively, then at least three entries are not overlapped and contribute to a distance of $3 * w_h$. So there is a limit for the number of non-zero entries in the application request matrix to achieve a distance smaller than D_A. This maximum number of non-zero values $N_{nz,\max}$ is calculated from line 2 to 6 in Algorithm 2.1. Next, the algorithm deals with each AR_i with an increasing number of nonzero entries i and computes a $AR_{covered,i}$. When i is larger than $N_{nz,A}$, there are also limitations about how these entries distribute in the collaboration matrix and the number of iterations could be further reduced as indicated in lines 11 and 12.

2.5.2 DDM extensibility

DDM extensibility measures the potential richness of a DDM by recombining collaboration archetypes.

Each archetype can be decomposed into multiple basic blocks. Each basic block describes collaborations among two or three parties of the same trust domain called primitives. Different collaboration archetypes may share the same primitives. The primitive set of a DDM is the union of primitives of its support-

Algorithm 2.1 Optimization algorithm for *coverage* calculation

Input D_A, $N_{nz,A}$ and w_h
if D_A is even **then**
  $N_{nz,\max} = \frac{D_A}{w_h} + N_{nz,A}$
else
  $N_{nz,\max} = \frac{D_A+1}{w_h} + N_{nz,A}$
end if
for $AR_i \in \{AR_0, AR_1, \cdots, AR_N\}$ **do**
  if $i \leq N_{nz,A}$ **then**
    compute $AR_{covered,i}$ by iterating all request $\in AR_i$
  else
    reduce $AR_i \rightarrow AR_{re,i}$ by restricting matrix deployment
    compute $AR_{covered,i}$ by iterating all requests $\in AR_{re,i}$
  end if
  $AR_{covered} = AR_{covered} + AR_{covered,i}$
end for

ing collaboration archetypes.

Suppose the primitive set of a DDM is $P = \{P_l | l = 1, 2, \cdots, N\}$ and a new collaboration archetype can be constructed as

$$A = r_1 P_1 + r_2 P_2 \cdots + r_N P_N = \sum_{l=1}^{N} r_l P_l \tag{2.5}$$

, where r_i denotes the number of repeating times of each primitive.

DDM extensibility is a measure of the ability to enrich DDM by archetype recombination. It can be measured as

$$\text{DDM } Extensibility = 1 - \frac{N_{A,o}}{N_{A,e}} \tag{2.6}$$

where $N_{A,o}$ denotes the number of original archetypes of a DDM and $N_{A,e}$ denotes the number of possible archetypes with the primitive combination.

2.5.3 Application extensibility

Application extensibility describes the elasticity of an individual application request in achieving a perfect match towards a given DDM. It is quantified as the percentage of unmodified soft entries over all the soft entries in the collaboration matrix. We set the metric as $-\infty$ if a zero distance is not reachable with this DDM by adjusting soft entries in the application. *Application extensibility* is calculated as

$$App\ extensibility = 1 - \frac{N_{m,soft}}{N_{soft}} \tag{2.7}$$

, where N_{soft} denotes the number of soft entries in a collaboration matrix and $N_{m,soft}$ denotes the number of modified soft entries for a perfect match. This metric is related to *flexibility* in Section 2.5.5 . This metric is conditional and is only valid when there are soft requests in the application request.

2.5.4 Precision

Precision describes how well the supported archetypes of a DDM match a specific application request of potential customers. This metric is calculated as

$$precision = 1 - \frac{D_{\min}}{D_{\mathrm{A}}} \tag{2.8}$$

$$D_{\min} = min(Distance(\mathrm{AR}, A_i)) \tag{2.9}$$

, where $D_{\min}$ denotes the distance between an application request AR and the optimum archetype in the DDM, D_{A} is aforementioned *affordable distance*.

If a perfectly matched archetype exists in a given DDM with $D_{\min} = 0$, *precision* regarding to the application request is 1. If $D_{\min}$ is exactly D_{A}, the *precision* turns out to be 0. Otherwise if $D_{\min}$ is significantly larger than D_{A}, *precision* results in a negative value.

2.5.5 Flexibility

The metric *flexibility* measures the strictness of an application request. It is quantified as

$$Flexibility = 1 - \frac{N_h}{N_E} \tag{2.10}$$

, where N_h denotes the number of hard request entries in a collaboration matrix, N_E denotes the number of entries containing efficient information, which can be calculated by equation 2.2.

2.5.6 Intelligent selection algorithm

With the values of proposed metrics for each DDM, the customer will get information about which DDM meets his or her application request best.

Algorithm 2.2 explains the concrete procedure of metric analysis. It aims to select the 'best' DDM who can provide a perfect matched collaboration archetype for the application request with minimum modification effort and relatively higher *coverage*.

First of all, the algorithm sorts all DDMs on *coverage* in a descending order to ensure that the winner always has the highest *coverage* among the qualified members.

In the first step, it analyzes *precision*, described from lines 3 to 8, to check if any DDM can provide a perfectly matched collaboration archetype without any modification. If so, it selects this DDM and ends the procedure.

In the second step, the algorithm checks which DDM can provide an exactly matched archetype by only extending the application request. This is done by analyzing metrics of *flexibility* and *Application Extensibility*. Line 9 checks if there are any soft requests in this application request. Line 10 checks whether the distance can be shortened to zero by just soft request adjustments. If so, the DDM with minimum modification of application request, a minimum value of *Application Extensibility*, is selected.

Finally, the algorithm enriches the DDM candidate pool by archetype recombination and checks whether a DDM in the enriched pool can fully satisfy the application request. They are indicated from lines 16 to 22.

Algorithm 2.2 Intelligent selection algorithm with a specific application request

1: Input application request $\rightarrow$ AR
2: Sort DDMs with *coverage* in descending order $\rightarrow \mathrm{DDM_{rank}}$
3: **for** $\mathrm{DDM}_i \in \mathrm{DDM_{rank}}$ **do**
4: **if** precision(DDM_i, AR) = 1 **then**
5: $\mathrm{DDM}_i \rightarrow \mathrm{DDM_{opt}}$
6: go to *output*
7: **end if**
8: **end for**
9: **if** flexibility(AR) > 0 **then**
10: **if** $\exists$ app extensibility ≥ 0 **then**
11: Select DDM_i with maximum app extensibility
12: $\mathrm{DDM}_i \rightarrow \mathrm{DDM_{opt}}$
13: go to *output*
14: **end if**
15: **end if**
16: Extend $\mathrm{DDM_{rank}}$ by primitive composition $\rightarrow \mathrm{DDM_e}$
17: **for** $\mathrm{DDM}_i \in \mathrm{DDM_e}$ **do**
18: **if** precision(DDM_i, AR) = 1 **then**
19: $\mathrm{DDM}_i \rightarrow \mathrm{DDM_{opt}}$
20: go to *output*
21: **end if**
22: **end for**
23: *output:*
24: Return DDM_{opt}

2.6 Performance evaluation and results analysis

In this section, we evaluate the effectiveness of the metrics with the data logistic use case and DL4LD archetypes we described in section 2.2.

2.6.1 Spatial distribution and mutual distances

We first investigate the spatial distribution of all seven DL4LD archetypes.

We computed the pair-wise mutual distances among all the archetypes. The corresponding results with four parties are shown in Table 2.2. The resulting matrix is upper-triangular because of the symmetry property of distances in space.

Table 2.2: Mutual distances between DL4LD archetypes.

	I	**II**	**III**	**IV**	**V**	**VI**	**VII**
I	0	10	12	12	12	12	12
II	-	0	14	5	4	2	4
III	-	-	0	16	16	16	16
IV	-	-	-	0	1	3	2
V	-	-	-	-	0	2	1
VI	-	-	-	-	-	0	3
VII	-	-	-	-	-	-	0

According to these relative distances, we can visualize the spatial distribution of those archetypes. As illustrated in Figure 2.10, archetypes I and III are more isolated with others and archetype II, IV, V, VI, and VII are clustered together. This computation result is in accordance with the similarity between archetypes.

2.6.2 Metrics evaluation from DDM provider perspective

In this section, we evaluate DDMs, who support different archetype sets by computing and analyzing *coverage* and *DDM extensibility*.

We assign the total seven archetypes into various subsets and suppose each of them is supported by an individual DDM. The number of all possible archetype combinations with a particular set size is shown in Table 2.3. We will compute *coverage* and *DDM extensibility* of all those individual DDMs.

Figure 2.11 shows the *coverage* of each archetype with affordable distance $D_A = 6$ and $D_A = 4$. Every archetype may have different capabilities to serve the overall request space with an identical pre-defined covered area. Archetype III has the highest *coverage*, which implies a higher density of feasible application

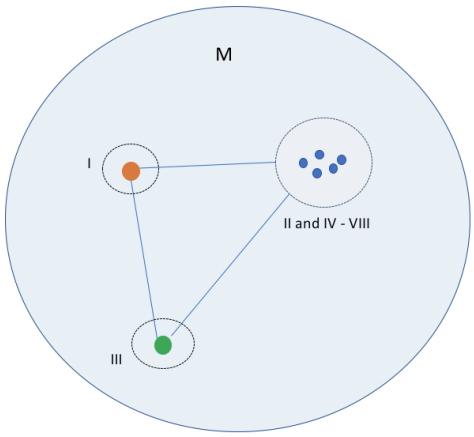


Figure 2.10: Spatial distribution of archetype collaboration models.

Table 2.3: The number of possible archetype combinations with increasing set size.

Archetype Set Size	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Number of Subsets	7	21	35	35	21	7	1

requests in its neighboring space. Also, the value of reasonable distance D_A plays an important role in *coverage*.

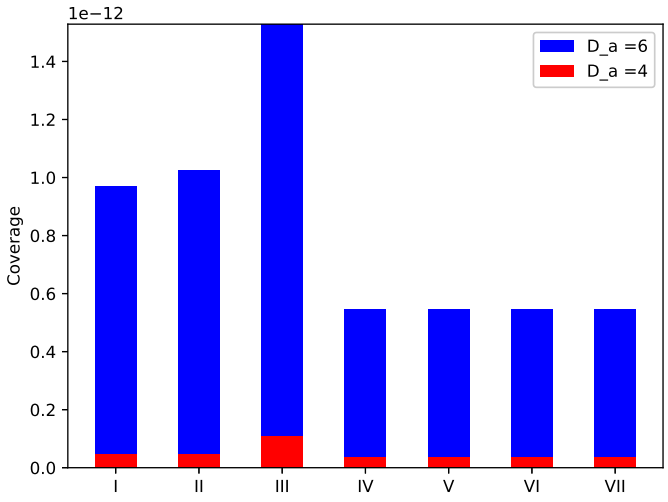


Figure 2.11: Individual *coverage* of each archetype, with $D_A = 4$ and $D_A = 6$ respectively.

A DDM provider may get complete information about its supported archetypes by computing and analysing metric *coverage*. For instance, the DDM provider may expect that implementing archetype III and corresponding infrastructures is more beneficial for the ability to meet overall collaboration requests.

More generally, *coverage* of all other archetype sets are computed with optimisation algorithm discussed in Section 2.4. The corresponding computation results are illustrated in Figure 2.12.

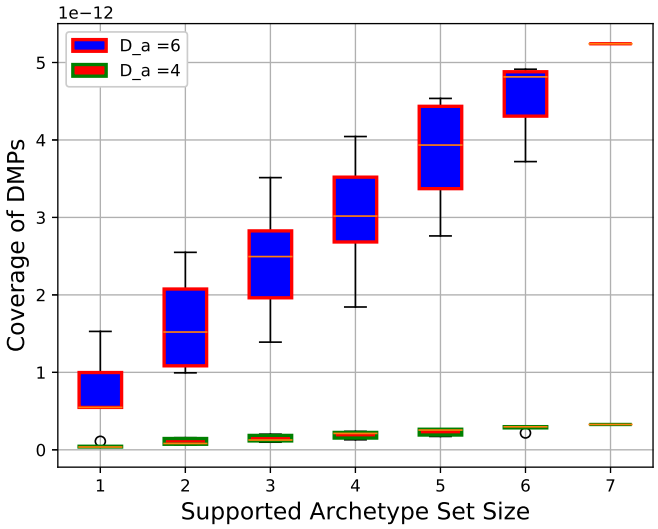


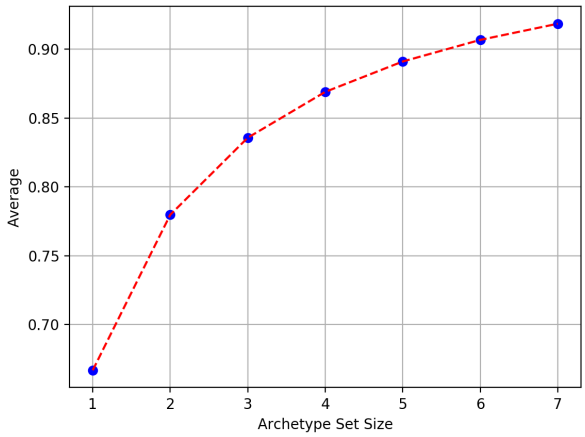
Figure 2.12: *Coverage* as a function of increasing archetype set size with $D_A = 4$ and $D_A = 6$ respectively.

In Figure 2.12, each group represents *coverage* of DDMs supporting archetype sets with equal size. It is not surprising that *coverage* increases approximately in a linear manner with a larger archetype set size. If a DDM provider implements and supports more collaboration archetypes, it certainly has a higher possibility of satisfying more requests. However, it is usually more expensive.

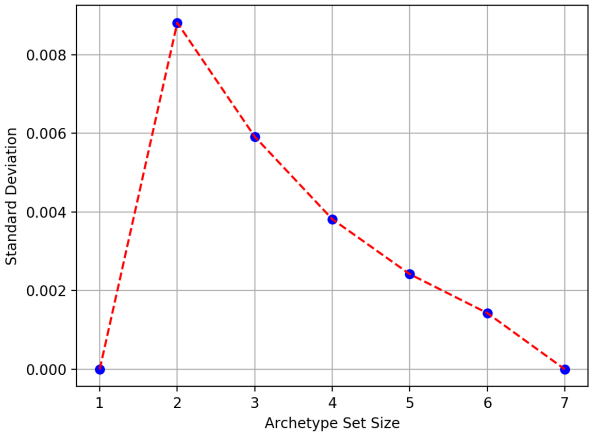
By analyzing data of proposed metrics, a DDM provider may find a better solution between implementation cost and achieved *coverage*. As shown in Figure 2.12, most inter-quartile range boxes have overlap values with their neighbors. This indicates that a DDM provider, which supports a larger number of archetypes, may result in a relatively lower *coverage*. One DDM provider or customer may beneficially select a specific archetype set that has higher *coverage* but lower archetype size.

Similar with *coverage*, *DDM extensibility* is also an evaluation metric defined from DDM provider perspective and independent of particular collaboration requests. It represents the richness a DDM can achieve by constructing new archetypes by primitive composition. In some scenarios, a DDM with lower *coverage* may have higher *DDM extensibility*.

Figure 2.13 shows statistic information about the values of *DDM extensibility* in DL4LD. *DDM extensibility* increases non-linearly with more supported archetypes. The mean value increases faster when the supported archetype size grows from 1 to 4 and becomes relatively stable after the number reaches 5. The standard deviation of *DDM extensibility* for DDMs with equal archetype set size is very small. It is because every archetype in DL4LD has only one primitive.



(a) Mean value of *DDM extensibility* for DDMs with equal archetype set size.



(b) Standard deviation of *DDM extensibility* for DDMs with equal archetype set size.

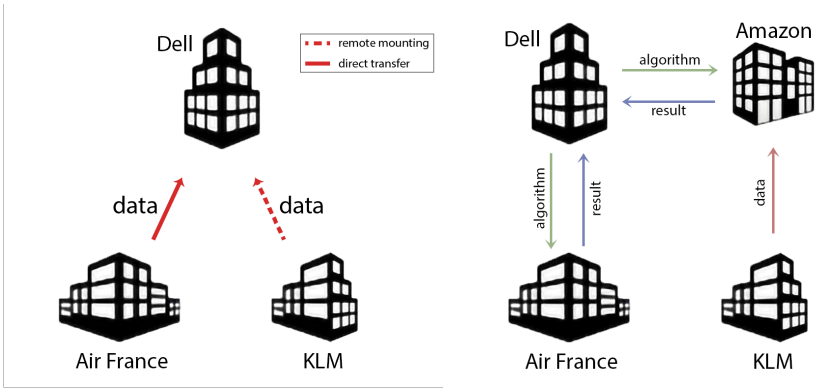
Figure 2.13: DDM extensibility as a function of archetype set size.

For *DDM extensibility*, it may be more interested to investigate how *coverage* or *precision* would increase after DDM extension. We would discuss some of them in next Section.

2.6.3 Intelligent selection of DDMs

This section evaluates multiple DDMs in data logistic use case by computing all the five metrics with two concrete application requests of the airline use case. An optimum DDM is selected for each scenario by analyzing those metrics intelligently with Algorithm 2.2.

Two scenarios describe collaboration among Airline Companies. The involved parties are KLM, AirFrance, and Dell.



(a) Scenario A (b) Scenario B

Figure 2.14: Two example application requests for a digital collaboration of airline companies in DL4LD.

Scenario A is illustrated in Figure 2.14a, both AirFrance and KLM trust Dell in *data scope* and provide their aircraft data to it. Dell aggregates the data and performs its AI algorithm on it. However, KLM prefers sharing its data only by a remote mounting and AirFrance allows the direct transfer, both of which are negotiable and belong to soft requests of this application.

Scenario B is more complicated and is described in Figure 2.14b. One data provider AirFrance does not trust Dell in *data scope* but Dell trusts it in *algorithm scope*. Dell first sends its AI algorithm to AirFrance, who would send the *intermediate result* back after operating on its local data. Another data provider KLM and Dell do not trust each other and agreed to use Amazon as a trusted third party to perform the computation and the *intermediate result* is also sent back to Dell. Finally, Dell can merge the *intermediate results* from both sides and offer a prediction result. All the asset sharing is through direct transfer, and no soft requests involve in this collaboration.

Now we show a concrete example of how to choose a suitable DDM with specific application requests among competing DDMs with the algorithm explained in Section 2.5.6. The application requests are described in detail as scenarios A and B and available DDMs are shown in Table 2.4. The table describes each DDM with its supported archetype set.

Table 2.4: Available DDMs and its supported archetypes defined in DL4LD.

DDM	Supported Collaboration Archetypes
DDM_1	I, II, III, IV, VII
DDM_2	I, II, III, V, VII
DDM_3	I, II, III, V, VI
DDM_4	I, III, IV, V, VII
DDM_5	II, III, IV, VI, VII

Table 2.5 shows the proposed metrics of all DDMs for application request A. Rank those DDMs with *coverage* in descending order and no DDM achieves a full *precision*. Existence of soft requests contributes to a non-zero *flexibility*, which is a pre-condition for calculating *application extensibility*. A positive *application extensibility* indicates that a perfect matched archetype can be provided by the DDM by modifying the application. Finally, DDM_1 is selected as optimum for this specific scenario.

Table 2.5: Metrics evaluation of various DDMs for scenario A.

	DDM_1	DDM_2	DDM_3	DDM_4	DDM_5
Coverage (1e−12)	4.29	4.28	4.26	3.69	3.65
Precision	0.83	0.83	0.83	0.83	-0.67
Flexibility	0.06	0.06	0.06	0.06	0.06
App extensibility	0.5	0.5	0.5	0.5	$-\infty$

Table 2.6: Metrics evaluation of various DDMs for scenario B.

	DDM_1	DDM_2	DDM_3	DDM_4	DDM_5
Coverage (1e−12)	4.29	4.28	4.26	3.69	3.65
Precision	0	0.17	0.33	0.17	0.33
Flexibility	0	0	0	0	0
App extensibility	–	–	–	–	–
Exact match after extension	F	F	T	F	T

The computed metrics of application request B for all available DDMs are shown in Table 2.6. Based on the value of *precision*, the fitness from those five DDMs to application request B is much lower than that of A. Since there is no

soft requests, $flexibility = 0$. Consequently, metric *application extensibility* is invalid under this scenario. Then we further explore whether a perfect match can be achieved by archetype recombination. According to the last row in Table 2.6, DDM_3 is selected as optimum for the ability to offer an exact match and relatively higher *coverage*.

2.7 Related work

DDMs are found in the literature to primarily describe specific online platforms that enable transactions among participating parties[11]. A very well known example is Airbnb [12], which is focused on putting peers, i.e. homeowners and short term renters, in contact. Business to business (B2B) platforms also relies on DDMs to create additional value for participating parties[13][14].

The typical approach to a DDM is that of a platform whereby the DDM provider becomes a trusted party[15]. This model entails that data and algorithms have to move to a secure trusted location provided by the provider. Our model of a DDM is a distributed model where autonomous parties build trust relations between them and move data and algorithms accordingly.

[16] defines DDM as a platform coordinating the supply and demand of digital products, a collection of data containing specific information among DDM providers and customers. They define a distributed business process model and corresponding supported P2P based network [17]. But no work is involved in linking digital agreement with digital infrastructures.

Our work is generically focused on modeling collaborations in DDMs and defining fundamental building blocks in such architectures. This is the first comprehensive step, to the best of our knowledge, towards a systematic description of DDMs.

Toward this general definition of DDMs, we built upon concepts that have been explored before, also in our research group. The two main concepts we adopt are trust and derived from trust policies.

[18] has been the first to identify the need for a thorough and comprehensive definition of trust among participants in the marketplace. They also saw trust as the starting point for the whole chain of resource and services authorization among parties. Subsequent work has further elaborated this concept, as we can see in [19]. We use this idea of trust as the underlying mechanism that allows us to model collaboration across scopes.

Trust is indeed the starting element in creating actionable policies. Policy-driven systems are well known in the literature [20][21]. In the work we presented here, we do not cover the implementation choices needed to translate the collaboration models into actual components, software and hardware, in the DDM. This is the focus of ongoing work.

2.8 Conclusions and future work

This chapter presented how we map a policy driven application into a best-match infrastructure pattern according to compatibility. The requested multi-lateral collaboration relationships and the provided infrastructure patterns are illustrated with archetypes. We modelled the archetypes, which are previously graphically represented, with 3D matrices for easier manipulation and reasoning. With the consistent numeric representations, we can identify the closeness of the collaboration request and the offered infrastructure by defining a proper similarity measurement metric, the Hamming distance. We also showed that the evaluation and comparison of competing DDM infrastructures are allowed and supported by having generic metrics, namely coverage, extensibility, precision and flexibility. We applied our model and metrics to a specific use case to demonstrate how these methodologies are applied in the real world.

Beyond the match between collaboration requests and the supported patterns of digital infrastructures, we also need to consider the security aspect. Chapter 3 describes how we quantitatively assess the remaining risk of a specific policy-driven data exchange application and allow DDM customers to select a digital infrastructure with minimum risk.

Chapter 3

Risk assessment framework

Security is a top concern for data exchange applications, and there is a basic need to assess the level of security guaranteed by the digital infrastructures for any application. Different applications may have different vulnerabilities and different infrastructures are equipped with security countermeasures. In Chapter 2, we answered RQ1 by introducing how to select infrastructures with best-fit collaboration patterns. In this chapter, to answer our RQ2, we propose a risk assessment system that allows to rank infrastructures in terms of security for a specific application. The system identifies threats of an application workflow, computes the severity weights with the modified Microsoft STRIDE/DREAD model [22] and estimates the final risk exposure after applying security countermeasures in the available digital infrastructures. We also conducted a detailed threat analysis for typical DL4LD data exchange applications. We additionally present a method to validate the stability and resolution of our ranking system with respect to subjective choices of the DREAD model threat rating parameters.

This chapter is based on:

- **Lu Zhang**, Arie Taal, Reginald Cushing, Cees de Laat, and Paola Grosso. "A risk-level assessment system based on the STRIDE/-DREAD model for digital data marketplaces." International Journal of Information Security 21, no. 3 (2022): 509-525.

3.1 Introduction

The DDM customers delegate their applications to a DDM infrastructure for better security and sovereignty. Therefore, it is a basic necessity for any DDM customer to estimate the guaranteed security level of such digital infrastructures.

We propose a system to assess the remaining after applying security countermeasures to existing infrastructure to solve this problem. The evaluation results can be used to rank available DDM infrastructures regarding guaranteed security. We proposed a risk assessment system that identifies threats semi-automatically by splitting the input application into transaction lists, assigns severity weights of each threat with the Microsoft STRIDE/DREAD model, and estimates the final risk exposure after applying security countermeasures in the available digital infrastructures.

The Microsoft STRIDE/DREAD model applies *risk attributes*, e.g. Damage and Affected Users, to measure the *likelihood* and impact of exploiting a vulnerability. Most recent work uses the STRIDE/DREAD model to rank threats based on their severities [23, 24, 25]. However, we adopt the model to compute the relative importance of each threat [22]. We also propose the new *risk attributes* for the DREAD model to fit the context of DDM infrastructures and define more fine-grained definitions of these attributes and their corresponding levels in our system to gain more objective assessment results. We additionally present a method to validate the stability and resolution of our ranking system with respect to subjective choices of the DREAD model threat rating parameters. The numerical values of *risk attributes* assigned to threats cannot be constant values during the life span of the system applying the model. Also, the choice of numeric values is not sufficiently objective. It is therefore important to analyse the stability and sensitivity of the STRIDE/DREAD model due to subjective choices of parameters in a real world use case. To quantify the robustness of our system for different values of risk parameters, we use three metrics: two well-known, Normalised Mean Square Error (NMSE) and Kendall's Tau and one we define ourselves, Granularity [26, 27]. The metric Granularity provides us insights into the resolution. Our experimental results show that our risk assessment methodology is stable to subjective choices of the *risk parameters* and able to provide sufficient resolution to discriminate the severity of real world threats in general. Additionally, we observe that methodology performance is highly dependent on the application scenarios and corresponding threat databases.

The rest of the chapter is organised as follows. Section 3.2 describes the system framework from a high level perspective. Section 3.3, 3.4 and 3.5 introduce the concrete methodologies we adopt in the 3 modules respectively in our proposed risk assessment system. Section 3.6 presents how we define metrics to measure the framework stability due to subjective choices of parameters. Section 3.7 and Section 3.8 describe and analyse the experimental results. Section 3.9 presents how our proposed risk assessment system compares with others in the related field. Section 3.10 discusses the conclusions and possible future directions.

3.2 System architecture

A Digital Data Marketplace (DDM) is a digital infrastructure that facilitates secure data exchange and federation. For instance, different DDM parties may want to gather their local data together and run a machine learning algorithm on their joint data to gain benefits from a more accurate prediction model. In the DDM community, there might be multiple DDM infrastructures with well implemented security countermeasures and devices. DDM customers delegate their data federation applications to one of the DDMs for better security governance. Currently, there are two primary typical DDM applications. One is training disease diagnosis models in the health care field; another is to predict aeroplane maintenance necessity for airline companies.

Different data exchange applications suffer from different vulnerabilities. Likewise, different DDM infrastructure providers apply varying sets of security countermeasures. When deploying an application, these varying threats and countermeasures contribute to different final risk levels depending on the DDM it runs in. Our risk assessment system is designed collaboratively to increase transparency and boost the trust of DDM customers in DDM providers.

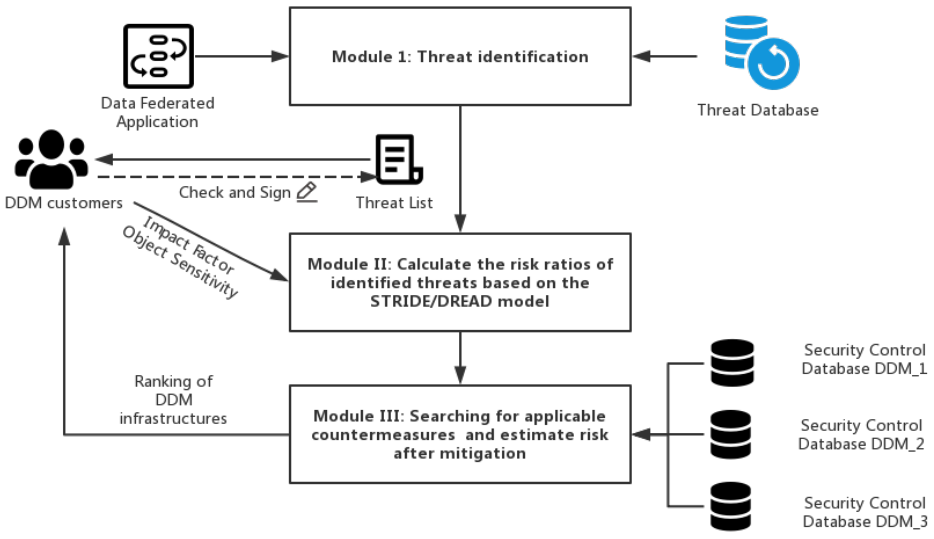


Figure 3.1: The architecture of our application-based risk assessment system.

