Generative Pattern Dissemination for Collaborative Intrusion Detection

A thesis presented for the degree of Master of Science
by Mike Petersen

First Reviewer: Prof. Dr. Ulrich Bühler
Second Reviewer: Prof. Dr. Sebastian Rieger

Department of Computer Sciences
University of Applied Sciences Fulda
Fulda, Germany
July 2021

Contents

Chapter 1

Introduction

1.1 Motivation

1.2 Objectives

1.3 Structure

Chapter 2

Preliminaries

2.1 Intrusion Detection

Classic security mechanisms, such as encryption or firewalls, are considered as preventive measures for protecting IT infrastructures. However, in order to be able to react to security breaches that have already occurred, additional reactive mechanisms are required. To complement preventive measures, Intrusion Detection Systems (IDSs) have been commercially available since the late 1990s [WM18, p. 27].

First, IDSs are described and categorized. In the context of the shortcomings of conventional IDSs for protecting large scale systems, Collaborative Intrusion Detection Systems (CIDSs) are introduced.

2.1.1 Intrusion Detection Systems

Generally, the main reason for operating an IDS is to monitor and analyze computer networks or systems in order to identify anomalies, intrusions or privacy violations [Hin+20]. Specifically, the following three advantages are significant [WM18, p. 391].

- IDSs can detect the preliminaries of attacks, in particular the organized gathering of information about networks and defense mechanisms (attack reconnaissance), and thus enable the prevention or mitigation of damage to information assets.
- IDSs can help protect information assets when known vulnerabilities cannot be fixed fast enough, notably in the context of an rapidly changing threat environment.
- The occurrence of unknown security vulnerabilities (zero day vulnerabilities) is not predictable, meaning that no specific preparations can be made for them. However, IDSs can identify processes in the IT system that deviate from the normal state and thus contribute to the detection of zero day attacks

For an effective IDS, it is important to be able to detect as many steps as possible within the typical attack sequence, also called kill chain [WM18, p. 393]. Since a successful intrusion into a system can be stopped at several points in this sequence, the effectiveness of the IDS increases with its functionality. The following categorization

of intrusion attempts according to [Ken99] reflects parts of the kill chain mentioned above:

Probing Probing refers to the preambles of actual attacks, also known as attack reconnaissance. This includes obtaining information about an organization and its network behavior (footprinting) and obtaining detailed information about the used operating systems, network protocols or hardware devices (fingerprinting).

Denial of Service (DoS) DoS refers to an attack aimed at disabling a particular service for legitimate users by overloading the target systems processing capacity.

Remote to Local (R2L) R2L attacks attempt to gain local access to the target system via a network connection.

User to Root (U2R) One step further, U2R attacks start out with user access on the system and gain root access by exploiting vulnerabilities.

Additionally, IDS are generally categorized by the platform being monitored and the employed attack detection method [Mil+15]. Hence, a distinction is made between Host-based Intrusion Detection System (HIDS) and Network-based Intrusion Detection System (NIDS). While a HIDS resides on a system, known as host, only monitoring local activities, a NIDS resides on a network segment and monitors remote attacks that are carried out across the segment. Furthermore, IDS are categorized into signature-based detection and anomaly-based detection. A signature-based IDS (also called knowledge-based detection) compares the system or network state against a collection of signatures of known attacks. The false positive rate is very low when using signatures, but only when assuming the system is confronted with already known attacks. Typically, this method cannot detect novel [WM18, p. 403], metamorphic or polymorphic attacks [Szo05, p. 236]. An anomaly-based IDS, on the other hand, creates a statistical baseline profile of the system's or network's regular state and compares it with the monitored activities. This allows both known and unknown attacks to be detected, but the frequent occurrence of false-positive estimations is a major challenge. Furthermore, hybrid models of the presented approaches from the respective categories exist.

2.1.2 Coordinated Attacks

2.1.3 Collaborative Intrusion Detection Systems

However, when considering the development of the current threat environment, the effectiveness of conventional intrusion detection systems is limited. Technology trends, such as the internet of things or cloud computing, are main drivers for increasingly blurring corporate boundaries in the context of interconnection of infrastructures and shared resources. This transformation increases the potential attack surface of productive computer systems for large-scale and high-velocity cyber attacks, which traditional IDSs have limited effectiveness due to their isolated nature. For example, such standalone IDS will not be able to create connections between security events that occur at different infrastructures simultaneously. Due to the mentioned increase of attack surface that is related to the size of current computer networks, attackers may attempt to obfuscate the characteristic overall sequence of the intrusion by spreading single attack steps.

n order to address the aforementioned security problems of large IT systems, Collaborative Intrusion Detection Systems (CIDSs) have been proposed. In general, a CIDS is a network of several intrusion detection components that collect and exchange data on system security. A CIDS is essentially specified by two different types of components, namely the detection units and the correlation units, and their communication among each other. The detections units can be considered as conventional IDS that monitor a sub-network or a host and by that, generate low-level intrusion alerts. The correlation unit is responsible for merging the low-level intrusion alerts and their further post-processing. This includes, for instance, the correlation of the alerts, the generation of reports or the distribution of the information to the participants of the network. CIDSs pursue the following two goals [Vas16, p. 24].

- The aggregation and correlation of data originating from different IDSs creates a holistic picture of the network to be monitored and enables the detection of distributed and coordinated attacks.
- CIDSs can monitor large-scale networks more effectively with the realization of a loadbalancing strategy. By sharing IDS resources across different infrastructures, short-term peak loads can be served, reducing the downtime of individual IDSs.

2.2 Gaussian Mixture Models

2.2.1 Gaussian Distribution

2.2.2 Density Estimation

2.2.3 Gaussian Mixtures

How many components should be chosen for fittin a GMM? A common method is to select the model M with the highest probability given the data D. Assuming that the model M is completely described by its set of parameters θ, the maximum likelihood function of the model M is given by

$$\hat{L} = p(D|\hat{\theta}, M), \tag{2.1}$$

answers the question of what is the probability that D is explained by M. Note, that the carat denotes the parameters that maximize the probability, i.e., the maximum likelihood function. Theoretically, in the case of a GMM, one could just increase the number of parameters K, e.g., the number of components, arbitrarily until the $K = N$