The risk assessment is performed by a broker, who is essentially a trusted

thrid party and closely cooperating with DDM customers and providers. The system estimates the risk level of all DDMs with respect to an application and provides a ranking of these DDMs to a DDM customer.

Figure 3.1 shows the architecture of the system. A collaboration of DDM customers first feed their applications, which is actually a list of transactions, into the risk assessment system. Module I identifies corresponding threats of the input application automatically by using a pre-constructed *Threat Database*. The *Threat Database* is constructed a-priori by identifying a wide range of threats for typical data exchange applications in DDMs. The *Threat Database* can be updated during the run time of the system, because new threats may occur and some existing threats may become obsolete. The list of identified threats is sent to the DDM customer and each collaborating party checks this threat list. They sign the list if they agree, or go into a negotiation phase if they disagree. Only with all the signatures from the collaborating parties, module II of the risk assessment system will process the approved threat list.

Module II estimates the risk level of each threat in the list with the modified STRIDE/DREAD model from Microsoft [22]. This model considers the possibility of an attack occurrence using 5 *risk attributes* and also the impact of each threat regarding the concrete application. DDM customers also provide *impact factors* and *object sensitivity* as inputs to module II. The *impact factors* reflect how the DDM customer perceives the influence of certain threats on their application. The *object sensitivity* reflects the sensitivity of the shared data of the application perceived by the DDM customer.

Module III matches the threats with corresponding security countermeasures provided by individual DDM providers. This module determines the risk reduction level of each threat provided by different DDMs and calculates the total remaining risk to this application. Finally, this module provides the DDM rankings back to the DDM customers.

3.3 Module I: Application-oriented threat identification

3.3.1 Mapping between Microsoft STRIDE model and security features

The STRIDE model is a threat modelling tool developed by Microsoft for analysing security flaws for cyber-security systems [22]. It groups threats into six categories: Spoofing (S), Tampering (T), Repudiation (R), Information disclosure (I), Denial of service (D), Elevation of privilege (E) [22]. All the identified threats for a data-exchange application belong to at least one of these categories.

We define a mapping of the threat categories in the STRIDE model onto

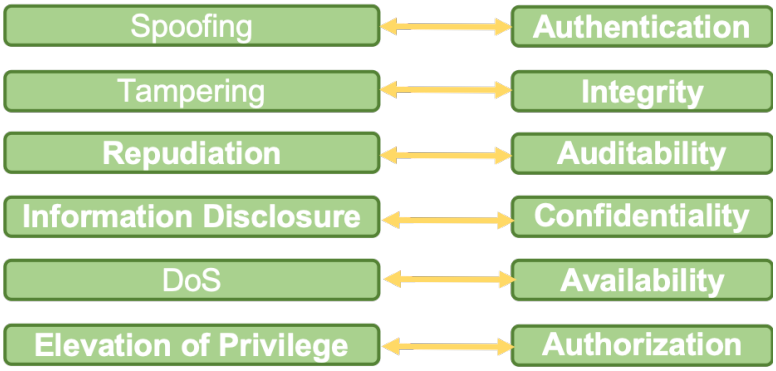


Figure 3.2: Correspondence between the threat categories in the STRIDE model (left) and the security features (right).

more generally understood security features, see Figure 3.2 [28]. So it is more intuitive and comfortable for the DDM customers to consider the impacts of each threat category for their applications. In this way, the DDM customer does not need to have background knowledge of the STRIDE model. A threat may have distinct risks for different applications because applications may have various security goals. For example, the threats belonging to the category of Information Disclosure damage the confidentiality of the shared data rather than integrity.

3.3.2 Applications of DDM-governed data exchange

As illustrated in Figure 3.1, DDM customers provide their application to the risk assessment system. In the DDM infrastructure, it is normal to use transaction lists to represent a data federation application. In this module, the input DDM application is split into multiple transactions. An example transaction list is shown below, with DO, CO, IR, DP, CP representing data objects, compute objects, intermediate results, data providers, compute object providers respectively.

Transactions of an example DDM data federation application
Trans 1: Trusted third party accesses DO from DP via remote mounting
Trans 2: Trusted third party accesses CO from CP via direct transfer
Trans 3: Trusted third party processes CO on DO with feature multi-tenancy, generating IR
Trans 4: CP accesses IR via direct transfer
Trans 5: CP processes on a trusted third party

We characterise each transaction as an attribute tuple:
$\langle stage, source, target, object, feature \rangle$

Source and *target* are DDM parties. *Feature* describes important aspects for threat identification. Direct transfer and remote mounting are two *features* for a transaction with *stage* transmission. For instance, the attribute tuple of trans 1 becomes:
⟨transmission, DP, Trustedthirdparty, DO, remote mounting⟩.

Object sensitivity

The risk assessment system requires *object sensitivity*. It determines the potential damage of a threat. The *object sensitivity* depends on an individual application. For example, data objects in health-care applications usually are more sensitive than others because they may contain private information of patients.

Impact factors

Due to the concrete security goal of an application, the impact of each threat category in the STRIDE model varies. The DDM customers are required to assign *impact factors* for each threat category based on its corresponding security feature. According to the work in [29], there are 5 levels, which are critical (1), high (0.75), medium (0.5), low (0.25) and none (0), to scale the *impact factor*. The *impact factor* indicates the degree of concern of DDM customers about each threat category. A DDM customer supplying the *impact factor* critical asks for the greatest concern and priority for a specific threat, and with *impact factor* none, the DDM customer has no concern about a given threat.

3.3.3 Threat modelling

Here we introduce a general methodology for identifying threats for applications in DDMs. Every application can be split into a sequence of transactions, each of which can be represented by a 5-tuple. The threats of each transaction can be identified primarily based on its *stage* and *feature*. The threats of an application are the union set of threats for all its transactions. In addition, DDMs, as distributed platforms for data federation applications, are based on virtualisation technologies for better isolation. We consider common vulnerabilities of virtualisation when modelling threats for a given application. For instance, threats caused by the multi-tenancy *feature* [30].

We classify the threats of a DDM application into 3 *stages*. Stage I is data in storage, and the main concern is confidentiality, availability, privacy of asset objects in storage. Stage II is data in transmission, which is related to issues such as end-to-end communication security. Stage III is data in execution, and it focuses on whether the procession by an algorithm on the data complies with the agreed policies.

There are some threats mainly depending on the attribute *stage* of the transaction. For example, the threat of 'data object leakage during end-to-end transmission' exists in nearly all transactions with *stage* of transmission. Attacks like 'man-in-the-middle' and 'eavesdropping' may exploit these threats. Similarly, the threat of 'malicious compute objects during execution' is also common for transactions with *stage* attribute of execution.

However, some threats are dependent on distinct *features* of a transaction. For instance, mounting a local file may give a 3rd party sufficient permission to suffer from the threat of data object tampering. The *feature* multi-tenancy indicates data objects are processed individually in separate containers on the same physical 3rd party platform. An example threat for this *feature* is the 'denial-of-service attack' by one of the malicious co-tenant containers.

According to the approach introduced before, we conduct threat modelling for DDM applications semi-automatically according to a pre-defined dynamic threat database.

ThreatName	Stage	Category	Archetype	DP	AC	SL	AU	ID
IP spoofing	II	S	ALL	SO	H	M	H	M
Identity spoofing: via remote data access	III	S	IV, V, VII	SO	H	L	M	H
Insecure data deletion	III	ID	ALL	SO	M	L	M	H
Malicious compute: Data Disclosure	III	ID	ALL	SO	L	H	H	M
Unauthorized disclosure: Eavesdropping	II	ID	ALL	SO	H	H	M	H
Weak Access Control	I	ID	ALL	SO	H	H	L	H
Malicious compute: high result correlation	III	ID	III	SO	L	H	H	M
Encryption Keys Leakage during exchange	II	ID	ALL	TOP	H	L	H	H

Figure 3.3: A screenshot of a SQL threat database for a DDM use case.

Figure 3.3 shows the screenshot of a pre-constructed SQL Threat Database for a DDM use case. Each a-priori identified threat has 9 different attributes, namely, *threatName, stage, category, archetype, DP, AC, SL, AU, ID.*

The *stage* describes in which stage this threat occurs. The *category* refers to the threat categories in the STRIDE model, as discussed in Section 3.3.1. It indicates to which category this threat belongs. *DP, AC, SL, AU, ID* are assigned values for the *risk attributes* for the given threat. The *archetype* describes the collaborating relationships among DDM members and each application follows at least one *archetype* [31]. The concrete *archetypes* were described in Chapter 2.

3.4 Module II: Risk assessment of an individual threat

Once threats have been identified by the methodology described in section 3.3 and approved by all collaborating parties, this module computes the application-dependent *risk ratio* of each threat with the modified Microsoft DREAD model.

The DREAD model is commonly used to rank individual threats based on their severities. In module II, we adopt the concept of the DREAD model to compute the relative importance of each threat according to the estimated risk level. Furthermore, we redefine five *risk attributes* to fit the context of DDM applications and increase objectivity in the assessment procedure.

3.4.1 Original DREAD model

The original DREAD part of the STRIDE/DREAD model proposed by Microsoft is used to assess and rank threats in terms of their risk [22]. It defines 5 *risk attributes* to estimate the probability of an exploitation of a vulnerability from distinct aspects. These attributes are Damage (D), Reproducibility (R), Exploitability (E), Affected users (A), Discoverability (Di) [32].

- Damage (D): How much are the assets affected?
- Reproducibility (R): How easily the attack can be reproduced?
- Exploitability (E): How easily the attack can be launched?
- Affected users (A): What's the number of affected users?
- Discoverability (Di): How easily the vulnerability can be found?

Each *risk attribute* is scaled into 3 qualitative levels as high, medium and low. Due to the property of a concrete threat, one of the 3 qualitative levels can be assigned for each *risk attribute*. All the five aspects need to be considered to assess the risk of a threat. The threat risk ranges from 0 to 10 and the DREAD model uses 3 integers 0, 5, 10, to represent the 3 corresponding levels numerically. In the STRIDE/DREAD model, we represent a threat by the following 5-tuple $(D_{t_i}, R_{t_i}, E_{t_i}, A_{t_i}, Di_{t_i})$ with D_{t_i},R_{t_i},E_{t_i}, A_{t_i}, $Di_{t_i} \in \{0, 5, 10\}$ of numeric numbers. The risk, represented as a metric called *risk score* $rs(t_i)$, is quantified as an average of the numeric values of those five *risk attributes*:

$$rs(t_i) = \frac{1}{5}(D_{t_i} + R_{t_i} + E_{t_i} + A_{t_i} + Di_{t_i}) \tag{3.1}$$

According to the *risk scores* of the threats, the DREAD model can rank all the threats regarding their risk.

However, the description of each risk parameter is obscure and there are no concrete definitions of each level for the original DREAD model. This probably increases the degree of subjectivity when assessing the risk level of a single threat with the original DREAD model.

3.4.2 Modified DREAD model for DDMs

We redefine five *risk attributes* and corresponding risk levels to better meet the requirement of the DDM applications. For example, we address the importance of monitoring and potential trust among collaborating parties in a DDM instance. Table 3.1 shows the defined *risk attributes* and qualitative descriptions of 3 scaled levels.

Table 3.1: *Risk attributes* of modified DREAD model and corresponding qualitative descriptions of 3 levels.

Risk Attributes	**Damage Potential (DP)**	**Accessibility (AC)**	**Skill Level (SL)**	**Affected Users (AU)**	**Intrusion Detectability (ID)**
Low	Low Data Sensitivity	By collaborating parties	Advanced skills	One party member	Detectable without monitoring
Medium	Medium Data Sensitivity	By collaborating parties or any trusted 3rd party	Malware existing in Internet or using attack tools	Partial party members	Detectable by monitoring
High	High Data Sensitivity	By outsiders of DDMs	Simple tools	All party members	Very hard to detect even by monitoring

Damage Potential (DP) describes the damage caused if a threat occurs. The assets of DDM applications are data objects, compute objects and intermediate results objects, which we have discussed in Section 3.3.2. The *object sensitivity* assigned by the DDM customer determines the corresponding level of the *risk attribute* Damage Potential (DP). For some threats like encryption key leakage during exchange, the DP is always set as the highest level regardless of the *objective sensitivity* of the application. In Figure 3.3, we use "TOP" to represent such threats for attribute DP.

Accessibility (AC) describes who can perform attacks to exploit a threat. If collaborating members of a DDM can only perform the attacks, the attribute is scaled as low due to the mutual trust among them. If a 3rd party of an application can also exploit the threat, i.e. more risk is included, AC is scaled as medium. The highest risk occurs if one entity can perform this attack, including malicious

parties outside the DDM.

Skill Level (SL) defines what skills are needed to exploit this threat. The probability is much lower if this exploitation requires complex programming or hacker skills. The risk is highest, scaled as high, if it just requires simple tools or even a web browser.

Affected Users (AU) is scaled into different levels according to how many collaborating parties are affected if a threat occurs.

Intrusion Detectability (ID) describes how easy monitoring tools can detect the exploitation of this threat. A threat is more severe if its exploitation is more difficult to detect, which indicates a higher success rate of attacks and more resulting damage.

Security experts can determine these *risk attributes* a-priori and reference information can be found in some public vulnerability databases, for instance, CAPEC [33]. Damage Potential (DP) and Affected Users (AU) are application-dependent and subjective in nature. Currently, we use 0, 5, 10 to represent the 3 *risk attribute* levels numerically. We further discuss the influence of other numeric representations on the stability and resolution of our methodology.

Integrating the application-dependent *impact factors* described in Section 3.3.2, we calculate the risk score $rs(t_i)$ of a threat t_i as the product of a *likelihood* LH and an *impact factor* IF:

$$rs(t_i) = LH(t_i) \cdot IF(t_i) \tag{3.2}$$

The *likelihood* $LH(t_i)$ and the *impact factor* $IF(t_i)$ are obtained as follows:

$$LH(t_i) = \frac{1}{5}(DP_{t_i} + AC_{t_i} + SL_{t_i} + AU_{t_i} + ID_{t_i}), \tag{3.3}$$

where DP_{t_i}, AC_{t_i}, SL_{t_i}, AU_{t_i}, ID_{t_i} denote the numeric values of the five *risk attributes* in Table 3.1 for threat t_i, and $IF(t_i)$ equals to the *impact factor* of the threat category in the STRIDE model that threat t_i belongs to.

We must observe that the *likelihood* LH is a linear combination of the five *risk attributes*. By the choice of a linear combination, Microsoft treats all attributes equally.

According to Equation 3.3, we can compute the *risk score* of each threat for the application, which represents the risk of each threat. A higher *risk score* indicates a more dangerous threat for the concrete application.

To determine the relative importance, we define a *risk ratio* rr of each threat t_i in the threat list of the application. This is calculated as follows:

$$rr(t_i) = \frac{rs(t_i)}{\sum_{t_i \in T} rs(t_i)}, \; with$$
$$\sum_{t_i \in T} rr(t_i) = 1, \tag{3.4}$$

where $rs(t_i)$ denotes the *risk score* of threat t_i, T denotes the threat list of the application identified by module I.

3.5 Module III: Risk mitigation and risk level evaluation

Module III of the risk assessment system matches security countermeasures to identified threats for an application, computes the mitigation level of each threat and calculates the total remaining risk of the application.

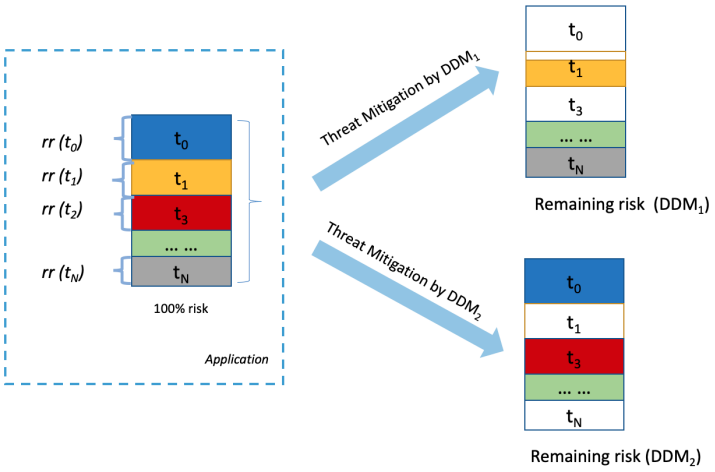


Figure 3.4: Functionality of module III. On the left side, we see the input of module III, the identified threats of an application with corresponding *risk ratios*. On the right side, we see the remaining risk of each threat after applying security countermeasures by the DDMs. White areas indicate zero risk, coloured areas indicate the remaining risk.

As illustrated in Figure 3.4, the input of module III is a list of threats with corresponding *risk ratios* $rr(t_i)$. Those threats constitute the original 100% risk of the application without any mitigation from DDMs and the proportion of each threat is equal to its *risk ratio* calculated by Equation 3.4. According to information of security countermeasures provided by DDMs, this module ranks DDMs regarding total remaining risk for the application.

3.5.1 Security countermeasures matching and threat mitigation

As shown in Figure 3.1, DDM providers publish the *CM Database*, a database of supporting security countermeasures. The risk assessment system accesses each *CM Database* and matches suitable security countermeasures for each threat of the application. Figure 3.5 illustrates the matching procedure. The module checks both the feasibility and necessity of applying a security mechanism to an application. Necessity indicates whether a security countermeasure can mitigate one or multiple threats identified for the application. Feasibility means whether a security countermeasure can fit the data type or data volume of the shared objects of the application. For instance, the watermarking techniques are only applicable to data objects of images. In Figure 3.5, an arrow from cm_j to t_i indicates countermeasure cm_j is both feasible and necessary to apply to the application to mitigate threat t_i. The matching from security countermeasures to threats can be one-to-one (1-1), one-to-multiple (1-N) or multiple to one (N-1). A one-to-multiple mapping indicates a security countermeasure is capable to mitigate multiple threats. A multiple-to-one mapping means multiple security countermeasures apply to only one threat.

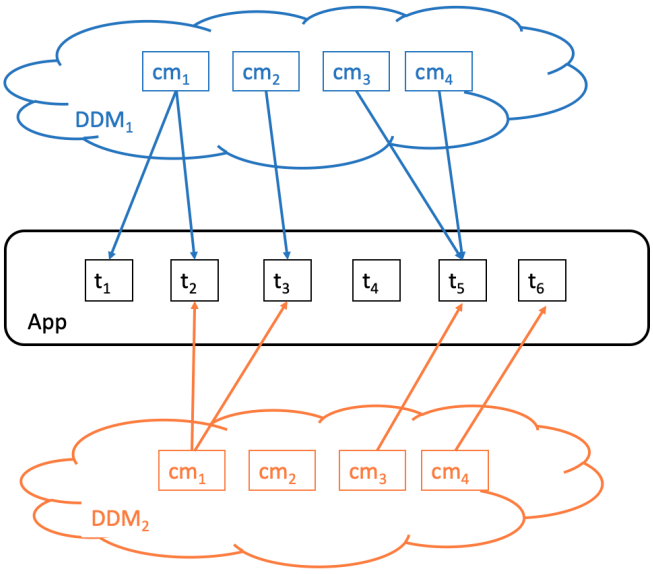


Figure 3.5: Threats mitigation by security countermeasures of each DDM provider. This results into a mitigation list of original threats for each DDM.

Every DDM_k applies a mitigation factor $f_{m;k} : CM_k \times T \to [0,1]$ by multiplying the $rs(t_i)$ of each $t_i \in T$ with $f_{m;k}(cm_j, t_i)$ for all $cm_j \in CM_k$, if cm_j does not apply to threat t_i, we define $f_{m;k} = 1$, i.e. it leaves $rs(t_i)$ unchanged; if cm_j can fully mitigate threat t_i, we define $f_{m;k} = 0$.

The mitigation factor $f_{m;k}$ is a measurement for the reduction of *likelihood* after applying a security countermeasure to a threat. For instance, it is much more difficult to perform an *eavesdropping* attack after end-to-end encryption than on plaintext. For a single threat, two factors influence the risk of a threat, *likelihood* LH and *impact factor* IF, according to Equation 3.3. The impact stays the same and the *likelihood* is reduced by $f_{m;k}(cm_j, t_i)$. That's why $f_{m;k}(cm_j, t_i)$ is serving as a scale factor of original threat risk score, subject to constraint $0 \leq f_{m;k}(cm_j, t_i) \leq 1$.

Security countermeasures $cm_j \in CM_k$ and identified threat t_i jointly determine the value of $f_{m;k}(cm_j, t_i)$. In DDM applications, monitoring techniques usually play a vital role to detect policy breaches. Hence we classify the security countermeasures into two categories, namely, prevention countermeasures and detection countermeasures. Prevention countermeasures are those security mechanisms aiming to stop an attack from occurring and prevent a policy breach, e.g. data access control and cryptographic mechanisms. Detection countermeasures are those aiming to detect any attacks or policy breaches during the data exchange procedures, e.g. system call monitoring. The *mitigation factors* in our risk assessment system for countermeasures that apply to a threat are defined as:

$$f_{m;k}(cm_j, t_i) = \begin{cases} 0, if\, t_i\, is\, prevented\, by\, cm_j \\ R_d, if\, t_i\, is\, detected\, by\, cm_j \end{cases}$$

R_d denotes the real time detection rate of the applied monitoring technologies. R_d is provided in the DDM countermeasure database offered by the DDM providers. Normally, DDM providers can achieve the estimated detection rate from IDS designers according to experimental evaluations. It is also possible for DDM providers to adjust R_d of a concrete countermeasure based on the historical data when apply to other DDMs.

For security countermeasures that prevent a threat, we assume the threat can be adequately mitigated and set the value as 0. It is also possible to recalculate *risk attributes* after applying the countermeasure according to Table 3.1 and determine the corresponding $f_{m;k}(cm_j, t_i)$. For security countermeasures that detect an intrusion, the *mitigation factor* is equal to the accuracy rate, denoted as R_d, of the implemented monitoring detection and algorithm. The value of R_d is typically gained with the historical data.

If multiple countermeasures are matched to a single threat, we need to consider interactions and redundancy among those security countermeasures when determining the joint mitigation level. The multiple security countermeasures are chained and the *joint mitigation factor* is calculated as:

$$F_{m;k}(t_i) = \prod_{j=1}^{N_k} f_{m;k}(cm_j, t_i) \tag{3.5}$$

$F_{m;k}(t_i)$ is the joint mitigation factor of threat t_i, $F_{m;k}(t_i) \in [0, 1]$. N_k denotes the

total number of security countermeasures for a threat t_i in CM_k and $f_{m;k}(cm_j, t_i)$ denotes the mitigation factor of countermeasure cm_j to threat t_i.

3.5.2 Total risk level of an application

The remaining risk of a threat after mitigation by DDM_k is computed as:

$$rr_{remain;k}(t_i) = rr(t_i) \cdot F_{m;k}(t_i) \tag{3.6}$$

$rr_{remain;k}(t_i)$ denotes the remaining risk of threat t_i after applying security countermeasures of DDM_k; $rr(t_i)$ denotes the original *risk ratio* of threat t_i.

The *risk level RL* of an application A provided by DDM_k is calculated as the summation of the remaining risk $rr_{remain;k}$ of all threats:

$$RL(A, DDM_k) = \sum_{t_i \in T} rr_{remain;k}(t_i) \tag{3.7}$$

Module III of the risk assessment system computes the risk levels for potential DDM providers and provides the rankings to DDM customers.

3.6 System stability due to subjective choices

Most risk assessment systems suffer from the problem of being too subjective. In this section, we investigate how the system ranking results fluctuate due to the subjective choices of the parameters. We call this the *stability* of the risk assessment system.

The subjective choices mainly occur in module II. As mentioned in Section 3.4, the STRIDE/DREAD model maps the 3 qualitative levels of each *risk attribute*, namely, low, medium, high, into 3 numeric values [0, 5, 10] with a bijective function. A function $f : X \to Y$ is bijective, if for all $y \in Y$, there is a unique $x \in X$ such that $f(x) = y$. The numeric combination of [0, 5, 10] indicates an equal risk increase between adjacent qualitative levels for all *risk attributes*, which fits the majority of risk assessment scenarios. However, it is also possible and reasonable to adopt other numeric values, e.g. [0, 1, 2] or even [1, 3, 8] entailing non-equalised risk increase. In the following, we will explore two questions: i) To which degree can numeric values be chosen objectively depending on system physical effects? ii) How these chosen numeric values relate to the system output, which is the risk rankings of DDMs.

3.6.1 Physical effect of value vectors

We put the numeric values in a 3-dimensional vector and name it as a *value vector*.

Every threat has a tangible effect on the system. With physical effect, we mean the measurable effects of the threat *risk attribute* values on the components in DDMs. Different *value vectors* express different physical effects. A *value vector* determines the quantitative risk increase between subsequent qualitative levels of each *risk attribute*, as explained in Table 3.1. For instance, the *risk attribute* Accessibility has three levels, which are only by consortium parties, by both consortium parities and 3rd party and by outsiders. If the system adopts a *value vector* [0, 5, 10], it means the risk level increases in equal steps as increasing qualitative levels. However, a *value vector* [1, 3, 8] implies that there is a higher risk increase from medium to high than from low to medium. This higher increase is because an attack from outsiders is considered more serious.

The choice of *value vector* should, in the first place, be determined by how the risk is supposed to increase between subsequent qualitative levels. We can classify those *value vectors* into two categories, namely, evenly spaced and non-evenly spaced *value vectors*. Evenly spaced *value vectors* indicate equal steps in risk increase between adjacent levels. If we represent a *value vector* as $[v_{i,1}, v_{i,2}, v_{i,3}]$, an evenly spaced *value vector* is a 3-term arithmetic progression $v_{i1} = a$, $v_{i2} = v_{i1} + \delta$, $v_{i3} = v_{i1} + 2\delta$. These evenly spaced *value vectors* are more interesting for us because they fit for most scenarios and share the same physical effect of the original *value vector* from the Microsoft STRIDE/DREAD model. Non-evenly spaced *value vectors* include some distortion and have different steps between neighbouring *risk attribute* levels. If one opts for an evenly spaced *value vector*, there are still many choices having the same physical effect, e.g. [0, 5, 10] versus [0, 1, 2]. The decision of which one to choose exactly turns to be subjective. So it is important to validate the methodology stability with distinct *value vectors* of similar physical effect. Particularly, we would like to investigate the system stability for the DL4LD use case.

We define a metric *Spreading Level* SL to characterise different *value vectors*. Those *value vectors* indicating similar physical effect should have the same SL. The *spreading level* of a *value vector* $\vec{v}_i = [v_{i,1}, v_{i,2}, v_{i,3}]$ is calculated as:

$$SL(\vec{v}_i) = (v_{i,2} - v_{i,1}) - (v_{i,3} - v_{i,2}) \tag{3.8}$$

3.6.2 Metrics definition

In this section, we further explore how the subjectively chosen parameters, *value vectors*, influence the output of the risk assessment system.

As introduced in section 3.4, module II computes the application-dependent risk of each threat with the modified STRIDE/DREAD model and calculates the *risk ratios* all the threats in the approved threat list. Obviously, both the threat risk and *risk ratios* are varying with the chosen *value vector*. Those fluctuated *risk ratios* further flow into module III in the system for threat mitigation. The security countermeasures and *risk ratios* jointly determine the DDM rankings. In

the ideal scenario, the risk assessment system would always generate the same ranking for DDMs for a given application regardless of subjective choices. Absolute values of *risk ratios* play a vital role.

Also, most users of the DREAD/STRIDE model, or our modified version, are focused on the rankings of threats in terms of their risk. We expect stable ranking results for those value vectors with the same physical effect. So we investigate the variance of threat risk rankings caused by a subjectively chosen *value vector*.

Two metrics are adopted to quantify the variance of *risk ratios* with various *value vectors*: Kendall's Tau and Normalised Mean Square Error (NMSE).

Kendall's Tau

We are able to rank the threats in terms of risk according to their *risk ratios*. Kendall's Tau is one of the commonly-used metrics to measure the similarity of two rankings [27]. We use it to measure the stability between threat rankings of different adopted *value vectors*.

The definition is as follows:

$$\tau(T_x, T_y) = \frac{\#\text{conc pairs}(\mathrm{T_x}, \mathrm{T_y}) - \#\text{disc pairs}(\mathrm{T_x}, \mathrm{T_y})}{\binom{N}{2}}, \tag{3.9}$$

where T_x represents a threat ranking according to *risk ratios* with *value vectors* $\vec{v_x}$ and T_y represents a threat ranking according to *risk ratios* with *value vector* $\vec{v_y}$, and N denotes the total number of threats in the list. This leads to a set of $\binom{N}{2}$ pairs. For any pair of *value vectors* $\vec{v_x}$ and $\vec{v_y}$, we calculate the *risk ratios* for the N threats: $(rr_x(t_i), rr_y(t_i))$, where $rr_x(t_i)$ is the *risk ratio* of threat t_i in T_x with $\vec{v_x}$ and $rr_y(t_i)$ is the *risk ratio* of threat t_i in T_y with $\vec{v_y}$. *#conc pairs* denotes the number of threat pairs that are concordant in both rankings T_x and T_y, and *#disc pairs* denotes the number of threat pairs that are discordant in both rankings. Threats t_i and t_j are considered a concordant pair if $rr_x(t_i) \leq rr_x(t_j), rr_y(t_i) \leq rr_x(t_j)$. Otherwise, they are considered as a discordant pair.

Normalised mean square error

We choose the metric Normalised Mean Square Error (NMSE) to quantify the variance of *risk ratios* due to different *value vectors* [26]. The reason we choose NMSE rather than other metrics are twofold. On the one hand, NMSE is sensitive to outliers. On the other hand, the results are not influenced by absolute values after normalisation. The definition of NMSE is as follows:

$$RR_x = \{{rr_1}^{(x)}, {rr_2}^{(x)}, {rr_3}^{(x)}, \cdots, {rr_N}^{(x)}\} \tag{3.10}$$

$$\overline{RR_x} = \frac{1}{N}\sum_i rr_i^{(x)} \tag{3.11}$$

$$\overline{RR_y} = \frac{1}{N}\sum_i rr_i^{(y)} \tag{3.12}$$

$$NMSE(RR_x, RR_y) = \frac{1}{N}\sum_N \frac{(rr_i^{(x)} - rr_i^{(y)})^2}{\overline{RR_x}\cdot\overline{RR_y}} \tag{3.13}$$

RR_x denotes the *risk ratios* of N threats using *value vector* $\vec{v}_x$. The *risk ratio* of the ith threat with *value vector* $\vec{v}_x$ is denoted by ${rr_i}^{(x)}$. $\overline{RR_x}$ denotes the average of all *risk ratios* in RR_x.

3.7 Experimental validation of system stability

In this section, we validate the stability of the risk assessment system. Here we focus on the stability of *risk ratios* because they influence the stability of DDM rankings of the risk assessment system. We compute and analyse the values of Kendall's Tau and NMSE of varying *value vectors* under different experimental settings.

3.7.1 Experimental design

In the experimental validation, we consider *value vectors* $\vec{v}_i = [v_{i,1}, v_{i,2}, v_{i,2}]$ with $v_{i,j} \in \{0, 1, \ldots, 10\}$. We construct a set V_{total} of 165 different *value vectors*. In particular, the *value vector* used by the original Microsoft DREAD model is called the baseline *value vector* $\vec{v}_{base}$, in our case $\vec{v}_{base} = [0, 5, 10]$.

Experiment A

In this experiment, we aim to explore the sensitivity of the threat risk rankings to the applied *value vectors* in a general sense. We compute the two metrics, Kendall's Tau and NMSE, between *risk ratios* for any *value vector* $\vec{v}_i$ in V_{total} and for $\vec{v}_{base}$, i.e. $\tau(T_x, T_y)$ and $NMSE(RR_x, RR_y)$ where $\vec{v}_x \in V_{total}$ and $\vec{v}_y = \vec{v}_{base}$. This results in a set of Kendall's Tau values and a set of NMSE values. The size of each set is equal to the size of V_{total}.

Experiment B

In this experiment, we aim to evaluate the fluctuations of threat risk rankings among *value vectors* of similar physical effect. According to the discussion in

Section 3.6.1, those *value vectors* with similar physical effect should have the same *spreading level*. Hence, we partition all the *value vectors* in set V_{total} in groups with equal SL. We calculate the two metrics, Kendall's Tau and NMSE, for any pair of *value vectors* in each equal SL cluster, i.e. $\tau(T_x, T_y)$ and $NMSE(RR_x, RR_y)$ for the *value vectors* $\vec{v}_x, \vec{v}_y \in V_{total}$ with $\vec{v}_x \neq \vec{v}_y$ and $SL(\vec{v}_x) = SL(\vec{v}_y)$. In this way we can achieve the variation of system outputs due to the subjective choice of *value vectors*.

3.7.2 Experimental threat database

We need to construct proper threat databases to compute and analyse *risk ratios* of a threat set. For simulation purposes, the assigned values of the five *risk attributes* described in Table 3.1 can uniquely identify each threat. In the current experiment, we consider two threat databases, namely, the theoretical threat database and the DL4LD threat database.

For the theoretical threat database, we consider all possible combinations of 5 *risk attributes*, each of which can be one of the 3 values in a *value vector*. The total number of threats in this database is 3^5 (243). Obviously, any real-world threat database, like the DL4LD threat database, is a subset of the theoretical threat database.

We have introduced 7 collaboration archetypes in Chapter 2. We model the threats for those archetypes and construct the DL4LD threat database. There are in total 22 threats for all archetypes in the DL4LD threat database as depicted in 3.2. For each threat, we read the related literature and determined the levels of 5 *risk attributes*.

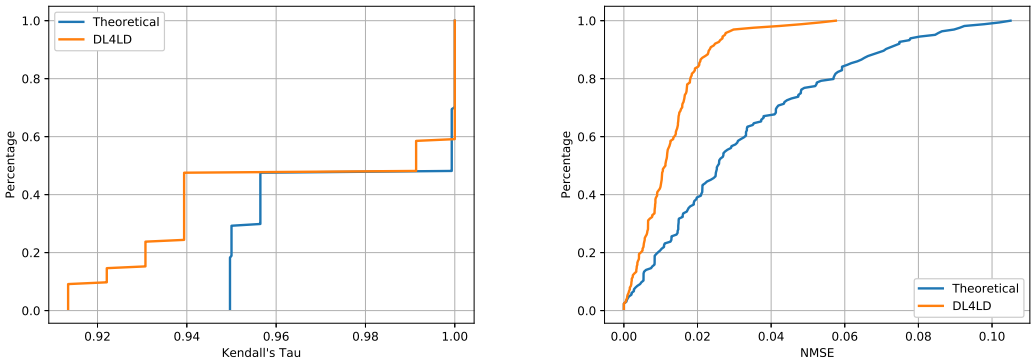
3.7.3 Analysis of Kendall's Tau values

We compute Kendall's Tau values between threat risk rankings generated by the baseline *value vector* [0, 5, 10] and any arbitrary *value vector* in set V_{total} for both theoretical and DL4LD threat database. For each database, we rank all threats according to their *risk ratios* computed in Equation 3.4.

Figure 3.6a shows the Cumulative Distribution Function (CDF) of those Kendall's Tau values for both theoretical and DL4LD threat database. For the theoretical threat database, all of the *value vectors* contribute to Kendall's Tau values higher than 0.95, and 50 % of the *value vectors* have Kendall's Tau values higher than 0.99. We conclude that the threat risk ranking is almost stable for all *value vectors* in the theoretical database. For the DL4LD threat database, approximately 50 % of the *value vectors* have Kendall's Tau values higher than 0.99, which is similar to the DL4LD use case. But another half have Kendall's Tau values between 0.91 and 0.94, which are lower than the minimum value for the theoretical threat database. The comparatively larger ranking variance for the DL4LD use case may be due to the characteristics of the DL4LD threat database.

Table 3.2: The threat list in DL4LD database. For each threat, we assign corresponding *Stage*, *Category* and *Risk Attributes* according to literatures. 'H', 'M', 'L' represent 'High', 'Medium', 'Low' respectively. 'SO' stands for the 'sensitivity of the object'. 'TOP' stands for the highest level of 'sensitivity of the object'.

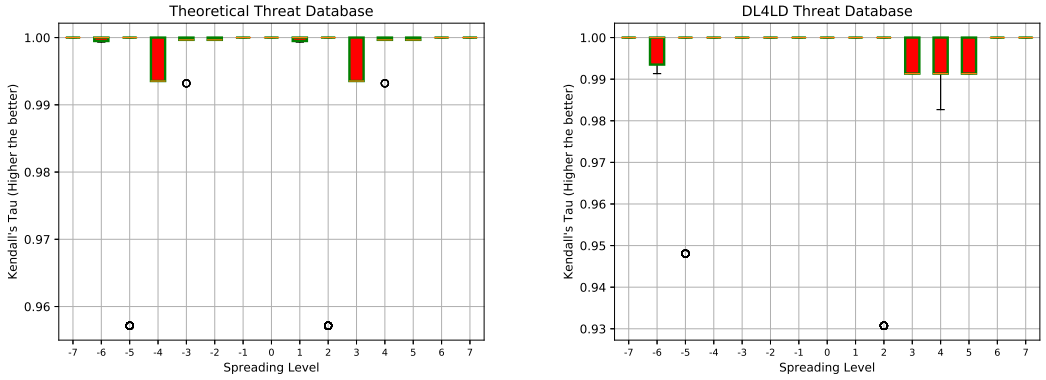
Threat Name	Stage	Category	Risk Attributes				
			DP	AC	SL	AU	D
IP spoofing	II	S	SO	H	M	H	M
Identity spoofing: Remote Data Access	III	S	SO	H	L	M	H
Insecure data deletion	III	ID	SO	M	L	M	H
Malicious compute: Data Disclosure	III	ID	SO	L	H	H	M
Unauthorized Disclosure: Eavesdropping	II	ID	SO	H	H	M	H
Weak Access Control	I	ID	SO	H	H	L	H
Malicious compute: High Correlation of Input and Output Data	III	ID	SO	L	H	H	M
Encryption Keys Leakage during Exchange	II	ID	TOP	H	L	H	H
Cross-tenant Side Channel Attack	III	ID	SO	M	L	H	H
Management Interface Compromise	I, III	ID, T	SO	M	M	M	M
Isolation Failure: Poorly Separated Container Traffic	III	ID	SO	L	L	H	H
Isolation Failure: Cross Container Attack	III	ID	SO	M	L	H	H
Insecure Running Environment	III	ID	SO	M	L	H	H
Man-in-the-Middle	II	T	SO	H	M	M	L
Malicious compute: Tamper Processed Data	III	T	SO	L	H	H	L
Log Files Tampering: illegal members delete or modify log files	I, II, III	T	TOP	L	L	H	L
Data Leakage/Loss	I	T	SO	H	L	M	L
Not-trustable Computing Environment	III	T, ID	SO	M	M	H	L
Denial of Service (DoS) Attack by Co-tenant Containers	III	DoS	SO	L	H	H	L
Container Runtime Escape	III	EP	SO	L	M	H	M
Repudiation Attacks	II	R	SO	M	L	H	L
Insufficient Auditing	II	R	SO	L	H	M	H



(a) Cumulative Distribution Function of Kendall's Tau values

(b) Cumulative Distribution Function of NMSE values

Figure 3.6: CDF of Kendall's Tau values and NMSE between all *value vector* and the baseline *value vector* [0, 5, 10] for both the theoretical and the DL4LD threat database.



(a) Box plots of Kendall's Tau in theoretical threat database

(b) Box plots of Kendall's Tau in DL4LD threat database

Figure 3.7: All value vectors in set V_{total} are grouped with identical *spreading level* ranging from -7 to 7. Each value vector indicates a bijective mapping from qualitative levels of risk attributes to numeric representations. For each value vector cluster with identical *spreading level*, we compute Kendall's Tau values between any pairs of value vectors and plot them as a box.

For two threats with higher diversity of *risk attributes* levels, their rankings are likely to flip with different *value vectors*. For instance, if we have two threats with *risk attributes* [L, L, L, L, L] and [H, H, H, H, H], the ranking will never alter no matter how you change the adopted *value vectors*, because [L, L, L, L, L] will always have the lowest *risk ratio* and [H, H, H, H, H] will always have the

highest. The rank of threats with *risk attributes* [L, H, L, M, L] and [M, L, M, H, M] most likely will flip after changing the *value vectors*. A higher proportion of such sensitive threats exists in the DL4LD threat database than in the theoretical threat database.

Figure 3.7a and Figure 3.7b show the box plot for Kendall's Tau values as a function of SL for the theoretical and the DL4LD threat database respectively. The Kendall's Tau values are computed according to *Experimental Design B* in Section 3.7.1. All the *value vectors* in set V_{total} are grouped with equal SL. Each box depicts the Kendall's Tau values computed between all possible pairs of *value vectors* within an equal SL group.

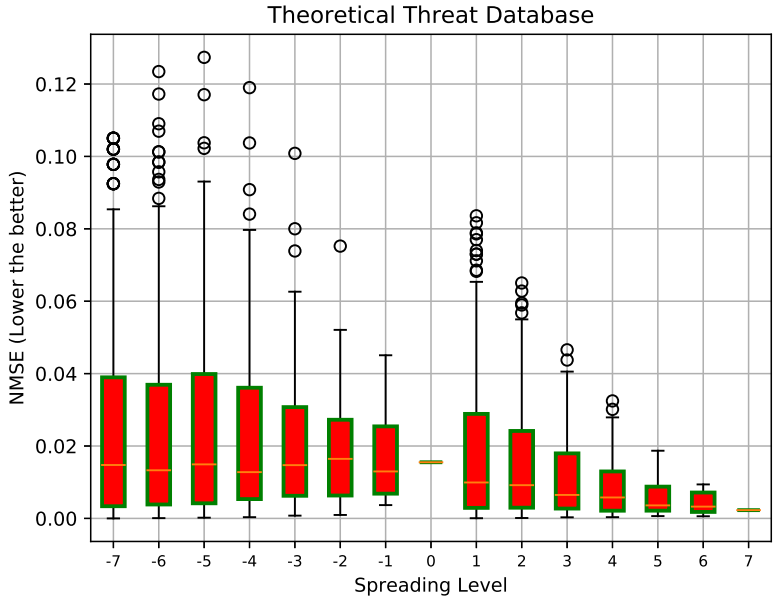
Figure 3.7a shows Kendall's Tau values among threats rankings for the theoretical threat database. We specifically focus on evenly spaced *value vectors* because they are most commonly used in reality. The Kendall's Tau values of evenly spaced *value vectors* ($SL = 0$) are all equal to 1. A subjectively chosen *value vector* with SL equals to 0 does not influence the risk ranking of all theoretical threats. Also, since all the real-world threat databases, e.g. DL4LD threat database, are a subset of the theoretical database, the Kendall's Tau values among evenly spaced *value vectors* should always be 1 for any threat database. The results illustrated in Figure 3.7b confirm this conclusion. More generally, as shown in Figure 3.7a, 9 out of 15 boxes have all values extremely close to 1, whereas 4 boxes have a slightly higher degree of dispersion, but the minimums are still larger than 0.99. Only two outliers around 0.955 occur for boxes $SL = -5$ and $SL = 2$ respectively. Subjective choices of *value vectors* having the same *spreading level* do not cause the risk rankings to fluctuate. As the theoretical database includes any real-world threat database, we may expect similar high stability achieved in any other threat database, e.g. DL4LD. Figure 3.7b shows the box plots for the DL4LD threat database. Similarly, most *value vector* clusters have Kendall's Tau values very close or all equal to 1. But the worst case, the two outliers in boxes with $SL = -5, 2$, have comparatively higher variance than for the theoretical database.

For both the theoretical and the DL4LD use case, there is almost no or neglectable influence due to the subjective choices of *value vectors* having the same physical effect (*spreading level*).

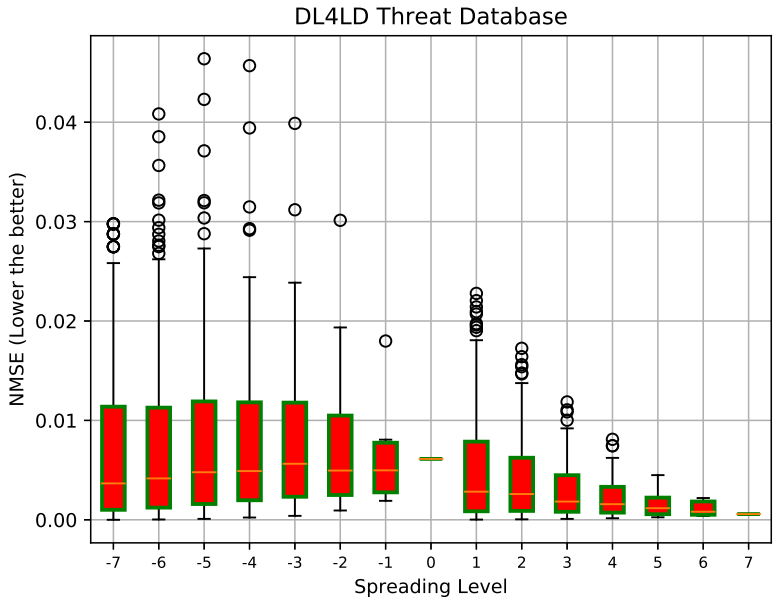
3.7.4 Analysis of normalised mean square error (NMSE)

The metric NMSE describes the variance of absolute values of *risk ratios* with different *value vectors*, which have a direct impact on final DDM exposure rankings. To explore the general sensitivity of *risk ratio* values to varying *value vectors*, we compute NMSE values between the baseline *value vector* [0, 5, 10] and any *value vectors* in set V_{total}.

Figure 3.6b shows the Cumulative Distribution Function (CDF) of NMSE values computed with all *value vectors* in set V_{total} for both the theoretical and



(a) Box plots of NMSE in theoretical threat database



(b) Box plots of NMSE in DL4LD threat database

Figure 3.8: All *value vectors* in set V_{total} are grouped with identical *spreading level* ranging from -7 to 7. Each value vector indicates a bijective mapping from qualitative levels of risk attributes to numeric representations. For each value vector cluster with identical *spreading level*, we compute NMSE values between any pairs of *value vectors* and plot them as a box.

the DL4LD threat database. For the theoretical threat database, approximately 50% of the *value vectors* result in an NMSE value smaller than 0.03 compared with the baseline *value vector*. An NMSE value of 0.03 means that the average shift between two data sets, *risk ratios* with two *value vectors*, is 3% of the product of mean values of the two data sets. For some specific *value vectors*, *risk ratios* vary unneglectable comparing to those computed with the baseline *value vector*. About 18% of *value vectors* in set V_{total} result in an NMSE value higher than 0.06, and the maximum value is 0.1. This is due to the non-linear mappings from qualitative levels to quantitative numbers in module II. However, the *risk ratios* are much less sensitive for threats in the DL4LD use case. Also shown in Figure 3.6b, approximately 95% of the *value vectors* in set V_{total} have NMSE values smaller than 0.03 for the baseline *value vector*. The maximum value of NMSE is only 0.06. One reasonable explanation is that absolute larger differences of *risk ratios* normally occur for threats with a smaller risk attributes level diversity, e.g. [L, L, L, L, L] and [H, H, H, H, H]. Such threats are not frequently included in the DL4LD threat database or any other real-world threat database. Hence we may expect the risk assessment system is quite robust against subjective choices of *value vectors* for the majority of use cases.

Figure 3.8 shows the box plots of NMSE values as a function of SL for both the theoretical and the DL4LD threat database. For *value vectors* of the same SL, we calculate NMSE values of *risk ratios* with every two *value vectors* in the group.

We first analyse stability for evenly spaced *value vectors*. Shown in Figure 3.8a and Figure 3.8b, the dispersion degrees of boxes for $SL = 0$ are very small. The pairwise NMSE values among evenly spaced *value vectors* for both threat databases are concentrated in the medians of the boxes, which are about 0.015, and 0.008 respectively. There are no outliers of relatively higher NMSE values. These NMSE values imply that the system is highly stable to subjective choices for *value vectors* with linear mappings from *risk attribute* qualitative levels to numeric representations.

Figure 3.8a shows the box plots for the theoretical threat database. Each box has a relatively high degree of dispersion, and the median value is around 0.01. An NMSE value of 0.01 is quite acceptable and has a relatively small probability of causing a ranking flip for DDMs in the final output of our system. The NMSE values in Figure 3.8a indicate that the ranking is stable for about 50% of *value vectors* for each equal SL cluster. However, outliers from 0.07 to 0.13 occur in most *value vector* clusters, especially for those with negative SL values. However, the system stability is much higher for the DL4LD use case shown in Figure 3.8b. All the boxes for the DL4LD threat database have median values of 0.005, which are much smaller than that of the theoretical threat database. Furthermore, the outliers are much acceptable, with the maximum value smaller than 0.05. The DL4LD use case is very robust to *value vector* variance.

3.8 Experimental validation of system resolution

In this section, we aim to validate the achieved resolution of our methodology provided by the output of module II, which are *risk scores*. We define a metric of Granularity to measure resolution quantitatively. We try to investigate how the chosen *value vectors* influence the system resolution for both theoretical and DL4LD threat database. In addition, we also explore whether the current methodology can provide sufficient Granularity for identified threats in the DL4LD use case.

3.8.1 Definition of granularity and experimental design

The metric Granularity aims to evaluate the resolution of our methodology in module II. Granularity is defined as **the total number of unique values of *risk scores* for a given threat database**. This metric describes the capability of distinguishing between threats in terms of assessed risk. It is usually not expectable that many threats result in the same risk level, which is equal to the value of computed *risk score*.

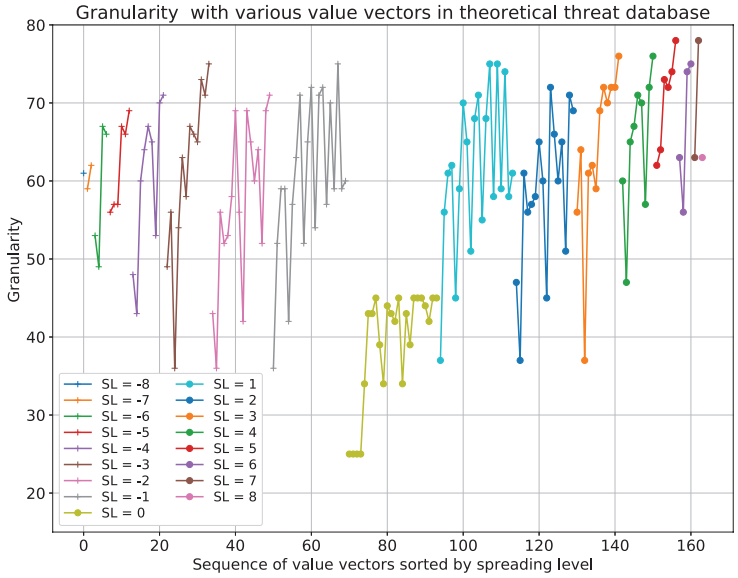
We adopt the same *value vector* set V_{total} as described in Section 3.7.1. We compute Granularity with each *value vector* in V_{total} for both theoretical and DL4LD threat database. As mentioned in Section 3.7.1, the number of threats is 243 in the theoretical database and 22 in the DL4LD threat database. We also consider the influence of assigned *impact factor*, which has been explained in Section 3.3.2, on the resulting system resolution. Any of the 5 *impact factors* scales a threat and the *risk score* of that threat is scaled accordingly.

3.8.2 Analysis of granularity values

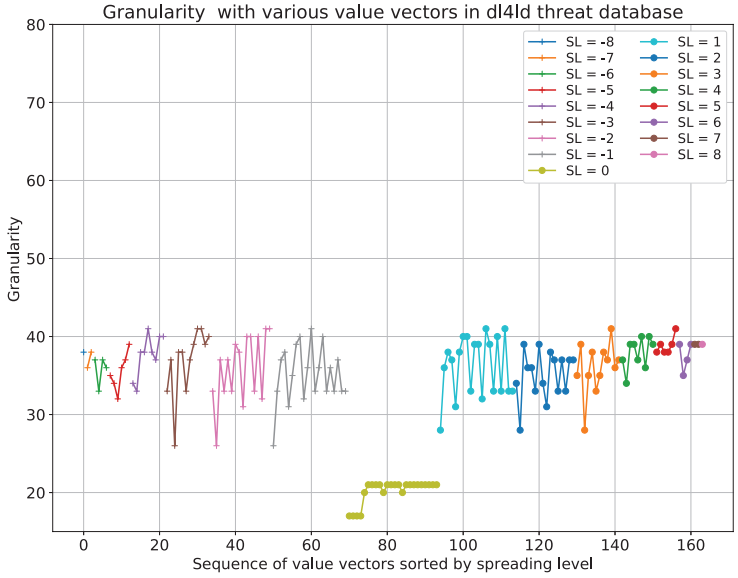
Figure 3.9a and Figure 3.9b show the values of granularity with various *value vectors* in the theoretical threat database and the DL4LD threat database respectively, with the *spreading levels* depicted in different colours.

Firstly, we investigate the relationship between achieved Granularity and *spreading level* of *value vectors*. For both threat databases shown in Figure 3.9, non-evenly spaced *value vectors* ($SL \neq 0$) normally gain much better resolution than evenly spaced ones ($SL = 0$). Also, the *value vectors* of different *spreading level* have a similar range of Granularity for all non-evenly spaced *value vectors*. Based on the conclusion drawn in Section 3.7, those evenly-spaced *value vectors* normally have comparatively higher stability. A higher resolution is achieved at the sacrifice of system stability.

The values of Granularity fluctuate for *value vectors* of identical SL. We recommend DDM customers to choose a *value vector* with relatively high Gran-



(a) Granularity of threats in the theoretical database with various *values vectors*



(b) Granularity of threats in the DL4LD database with various *values vectors*

Figure 3.9: Values of Granularity with varying *value vectors* for both the theoretical and the DL4LD threat database. The value vectors are firstly sorted with increasing *spreading level*. For those with equal *spreading level*, the *value vectors* are in lexicographic sorted order.

ularity and to avoid those with very low resolution. As shown in Figure 3.9a and Figure 3.9b, the relative relations of Granularity values among same *value vectors* for both threat databases are similar. For each group with equal SL, there is a *value vector* resulting in a low Granularity value. Those *value vectors* are sorted in lexicographic order for an equal-SL cluster. Hence those *value vectors* with the first element as 0 contribute to relative worse resolution compared with those of similar physical effect. According to the above observation and discussion, our system can warn system users when they use such *value vectors*.

We further discuss the absolute values of Granularity for both threat databases. As shown in Figure 3.9a, evenly spaced *value vectors* have Granularity between 25 and 45. For a threat database of 243 threats with 5 different *impact factors*, on average 27 threats may result in the same risk level even adopting the *value vector* with the highest resolution. The performance for non-evenly spaced *value vectors* is better with Granularity values between 35 to 80. Nevertheless, there are still on average 15 different threats that are not distinguishable for their risk with the current methodology in the theoretical threat database. As indicated in Figure 3.9b, the DL4LD use case performs very well regarding the small size of the threat database of only 22 threats, each of which may have 5 different *impact factors*. The Granularity values are around 20 for evenly spaced *value vectors* and vary from 24 to 45 for non-evenly spaced ones. Compared to the theoretical threat database, the DL4LD threat database can achieve approximately half of the Granularity with only one-tenth of the threats number. The discussion above indicates the system provides sufficient resolution to distinguish threats in the DL4LD use case.

3.9 Related work

Recent research assesses the security provided by digital infrastructures, e.g. clouds. [34] proposed an approach to assess the security of a cloud platform, but it only focuses on individual threats separately and provide only qualitative evaluation results. [35] proposed a methodology to assess the security level of a Security Service Level Agreement (SecSLA) with respect to customers' requirements. The work allows cloud clients to compare SecSLAs provided by different cloud service providers (CSP) and aims to provide costumers with a general view of security coverage of the provided infrastructures. The security controls of the cloud infrastructures are classified based on the Cloud Control Matrix (CCM) taxonomy, which makes the proposed system very difficult to migrate to another application context, e.g. Digital Data Marketplaces. [36] advocate a similar security evaluation methodology of SLAs by using a trust model. Different security countermeasures are chained according to the taxonomy defined in this trust model. The system measures the security strength from multiple dimensions and computes a trust value. Nevertheless, these authors fail to consider that the vul-

nerabilities, normally varying with each application, have a strong influence on the effectiveness of applied security countermeasures. [37] proposed an off-line risk assessment framework to evaluate the security level of an application for a specific CSP. They first identify the threats for a given application and estimate how much risk can be mitigated with the CSP's infrastructures. However, the system treats all the identified threats with equal severities and this is not what happens in real world scenarios.

There are multiple risk management frameworks for information systems. The ISO provides standards with which an information system can gain adequate security. The standards describe the control objectives, required security controls and guidelines. The ISO standards are widely used as certifications for companies to verify the security of their information systems and promote customer's trust [38]. The National Institute for Standards and Technology (NIST) cybersecurity framework also offers guidance to facilitate risk management within specific organisations [39]. This framework aims to keep an information system safe by identifying security gaps. OCTAVE is also a risk-based assessment and planning process. It identifies the infrastructure vulnerabilities and develops protection strategies in design [40]. However, all the work mentioned above focuses on risk management while establishing a digital infrastructure. After risk analysis, the output of those frameworks are implementation requirements, e.g. security countermeasures, user action guidance's, for a single information system. Our proposed framework aims to choose the most secure digital infrastructures in an application-based manner with risk scores. CORAS is a model-driven risk assessment framework. It identifies the threats of a use case, assess the risk of each threat and develop treatments [41]. But it does not consider the relative importance of each threat and does not provide a total risk score for ranking different DDM infrastructures.

The Microsoft STRIDE/DREAD model provides a threat modelling approach and assesses a single threat risk by proposing attributes measuring difficulties of exploiting the vulnerability [22]. Most studies of the STRIDE/DREAD model focus on risk evaluation of an individual threat and provide threats ranking regarding their risk [42, 43]. In [29], the authors used' the STRIDE/DREAD model to assess and prioritise threats in a cloud environment. They adapt the original risk parameters to the cloud environment and assign *impact factors* to each threat category. However, their model does not fit the context of DDMs, where applications are modelled as workflows and trust among collaborating parties plays a vital role. Moreover, all the studies in [29, 42, 43] just inherit the numeric values of *risk attributes* from the original Microsoft STRIDE/DREAD model without validating the objectivity of the choices.

From this overview it is clear that our work covers a currently unexplored area. Firstly, it specifically caters to DDMs and their customers. Secondly, it assigns severity weights for identified threats, with the modified STRIDE/DREAD model, to achieve more objective risk assessment results. Thirdly, we investigate

the robustness and resolution of our proposed system against subjective choices of risk parameters in the original STRIDE/DREAD model with real world security threats.

3.10 Conclusions and future works

Customers of DDMs, or other digital infrastructures, need to know what is the risk level associated with running their applications in any specific DDM. We propose a risk assessment system to quantitatively assess the risk level. This system allows customers to rank available digital infrastructures in terms of guaranteed security and select the optimal one for their applications.

To increase transparency, the system collaborates interactively with all involved. It addresses the complexity of DDMs by considering a number of influencing factors, such as application archetypes, security requests of DDM customers, interactions of security countermeasures. Our proposed system considers the relative importance of each threat and is able to capture the dynamic feature of risk levels in data exchange applications in DDMs.

We validated the stability and resolution of the Microsoft STRIDE/DREAD model in our risk assessment system with a concrete DDM use case, DL4LD. Our experimental results show that subjective choices of users have a very subtle influence on the final DDM rankings of the system. In addition, the risk assessment system provides sufficient resolution and works very well in terms of stability for the DL4LD use case.

In this chapter, we conducted a threat analysis for the DL4LD dataset. We observed that there is a lack of countermeasures to deal with the vulnerabilities in the execution stage of a data exchange application. We focus on designing DDM components that help increase policy enforcement capability in the execution stage in Chapter 4 and Chapter 5.

Chapter 4

Policy compliance detection with syscall profiling

It is vital to improve the policy compliance detection capability of the digital infrastructures in the execution stage, where the algorithms and data meet. Multiple containers may run on the same platform with permissions to execute on different data sets. It is crucial to ensure that each data set is only accessed by the authorised algorithm defined in the policy. The algorithms in DDMs are normally encapsulated in containers to gain better portability and higher isolation level. Normally the DDM infrastructures are not allowed to access the containers or the algorithm source code directly to avoid information leakage. In this chapter, we propose a profiling architecture to distinguish containerised algorithms by external monitoring and start to answer our RQ3. We profile dynamic behaviours of running algorithms with Linux-kernel system call traces and investigate the stability of the proposed methodology with seven typical containerised machine learning applications over different execution platform OSs and training data sets.

This chapter is based on:

- **Lu Zhang**, Reginald Cushing, Ralph Koning, Cees de Laat, and Paola Grosso. "Profiling and Discriminating of Containerized ML Applications in Digital Data Marketplaces (DDM)." In ICISSP, pp. 508-515. 2021.

4.1 Introduction

The collaborating parties, e.g. data providers and algorithm providers, normally come to a DDM with a pre-agreed policy. For example, a typical policy rule is *A data object can only be accessed by a specific compute object pre-defined in the policy, but not by the others.* To ensure the security of such policy-driven

applications, a DDM infrastructure should include components that detect policy non-compliance.

The container level virtualisation has gained significant attention in recent years. Containers are operating system level virtualisation abstractions. A container image is a lightweight, executable package including source code, program runtime and libraries [44]. The compute objects in a DDM are normally containerised so that they are easily adapted to various execution platforms[45]. According to different collaboration models, the execution process can occur on the data provider side, algorithm provider side or a third-party platform. Most of the platforms do not have the privilege to check the source code for confidentiality reasons. Therefore, it is essential for DDM infrastructures to implement components that can distinguish containerised applications only depending on external monitoring.

System calls provide an interface to the services provided by an operating system. They are used by user-level applications for various reasons, including file management, process control, device management and communication. These system calls give us a way to monitor what user-level applications are doing. The behaviours of a computer program can be well modelled as system call sequences [46].

We propose an architecture to distinguish running containers by establishing and verifying system call profiles. An authorised party builds an authorised profile of a specific computing algorithm after verifying the source code. The program behaviour is modelled with the occurrence of fixed length subsequence of system calls (*n-grams*). The execution platform monitors system calls in real time and can, at the end of execution, distinguish programs running inside containers based on system call profiles. The dissimilarity between profiles is computed with cross entropy. The system will trigger an alarm if there is a mismatch between the policy and classification results.

System calls are highly localised, and the generated trace files vary with the configurations of the execution platforms and the performed data sets [47]. There is a basic requirement for the proposed architecture to allow authorised profile reuse. To address these points, we investigate the suitability of the proposed methodology over different Linux distribution operating systems and with different training sets with 7 typical ML applications in DDMs.

The remaining of the chapter is structured as follows. We introduce system architecture in Section 4.2 and the proposed methodology to gather and generate profiles in Section 4.3, Section 4.4 and Section 4.5. Section 4.6 and Section 4.7 introduce the definitions of self-variance and stability respectively. Section 4.8 presents the experimental results of classification accuracy. Our methodologies and results, as well as next steps, are discussed in Section 4.9. In Section 4.10 we compare our architecture with existing ones and Section 4.11 concludes this chapter.

4.2 Architecture

We propose an architecture to enforce the data access and usage policy among collaborating parties during the execution phase of a data sharing application. The architecture can be implemented as one of the enforcement components for DDM infrastructures.

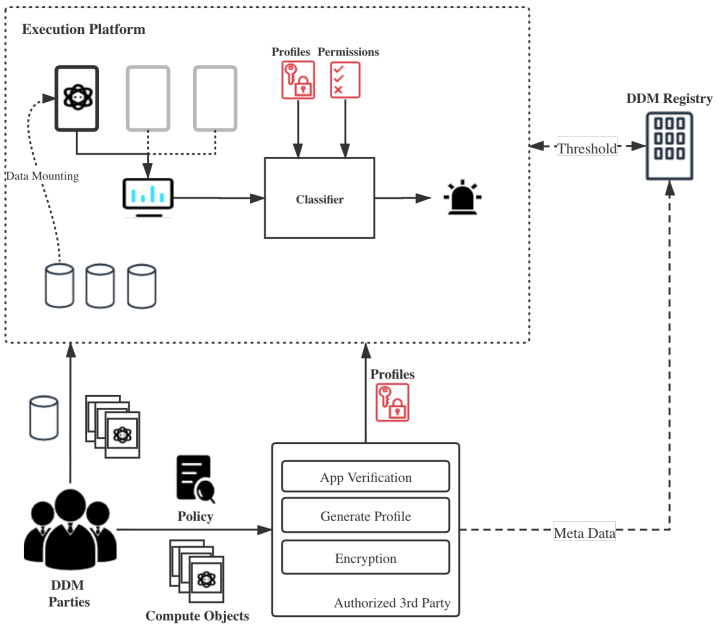


Figure 4.1: A DDM enforcement component to distinguish and verify running algorithms inside containers with system call profiles.

As shown in Figure 4.1, the data providers and algorithm providers first agree on a policy for the data sharing application. The policy describes the purpose of the computing algorithm, e.g. the algorithm can perform on which specific data set. For portability, the computing algorithms in DDMs are encapsulated into containers.

Secondly, the algorithm provider sends its compute algorithm, one or multiple images, and the policy to an authorized party for verification. The authorized party can either be a DDM component or an external trusted 3rd party. This checks the source code and verifies whether the given algorithm complies to the policy. If so, the authority party generates profiles of the application images with system call traces and sends them to the execution platform. These profiles are digitally signed and encrypted with the public key of the execution platform. Thus the risk of mimic attacks for profiling is highly reduced [48].

Thirdly, both data objects and compute objects are sent to the execution platform. To keep confidentiality of the algorithm, the compute containers are

normally executed by its owners remotely. The platform monitors system calls of running containers and feeds the information to a classifier. The classifier discriminates running programs inside each container in real time. The classifier computes the dissimilarities between the observed tracefiles with all authorized profiles.

With the permission of DDM collaborating parties, the authorized party and execution platform can choose to send metadata to a DDM registry. The metadata contains the hashed application name and pre-processed profiles. For example, a subsequence of system calls can be encoded into binary identifiers. The registry can exploit those data and compute a reference threshold for outlier detection in the classification algorithm.

4.3 Profile generation

In most related works, the system call profiles are used to detect anomalies during the execution[47, 49]. The interest, in this case, is on a specific subsequence that indicates malicious behaviour. We instead use the profiles to distinguish among running applications. Hence it is important to consider the behavioural variance of system call trace files when containerized applications are running with different configurations. For instance, the system call traces of a specific ML algorithm may change with different training data sets or with multiple runs. This may cause false alarms when we use system call profiles to distinguish compute applications.

Figure 4.2 shows the graphical representation of profiles of two different applications. The red dots represent the system call profiles generated by Application 1. The scatter pattern we observe is caused by different training platforms, different training data sets or even different runs. The blue triangles represent the profiles of a different application. Good profiling and similarity computation methodology will reduce the noise and only extract the information which actually represents the algorithm behaviours.

We define *self-variance* of a profiling methodology to be the dissimilarities among system call profiles of the same application. As illustrated in Figure 4.2, the *self-variance* reflects the degree of dispersion of red dots (profiles of App1) or blue triangles (profiles of App2). Higher *self-variance* indicates a larger dashed circle. It represents how sensitive a profiling methodology is to the intrinsic variability of the running application. The quantitative definition of *self-variance* depends on dissimilarity measures in the classifier. The *self-variance* conveys important information when profiles are used for application discrimination. Higher self-variance is likely to cause a higher false negative rate when making the decisions.

In the next sections we will present our methods and results on classification. To investigate our proposed methodologies, we set up an experimental environment illustrated in Figure 4.3. We want to emulate the action of the authorized

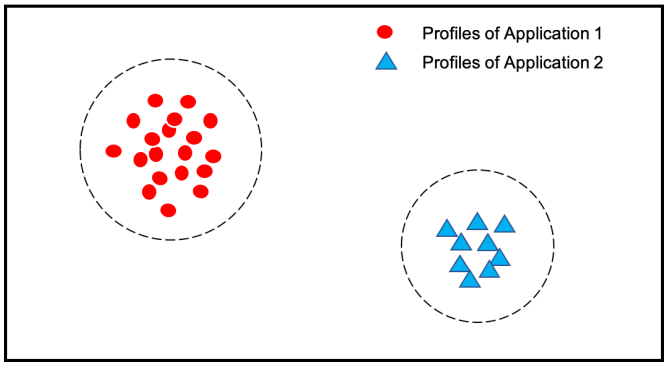


Figure 4.2: Graphical representation profiles of different applications. Profiles from the same application cluster together (red dots); a different application profile (blue triangle) is distinctly separate.

3rd party in Fig. 4.1 when it generates the profile as well as the actions of the execution platform when it is monitoring running applications.

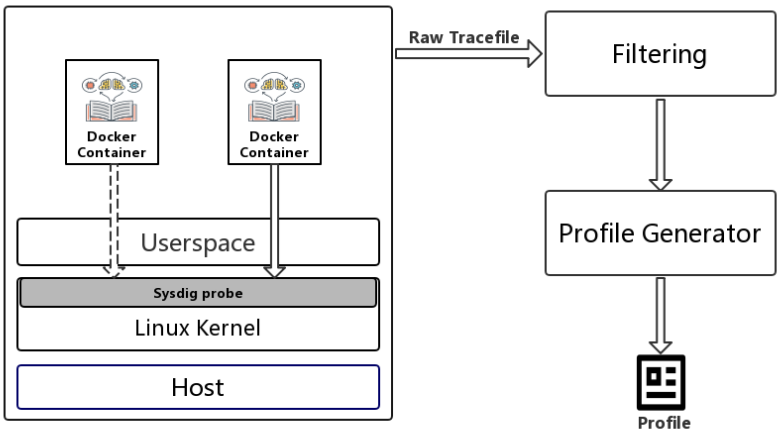


Figure 4.3: Profile generation setup: a single physical host running Docker containers and the Sysdig probe in the Linux Kernel

We run containerized applications within a physical node. We use Sysdig to monitor the generated system calls because it is specifically designed for containers [50]. The Sysdig probe is placed in the Linux kernel of the host machine and it traces all the system calls generated by the container. The input data is accessed as the volume of the Docker container.

As shown in Fig 4.3, the result of this setup is the system call raw trace file which will serve as the input of *Filtering* component. The *Filtering* component filters out all system calls generated by the container runtime. Finally, the *Profile*

Generator will generate profiles with the proposed methodology, which will be introduced in Section 4.5.

4.4 Classic n-gram profiles and limitations

Recently, there are plenty of work using fixed length subsequence to detect characterize behaviors of running processes [48, 51, 52]. They segment a system call trace into fixed length sub-sequence with a sliding window of length n (normally 3 - 6). The subsequences are called *n-grams*, with n being the length of the subsequence. Suppose we have a sequence of system calls such as $fstat, mmap, close, open, read, write \cdots$. This can be segmented into a list of n-grams with length 3: $\{fstat, mmap, close\}$, $\{mmap, close, open\}$, $\{close, open, read\}$, $\{open, read, write\}$, $\{read, write \cdots\}$. The underlying principle of this methodology is that each n-gram represents a small execution code path.

The traditional n-gram profiling methodology was proposed in [46]. It builds the profile as an enumeration of all occurring n-grams and the deviation between two distinct profiles is computed as the number of n-grams that are distinct in two profiles. With the setup illustrated in Figure 4.3, we aim to investigate the *self-variance* of traditional n-gram profiles. The application is a containerized machine learning algorithm training a classifier for fraud detection. The operating system for the execution platform for this experiment is *Ubuntu 18.04*. We first run a docker container with the same training data set for 10 times and investigate the *self-variance* of the generated n-gram profiles.

As seen in Figure 4.3, Sysdig traces system calls are generated by an entire container runtime. It is interesting when looking at the self variance to filter out the runtime generated system calls from the raw trace file. We filter out these calls with a text processing script we wrote for this purpose, as illustrated in the *Filtering* module in Figure 4.3.

Table 4.1: Average number of n-grams for the fraud detection application, before and after filtering of runtime generated system calls in three categories: all n-grams, pair-wise common n-grams and overall common n-grams.

	# ngrams	# pairwise common ngrams	# overall common ngrams
Before Filtering	1520	1310	1205
After Filtering	1370	1200	1125

Table 4.1 shows the average number of n-gram in the 10 profiles for the same application. We distinguish between profiles generated by the entire container, including application and container runtime and profiles with container runtime calls filtered out. Not surprisingly, we observe that all cases the number of n-

grams for filtered trace files is lower since the runtime-generated noise is filtered out.

The similarity of two n-gram profiles can be measured as the proportion of n-gram entries that are common in both files. We do a pair-wise comparison of the n-grams in the 10 trace files and show the average number of pair-wise common n-grams in the middle column. Finally we present the average number of overall common n-grams in all 10 trace files in the last column.

We first focus on unfiltered trace files. We can see that the average number of unfiltered n-grams is 1520; this reduces to 1310 when we do the pair-wise comparison and finally only 1200 n-grams are common to all 10 tracefiles. We can calculate that there are only around 86% of the n-gram entries are common between arbitrary two profiles, as this is the ratio between 1200/1520. In addition, the overall similarity is only 78%.

Looking at the filtered profiles we see that the pair-wise similarities are approximately 88% and the overall similarity is 82%. This is only slightly better than the unfiltered case and the likelihood of false negatives is still high.

In addition, larger variance is expected if we further consider the effect of different training data sets or different execution platform OSs due to the observation the generated system calls are varying with execution environment [47].

From this we can conclude that the classic n-gram profiling and distance computation methodology are likely to generate false alarms if we adopt it for discriminating containerized applications. In the next sections we will propose a different profiling methodology that can solve this issue.

4.5 Profiling with n-gram frequency distributions

The traditional n-gram method presented in section 4.4 builds the profile with distinct fix-length subsequences without information on the corresponding frequency. However, the frequency distribution conveys information for describing the behavior of a program. Using the same application (fraud detection) and using the 10 filtered profiles we have obtained in the previous experiment we produce the occurrence distribution.

Figure 4.4 shows the CDF of the occurrence distribution of each n-gram in one run of our machine learning application. The axis is the number of occurrence of each n-gram.

There are in total 1350 distinct n-grams in this profile. The occurrence numbers of those n-grams range from 1 to as high as 10^5. As shown in the Figure, more than 50% of the n-grams occur very rarely with occurrence numbers of 1. We expect that those rarely occurred n-grams contain little or at least less important information about the actual program behaviors. On the other hand, n-grams which occur often provide a clear *signature* of the application.

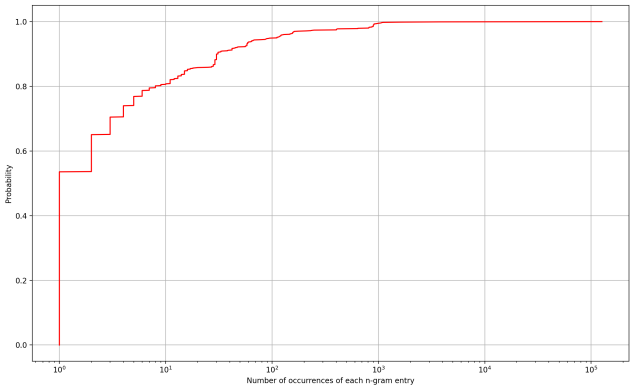


Figure 4.4: CDF of occurrence distribution of n-grams in one filtered tracefile - a tracefile containing only the container runtime calls.

This observation is in fact confirmed when we compare the n-gram entries that are different between two profiles of our application. We observe that almost all of those distinct n-gram entries occur only once in the profile. In this sense, we confirm that the traditional n-gram profiling methodology which treats all the distinct n-grams entries equally does not model the actual behaviour of an application well and may result in a lower classification rate. Hence we propose to profile a containerized program with the frequency distribution of n-gram entries. We do expect that the distinctive signatures of the application's profiles will result in lower variance, i.e. a close clustering of profiles of the same application as in Fig. 4.2 and a large distance from profiles of other applications.

To confirm our assumption we need to identify a method to calculate the profile distances when they are expressed as frequency distributions of n-grams. For this we introduce cross-entropy.

The dissimilarity of two profiles will then be computed using the cross entropy.

Suppose $OD_p = \{nc_1^{(p)}, nc_2^{(p)}, \cdots, nc_N^{(p)}\}$ denotes occurrence distribution in tracefile tr_p. $nc_i^{(p)}$ indicates occurrence counts for i the n-gram in the tracefile tr_p and N denotes the total number of distinct n-grams. Similarly, $OD_q = \{nc_1^{(q)}, nc_2^{(q)}, \cdots, nc_M^{(q)}\}$ denotes occurrence count of M n-gram entries in tracefile tr_q. In most scenarios, we have two distributions with $M \neq N$.

To obtain the cross entropy, two input distributions need to have an equal number of entries. To accomplish this we define a procedure to adjust the sets. We first compute the union set of OD_p and OD_q:

$$OD_u = OD_p \bigcup OD_q = \{nc_1, nc_2, \cdots, nc_L\} \tag{4.1}$$

$$L \geq M, L \geq N \tag{4.2}$$

L denotes the cardinality of the union set OD_u. We create two sets $\hat{OD}_p$ and

$\hat{OD}_q$ with L entries; we add all the n-grams contained in OD_u but not in the original OD_p with an occurrence of zero to form the new set $\hat{OD}_p$. We do the same procedure to form $\hat{OD}_q$.

$$\hat{OD}_p = \{nc_1, nc_2, \cdots, nc_N, 0, 0\} \tag{4.3}$$

$$\hat{OD}_q = \{nc_1, \cdots, nc_N, 0, 0, 0\} \tag{4.4}$$

To avoid zero probability, we adopt Laplace smoothing, specifically add-one smoothing, to calculate the frequency distribution.

$$laplace\, smoothing = \frac{nc_i + 1}{\sum_i^L nc_i + 1} \tag{4.5}$$

$$laplace\, smoothing : \hat{OD}_p \rightarrow FD_p \tag{4.6}$$

$$laplace\, smoothing : \hat{OD}_q \rightarrow FD_q \tag{4.7}$$

$FD_p = \{fd_1^{(p)}, fd_2^{(p)}, \cdots, fd_L^{(p)}\}$ denotes the frequency distribution of each n-gram entry after applying laplace-smoothing for tracefile tr_p. Similarly, FD_q denotes the smoothed frequency distribution for tr_q.

After this procedure, we can compute the cross entropy of two tracefiles tr_p and tr_q as:

$$C(tr_p, tr_q) = \sum_{i=1}^{K} (fd_i^{(p)} - fd_i^{(q)}) \cdot \log \frac{fd_i^{(p)}}{fd_i^{(q)}} \tag{4.8}$$

The value of cross entropy has a lower bound of 0 if two distributions are identical. A small value of cross entropy indicates a large similarity between two distributions.

4.6 Self variance and mutual distance

In this section we validate our frequency distribution profile methodology by looking at the self-variance and the distance of profiles calculated with cross-entropy, which we define as follows.

Suppose $T_M = \{t_1^{(M)}, t_2^{(M)}, \cdots, t_P^{(M)}\}$ represents the set of tracefiles for Application M. Suppose $T_N = \{t_1^{(N)}, t_2^{(N)}, \cdots, t_Q^{(N)}\}$ represents the set of tracefiles for Application N.

The *self-variance* of Application M with tracefile set T_M is the average value of the cross entropy for any pair of tracefiles in set $T_M \times T_M$:

$$\text{self variance}(T_M) = average(C(t_i^{(M)}, t_j^{(M)})) \tag{4.9}$$

$$\forall (t_i^{(M)}, t_j^{(M)}) \in T_M \times T_M \tag{4.10}$$

Table 4.2: Typical Applications in DDMs and corresponding libraries and trace-file size (average number of distinct n-grams of length 6). The experiments are conducted in Ubuntu 18.04.

	Application name	**Library**	**# ngrams**
APP 1	Unbalanced Classifier	TensorFlow, Numpy, csv	1370
APP 2	Text Classification from scratch	TensorFlow, Numpy	5491
APP 3	Collaborative Filtering	TensorFlow, Numpy, Pandas	1394
APP 4	Federated learning	Syft, aiomqtt, signal, Pandas, Numpy	1140
APP 5	S2S learning to perform addition	TensorFlow, numpy	1452
APP 6	Train a LSTM	TensorFlow, Numpy	1948
APP 7	Train a quisi-svm	TensorFlow	2204

Similarly, the average *mutual distance* between two applications M and N is calculated as:

$$\text{Mutual Distance}(T_M, T_N) = average(C(t_i^{(M)}, t_j^{(N)})) \tag{4.11}$$

$$\forall (t_i^{(M)}, t_j^{(N)}) \in T_M \times T_N \tag{4.12}$$

To validate our methodology we select 7 typical DDM applications in the related field of machine learning. Each application is encapsulated into a Docker container [53]. All the algorithms are written in Python and primarily rely on TensorFlow, an open source library that helps to develop and train ML models [54]. The Docker images of all applications have a common underlying image of *Python Strech 3.6* and the algorithm script is running on top of it. The applications and the used libraries are shown in detail in Table 4.2. We run containers of all 7 applications as the experimental setup depicted in Figure 4.3. The kernel operating system is *Ubuntu 18.04* and we run 4 times each application.

We compute *self variance* and average *mutual distances* among all 7 containerized applications. The computation results are shown in Table 4.3.

The *self variance* of each application is shown as the value on the diagonal line. The *self variance* is small compared to the mutual distances. This implies our proposed classification methodology can provide a lower false negative rate than the classic n-gram method.

When we look at the mutual distances between application we observe a number of interesting features. First we see that distances range from a value of 0.38 (App3-App5 pair) to 23 (App4-App6 pair). This large variation can be explained by observing that a number of our applications use the same libraries; we therefore expect that those applications are more difficult to distinguish. This is in fact the case for App4 which has a very unique set of libraries and therefore gives larger distances.

Table 4.3: Self variance and mutual distances among all 7 containerized applications.

	APP 1	**APP 2**	**APP 3**	**APP 4**	**APP 5**	**APP 6**	**APP 7**
APP 1	**0.064**	5.21	1.42	12.5	3	3.35	5.56
APP 2	-	**0.038**	4.81	20	6.09	6.68	9.23
APP 3	-	-	**0.002**	18	0.38	0.66	5.2
APP 4	-	-	-	**0.045**	22.3	23	19
APP 5	-	-	-	-	**0.0007**	0.58	2.76
APP 6	-	-	-	-	-	**0.0009**	4.74
APP 7	-	-	-	-	-	-	**0.02**

4.7 Stability of proposed methodology

The system call trace files of a running program depend on configurations of the execution environment. As mentioned in Section 4.3, the generated profiles of a specific application should be closely clustered together. In this section, we will investigate the stability, quantified as self-variance, of generated profiles over different operating system platforms and different training data sets.

In Section 4.6 we discussed the self-variances of our proposed methodology for 7 ML applications. Now we mainly focus on 3 of them: Application 1 - *Unbalanced Classifier*, Application2 *Text Classification from scratch* and Application3 - *Collaboration Filtering* as they are widely used ML algorithms in DDMs.

A trace file database is established with various configurations shown in Table 4.4.

Table 4.4: Tracefile database for three model application with different dataset sizes

App 1	App 2	App 3
Ubuntu 18.04	CentOS 7	Debian GNU Linux 9
DS0 240000 items	DS0: Training: 25000; Test:25000	DS0: 100000 items
DS1: 120000 items	DS1: Training: 32000; Test:10000	DS1: 80000 items
DS2: 80000 items	DS2: Training: 20000; Test:20000	DS2: 50000 items
DS3: 50000 items	DS3: Training: 16000; Test:10000	DS3: 40000 items
DS4: 20000 items	DS4: Training: 10000; Test:10000	DS4: 30000 items

The 3 applications are containerized with Docker. For each application, we run the Docker container in 3 hosts with different Linux operation systems: *Ubuntu 18.04 (kernel version 4.15.0)*, *CentOS 7 (kernel version 3.10.0)* and *Debian GNU Linux 9 (kernel version 4.9.0)*. In each host, the containerized ML algorithm is

trained with 5 different training sets. Hence there are in total 15 configurations, ie. operating system and dataset pairs, per application. The container runs 3 times under each configuration setting; and we have at the end 45 trace files available. For all trace files, system calls generated by the container runtime are filtered out.

4.7.1 Stability over different platform OSs

One of the benefits of application containerization is the portability over different platforms. In DDMs environments it can be expected that the same application will run at different times on different platforms, and that the operating system (OS) of the execution platforms chosen to run on may vary. It would be obviously very convenient if we could determine that a baseline profile generated by an application in a certain OS can be used for classification in another OS. To assess this we need to determine what is the stability of generated profiles over different platforms as this influences the classifier accuracy.

In our experiment we ran the three applications on 3 host machines, each one configured with the following operating systems: Ubuntu 18.04, CentOS 7 and Debian GNU Linux 9. For each application, the container is running 5 times with the same training data set. We then calculated the cross entropy for all application profiles produced in one OS and the cross entropy for pairwise comparison of application profiles in different OSs.

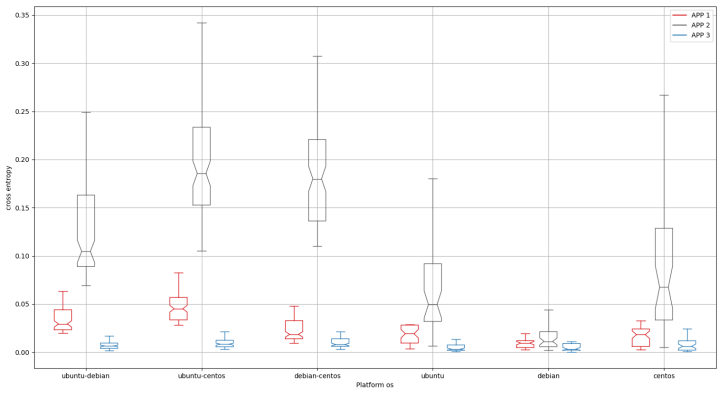


Figure 4.5: Stability of the profiles, expressed as cross entropy, for 3 model applications over execution platforms running three OSs: Ubuntu, Debian and CentOS.

As shown in Figure 4.5, there are in total of 6 groups of boxes. Each group contains the results for the runs of the same application: application 1 in red, application 2 in black and application 3 in blue. The 3 groups on the right show the

values of cross entropy of the three application's profiles generated on machines with a specific OS. The 3 groups on the left show the cross-platform variance of the profiles for the same application when running on two host machines with a different OS.

Not surprisingly, we observe that the variance among different platforms is application-dependent. According to Figure 4.5, the profiles of application 2 suffers from more variance compared to the other two. One explanation is that application 2 is more resource intensive and that more device management system calls are inserted into the program behavioral traces. This generates more rarely occurring n-grams that contribute to higher variance. Also, we can observe that *Debian GNU Linux 9* provides the most stable profiles for all 3 applications.

When we compare the variances for runs in the same OSs, with values ranging from 0 to 0.27, to the runs with different OSs pairs, with values ranging from 0 to 0.34, the cross-platform variance is only slightly larger. In addition, as shown in Table 4.3, the mutual distances among the 3 applications are 5.21 (APP1- APP2), 1.42 (APP1 -APP3) and 4.81 (APP2 - APP3). The absolute values of the variance cross platforms are smaller than the mutual distances. This indicates the variance of profiles over different platform os is unlikely to cause false negatives. The results strongly suggest that our proposed methodology will support container portability with quite stable performance, at least for typical DDM ML algorithms.

4.7.2 Stability over different training data sets

Not only it is important to determine the stability of profiles across OS-es, it is likewise essential to investigate the stability over different training data sets. There are two reasons for this. First, to reduce the risk of data leakage in a DDM, the authorized party is not allowed to access the data objects directly and to execute the compute algorithm on them. This means that the data used to generate the profile in the authorized party is normally different from the one used for classification in the execution platform. Secondly, a DDM customer may retrain a machine learning model multiple times with different training data sets for higher accuracy.

To determine the stability of each application in this case we trained the ML model with the data set shown in Table 4.4. The operating system of the host machine we used is *Ubuntu 18.04*. We chose this for its wide adoption in DDMs. For each application there are 5 training data sets of various sizes. The application containers are running 5 times with every training data set. At the end, we obtain 25 trace files for each application.

Figure 4.6 shows the stability of the application profiles expressed as cross entropy values for all pairs in the Cartesian product of the tracefile set. The cross entropy between profiles of the same application is small, in the large majority of cases with values lower 0.1. In particular, the profiles of application 1 are the most stable across the 5 training data sets, and the cross entropy has values

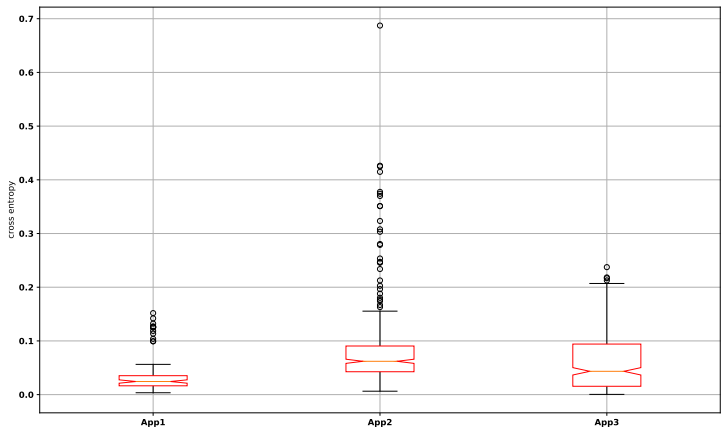


Figure 4.6: Stability of the three model application profiles over all data sets in Table 4.4 expressed as cross entropy.

ranging from 0 to 0.15 including outliers. Application 2 and 3 have a few outliers with cross entropy higher than 0.1. In particular application 2 has 12 outliers of relatively high entropy values, ranging from 0.17 to 0.7. As explained in Section 4.7.1 this is because more device management system calls are generated in this case and inserted into the program behavioral traces. Outliers decrease the classification accuracy rate.

From our two experiments, stability over different OSs and stability over different data sets, we can conclude that profiles are fairly closely clustered together, as seen by the low value of cross entropy. This means that we expect that our classification methodology will be able to deliver good results independently from the OSs and training data set the reference profiles has been generated with.

4.8 Classification accuracy

In this section, we will investigate how well our proposed classifier works for distinguishing containerized applications. First we will describe our experiments and then present our results.

4.8.1 Procedure

To estimate the accuracy rate of our methodology, we will choose one profile per application to be the reference profile, which is a real life scenario would be the one produced by an authorized party. We will then present to the classifier profiles from a specific application and determine if it correctly identifies the application.

As shown in Table 4.3, application 4 uses distinct libraries and it has larger distances to all other 6 applications. we will therefore not consider it further in our experiment, as we expect to be always distinguishable.

The tracefile database T we use contains 45 tracefiles for all the other 6 applications running with different configurations - OS and training data set. We denote the tracefile database for application k as T_k.

The procedure we follow is shown in Algorithm 4.1. We first input the tracefile database T in line 1. We run the experiment for R times and we will have different authorized profiles in each iteration. From line 3 to line 7, we randomly select a tracefile from each application tracefile database T_k and compute the authorized profile $p_{A,k}$ for the specific iteration. Then we input all tracefiles in T into the classifier, excluding those used for generating $p_{A,k}$. From line 8 to 15, we classify each tracefile into an application category with the shortest distance principle. Finally, from line 16 to 24, we compare the determined result with the tracefile label and compute the accuracy rate for R rounds.

Algorithm 4.1 Classification accuracy procedure

1: $T \leftarrow$ Tracefile database
2: **for** $r = 1, 2, \cdots, R$ **do**
3: **for** $T_k \in T$ **do**
4: Randomly choose a tracefile $tr_j^k \in T_k$
5: $tr_{a,k} \leftarrow tr_j^k$
6: Generate authorized profile $p_{A,k}$ for $tr_{a,k}$
7: **end for**
8: **for** $tr_m \in T$ **do**
9: **if** $tr_m \in \{tr_{A,1}, tr_{A,2}, tr_{A,3}\}$ **then**
10: Go to 8
11: **end if**
12: Generate profile p_m for tr_m
13: **for** $k = 1, 2, \cdots, K$ **do**
14: $C_{m,k} \leftarrow cross\,entropy(p_m, p_{A,k})$
15: $K_{D,m} = \arg\min_k \left\{C_{m,1}, C_{m,2}, \cdots, C_{m,K}\right\}$
16: **if** $K_{D,m} = k$ **then**
17: $PD_{pos} += 1$
18: **else**
19: $PD_{neg} += 1$
20: **end if**
21: $R_a = \frac{PD_{pos}}{PD_{pos} + PD_{neg}}$
22: **end for**
23: **end for**
24: **end for**

4.8.2 Results

Table 4.5 shows the accuracy confusion matrix of the classifier for 6 containerized ML algorithms for $R = 40$. In each row we report the classification result of the specific application. The last column shows the mean value and standard variation of classification accuracy rates for application k among 40 rounds.

Table 4.5: The confusion matrix of the classifier for 6 applications running with various platform OSs and training data sets.

	APP 1	APP 2	APP 3	APP 5	APP 6	APP 7	mean (%) ± std
APP 1	**1529**	0	209	0	22	0	86.7 ± 0.15
APP 2	0	**1760**	0	0	0	0	100 ± 0
APP 3	0	0	**1623**	137	0	0	92.2 ± 0.15
APP 5	0	0	61	**1483**	216	0	84.2 ± 0.21
APP 6	0	0	0	0	**1760**	0	100 ± 0
APP 7	0	0	0	0	0	**1760**	100 ± 0
							93.85

We can observe that the classifier can always achieve an accuracy rate of 100% for App2, App6 and App7 no matter how we built the authorized profiles. The prediction accuracy is only 86.7% for App1, with 209 samples classified as App3 and 22 samples classified as App6. False classifications mainly occur between App3 and App5, App5 and App6. This is because of their mutual distance, as shown in Table 4.3, are only 0.38 and 0.58. For those applications with lower average classification accuracy, App1, App3 and App5, the *std* is also higher. This indicates that the selection of authorized profiles plays an important role in the performance of the classifier. The classifier can predict profiles with 100% accuracy for some applications sharing the same libraries and overall accuracy for all applications is as high as 93.85%.

4.9 Discussion

The work presented so far mainly focuses on distinguishing applications running inside the containers based on system call monitoring. We must stress our methodology is not concerned with the maliciousness of the code. The focus is on whether the code is authorized to run. Detecting malicious code and intrusions in real time is out of the scope for the current paper, but we will extend our architecture in Fig. 4.1 to include this as a separate component.

The performance of our proposed methodology is lower for applications that are pretty similar to each other. It will be a focus of our future work to determine if more fine-grained classifiers, namely Support Vector Machine (SVM), Decision

Tree, KNN(K-nearest neighbour), can improve the current classification accuracy. These more refined methods require many more tracefiles than the ones we currently. Still, in many DDMs this larger data set tracefiles is difficult to collect, hence our methods will still be the one adopted, and we believe deliver more than sufficient discrimination power, as seen by its overall accuracy of 93.85%.

Another interesting aspect to consider is the effect of the tracefile size, i.e. the number of distinct n-grams contained in it. As we can see from Table 4.2 the six applications we chose produce tracefiles which contain between 1140 and 5941 distinct n-grams. When we look at Table 4.5 we can see that there is no obvious correlation between number of n-grams and accuracy, which is also a good performance indicator for the suitability of our method as it is application size agnostic.

4.10 Related work

There are recent studies modelling program behaviour with system calls. The work in [55] proposed to use frequency of individual system calls to build the normal profiles of an application. Such profiles may lose some important information about the application behaviours because they do not capture the sequential relationships among systems calls. In addition, this method is vulnerable to mimic attacks. Exploiting the profiles, the adversaries can mimic the benign program to perform malicious actions [48]. [46] proposed to profile normal behaviours of a running process as a set of short sequences of system calls. The profile is an enumeration of all fixed-length subsequence and an anomaly is flagged if a sufficient number of new short subsequences occur. [56] proposed a similar version of program profiling methodology, which is called STIDE. In this case the authors use Hamming distances to calculate the dissimilarities to detect anomalies. These works provide us with a good starting base for our research.

[57] proposed in their recent work to use Hidden Markov Chain to model normal behaviours of applications. The work in [49] proposed to train an LSTM model with benign behaviours of a running program. However, those methods are very computationally expensive and require a large amount of data.

The work in [58] aimed to analyse Android App behaviours using system calls. They built the App profile based on the frequency distribution of individual system calls. Their conclusion was that system calls are not sufficient to classify application behaviour. In our work we will demonstrate that this possible as long as the profiling methodology is more refined than the one adopted by these authors.

4.11 Conclusions and future works

In this chapter, we introduced architecture to distinguish algorithms running inside containers by monitoring system calls during the execution stage. We proposed to profile a containerised algorithm with n-gram occurrence distribution and compute dissimilarity with Laplace smoothing and cross entropy. With our experimental results, we demonstrated that the methodology allows profile reuse across execution platforms with different OSs to support container portability and profile reuse across different datasets to allow retraining. We showed that we could gain high classification accuracy with typical ML applications.

In Chapter 5, we further extend our architecture to include intrusion detection modules. This allows us to distinguish an algorithm as being authorised or not, and determine its behaviour at runtime.

Chapter 5

Real time intrusion detection systems

The collaborating parties, especially the data providers, normally are concerned about the security of their outsourced data. The pre-agreed policy normally contains rules to forbid data leakage and data modification during the execution stage, which are also threats we identified in Chapter 3. The DDM infrastructures should have the capacity to help enforce such policy rules. In this chapter, we extend the architecture described in Chapter 4 and introduce a real time intrusion detection system(IDS) that monitors the system calls of a running container based on the one-class support vector machine (OC-SVM) algorithm. We also investigate the influence of various feature extraction methods, kernel functions and segmentation length with four metrics. Same as in Chapter 4, this chapter contributes to answer *RQ3*.

This chapter is based on:

- **Lu Zhang**, Reginald Cushing, Cees de Laat, and Paola Grosso. "A real-time intrusion detection system based on OC-SVM for containerized applications." In 2021 IEEE 24th International Conference on Computational Science and Engineering (CSE), pp. 138-145. IEEE, 2021.

5.1 Introduction

In a DDM, there is a unique identifier for each data and compute object. The parties agree on permissible actions on specific data and compute objects and express them into a policy [59]. The compute objects are containerised for better portability. The DDM infrastructure implements policy enforcement components to mitigate possible vulnerabilities faced by such data exchange applications, such as container escalation attack [60]. Preventive countermeasures must be

implemented as the first line of defence. However, attackers will keep developing novel techniques to bypass the existing security mechanisms. Hence, a real-time intrusion detection system (IDS) is essential to enhance the operations of such digital infrastructures.

An IDS can be mainly classified into two categories, namely, signature-based and anomaly-based. The signature-based IDS detects attacks by matching the monitoring metrics with existing patterns. Consequently, it can not identify zero-day attacks. An anomaly-based IDS learns a profile describing normal behaviours and flags a potential attack if a sufficient deviation from the normal profile occurs [61].

We propose a hybrid real-time intrusion detection system with system calls generated by a running container. We adopt the One-Class Support Vector Machine (OC-SVM) as the anomaly detection algorithm due to its capability to deal with complex non-linear problems. To detect malicious behaviours in a real-time manner, the streaming system calls are separated into segments before being mapped into feature vectors. Then we apply the signature-based methodology to reduce false alarms. An anomaly detection algorithm is trained For each compute object if used for the first time. Adapting to the dynamic characteristics of the application behaviour, the anomaly detection algorithm is retrained whenever new data is available. It is vital to ensure that both training and retraining data are attack-free. In our proposed system, the training data is collected in a secured environment and retraining data is analysed and sanitised.

We also evaluate how the OC-SVM algorithm works for detecting anomalies with system call traces. The performance highly depends on the statistical distribution of the attack traces and modern attacks are more difficult to distinguish [62]. We construct a new dataset including system call traces of modern container-specific attacks and adversarial ML attacks. Three numeric metrics are measured to evaluate the performance, namely, True Positive Rate (TPR), False Positive Rate (FPR) and Area under the ROC curve (AUC). We also investigate how the different feature extraction methods, kernel functions, segmentation lengths, influence the IDS performance.

The rest of this chapter is organised as follows. We present our proposed system architecture in Section 5.2. The details of the *Detection Engine (DE)* are explained in Section 5.3. Section 5.4 and Section 5.5 present the constructed dataset and the experimental setups. We introduce our results, analysis and discussions in Section 5.6 and Section5.7. In Section 5.8 we compare our methodology with existing ones. In Section 5.9, we introduce the conclusions of this chapter.

5.2 System description

Figure 5.1 describes the architecture of our proposed real-time intrusion detection system based on OC-SVM. In general, the training stage is conducted offline in a secured environment, a *authorized party*, in a centralized manner and then distributed to the *endpoint execution platform*, for real time anomaly detection. The *centralized authorized party* consists of an *Initial Training* module, an *Integrity Verification and Retraining* module and a *Model Database*. The *Endpoint Execution Platform* contains a *System Call Monitoring* module and a *Decision Engine*.

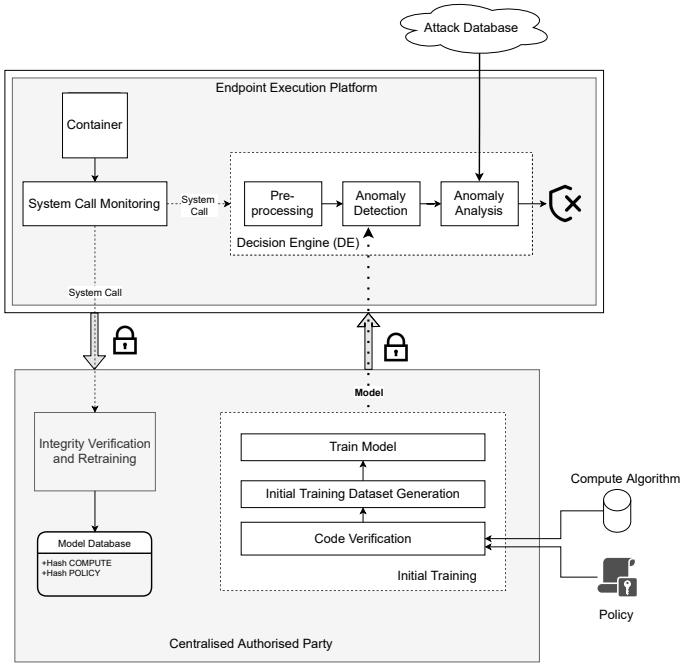


Figure 5.1: The architecture of the intrusion detection system.

Before delegating a data exchange application to a DDM, the collaborating parties first agree on a policy describing permissible actions on the data and compute objects. The policy and compute objects are sent to a trustworthy *authorized party*. The security experts check the source code manually and verify whether it complies with the policy. Then the *authorized party* generates the initial training data and trains the anomaly detection algorithm. This ensures that the training data is clean and not contaminated by adversaries. Both the pre-trained detection model and verified compute objects are sent to the *endpoint execution platform* via a secured communication channel. The *authorized party* signs and encrypts the pre-trained detection model and the compute object, normally a container image, with the public key of the *endpoint execution platform* to ensure integrity.

On the *endpoint execution platform*, the containerized compute object may

perform operations on the data objects agreed in the policy. The *system call monitoring* module gathers the system call traces with Sysdig. It sends the streaming system calls to the *Detection Engine* module. This module detects anomalies with the pre-trained OC-SVM model and decides whether it is necessary to apply countermeasures. The details of this module will be discussed in Section 5.3.

In the meantime, the *System Call Monitoring* module also sends the system calls back to the *authorized party* for model retraining if needed. The behaviors of some applications are dynamic in nature, so it is necessary to retrain the model periodically. The *Integrity Verification and Retraining* module checks the integrity of the received system call traces and verifies whether they are attack free. The same anomaly detection model, with exactly the same parameters, is run in parallel with the received streaming system call traces in the *authorized party*. The receiving system calls are recognized as attack-free if the alarm rate is close to the recorded FPR value of this model in the initial training. Then these system call traces are allowed to retrain the model. We adopt this mechanism to reduce the likelihood that the model is retrained with contaminated samples.

5.3 Detection engine

The Detection Engine (DE) is comprised of three components: a Pre-Processing Module, an Anomaly Detection module and an Anomaly Analysis model.

5.3.1 Pre-processing module

When the container runs, the *System Call Monitoring* module captures the system calls and passes them to the *Pre-processing* module. This divides the streaming system calls into segments with a fixed window size: this is needed because the inputs of the classic ML algorithms are fixed length vectors. Furthermore this segmentation allows us to perform our detection while the systems calls are coming in real time.

The Pre-Processing module also maps these segments system call traces into vectors in the feature space. In our work we considered three feature extraction methods, namely *tf*, *tf-idf* and *ngram*. We will describe them in more detail in Section 5.5.2

5.3.2 Anomaly detection module

A pre-trained IDS learning algorithm is running in the *Anomaly Detection* module and determines whether an input feature vector is anomalous or not. Here we adopted One-class Support Vector Machine (OC-SVM) as the IDS learning algorithm. Our choice stems from the fact that OC-SVM is good at dealing with

complex non-linear problems. This results in it being widely used for intrusion detection systems [63].

The general idea of an SVM algorithm is to find a hyperplane that separates the normal and abnormal data points with a maximized geometry margin by solving an optimization problem. It maps the input data points into a new feature space of higher or even infinite dimensions with kernel functions. Therefore, the original linearly non-separable data patterns may be converted into, with high likelihood, linearly separable patterns in the high dimensional space [64].

Similar to standard SVM, the OC-SVM also aims to find a decision boundary with a maximum geometry margin [65]. However, it is an unsupervised learning algorithm and does not require any labeled training data. This is suitable for usage in anomaly detection, where training datasets are normally unbalanced. The OC-SVM algorithm considers the anomalous data points to be close to the origin, while the normal data points are far from the origin. It uses a spherical decision boundary instead of a plane in the higher-dimension feature space [66]. The algorithm aims to find a sphere with minimal volume that contains the most normal data points. The objective function is:

$$\begin{aligned} &\min_{R,\vec{a}} R^2 + C\sum_{i=1}^{N}\xi_i \\ &\text{subject} \quad \text{to}: \\ &\|\Phi(X_i) - \vec{a}\| \leqslant R^2 + \xi_i, \quad for \quad i = 1, 2, \cdots, N \\ &\xi_i \geq 0, \quad for \quad i = 1, 2, \cdots, N \end{aligned} \tag{5.1}$$

The spherical decision boundary is characterized with its center $\vec{a}$ and the radius R. X_i is the ith training data in input space I and N is the total number of training samples. $\Phi()$ denotes a feature map from the input space I to a high-dimensional feature space F. ξ_i is a slack variable to prevent over-fitting from some noisy data points by creating a soft margin. It allows some data points to lie within the margin. C is a constant to determine the trade-off between the sphere volume and the number of data points it can hold.

A kernel function is defined as:

$$K(X_i, X_j) = \Phi(X_i) \cdot \Phi(X_j)^T \tag{5.2}$$

Given two data points X_i, X_j, the output of the kernel function is the dot product of their mapping in new space F. As the decision boundary of an SVM algorithm only relies on the dot product in feature space F, the explicit projection is not necessary. In our work, we chose two popular kernels, linear and Gaussian, to evaluate the performance of our system. The details will be discussed in section 5.5.3.

5.3.3 Anomaly analysis module

The *Anomaly Analysis* module analyses the anomalies identified by the *Anomaly Detection* module and provides information to the security experts determining whether to take countermeasures or not. For an input segment tracefile, which is flagged as an anomaly, this module conducts the following operations:

- Match the system call sequences with existing attack database;
- Check the neighboring segments of system call traces and determine whether this is a standalone anomaly or not;

An identified anomaly is recognized as standalone if all the data points in the neighboring area (N_s points before and N_s points after) are identified as normal. N_s is an adjustable parameter to define the range of the neighboring area of a data point. The *Anomaly Analysis* module assigns each anomaly a level based on the outcomes of the above operations. An anomaly is marked as 'High' if the traces match any existing attack in the database. An anomaly is marked as "Low" if it is a standalone point. The remains are marked as "Medium". The detailed matching approach and its performance has been explained in our previous paper [67]. The effectiveness of recognizing standalone points will be discussed in Section 5.7.

5.4 Experimental dataset construction

The emerging microservices pose many challenges for real-time intrusion detection systems, due to their highly dynamic natures. Also, the performance of IDS learning algorithms highly depends on the statistical properties of the training dataset. When starting our evaluation we found out that there are currently no public databases containing system call traces of container-specific applications or attacks. We therefore set out to construct our own training dataset, in order to validate how *oc-svm* works for detecting modern attacks for containerized applications.

To tailor our work to the DDMs we first identified the most typical applications expected to run in such environments: databases services and training machine learning models are the most likely type of use cases. The former are examples of dynamic applications with many users interacting with the system; the latter are more static applications that do not have many users involved and have a more constant execution path every time. For each one of them we implement an attack with penetration tools which provides us with a suitable dataset.

5.4.1 Dynamic applications: CouchDB and MongoDB

As example of dynamic applications we select two NoSQLMap databases, CouchDB and MongoDB, to investigate the performance of the proposed intrusion detection system.

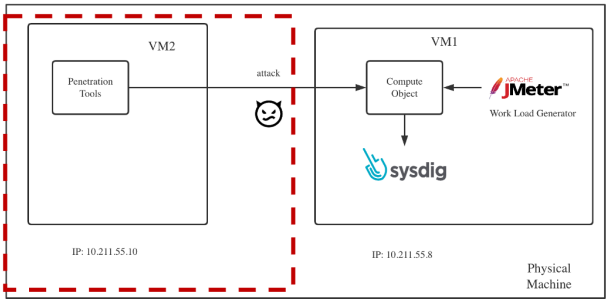


Figure 5.2: Platform for generation of normal and abnormal traces for dynamic applications (CouchDB and MongoDB).

Figure 5.2 shows the experimental platform we built for generating normal and abnormal system call traces. To avoid the inferences of other user activities, we set up a Docker container running CouchDB or MongoDB server in a virtual machine (VM1). Sysdig is implemented in the kernel space to collect system call traces generated by the running container. To simulate the dynamic behaviors of real-world database users, we send requests to the container from the same virtual machine (VM1). This is conducted with Apache JMeter, which is an open-source workload generation tool. For abnormal traces, we conduct attacks with exploitation tools, e.g Nmap and Metasploit, in another virtual machine (VM2).

For CouchDB, we use the HTTP sampler of JMeter. This sampler enables choosing the proper HTTP traffic types, e.g GET or POST. For MongoDB we use JSR 223 Sampler of JMeter to generate the traffic. Web requests are sent to the server to induce database operations. For each application, there are two threads in JMeter, thread 1 and thread 2, which send requests to the containerized server simultaneously. Thread 1 includes 100 users, who perform operations of inquiring documents and basic information in different databases. Thread 2 includes 3 users, who perform operations including updating, creating and deleting databases. While JMeter is generating dynamic traffic, Sysdig monitors the system calls of the server container, which is used as the normal traces of the application. Table 5.1 shows the information of the performed attacks and tracefile sizes for the two dynamic applications.

Table 5.1: Applications and attacks of the constructed dataset for the two dynamic applications

Application	Attack	Exploitation Tool	Jmeter Sampler	# Syscall Symbols	
				Normal	Abnormal
CouchDB	Container Escalation	Metasploit	HTTP	7458046	5199817
MongoDB	Brute Force	Nmap	JSR 223	40463150	4449052

5.4.2 Static application: machine learning applications

Another common application for DDM is to train a ML learning model with data from multiple parties. We chose image classification with Concurrent Neural Networks (CNN) with MINST dataset to validate the performance of our anomaly detection system.

Recent research shows that deep learning algorithms are vulnerable to adversarial attacks. [68]. This attack generates adversarial training samples in the runtime. The adversarial samples are nearly unnoticeable for humans but can fool the model with high confidence. There are a number of adversarial sample generation approaches in the literature [69]. Concretely, we adopted the Projected Gradient Descent (PGD), Basic Iterative Method (BIM), Carlini and Wagner (CW), Fast Adaptive Boundary (FAB), multiple steps fast gradient symbol method (MIFGSM), PGDDLR, Square and TPGD method to generate adversarial samples [70].

The ML algorithms is encapsulated with a Docker container. For normal traces of the application, Sysdig collects system calls for the entire training stage. For abnormal traces, Sysdig gathers system calls when the block of code, which generates the adversarial samples, is executed. Table 5.2 shows the information of the performed attacks and tracefile size for the image classification application.

5.5 Experimental design

In this section, we implement the proposed architecture and evaluate its performance. The key question we want to answer is: *How does the OC-SVM based DE perform for detecting modern attacks?*

More precisely, we want to investigate the influence of different feature extraction methods, different kernel functions and different segmentation lengths.

The experiments are conducted with the dataset discussed in Section 5.4. OC-SVM is an unsupervised learning algorithm; the model is trained with only normal data and tested with both normal and abnormal data. To avoid over-fitting, in our experiments we use K-fold ($K = 10$) cross validation. We first shuffle the normal data points randomly and split them into K folds. For each interaction

Table 5.2: Applications and attacks of the constructed dataset for the ML static application

Application	Attacks	Exploitation Tool	# Syscall Symbols	
			Normal	Abnormal
Image Classification	AML: PGD	Proof of Concept	2798258	4360000
	AML: BIM			960967
	AML: CW			4728113
	AML: FAB			1394676
	AML: MIFGSM			118007
	AML: PGDDLR			466024
	AML: Square			182813
	AML: TPGD			87884

$k \in [\![1, K]\!]$, one fold of the normal data points and all the abnormal data points are used for testing. The remaining 9 folds are used for training the model. After K interactions, every fold has been used once for testing. The final value of the evaluation metric is the average of K values [71].

We deployed our experiments in a VM equipped with 4 CPU cores at 2.9 GHz and 16 GB memory. The OS is Ubuntu 18.04 LTS, kernel 4.15.0.

5.5.1 Segmentation length

As already discussed in Section 5.2, the streaming system calls are divided into segments to achieve detection results in the real time. The window size that segments the trace is called *segmentation length* and denoted as L_s. Hence a trace of L system call symbols can be spit into $\lfloor L/L_s \rfloor + 1$ segments. To investigate the influence of different *segmentation lengths*, we set $L_s \in \{1000, 2000, 5000, 10000, 15000, 20000, 25000, 30000, 50000\}$.

5.5.2 Feature extraction

Three feature extraction methods are used in our experiment, namely term-frequency (*tf*), term frequency-inverse document frequency (*tf-idf*) and *ngram*. We use S to denote a system call symbol, T_s to denote the trace of a segment. $tf(S,T_s)$ denotes the weight of symbol s in trace T_s and is computed as the occurrence frequency.

$$tf(S,T_s) = \frac{count\,of\,symbol\,S\,in\,T_s}{count\,of\,system\,calls\,in\,T_s} \tag{5.3}$$

tf-idf also considers how rare a system call symbol occurs in an entire trace set. We use T to represent the entire trace set and N_T to represent the total number of traces in T. The *tf-idf* of a symbol s is the product of term-frequency and the inverse-document frequency of that symbol.

$$\begin{aligned} tfidf(s,T_s,T)) &= tf(s,T_s)\cdot idf(s,T) \\ idf(s,T) &= log(\frac{N_T}{count(T_s \in T : s \in T_s)+1}) \end{aligned} \tag{5.4}$$

ngram captures the sequential information of system call traces. A trace is divided into fixed length sub-sequences, called *n-grams*, with a sliding window of length n. The feature vector is essentially the distinct n-grams weighted by their occurrence frequency. In our experiment, we use n equal to 3.

5.5.3 Kernel functions

We use two kernel functions for the OC-SVM learning algorithm in our experiment, namely linear and Gaussian. As described in Equation 5.2, the kernel function computes the dot product of two feature vectors in feature space F with a function of vectors in input space I. Let X_i and X_j be two feature vectors in input space I. The linear kernel is simply the dot production of X_i and X_j.

$$K(X_i, X_j) = X_i \cdot (X_j)^T \tag{5.5}$$

The Gaussian kernel is computed as following:

$$K(X_i, X_j) = exp(-\frac{\left\|X_i - X_j\right\|^2}{2\sigma^2}) \tag{5.6}$$

5.5.4 Evaluation metrics

To evaluate the performance of the DE, we measure four metrics, namely true positive rate (TPR), false positive rate (FPR), area under the ROC curve (AUC) and execution time.

The values of TPR and FPR are calculated as following:

$$TPR = \frac{TP}{TP+FN} \tag{5.7}$$

$$FPR = \frac{FP}{TP+FP} \tag{5.8}$$

TP (true positive) indicates the number of anomalies that are classified correctly. FN (false negative) indicates the number of anomalies that are not detected by the classifier. FP (false positive) represents the number of normal samples that are classified as anomalies.

A ROC curve is a graphical plot that illustrates the performance of a classifier with different discrimination thresholds. This curve plots TPR (y-axis) against FPR (x-axis). The OC-SVM algorithm essentially does not provide any probability score. In the experiment, we approximate the score as a function of the distance from the input data point to the decision boundary. AUC measures the total 2-dimensional area under the ROC curve. It summaries the information of the ROC curve and measures the capability of a classifier to distinguish between positive and negative classes. The higher the AUC value, the better performance a classifier can achieve. The AUC value ranges from 0 to 1 and the classifier is perfect if $AUC = 1$.

5.6 Performance of anomaly detection module

We focused our attention to the evaluation metrics, namely TPR, FPR, AUC and execution time, of the *Anomaly Detection* module. In Figure 5.3 we show their values for different kernels and feature vectors as function of the *segmentation lengths*. In Table 5.3 we evaluate the same metrics for different applications and attacks.

Comparison among different segmentation length

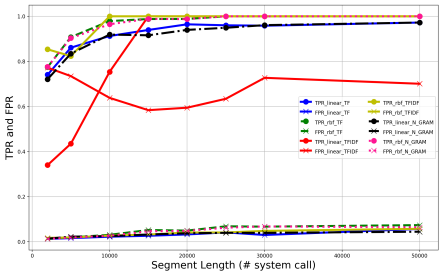
The choice of a specific *segmentation length* is part of the DE configuration and it is important for us to determine what is the impact of its value.

As shown in Figure 5.3, both TPR and FPR values show a growing trend with larger *segmentation length* values for all applications and features. The TPR converges at a specific point with a value close to or equal to 1. The performance of the *Anomaly Detection* model degrades with higher FPR if the *segmentation length* exceeds that point. The performance of the DE degrades significantly with an improper *segmentation length*, particularly in the region of lower values. It is therefore vital to choose the most appropriate value.

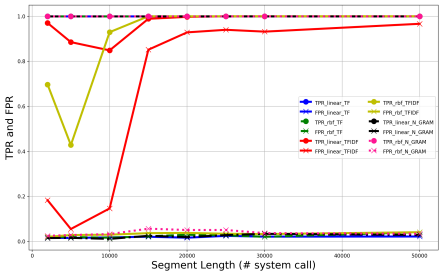
We observe that there is a *segmentation length* which can provide optimal performance for the module. According to the experimental results of the three applications we used, the optimal *segmentation length* is 30000.

Comparison among different features and kernels

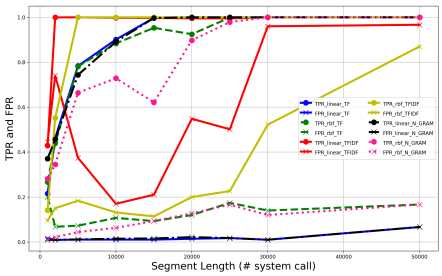
As seen in Figure 5.3, the feature *tfidf* performs significantly worse than the *ngram* and *tf* features for all applications. In the worst case, the FPR values can be as high as 0.78, 0.91, 0.92 and the TPR values can be as low as 0.38, 0.42, 0.15 for CouchDB, MongoDB and Image Classification respectively. This is not an acceptable performance of an IDS and there are two possible explanations. Firstly, we must observe that the *idf* factor is extracted from only the training dataset to avoid information leakage. A distortion will occur when applying it



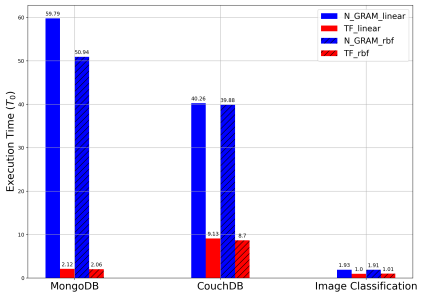
(a) CouchDB and Arbitrary Code Execution Attack



(b) MongoDB and Brute Force Attack



(c) Image Classification and Adversarial ML Attack (PCA)



(d) Execution Time

Figure 5.3: The FPR and TPR values (Figure 5.3b, 5.3a, 5.3c) and execution time (Figure 5.3d) of the OC-SVM with different *segmentation lengths*, features and kernel functions. The TPR values are shown as circles and the FPR values are shown as crosses.

to the testing dataset, especially for the abnormal data. Secondly, our results indicate that the rareness of a syscall symbol across traces in the entire dataset (measured with *idf*) is not an effective indicator for real anomalies (intrusions).

ngram and *tf* produce nearly identical results along various *segmentation lengths* for a given kernel.

To determine the preferred feature vector we focused our attention to the execution for both of them. Figure 5.3d illustrates the execution time of training the model with *ngram* and *tf* features for all 3 applications with the ideal segmentation lengt of 30000. The execution time includes parsing the raw traces, extracting features, training and testing the model. The Image Classification with feature *tf* and linear kernel takes minimum time and we use it as the basic time unit T_0 ($T_0 = 8.2s$). All the execution times are represented as multiples of T_0. As shown in Figure 5.3d, *ngram* always takes a longer time compared to *tf* for each application and the time difference is positively related to the number of system call symbols (see Table 5.1 and Table 5.2).

Given our results, we can conclude that the feature *tf* is the best choice since it provides almost the best detection performance with a lower workload.

The optimal kernel mainly depends on the spatial distribution of the normal (application traces) and abnormal data points (attack traces) in the input space I. Linear kernel works better if the data points are essentially linearly separable and vice versa.

Comparison among applications and attacks

Table 5.3 summarizes the AUC, TPR and FPR values of the OC-SVM algorithm for all applications and attacks described in Section 5.4. The OC-SVM model is trained with *tf* feature extraction method and Gaussian kernel. The *segmentation length* is set to 30000. We chose these parameters because they are the optimal choices according to our discussion in Section 5.6 and Section 5.6.

The attacks *arbitrary code execution* and *brute force* performed on dynamic applications are easier to detect. The DE is able to detect 100% of attacks at a FPRof 6.7% for *arbitrary code execution* and 2% for *brute force*. The AUC values can reach as high as 0.995 and 0.959.

It is more difficult to detect adversarial machine learning attacks because the distinctions between normal and anomalous traces are weak. For the Image Classification application, the FPR is relatively high (12%). The TPR rate varies with different adversarial sample generation methods. For PGD, MIFGSM, PGDDLR and Square, 100% of the attacks can be detected. However, only 55% attacks of TPGD can be caught by the DE of the IDS.

Table 5.3: AUC, TPR, FPR values of different applications and attacks.

Application	Attack	AUC	TPR	FPR
CouchDB	Execute Arbitrary Code	0.995	1	0.067
Mongodb	Brute Force	0.959	1	0.020
Image Classification	PGD	0.917	1	0.12
	BIM	0.949	0.972	0.12
	CW	0.929	0.988	0.12
	FAB	0.951	0.961	0.12
	MIFGSM	0.851	1	0.12
	PGDDLR	0.857	1	0.12
	Square	0.858	1	0.12
	TPGD	0.799	0.55	0.12

5.7 Performance of anomaly analysis module

We evaluated the effectiveness of the *Anomaly Analysis* module discussed in Section 5.3.3. The *Anomaly Analysis* module recognizes the standalone anomalies and we measured the TPR and FPR values before and after filtering out those standalone anomalies. Figure 5.4 shows the metrics for different *segmentation lengths* for the MongoDB application with feature *tf* and linear kernel.

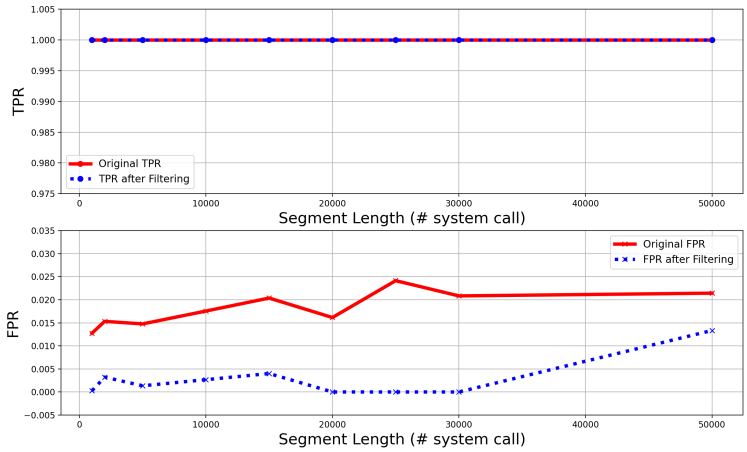


Figure 5.4: The TPR and FPR values as function of the *segmentation length* before and after filtering standalone anomalies in the case of the MongoDB application with *tf* feature and linear kernel

As shown in Figure 5.4, the TPR values are equal to 1 for all *segmentation*

lengths before and after filtering. This indicates there is no additional performance loss, in terms of TPR, after filtering.

The FPR values drop significantly. The original FPR ranges from 0.013 to 0.021 with various *segmentation lengths*. After filtering, the maximal FPR (with segmentation length equal to 50000) is only 0.014. The values can even reach 0 when optimal *segmentation length* is chosen.

5.8 Related work

There is a large amount of literatures using system calls to detect potential malicious behaviors.

The work in [56] was one of the first to use system call traces to characterize the behaviors of a running program. It builds a dataset of normal behaviors with a fixed length system call subsequences. After this, a test trace is identified as anomalous if the number of mismatch subsequences exceeds a user-defined threshold. The problem with this approach is that it lacks generalization and consumes huge storage capacity. The work in [72] uses system call to model the normal profiles as a Hidden Markov Chain (HMM), but tuning parameters of an HMM is extremely time consuming.

Recently, machine learning techniques have started to also be widely used for building anomaly-based IDS. The work in [73] proposes an IDS with K-nearest-neighbor (KNN) algorithm and evaluates the performance with the DARPA dataset. This approach does not require a separate profile for each program but the detection accuracy for novel attacks is only 75%. The work in [74] uses ngrams as feature vectors and compares multiple learning algorithms, namely, Support Vector Machine (SVM), Multilayer Perceptron (MLP) and Naive Bayes. They evaluate the performance with the ADFA-LD public dataset and observed that SVM outperforms the other two. There are also work that similarly to ours use OC-SVM as underlying technique. [63] evaluates the performance of the OC-SVM algorithm with different kernel functions also with the ADFA-LD public dataset. The work shows that OC-SVM can gain satisfactory performance with low computational cost. [75] proposes an OC-SVM based anomaly detection system using frequency distribution of various length n-grams as the feature vector. They also conclude that OC-SVM outperforms sequential anomaly detection models with the ADFA-LD dataset. However, both works fail to address the real time requirement of modern IDS systems, which we do cover in our work.

5.9 Conclusions and future works

This chapter extends the policy non-compliance detection component with an OC-SVM-based real time intrusion detection system that monitors and analyses

system calls. For each uniquely identifiable compute object, an IDS model is trained centrally in an authorised party and distributed to local nodes via a secure channel. An anomaly analysis model was implemented to reduce false alarms of the system. We demonstrated the detection capability with a customised attack dataset. The dataset contains modern attacks, such as adversarial machine learning attacks which modifies the data in the runtime and container escalation attack, giving the attacker potential privilege to copy out or modify data.

The experimental results showed that the OC-SVM algorithm could successfully detect modern attacks with satisfactory FPR, ranging from 0.02 for the brute force attack up to 0.12 for adversarial ML attacks. In addition, we observed that the system gains the optimal performance with feature extraction method *term-frequency* with TPR equal to one for large part of our attacks. Furthermore, the choice of *segmentation length* is crucial. The system performance degrades significantly if the *segmentation length* is too small. For the applications, we examined the optimal *segmentation length* is 30000 in terms of number of system call symbols. In addition, we observed that the optimal kernel functions are application dependent.

The proposed architecture can improve the policy enforcement capability of digital infrastructures by effectively detecting whether the algorithm behaviour complies to the policy or not. However, we noticed that the experimental results are under the assumption that the training data of the OC-SVM based IDS is completely clean. Collecting re-training data from end-point execution platforms to adapt the dynamic properties of an algorithm may give external attackers chances to inject malicious samples into the training dataset and degrade the IDS's performance. In Chapter 6, we further strengthen our architecture with a sanitization module to mitigate such malicious injections.

Chapter 6

Defending against poisoning attacks for IDS

Machine learning techniques are widely used to detect intrusions in the cyber security field. In Chapter 5, we demonstrated that our proposed architecture can effectively detect containerised attacks for policy-driven data exchange applications with an OC-SVM based IDS. However, most machine learning models are vulnerable to poisoning attacks, in which malicious samples are injected into the training dataset to manipulate the classifier's performance. In this chapter, we provide answers for our RQ4 by introducing a sanitization module in the architecture. We evaluate the accuracy degradation of OC-SVM classifiers with three different poisoning strategies with a public ADFA-LD dataset and a real world dataset that we developed in Chapter 5. We also propose a sanitization mechanism based on the DBSCAN clustering algorithm. The experimental results show that the poisoning attacks can degrade the performance of the OC-SVM classifier to a large degree and the proposed sanitization method can filter out poisoned samples effectively for both datasets.

This chapter is based on:

- **Lu Zhang**, Reginald Cushing, Paola Gross. "Defending OC-SVM based IDS from poisoning attacks." In 5th IEEE Conference on Dependable and Secure Computing (IEEE DSC 2022)/The 4th International Workshop on Secure Smart Societies in Next Generation Networks (SECSOC 2022).

6.1 Introduction

Machine learning (ML) techniques are increasingly being adopted in the security domain, for example, in an IDS [76]. However, in an adversarial setting, the adversary may inject specially crafted samples into the training data, which can make the decision boundary severely deviate and cause classification errors [77]. In the real world, initial training data is collected from open datasets, and periodic retraining is necessary. This may allow adversaries to carry out attacks by poisoning this public data.

In the last chapter, we introduced a real-time IDS system based on the OC-SVM algorithm by monitoring system calls to detect anomalous behaviours. Here we ask ourselves the question: *How is the OC-SVM based IDS resistant to the poisoning attacks and how can we defend against them?*

To answer this question, we first evaluate the performance degradation caused by 3 poisoning strategies with two syscall datasets: the public ADFA-LD dataset and a real-world dataset [78]. We construct tainted training datasets by injecting adversarial samples with different poisoning attack strategies. We measure the *accuracy* of the OC-SVM classifiers trained with the benign dataset and the contaminated dataset, respectively.

Next, we propose a sanitization process based on the DBSCAN clustering algorithm because it does not require any pre-knowledge of the normal data and can separate clusters of any shape [79]. We evaluate the effectiveness of the proposed methods and investigate the influences of different distance metrics and dimensionality reduction techniques. We demonstrate that the sanitization process achieves very good performance for all applications in both datasets.

The rest of the chapter is organised as follows. Section 6.2 introduces the basic concepts and categories of adversarial machine learning attacks. Section 6.3 describes the 3 label flipping strategies we adopt to evaluate the performance degradation. In Section 6.4, we present our proposed sanitization mechanism based on the DBSCAN clustering algorithm. We further describe the experimental design to evaluate the performance degradation caused by various poisoning strategies and corresponding improvement after applying proposed sanitization mechanisms in Section 6.5. Section 6.6 analyses the experimental results. Section 6.7 discusses the influences of different distance metrics. Section 6.8 presents the related work and Section 6.9 concludes the chapter.

6.2 Background

In poisoning attacks, the attacker adds adversarial samples to the training data so that the ML model's decision boundary can be manipulated. The adversarial samples can be crafted either by flipping the labels, e.g. inject malicious samples in the normal training set, or by distorting the training samples, e.g. adding de-

liberately calculated noise to the feature vectors. The latter is commonly applied and is more effective to ML models that deal with image data. In our work, where we deal with time series data, we mainly focus on the label flipping attacks.

In our work, we use OC-SVM, an unsupervised learning algorithm where only normal data is required for model training. The label flipping poisoning attacks for unsupervised learning algorithms such as OC-SVM can be understood as the strategies needed to select the malicious samples to inject under a predefined cost. Nearest first label flipping, furthest first label flipping and optimization label flipping are three commonly used poisoning attacks. We must note that the last one, the optimization label flipping method, tries to find a combination of malicious samples by solving an optimization problem. This optimization is required because performing an exhaustive search for all possible combinations is extremely computationally intensive and normally not feasible. In the next section, we will describe only one specific type of optimization attack, ALFA, because of its popularity.

6.3 Poisoning strategies

To evaluate the performance degradation, we use 3 poisoning strategies, nearest first attack, furthest first attack and adversarial label flip attack (ALFA). All the 3 attacks are white-box attacks, which means the adversary needs to know the model parameters and feature extraction methods a-priori. In practice, it is expected that the adversary's capability is bounded, and we assume that the attacker can only inject a limited number of malicious samples.

6.3.1 Nearest first attack

For the nearest first attack, the adversary first injects malicious samples which have the smallest distances to the decision hyperplane of the OC-SVM classifier in the feature space. These samples are normally hard to distinguish as they can occur also in absence of an attack: either they are caused by incorrect labeling or by the intrinsic classification error rate present in a ML-based IDS in the scenario of model-retraining.

6.3.2 Furthest first attack

For the furthest first attack, the adversary first inserts malicious samples which have the largest distances to the decision hyperplane in the feature space. This strategy is intuitively most effective since it aims to shift the decision boundary of an OC-SVM classifier to a big degree and is also computationally efficient.

6.3.3 Adversarial label flips attack (ALFA)

The adversarial label flips attack (ALFA) poison strategy is an optimization label flipping attack [80]. It aims to find adversarial samples that jointly deteriorate the accuracy of a classifier to a maximal degree under a given cost. It adopts a relaxed optimization framework that can achieve near-optimal results with less computational effort. In ALFA, the optimization problem is decomposed into two sub problems: a quadratic programming (QP) program and a linear programming (LP) program. The QP program is used to compute a decision boundary of the classier with the latest updated tainted training dataset. The LP program is used to update the training dataset with maximal hinge error with respect to the latest decision boundary. The ALFA algorithm devises an iterative approach to minimize QP and LP alternatively. The process is repeated until convergence. However, the proponents of this attack formulated an optimization framework in the supervised setting. We adapt the framework to unsupervised learning algorithms by changing the objective functions and the constraints in the LP program.

6.4 The data sanitization with DBSCAN

6.4.1 The DBSCAN clustering algorithm

DBSCAN is a density-based clustering algorithm. It separates the data points in the feature space into clusters and assigns to each point a label. The general idea is to find areas that reach the minimum density level and are separated by lower-density areas. [79]

The DBSCAN cluster algorithm is illustrated in Figure 6.1. Any data point is assigned one of the 3 labels, which are *core point* in red, *border point* in blue and *outlier* in green. In a DBSCAN cluster model there are 2 predefined parameters, ϵ and $minPts$. ϵ defines the maximal distance between two points that can be considered as neighbors. It is the radius of the circles in the Figure 6.1. The distance measure can be arbitrary. $minPts$ defines the threshold of neighbor numbers that reaches a minimum density level. A point is labeled as a *core point* if it has at least $minPts$ neighboring points within the radius ϵ. A point is labeled as a *border point* if it has less than $minPts$ neighboring points within the radius ϵ, but is the neighbor of any *core point*. Other points are labeled as *outliers*.

The choice of parameters ϵ and $minPts$ has a direct influence on the DBSCAN clustering algorithm's performance, especially in high-dimensional data space [81]. We will describe how we chose parameters in our sanitization process with details in section 6.4.2.

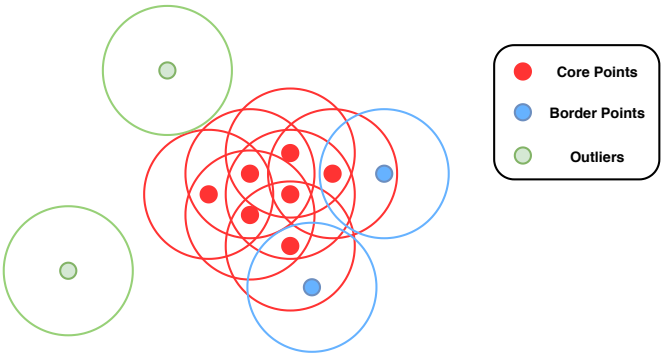


Figure 6.1: The DBSCAN clustering algorithm.

6.4.2 Sanitization flowchart

There are multiple reasons for the adoption of the DBSCAN clustering algorithm for sanitizing the training dataset against poisoning attacks. Firstly, it deals well with nonlinear data. It can find non-linearly separable clusters of any shape, while KNN and k-means cannot do this well. Secondly, the DBSCAN algorithm does not require a priori knowledge of the normal data. On the one hand, we do not need to specify the number of clusters as needed in other clustering mechanisms. On the other hand, we do not require normal data as for other outlier detection mechanisms. It is not a trivial task to get quality-guaranteed normal data, especially for the initial IDS training.

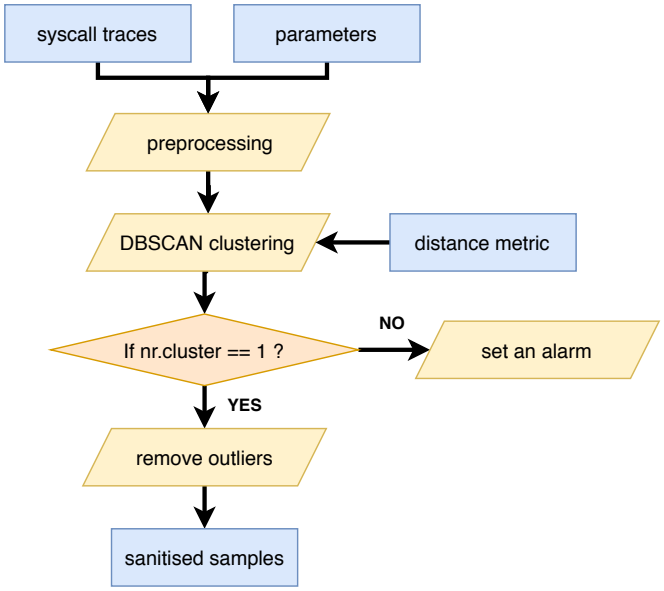


Figure 6.2: The flowchart of the sanitization process.

Figure 6.2 shows the flowchart of the sanitization process. The *syscall traces*

and *parameters* are inputs of the sanitization process. To select proper parameters ϵ and $minPts$, we conduct a grid search if any normal data is available. As the feature vectors are normalized frequency distributions, the pairwise distances normally lie in the interval between 0 and 1 for most popular distance metrics, such as cross entropy and euclidean. The parameter $minPts$ also reaches an upper limit of the total number of available normal data points. We iterate combinations of ϵ and $minPts$. Among all the combinations where the input normal points end with a single cluster, we select the one with a relatively small ϵ value and a moderate $minPts$ value. We also investigate the parameter sensitivity over different applications and distance metrics in section **??**.

In the *preprocessing* module, the *syscall traces* are mapped to data points in the feature space. Each trace is parsed and vectorized as the frequency distribution of the system call symbols. The algorithm performs dimensionality reduction, if necessary, and computes pair-wise distances with the given distance metric. In the *DBSCAN clustering* module, the algorithm separates the clusters and labels each data point, either in the original feature space or in the space of lower dimension. The sanitization process first checks the number of clusters. If there is more than one cluster, the process sets an alarm and expert effort would be required. Otherwise, the process removes all the outliers and generates the *sanitized samples* in the original feature space.

6.5 Experiments and dataset

We design experiments to investigate the performance degradation of the OC-SVM classifier due to poisoned training data and the effectiveness of the proposed DBSCAN-based sanitization algorithm.

6.5.1 Dataset

Table 6.1: Applications and attacks of the public dataset and the real world dataset.

Dataset Name	Application	Attacks	Number of traces	
			Normal	Attack
ADFA-LD public dataset	Linux web server	Add user	833	91
		Java meterpreter		125
		Web shell		118
The real world dataset	CouchDB	Container escalation	248	173
	MongoDB	Brute force	1348	148

To perform the experiments we adopt two datasets of system call traces: a

public dataset and a real world dataset. The ADFA-LD dataset is a benchmark dataset for evaluating anomaly detection systems of system call traces. It was released recently and it incorporates the characteristics of modern attacks [78]. In the ADFA-LD dataset, the normal system call traces are collected from a contemporary Linux server and abnormal traces are generated by 6 types of modern attacks. In our experimental evaluation, we choose 3 attacks, *web shell*, *java interpreter* and *add user*. We also run the experiments with a real world dataset, the DL4LD use case. We run two dynamic applications with Docker containers: CouchDB and MongoDB. To emulate the dynamic behaviors of real-world database users, we send requests to the container with Apache JMeter, an open-source workload generation tool. We craft attacks with exploitation tools, e.g Nmap and Metasploit, to generate abnormal traces. Table 6.1 describes the detailed information of our two datasets: the corresponding applications, the crafted attacks and the associated number of traces. In the real world dataset, the monitored streaming traces are segmented with a fixed window size of 30000 syscall symbols, which was determined to be the optimal choice in our previous work [82].

6.5.2 Experimental design

There are two goals in our experiments. On the one hand, we aim to investigate the performance degradation of the OC-SVM based IDS with the poisoned training dataset. We also compare different poisoning strategies described in section 6.3. On the other hand, we try to explore the effectiveness of our proposed defense mechanisms with various distance metrics and dimensionality reduction methodologies.

Dataset split

We use the metric *accuracy* to evaluate the performance of a classifier. *Accuracy* describes the proportion of correct predictions. It is computed as:

$$accuracy = \frac{Nr.\,correct\,predictions}{Nr.\,total\,predictions} \tag{6.1}$$

Figure 6.3 illustrates how we split the dataset to perform our evaluation. First of all, each system call trace is vectorized to a fixed length vector with the frequency distribution of all system call symbols. We call those feature vectors samples. As shown in Table 6.1, a dataset includes normal and attack traces for each application. Secondly, we split the *normal samples* into a *train normal dataset* and a *test normal dataset* with a given ratio. In our experiment we set this to 0.9. Thirdly, we randomly select attack samples to construct the *test attack dataset*. To construct a balanced *untainted test dataset*, the number of samples in the *test attack dataset* is equal to that of the *test normal dataset*. This means

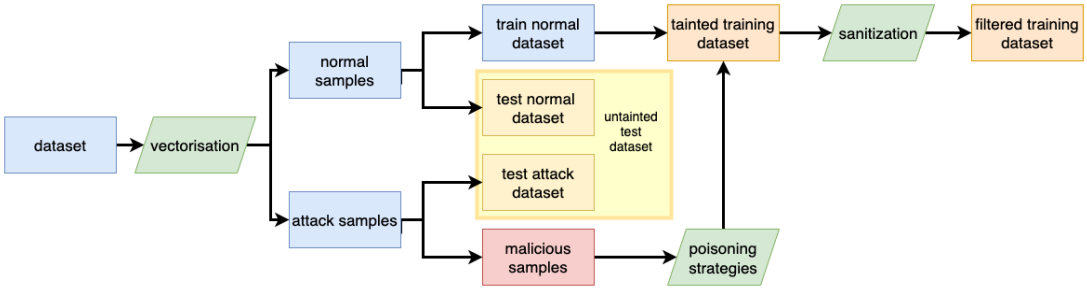


Figure 6.3: The experiment procedure to obtain the tainted and the filtered training dataset. The procedure consists of three operations (in green): vectorisation, poisoning and sanitization which in terms generate the corresponding datasets.

the worst performance of the classifier is 50% consistent with a random guess. With different *poisoning strategies*, the *training normal dataset* is tainted with a number of deliberately selected malicious samples. We call the poisoned dataset the *tainted training dataset*.

We limit the attacker's capability so that they can only inject a specific portion of malicious samples. Here we define the metric *poison portion* as the fraction of the number of the malicious samples over the number of the benign samples:

$$poison\,portion = \frac{Nr.\,malicious\,training\,samples}{Nr.\,benign\,training\,samples} \tag{6.2}$$

In our experiment, we choose the *poison portion* in the range 0.05 - 0.2.

Performance degradation and improvement

To evaluate the performance degradation, we first train the OC-SVM classifier with the *train normal dataset* and compute the *accuracy* with the *untainted test dataset*. This indicates the original performance of the OC-SVM based syscall IDS. Secondly, we train the OC-SVM classifier with the *tainted training dataset* and test its performance also with the *untainted test dataset*. By comparing the *accuracy* values achieved by the above two settings, we can numerically determine the performance degradation.

To demonstrate the effectiveness of our proposed sanitization process, we train the OC-SVM model with the *filtered training dataset* and test the classifier again with the *untainted test dataset*.

Distance metrics

We also investigate the influence of different distance metrics on the performance of the sanitization process. We adopt various distance metrics in the *DBSCAN clustering* module in Figure 6.2 and compare the performance improvement.

Dimensionality reduction

Dimensionality reduction is conducted in the *preprocessing* module in Figure 6.2. The technique transforms data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data. We adopt two dimensionality reduction techniques in our experiment: principal component analysis (PCA) and truncated singular value decomposition (SVD). Then we compute the *accuracy*.

6.6 Results analysis of performance degradation and improvement after sanitization

We investigate the performance degradation of the classifier with different poisoning strategies and the effectiveness of the sanitization process for both datasets. We conduct experiments with the procedures explained in the section 6.5.2. We use the Gaussian kernel to train the OC-SVM classifiers with two different datasets [83]. In the DBSCAN clustering algorithm, we compute pairwise distances between data points with Euclidean distance metric in the original feature space without any dimensionality reduction techniques. With the parameter calculation method described in Section 6.4.2, we chose ϵ as 0.3 and $minPts$ as 100 for both datasets.

Figure 6.4 and Figure 6.5 show the performances of the OC-SVM classifier trained with the original dataset (in blue), the *tainted training dataset* (in orange) and the *filtered training dataset* (in green) with different *poison portion* for the public ADFA-LD dataset and the real world dataset respectively.

6.6.1 Performance degradation

We first focus on the blue and orange lines in Figure 6.4 and Figure 6.5 to investigate the performance degradation. It is not surprising to notice that the *accuracy* value decreases as the *poison portion* increases.

For both datasets, the furthest first attack always degrades the classifier performance to the largest degree. It can deteriorate the *accuracy* to a value as low as 0.5 already from a *poison portion* percentage of less than 0.1. The nearest first attack shows performance degradation when the *poison portion* exceeds a threshold, which depends on concrete applications and attacks. The *accuracy* for the ALFA attack shows a decreasing trend but fluctuates. This behaviour is because the algorithm computes a combination of adversarial samples with sub-optimal choices. In fact, adversarial samples that were selected with a smaller *poison portion* and that contributed to a big decision boundary shift may actually not be selected again with a larger *poison portion*. According to our experimental results, the ALFA attack strategy is not so good as expected when it is used in

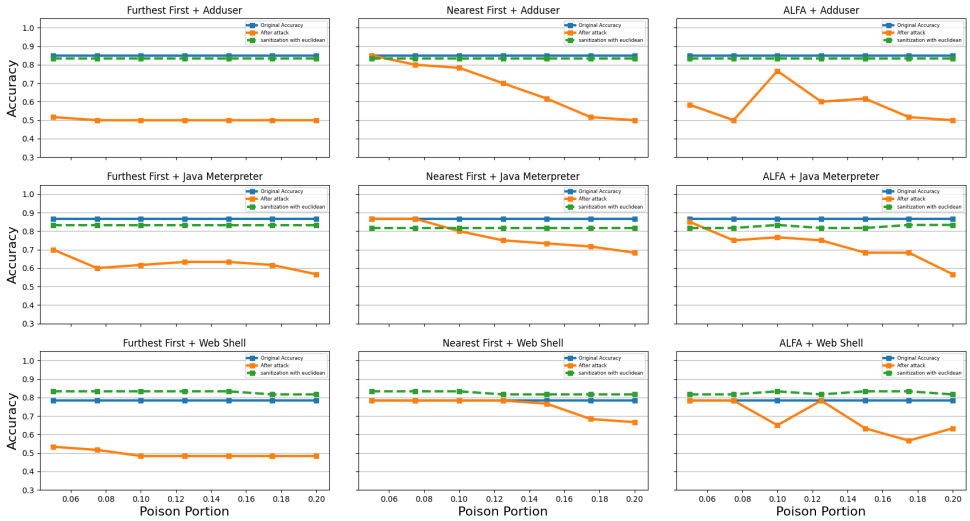


Figure 6.4: The plot of *accuracy* vs *poison portion* for the public ADFA-LD syscall dataset of the OC-SVM based IDS. Each column corresponds to a specific poisoning attack strategy: furthest first, nearest first and ALFA. Each row corresponds to a specific attack in the dataset: Add user, Java meterpreter and Web shell. and with sanitation process for the ADFA LD dataset. In each subplot, the blue line represents the original performance of the classifier (without pure clean training data), the green line represents the performance with tainted training dataset of various *poison portion* and the red line represents the performance of the classifier with proposed sanitization process.

the unsupervised learning scenario. A second reason is that this method is not well suited for the data distribution of system call traces.

In short, the *performance* of an OC-SVM classifier can be significantly affected by deliberately crafted malicious samples even when the *poison portion* is very small. This can be considered as a serious vulnerability when applying the OC-SVM based IDS in real world settings. From the attacker's perspective, the furthest first attack seems to be the optimal choice because it leads to a larger *accuracy* degradation level with a lower computational effort.

6.6.2 The effectiveness of the sanitization process

Observing Figure 6.4 and Figure 6.5, we can conclude that the *accuracy* after the sanitization process (green line) is pretty close the original *accuracy* of the OC-SVM classifier (blue line) for all applications in both datasets. This indicates that the tainted samples can be easily filtered out by the DBSCAN clustering algorithm and that our method is effective regardless of the underlying data

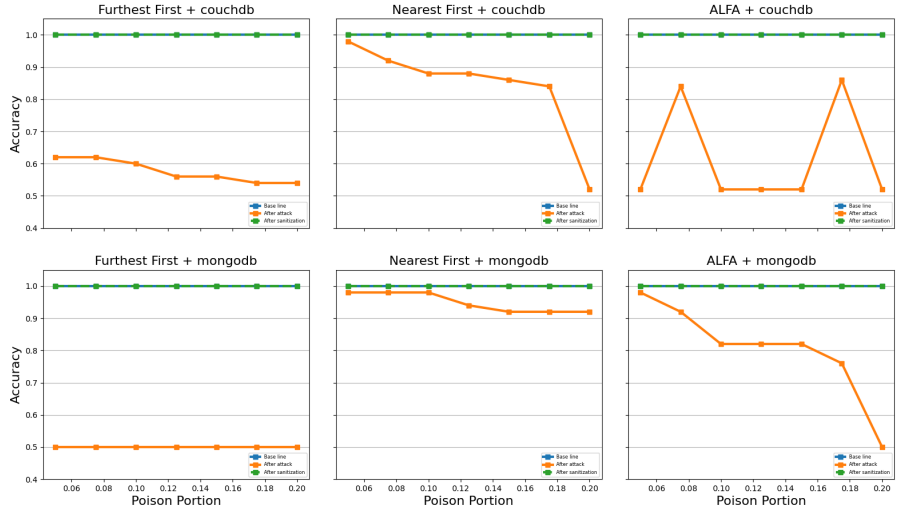


Figure 6.5: The plot of *accuracy* vs *poison portion* for the real time syscall dataset of the OC-SVM based IDS. Each column corresponds to a specific poisoning attack strategy: furthest first, nearest first and ALFA. Each row corresponds to a specific application in the dataset: CouchDB and MongoDB. In each subplot, the blue line represents the original performance of the classifier (without pure clean training data), the green line represents the performance with tainted training dataset of various *poison portion* and the red line represents the performance of the classifier with proposed sanitization process.

distribution patterns of applications and attacks.

We can also make a number of interesting and more specific observations. First when there is a good separation between normal and abnormal data points in the feature space of the syscall traces the original *accuracy* and the one after sanitization are essentially the same, as seen in Figure 6.5 where the blue and green line overlap. Second, it can occur that the *accuracy* after sanitization is even slightly better than the original one. We see this for the *Web shell* attack in the public ADFA-LD dataset, in the 3rd row in Figure 6.4. One possible explanation is that the sanitation process can also filter out noisy points that do not follow the distribution pattern of the general dataset. Third, we observe that different data distribution patterns have a larger impact on the performance of the sanitization process than the poisoning strategies themselves. This is visible in Figure 6.4 where the accuracy after sanitization is very similar on the same row, ie the same application, more than in the same column, ie the same method. Finally, we can observe that the *accuracy* of the OC-SVM classifier remains almost constant when the *poison portion* increases for all settings in both datasets.

Table 6.2: The chosen parameters (ϵ and $minPts$) of the sanitization process with different distance metrics and applications.

	Euclidean	Cross Entropy	Square	Cosine	Manhanttan	Minkowski (order =3)
ADFA-LD dataset	(0.3, 100)	(0.81, 60)	(0.21, 100)	(0.41, 100)	(0.71, 60)	(0.31, 100)
The real world dataset	(0.3, 100)	(0.81, 60)	(0.21, 100)	(0.41, 100)	(0.71, 60)	(0.31, 100)

6.7 Influences of distance metrics and dimensionality reduction techniques

We evaluate the influence of different distance metrics and dimensionality reduction techniques in the sanitization process and try to find the optimal choice for the underlying data patterns for the system call traces.

With the strategies described before, the parameters of different distance metrics are shown in Table 6.2. We observe that the optimal parameters ϵ and $minPts$ vary over different distance metrics along the row but are the same among different applications and attacks. This indicates that we can adopt the historical parameters of the same distance metric and expect good performance even if there is not any normal training data available for a new specific application.

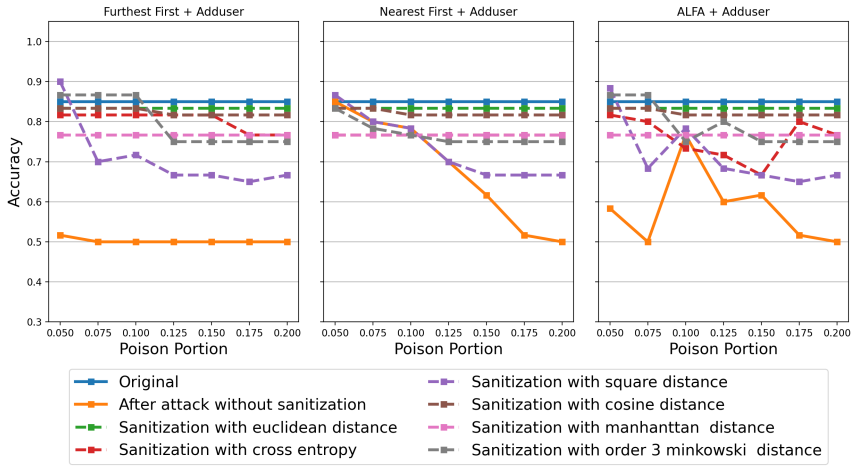


Figure 6.6: Performance improvement with different distance metrics for the public dataset with Add user attack.

In Figure 6.6, we show the comparison results with 6 distance metrics for the *Add user* attack in the ADFA-LD dataset.

Among all the distance metrics, Euclidean distance (blue) has the best performance for all 3 label flipping strategies. The *accuracy* has a constant value of 0.84 as the *poison portion* increases, which is very close to the original value of 0.85. The cosine distance (brown) and Manhattan distance (pink) have similar

performance but contribute to lower *accuracy* values, which are about 0.84 and 0.76 respectively. This means those 3 distance metrics can represent the similarities of data points, mapped by frequency distributions of syscall symbol, in the feature space well. In addition they provide good separation between normal and abnormal traces, with the Euclidean distance having the best capability to distinguish between similar ones. The 3rd order Minkowski distance (grey) can filter out samples that are far from the decision boundary. It leads to an *accuracy* even higher than the original value when the *poison portion* is smaller than 0.1 for the furthest first attack and 0.075 for the ALFA attack. This is because the 3rd order polynomial over-stretches the larger values and over-shrinks the lower values. The square distance (purple) and cross entropy(red) do not work well and fail to extract the similarity information from the feature vector.

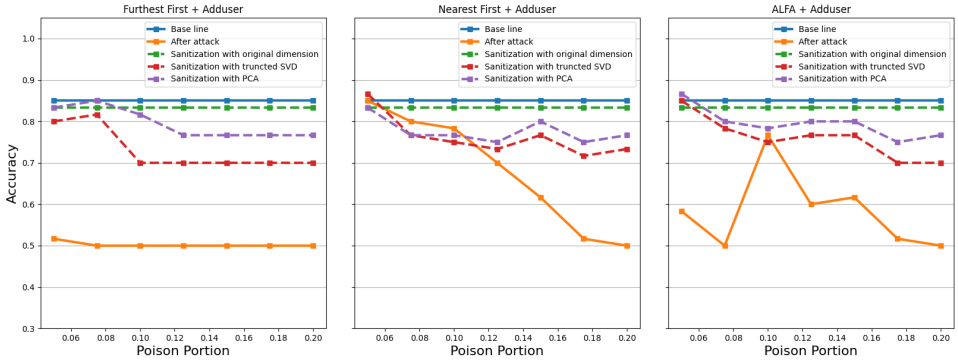


Figure 6.7: Performance improvement with different dimensionality reduction techniques for the public dataset with Add user attack.

Figure 6.7 shows the *accuracy* with 2 dimensionality reduction techniques, PCA and trucated SVD, applied in the sanitization process for the *Add user* attack in the ADFA-LD dataset. We adopt the Euclidean distance metric as it is demonstrated as optimal and the reduced dimension is set to 10 with grid search and correlation analysis. Same as our previous observation, the *accuracy* is constant at 0.84 over different *poison portion* values. The *accuracy* values fluctuate slightly with dimensionality techniques, ranging from 0.8 to 0.86 for truncatedSVD (in red) and from 0.8 to 0.89 for PCA (in purple). We observe that performing dimensionality reduction does not increase the effectiveness of the sanitization process obviously.

6.8 Related work

There are plenty of recent works investigating the performance degradation of ML-based IDS caused by poisoning attacks. [84] investigated the influences of the poisoning attacks for PDF malware detectors using deep learning models.

But the PDF files, same as image data, require the adversarial samples to have the same functionality as the normal data. The work in [85], [86] and [87] studied the impact of poisoning attacks on the network IDS with public dataset of network traffics, which are also time-series. However, all those IDS adopt deep learning models, which have different label flipping strategies from classic machine learning models. To the best of our knowledge, we are the first to evaluate the sensitivity of host-based IDS with the monitoring metric of syscall traces. In the research area of defending adversarial machine learning attacks, some works aim to make the training algorithms more resistant. [88] used nonlinear data projections and game theory to improve the resilience of SVMs against adversarial samples. The work in [89] proposed an adversarial SVM model, which can gain higher robustness with a modified loss function. However, these approaches sacrifice the performance of the classifiers and are not general. [90] proposed to reduce the influence of the adversarial samples by measuring the impact of each sample, but the method is very computationally expensive. [91] used KNN to filter out poisoned samples but it has limited effect for high dimensional data. [92] proposed to use weight initialization to remove the adversarial samples in the training set, but it assumes a small untainted dataset is available.

6.9 Conclusions and future works

We evaluated and compared the performance degradation of OC-SVM caused by 3 different poisoning attacks: furthest first attack, nearest first attack and ALFA. From the attacker's perspective, the furthest first attack is optimal with a higher degradation level and lower computational effort. From the defender's perspective, we noticed that even a small portion of injected malicious samples can deteriorate the classifier's performance dramatically.

We proposed a sanitization process to filter out malicious samples before training. DBSCAN is used to separate clusters and label outliers according to the density. The sanitization process automatically removes all outliers if the output cluster number is 1. Our experimental results demonstrated the sanitization process is effective for all applications and poisoning strategies we considered. We compared the performance of 6 distance metrics and observed that the Euclidean distance is optimal since it best fits to the data patterns of the system call traces. Dimensionality reduction techniques do not contribute to an obvious performance improvement even with properly selected dimension numbers. DBSCAN parameters are sensitive to distance metrics but not the type of application, hence parameter reuse is therefore feasible.

Together with the functionalities described in Chapter 4 and Chapter 5, our proposed architecture is easily incorporated into the DDM digital infrastructures as a policy enforcement component.

Chapter 7

Conclusions and future works

Data sharing and federation can significantly increase efficiency and lower the cost of digital collaborations. More useful information can be found if more diverse data is available. Motivating data owners to share their data and engage into such collaborations is critical. Therefore, it is important to convince them that their outsourced data will be used in a secure and controlled manner. Constructing a policy governing concrete data usage rules among all parties is an essential step. It is even more important to build digital infrastructures to enforce the policy effectively. In this thesis, we presented essential functional components for secure data sharing in the context of a DDM prototype. In Chapter 2 and 3, we introduced concrete methodologies to link high-level applications and underlying infrastructures. In Chapter 4, 5 and 6, we described a distributed architecture to detect suspicious behaviors that break the policy during the execution stage. We evaluated the performance of the proposed methodologies with the data logistic use case. By examining the experimental results, we gained insights into how the concept of DDM and proposed functional components work in practice. The proposed methodology can also be integrated into other data exchange infrastructures.

The main contributions we achieved from this work are:

- an approach to model and measure mutual similarities of multi-lateral collaboration relationships,
- a framework to quantitatively assess and compare risk exposure of data exchange infrastructures,
- a hybrid intrusion detection system,
- a methodology to profile and discriminate running behaviors of a containerized algorithm,
- a defense mechanism against poisoning attacks targeted to machine learning based IDS.

7.1 Answers to research questions

With the above-mentioned outcomes and the analysis of evaluation results, we answer our main research question defined in Chapter 1:

How to select optimal application-tailored infrastructures and enhance policy compliance capabilities?

The research work was conducted in 4 phases: modeling, selecting, monitoring and defending. Figure 7.1 illustrates the concrete procedures.

We started from the high level framework of a DDM data sharing infrastructure, a concept proposed by the research project DL4LD to facilitate data sharing among Dutch logistic parties. In this thesis, our main focus is building functional mechanisms along the arrow chain in the center, from policy to data exchange infrastructures, via infrastructure patterns. The DDM customers come with a concrete application with pre-agreed policy rules, for instance, the airline use case. We first introduced a component linking the policy to the infrastructure patterns in Chapter 2. The mechanism selects digital infrastructure patterns that satisfy the collaboration request to a maximal degree by modeling and closeness identification. Secondly, we presented a threat-analysis driven risk assessment framework in Chapter 3. After delegating an application to a potential secure infrastructure, it can quantitatively assess the remaining risk. The risk assessment results allow DDM customers to rank and select the most secure digital infrastructures. Also, the quantitative assess results increase the transparency and boost the confidence of data owners to share their data. The optimal digital infrastructure for a specific data federation application is the one which can support the requested collaboration model and provides the best security guarantee. Then we presented a distributed architecture that detects policy compliance when an algorithm executes on the data. There are 3 main components, which are profile generation and validation module introduced in Chapter 4, a hybrid IDS module introduced in Chapter 5 and a sanitization module introduced in Chapter 6. A profile and a pre-trained IDS model are built for every uniquely identifiable compute object. They are initially trained in a secure environment and then distributed to endpoint execution platforms via a secure channel. The system call traces are monitored, analysed and verified in endpoint points platforms. The IDS model is retrained periodically to increase generalization with the empirical data. A sanitization algorithm is implemented to filter out malicious samples to further defend the architecture against adversarial machine learning attacks. This architecture can be integrated into a DDM infrastructure or other data sharing platforms to help enforce the policy during execution.

To be more specific, we answer the sub research questions with more details.

- RQ1: *How to map an application request to a best-fit digital infrastructure based on collaboration models?*

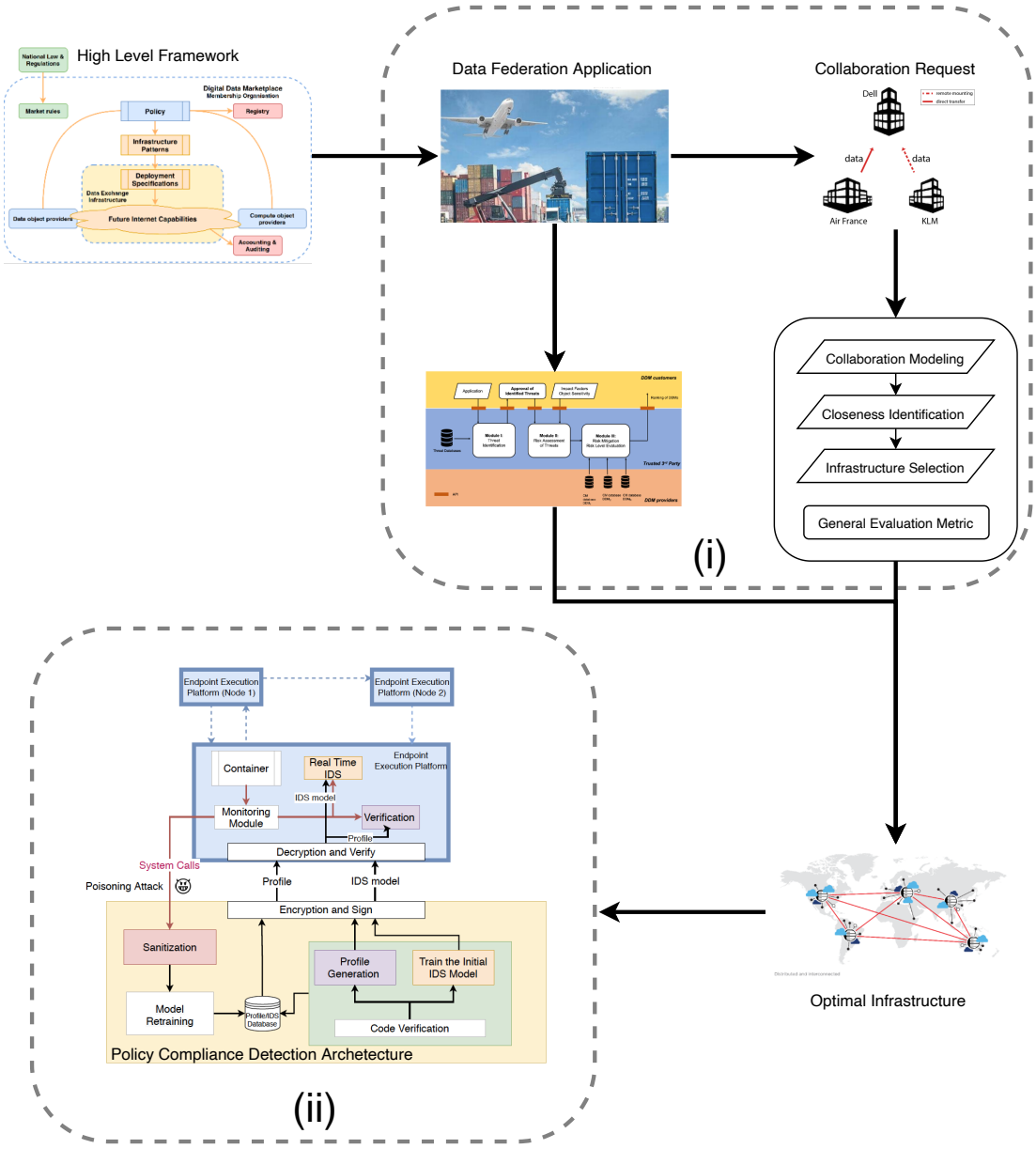


Figure 7.1: Functional components of a DDM prototype: (i)components to select optimal digital infrastructures given a data federation application considering both compatibility and security. (ii) components to improve the policy enforcement capability of a DDM by effectively detecting rule compliance during the execution stage.

We map an application request to an optimal infrastructure with the procedures described in Chapter 2: 1) represent both the application request and the provided architectural patterns with 3-D matrices according to the modelling methodology 2) define similarities measures with the mathematical representations to identify the closeness between any two collaboration models 3) select an archetype that fulfils the hard request and is most similar to the soft request of the application. By answering this question, we addressed the influence of underlying multi-lateral trust relationships among the involved parties. With the proposed matching mechanisms, the DDM customers can select a digital infrastructure which supports archetypal patterns for their data federation applications.

- RQ2: *How to select an optimal digital infrastructure with minimum risk?*

We answer this question by introducing a risk assessment framework in Chapter 3. The framework can quantitatively estimate the remaining risk of a data federation application operated on specific digital infrastructure. In-depth threat analysis was conducted. The data federation application is represented as a list of transactions and the threats are identified in a semi-automatic manner. The proposed framework estimates the relative severity of each threat with a modified version of Microsoft DREAD model. New risk attributes were defined to address the impacts of important aspects of a data sharing platform, such as application-dependent security requests, the essential role of monitoring and so on. Since most risk assessment systems are criticized to be over-subject, we addressed this issue from two perspectives. 1) We defined more fine-grained guidance for risk attribute level assignment. 2) We evaluated the robustness of the DREAD model caused by the subjective choices of parameters. We showed with our experimental results that DREAD model can gain good stability as we adopted it to compute the risk ratios rather than the absolute values. We also proved that our method can offer good precision to distinguish between individual threats.

After answering the previous two sub research questions, potential customers are able to select a best-fit digital infrastructure when they come with a data sharing application request. We further investigated how to enhance the policy enforcement capability in a data exchange infrastructure, especially when the data and algorithm meet. Detection plays an important role. Here comes our third sub research question:

- RQ3: *How to develop policy compliance detection components during execution?*

The proposed architecture detects whether an algorithm conforms to a policy by monitoring and analyzing Linux system call traces generated by the running container. Firstly, we introduced a methodology to profile and verify the running behaviors of an algorithm in Chapter 4. This allows us to distinguish whether an algorithm running inside a container is the one claimed to be. For instance,

this method can effectively detect the compliance of policy rules such as *the data object can only be accessed by compute object A, but not the others.* The profile is built with the frequency distribution of n-grams of system call symbols and the verification is conducted by calculating dissimilarity with Laplace smoothing and cross-entropy. We showed that the proposed methodology can provide high accuracy and support profile reuse over different platforms and input data. Secondly, we introduced a hybrid real-time IDS in Chapter 5. It uses OC-SVM as the anomaly detection algorithm and applies a signature-based methodology to reduce false alarms. The streaming systems calls are segmented with a constant time window. We demonstrate that the proposed IDS can achieve outstanding performance with high TPR and relatively low FPR values for detecting modern containerized attacks.

- RQ4: *How to defend against adversarial machine learning attacks for the monitoring components?*

To answer this sub research question, we first conducted experiments to evaluate OC-SVM based IDS sensitivity for poisoning attacks. By examining the experimental results, we proved the necessity of implementing a defending mechanism due to the observations that the classifier's performance may degrade dramatically due to a very small portion of malicious samples. Then, we introduced a sanitization mechanism based on the DBSCAN clustering algorithm. The advantages mainly lie in two aspects. i) It does not require any pre-knowledge of the normal data or the machine learning models. ii) It can separate the data of any shape which conforms to the statistical property of system calls. Finally, we demonstrated that the proposed mechanism effectively filters out malicious samples and achieves almost equal accuracy with an untainted training dataset.

7.2 Future works

Our work can be extended in the following two directions in the future.

7.2.1 Improve the anomaly-based IDS

Since we aim to detect potential intrusions for a distributed application, we will extend the IDS to detect anomalies by monitoring multi-dimensional metrics. The metrics comprise two main categories: the interactions between distributed execution platforms and the execution information of a single execution platform. The former includes information such as traffic volumes and traffic patterns. The latter includes information such as the CPU or GPU usage, log information and memory access. The detection engine can decide by analyzing all these monitored metrics jointly. This will allow us to achieve richer information and detect host-based malicious behaviors more accurately. In addition, this enables the IDS to

detect intrusions during the data transmission stage. After extending the IDS to analyze multi-variant data, it is also interesting to combine different machine learning models and detect the anomalies in parallel. It is important to investigate the performance of other anomaly detection models, such as auto-encoder, generative adversarial networks, isolation forest, with different monitoring metrics. Therefore we can design the distributed IDS that achieve an optimal trade-off between detection performance and cost.

It is also interesting to explore and expand the confidence area of a distributed IDS. In this work, we train an individual IDS model for each application. We will investigate the possibility that a group of similar applications can share one pre-trained IDS model with sufficiently good detection performance. Hence, we can have richer training data and increase the overall efficiency. The main challenges lie in clustering the different applications and defining the confidence area of a trained IDS model.

7.2.2 Improve the sanitization mechanism

We have demonstrated that our DBSCAN based IDS has gained good performance for public and customized datasets. We did explain why the DBSCAN is expected to gain better performance. In the future, we will validate this hypothesis by experimentally comparing its performance with other outlier detection algorithms. Furthermore, it is also interesting to investigate the generalization of our proposed approach. To achieve this goal, we will apply the sanitization mechanism to other anomaly-based machine learning models and study the corresponding performance improvements in classification accuracy.

Bibliography

[1] EarthWeb. (2022) How much data is created every day in 2022. [Online]. Available: https://earthweb.com/how-much-data-is-created-every-day/

[2] J. A. Kassem, C. De Laat, A. Taal, and P. Grosso, "The epi framework: A dynamic data sharing framework for healthcare use cases," *IEEE Access*, vol. 8, pp. 179 909–179 920, 2020.

[3] H. Ma, R. Zhang, G. Yang, Z. Song, K. He, and Y. Xiao, "Efficient fine-grained data sharing mechanism for electronic medical record systems with mobile devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 5, pp. 1026–1038, 2018.

[4] X. Lu and X. Cheng, "A secure and lightweight data sharing scheme for internet of medical things," *IEEE Access*, vol. 8, pp. 5022–5030, 2019.

[5] H. Lei, Y. Yan, Z. Bao, Q. Wang, Y. Zhang, and W. Shi, "Sdsbt: A secure multi-party data sharing platform based on blockchain and tee," in *International Symposium on Cyberspace Safety and Security*. Springer, 2020, pp. 184–196.

[6] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 57–64.

[7] R. N. Zaeem and K. S. Barber, "The effect of the gdpr on privacy policies: Recent progress and future promise," *ACM Transactions on Management Information Systems (TMIS)*, vol. 12, no. 1, pp. 1–20, 2020.

[8] S. Prabhakaran, S. Raman, J. E. Vogt, and V. Roth, "Automatic Model Selection in Archetype Analysis." Springer, Berlin, Heidelberg, 2012, pp. 458–467. [Online]. Available: http://link.springer.com/10.1007/978-3-642-32717-9{_}46

[9] A. Jøsang, E. Gray, and M. Kinateder, "Simplification and analysis of transitive trust networks," *Web Intelligence and Agent Systems: An International Journal*, vol. 4, no. 2, pp. 139–161, 2006.

[10] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, "Hamming distance metric learning," in *Advances in neural information processing systems*, 2012, pp. 1061–1069.

[11] S. Cisneros-Cabrera, A. Ramzan, P. Sampaio, and N. Mehandjiev, "Digital marketplaces for industry 4.0: a survey and gap analysis," in *Working Conference on Virtual Enterprises*. Springer, 2017, pp. 18–27.

[12] A. Fradkin, "Search, matching, and the role of digital marketplace design in enabling trade: Evidence from airbnb," 2017.

[13] D. Zahay, D. Schultz, and A. Kumar, "Reimagining branding for the new b2b digital marketplace," *Journal of Brand Strategy*, vol. 3, no. 4, pp. 357–372, 2015.

[14] A. Ordanini and A. Pol, "Infomediation and competitive advantage in b2b digital marketplaces," *European Management Journal*, vol. 19, no. 3, pp. 276–285, 2001.

[15] M. Schoop and T. List, "To monitor or not to monitor-the role of trusted third parties in electronic marketplaces," in *Information Age Economy*. Springer, 2001, pp. 605–618.

[16] M. Schmees, "Distributed digital commerce," in *Proceedings of the 5th international conference on Electronic commerce*. ACM, 2003, pp. 131–137.

[17] H. Tran, M. Hitchens, V. Varadharajan, and P. Watters, "A trust based access control framework for p2p file-sharing systems," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. IEEE, 2005, pp. 302c–302c.

[18] L. Gommans, J. Vollbrecht, B. Gommans-de Bruijn, and C. de Laat, "The service provider group framework: A framework for arranging trust and power to facilitate authorization of network services," *Future Generation Computer Systems*, vol. 45, pp. 176–192, 2015.

[19] A. Deljoo, T. van Engers, R. Koning, L. Gommans, and C. de Laat, "Towards trustworthy information sharing by creating cyber security alliances," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 1506–1510.

[20] M. Sloman, "Policy driven management for distributed systems," *Journal of network and Systems Management*, vol. 2, no. 4, pp. 333–360, 1994.

[21] C. Bennett, E. Esseiva, T. Kol, and R. Stevens, "Policy driven access to electronic healthcare records," Jun. 28 2007, uS Patent App. 11/316,262.

[22] Miscosoft. (2020) Microsoft security development lifecycle (sdl). [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl/

[23] V. Shivraj, M. Rajan, and P. Balamuralidhar, "A graph theory based generic risk assessment framework for internet of things (iot)," in *2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. IEEE, 2017, pp. 1–6.

[24] S. Sicari, A. Rizzardi, D. Miorandi, and A. Coen-Porisini, "A risk assessment methodology for the internet of things," *Computer Communications*, vol. 129, pp. 67–79, 2018.

[25] P. Anand, J. Ryoo, H. Kim, and E. Kim, "Threat assessment in the cloud environment: A quantitative approach for security pattern selection," in *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication*, 2016, pp. 1–8.

[26] A. A. Poli and M. C. Cirillo, "On the use of the normalized mean square error in evaluating dispersion model performance," *Atmospheric Environment. Part A. General Topics*, vol. 27, no. 15, pp. 2427–2434, 1993.

[27] F. Lindskog, A. McNeil, and U. Schmock, "Kendall's tau for elliptical distributions," in *Credit Risk*. Springer, 2003, pp. 149–156.

[28] B. Lundgren and N. Möller, "Defining information security," *Science and engineering ethics*, vol. 25, no. 2, pp. 419–441, 2019.

[29] P. Anand, J. Ryoo, H. Kim, and E. Kim, "Threat assessment in the cloud environment: A quantitative approach for security pattern selection," in *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication*, 2016, pp. 1–8.

[30] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang, "Containerleaks: Emerging security threats of information leakages in container clouds," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 237–248.

[31] L. Zhang, R. Cushing, L. Gommans, C. De Laat, and P. Grosso, "Modeling of collaboration archetypes in digital market places," *IEEE Access*, vol. 7, pp. 102 689–102 700, 2019.

[32] STRIDE/DREAD. (2020) The dread approach to threat assessment. [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/driversecurity/threat-modeling-for-drivers

[33] CAPEC. (2020) Common attack pattern enumeration and classification. [Online]. Available: https://capec.mitre.org/

[34] X. Zhang, N. Wuwong, H. Li, and X. Zhang, "Information security risk management framework for the cloud computing environments," in *2010 10th IEEE international conference on computer and information technology*. IEEE, 2010, pp. 1328–1334.

[35] J. Luna, H. Ghani, T. Vateva, and N. Suri, "Quantitative assessment of cloud security level agreements: A case study," *Proc. of Security and Cryptography*, pp. 64–73, 2012.

[36] R. Shaikh and M. Sasikumar, "Trust model for measuring security strength of cloud computing service," *Procedia Computer Science*, vol. 45, pp. 380–389, 2015.

[37] A. Sen and S. Madria, "Off-line risk assessment of cloud service provider," in *2014 IEEE World Congress on Services*. IEEE, 2014, pp. 58–65.

[38] G. Disterer, "Iso/iec 27000, 27001 and 27002 for information security management," 2013.

[39] L. A. Gordon, M. P. Loeb, and L. Zhou, "Integrating cost–benefit analysis into the nist cybersecurity framework via the gordon–loeb model," *Journal of Cybersecurity*, vol. 6, no. 1, p. tyaa005, 2020.

[40] C. Alberts, A. Dorofee, J. Stevens, and C. Woody, "Introduction to the octave approach," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, Tech. Rep., 2003.

[41] F. Den Braber, I. Hogganvik, M. S. Lund, K. Stølen, and F. Vraalsen, "Model-based security analysis in seven steps—a guided tour to the coras method," *BT Technology Journal*, vol. 25, no. 1, pp. 101–117, 2007.

[42] D. Seifert and H. Reza, "A security analysis of cyber-physical systems architecture for healthcare," *Computers*, vol. 5, no. 4, p. 27, 2016.

[43] M. Cagnazzo, M. Hertlein, T. Holz, and N. Pohlmann, "Threat modeling for mobile health systems," in *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2018, pp. 314–319.

[44] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys european conference on computer systems 2007*, 2007, pp. 275–287.

[45] R. S. Canon and A. Younge, "A case for portability and reproducibility of hpc containers," in *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, 2019, pp. 49–54.

[46] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Proceedings 1996 IEEE Symposium on Security and Privacy*. IEEE, 1996, pp. 120–128.

[47] S. Forrest, S. Hofmeyr, and A. Somayaji, "The evolution of system-call monitoring," in *2008 annual computer security applications conference (acsac)*. IEEE, 2008, pp. 418–430.

[48] S. M. Varghese and K. P. Jacob, "Process profiling using frequencies of system calls," in *The Second International Conference on Availability, Reliability and Security (ARES'07)*. IEEE, 2007, pp. 473–479.

[49] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and lstm," *Multimedia Tools and Applications*, vol. 78, no. 4, pp. 3979–3999, 2019.

[50] Sysdig, "Sysdig," https://www.sysdig.com/, January 2020. [Online]. Available: https://www.sysdig.com/

[51] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, and C. Talhi, "An anomaly detection system based on variable n-gram features and one-class svm," *Information and Software Technology*, vol. 91, pp. 186–197, 2017.

[52] B. Subba, S. Biswas, and S. Karmakar, "Host based intrusion detection system using frequency analysis of n-gram terms," in *TENCON 2017-2017 IEEE Region 10 Conference*. IEEE, 2017, pp. 2006–2011.

[53] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[54] TensorFlow, "Tensorflow," https://www.tensorflow.org/, January 2020. [Online]. Available: https://www.tensorflow.org/

[55] S.-H. Paek, Y.-K. Oh, J. Yun, and D.-H. Lee, "The architecture of host-based intrusion detection model generation system for the frequency per system

call," in *2006 International Conference on Hybrid Information Technology*, vol. 2. IEEE, 2006, pp. 277–283.

[56] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.

[57] S. Suratkar, F. Kazi, R. Gaikwad, A. Shete, R. Kabra, and S. Khirsagar, "Multi hidden markov models for improved anomaly detection using system call analysis," in *2019 IEEE Bombay Section Signature Conference (IBSSC)*. IEEE, 2019, pp. 1–6.

[58] P. K. Das, A. Joshi, and T. Finin, "App behavioral analysis using system calls," in *2017 IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2017*. Institute of Electrical and Electronics Engineers Inc., nov 2017, pp. 487–492.

[59] R. Cushing, R. Koning, L. Zhang, C. de Laat, and P. Grosso, "Auditable secure network overlays for multi-domain distributed applications," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 658–660.

[60] O. Tunde-Onadele, J. He, T. Dai, and X. Gu, "A study on container vulnerability exploit detection," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 121–127.

[61] T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, R. A. Bridges *et al.*, "A survey of intrusion detection systems leveraging host data," *arXiv preprint arXiv:1805.06070*, 2018.

[62] G. Creech and J. Hu, "Generation of a new ids test dataset: Time to retire the kdd collection," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2013, pp. 4487–4492.

[63] M. Xie, J. Hu, and J. Slay, "Evaluating host-based anomaly detection systems: Application of the one-class svm algorithm to adfa-ld," in *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. IEEE, 2014, pp. 978–982.

[64] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[65] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, J. C. Platt *et al.*, "Support vector method for novelty detection." in *NIPS*, vol. 12. Citeseer, 1999, pp. 582–588.

[66] L. M. Manevitz and M. Yousef, "One-class svms for document classification," *Journal of machine Learning research*, vol. 2, no. Dec, pp. 139–154, 2001.

[67] L. Zhang., R. Cushing., R. Koning., C. de Laat., and P. Grosso., "Profiling and discriminating of containerized ml applications in digital data marketplaces (ddm)," in *Proceedings of the 7th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, INSTICC. SciTePress, 2021, pp. 508–515.

[68] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[69] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[70] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[71] Y. Bengio and Y. Grandvalet, "No unbiased estimator of the variance of k-fold cross-validation," *Journal of machine learning research*, vol. 5, no. Sep, pp. 1089–1105, 2004.

[72] W. Khreich, E. Granger, A. Miri, and R. Sabourin, "A survey of techniques for incremental learning of hmm parameters," *Information Sciences*, vol. 197, pp. 105–130, 2012.

[73] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," *Computers & security*, vol. 21, no. 5, pp. 439–448, 2002.

[74] B. Subba, S. Biswas, and S. Karmakar, "Host based intrusion detection system using frequency analysis of n-gram terms," in *TENCON 2017-2017 IEEE Region 10 Conference*. IEEE, 2017, pp. 2006–2011.

[75] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, and C. Talhi, "An anomaly detection system based on variable n-gram features and one-class svm," *Information and Software Technology*, vol. 91, pp. 186–197, 2017.

[76] Z. Chiba, N. Abghour, K. Moussaid, M. Rida *et al.*, "Intelligent approach to build a deep neural network based ids for cloud environment using combination of machine learning algorithms," *computers & security*, vol. 86, pp. 291–317, 2019.

[77] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[78] M. Xie, J. Hu, and J. Slay, "Evaluating host-based anomaly detection systems: Application of the one-class SVM algorithm to ADFA-LD," *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2014*, pp. 978–982, 2014.

[79] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[80] H. Xiao, H. Xiao, and C. Eckert, "Adversarial label flips attack on support vector machines," *Frontiers in Artificial Intelligence and Applications*, vol. 242, pp. 870–875, 2012.

[81] M. E. Houle, H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Can shared-neighbor distances defeat the curse of dimensionality?" in *International conference on scientific and statistical database management*. Springer, 2010, pp. 482–500.

[82] L. Zhang, R. Cushing, C. de Laat, and P. Grosso, "A real-time intrusion detection system based on oc-svm for containerized applications," in *The 24th IEEE International Conference on Computational Science and Engineering*, 2021.

[83] B. Jin, Y. Chen, D. Li, K. Poolla, and A. Sangiovanni-Vincentelli, "A one-class support vector machine calibration method for time series change point detection," in *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*. IEEE, 2019, pp. 1–5.

[84] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[85] C.-H. Huang, T.-H. Lee, L.-h. Chang, J.-R. Lin, and G. Horng, "Adversarial attacks on sdn-based deep learning ids system," in *International Conference on Mobile and Wireless Technology*. Springer, 2018, pp. 181–191.

[86] Z. Wang, "Deep learning-based intrusion detection with adversaries," *IEEE Access*, vol. 6, pp. 38 367–38 384, 2018.

[87] J. Clements, Y. Yang, A. Sharma, H. Hu, and Y. Lao, "Rallying adversarial techniques against deep learning for network security," *arXiv preprint arXiv:1903.11688*, 2019.

[88] S. Weerasinghe, S. M. Erfani, T. Alpcan, and C. Leckie, "Support vector machines resilient against training data integrity attacks," *Pattern Recognition*, vol. 96, p. 106985, 2019.

[89] Y. Zhou, M. Kantarcioglu, B. Thuraisingham, and B. Xi, "Adversarial support vector machine learning," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 1059–1067.

[90] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter." *LEET*, vol. 8, pp. 1–9, 2008.

[91] A. Paudice, L. Muñoz-González, and E. C. Lupu, "Label sanitization against label flipping poisoning attacks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2018, pp. 5–15.

[92] P. P. Chan, F. Luo, Z. Chen, Y. Shu, and D. S. Yeung, "Transfer learning based countermeasure against label flipping poisoning attack," *Information Sciences*, vol. 548, pp. 450–460, 2021.

Publications

Journal Publications

- **Lu Zhang**, Reginald Cushing, Leon Gommans, Cees De Laat, and Paola Grosso, "Modeling of Collaboration Archetypes in Digital Market Places." IEEE Access 7 (2019): 102689-102700.
- **Lu Zhang**, Arie Taal, Reginald Cushing, Cees de Laat, Paola Grosso, "A risk level assessment system based on the STRIDE/DREAD model for Digital Data Marketplaces." International Journal of Information Security 21, no. 3 (2022): 509-525.

Conference Publications

- **Lu Zhang** "Management of collaborations in digital marketplaces." In 2019 International Conference on High Performance Computing and Simulation (HPCS), pp. 1014-1016. IEEE, 2019.
- Reginald Cushing, Ralph Koning, **Lu Zhang**, Cees de Laat, Paola Grosso, "Auditable secure network overlays for multi-domain distributed applications.' In 2020 IFIP Networking Conference (Networking), pp. 658-660. IEEE, 2020.
- **Lu Zhang**, Reginald Cushing, Ralph Koning, Cees de Laat, Paola Grosso, "Profiling and discriminating of containerized ML applications in Digital Data Marketplaces (DDM)." In ICISSP, pp. 508-515. 2021.
- **Lu Zhang**, Reginald Cushing, Cees de Laat, Paola Grosso, "A real-time intrusion detection system based on OC-SVM for containerized applications." In 2021 IEEE 24th International Conference on Computational Science and Engineering (CSE), pp. 138-145. IEEE, 2021.

- **Lu Zhang**, Reginald Cushing, Paola Grosso, "Defending OC-SVM based IDS from poisoning attacks." In proceeding of 5th IEEE Conference on Dependable and Secure Computing (IEEE DSC 2022)/The 4th International Workshop on Secure Smart Societies in Next Generation Networks (SEC-SOC). IEEE, 2022.

Demonstrations

- Reginald Cushing, **Lu Zhang**, Paola Grosso, Tim van Zalingen, Joseph Hill, Leon Gommans, Cees de Laat, Vijaay Doraiswamy, Purvish Purohit, Kaladhar Voruganti, Craig Waldrop, Rodney Wilson, Marc Lyonnais: "Dataharbours: computing archetypes for digital marketplaces", a demonstration (short paper description) at supercomputing 2018 (SC 2018).

- Reginald Cushing, **Lu Zhang**, Yuri Demchenko, Cees de Laat, Paola Grosso, "Data Harbours: Computing archetypes for digital marketplaces", a demonstration at the 2019 International Conference on High Performance Computing and Simulation (HPCS 2019).

- Ralph Koning, Reginald Cushing, **Lu Zhang**, Cees de Laat, Paola Grosso, Kaladhar Voruganti, Rodney Wilson, Marc Lyonnais: "A secure network overlay for tracking and enforcement of data transaction rules", a demonstration (short paper description) at supercomputing 2019 (SC 2019).

Summary

Data sharing and aggregation can generate great value. To motivate such digital collaborations, it is essential to establish digital infrastructures, e.g. digital data marketplaces, to facilitate policy-driven data sharing and federations in a secure and trustworthy manner. In this thesis, we are motivated to investigate how to select optimal application-tailored infrastructures and enhance policy compliance capabilities.

To match a policy-driven data exchange application to a best-fit digital infrastructure, we mainly consider two aspects, namely, compatibility and security. For compatibility, we presented a mechanism for selecting digital infrastructure patterns that satisfy the collaboration request to a maximum degree by modelling and closeness identification. We also proposed four metrics to allow evaluating and comparing competing DDMs from more complete dimensions: coverage, extensibility, precision and flexibility. We validated the effectiveness of our methodology with a real-world use case.

Security is also an important concern. We presented a threat-analysis driven quantitative risk assessment framework. An in-depth threat analysis was conducted. The data federation application is represented as a list of transactions and the threats are identified in a semi-automatic manner. The framework estimates the relative severity of each threat with a modified version of the Microsoft DREAD model and evaluates the remaining risk after applying the security controls of a DDM infrastructure. We also validated the stability and resolution of our proposed framework with a real world use case. We proved that our proposed system is robust against unavoidable subjective choices of the STRIDE/DREAD model parameters and can provide sufficient resolution.

We further investigated how to enhance the policy enforcement capability in the execution stage. We presented a distributed architecture that detects policy compliance by monitoring and analyzing Linux system call traces generated by the running container. The architecture works as follows. A profile and a pretrained IDS model are built for every uniquely identifiable compute object. They

are initially trained in a secure environment and then distributed to endpoint execution platforms via a secure channel. The system call traces are monitored, analysed and verified in endpoint points platforms. The IDS model is retrained periodically to increase generalization with the empirical data. A sanitization module is implemented to filter out malicious samples to further defend the architecture against adversarial machine learning attacks.

Firstly, we introduced a methodology to profile and verify the running behaviors of an algorithm. This allows us to distinguish whether an algorithm running inside a container is the one claimed to be. With experimental results, we showed that the proposed methodology can provide high accuracy and support profile reuse over different platforms and input data.

Secondly, we presented a hybrid real-time IDS. We adopt the OC-SVM algorithm to detect anomalies and apply a signature-based methodology to reduce false alarms. The streaming systems calls are segmented with a constant time window. We evaluated that the proposed IDS can achieve outstanding performance with high TPR and relatively low FPR values for detecting modern containerized attacks. In addition, we investigated the influence of various feature extraction methods, kernel functions and segmentation length with four metrics.

Thirdly, we conducted experiments to evaluate OC-SVM based IDS sensitivity for poisoning attacks. We proposed a sanitization mechanism based on the DBSCAN clustering algorithm. We demonstrated that the proposed mechanism effectively filters out malicious samples and achieves almost equal accuracy with an untainted training dataset.

Samenvatting

Het delen en samenvoegen van gegevens kan grote waarde genereren. Om dergelijke digitale samenwerkingen te motiveren is het essentieel om digitale infrastructuren op te zetten, bijvoorbeeld digitale datamarkten, om beleidsgestuurde gegevensdeling en federaties op een veilige en betrouwbare manier te faciliteren. In dit proefschrift zijn we gemotiveerd om te onderzoeken hoe optimale infrastructuren op maat van de toepassing kunnen worden geselecteerd en hoe de mogelijkheden om het beleid na te leven kunnen worden verbeterd.

Om een beleidsgestuurde toepassing voor gegevensuitwisseling af te stemmen op een best passende digitale infrastructuur, beschouwen wij hoofdzakelijk twee aspecten, namelijk compatibiliteit en veiligheid. Voor compatibiliteit hebben wij een mechanisme voorgesteld voor het selecteren van digitale infrastructuurpatronen die maximaal voldoen aan de samenwerkingsvraag door middel van modellering en identificatie van nabijheid. Wij hebben ook vier metrieken voorgesteld om concurrerende DDM's te kunnen evalueren en vergelijken vanuit vollediger dimensies: dekking, uitbreidbaarheid, precisie en flexibiliteit. Wij hebben de doeltreffendheid van onze methodologie gevalideerd met een praktijkgeval.

Veiligheid is ook een belangrijk punt van zorg. Wij presenteerden een kader voor kwantitatieve risicobeoordeling op basis van dreigingsanalyse. Er werd een diepgaande dreigingsanalyse uitgevoerd. De datafederatie-applicatie wordt voorgesteld als een lijst van transacties en de bedreigingen worden op semi-automatische wijze geïdentificeerd. Het raamwerk schat de relatieve ernst van elke bedreiging met een aangepaste versie van het Microsoft DREAD-model en evalueert het resterende risico na toepassing van de beveiligingscontroles van een DDM infrastructuur. We hebben ook de stabiliteit en resolutie van ons voorgestelde raamwerk gevalideerd met een praktijkgeval. Wij hebben aangetoond dat ons voorgestelde systeem robuust is tegen onvermijdelijke subjectieve keuzes van de STRIDE/DREAD modelparameters en voldoende resolutie kan bieden.

Verder hebben we onderzocht hoe we het vermogen tot handhaving van het

beleid in de uitvoeringsfase kunnen verbeteren. We hebben een gedistribueerde architectuur gepresenteerd die naleving van het beleid detecteert door het monitoren en analyseren van Linux system call traces die door de draaiende container worden gegenereerd. De architectuur werkt als volgt. Voor elk uniek identificeerbaar computerobject worden een profiel en een voorgetraind IDS-model gemaakt. Ze worden eerst getraind in een beveiligde omgeving en vervolgens via een beveiligd kanaal gedistribueerd naar platforms voor eindpuntuitvoering. De systeemaanroepsporen worden gecontroleerd, geanalyseerd en geverifieerd op eindpuntplatforms. Het IDS-model wordt periodiek bijgeschoold om de generalisatie met de empirische gegevens te verhogen. Een saneringsmodule is geïmplementeerd om schadelijke samples uit te filteren om de architectuur verder te verdedigen tegen machine learning-aanvallen van tegenstanders.

Ten eerste hebben we een methode geïntroduceerd om het gedrag van een algoritme te profileren en te verifiëren. Zo kunnen we onderscheiden of een algoritme dat in een container draait, het algoritme is waarvan wordt beweerd dat het draait. Met experimentele resultaten hebben wij aangetoond dat de voorgestelde methodologie een hoge nauwkeurigheid kan bieden en hergebruik van profielen over verschillende platforms en invoergegevens kan ondersteunen.

Ten tweede hebben wij een hybride IDS gepresenteerd. Wij gebruiken het OC-SVM algoritme om anomalieën te detecteren en passen een op handtekeningen gebaseerde methodologie toe om valse alarmen te verminderen. De streaming systeemoproepen worden gesegmenteerd met een constant tijdvenster. Wij hebben geëvalueerd dat het voorgestelde IDS uitstekende prestaties kan leveren met een hoge TPR en relatief lage FPR waarden voor het detecteren van moderne gecontaineriseerde aanvallen. Bovendien onderzochten wij de invloed van verschillende methoden voor kenmerkextractie, kernelfuncties en segmenteringslengte met vier metrieken.

Ten derde hebben we experimenten uitgevoerd om de gevoeligheid van OC-SVM-gebaseerde IDS voor poisoning-aanvallen te evalueren. We hebben een zuiveringsmechanisme voorgesteld op basis van het DBSCAN-clusteralgoritme. Wij hebben aangetoond dat het voorgestelde mechanisme effectief schadelijke monsters uitfiltert en bijna dezelfde nauwkeurigheid bereikt als een onbezoedelde trainingsdataset.

Acknowledgements

Time flies. I remember the first time I came to the science park for the interview, the first day I started my PhD and worked in my office. It was an amazing adventure to experience all that happened during the last four years. It would be impossible for me to finish this journey alone. There are all of you providing me kindness and endless help along the path.

I would like to first thank my promotors Cees and Leon. Cees, I always got great inspiration from the meetings and discussions we have had. You are always kind and supportive. You listen to my questions patiently and offer fruitful suggestions and advice. I deeply appreciate your input and help in the past four years. Leon, I started my PhD work with the famous archetypes you proposed for the project. During the project meetings, you always provide me with valuable advice. Thank you for your effort of helping me write my thesis. You taught me how to convey my ideas in a more concise and accurate way.

I would like to express my deep gratitude to my promotor and supervisor, Paola. Thank you for always being there offering your endless support and kind help. I deeply appreciate all the input and guidance that you devoted to me and my research. You helped me patiently in writing my first academic paper, encouraged me to present confidently in public, and provided constructive advise when I was stuck in research. You have taught me so much, which is not only about knowledge, but also the way to solve questions, the way to conquer barriers. It was a fantastic experience working with you for the past four years.

Thanks to my committee members for taking the time to read and evaluate my thesis. I appreciate all your fruitful feedback.

Sara, Ruyue and Jamila, nice to meet you here. Dear Sara, we experienced this adventure together. Your accompaniment and encouragement make this journey more interesting and less lonely. Ruyue and Jamila, I treasure all the joyful moments with you, the chats and laughs. All of these make this trip more colorful. I am also grateful to all my officemates, Yixian, Zeshun, Xiaofeng, Lourens and Henk.

I am also grateful towards my colleagues in the project DL4LD: Reggie, Ralph, Xin, Mostafa, Giovanni, Lydia, Adam and Arie. I got great inspiration for my research from the discussions with you. Reggie, thank you for being my daily supervisor. You are always willing to help and answer my questions patiently. Arie, thank you very much for guiding me write articles with your insights about notations and formulas.

I would like to thank all my colleagues from the MNS group and the SNE cluster. It was a nice experience working with you. I am thankful to the people who I have interactions with during various group events and coffee breaks: Ameneh, Ana, Chrysa, Cyril, Dolly, Florian, Garazi, Henk, Hongyun, Huan, Joseph, Julius, Jun, Llorenç, Leonardo, Lu-chi, Lukas, Lukasz, Marco, Marian, Milen, Misha, Na, Peter, Riccardo, Ruyue, Saba, Saeedeh, Spiros, Uraz, Xiaotian, Yang, Yuandou, Yuri, Zhiming.

Finally yet importantly, I would like to express my very deep appreciation to my parents. Baba and Mama, thank you so much for your endless love and support even though we are far away. I love you;-). I also want to thank my cats, cola, dragon and tiger, for their love (I assume). Luckiness is mine to meet you, Wei! Thank you for your indispensable support during my journey to a PhD and for always being with me.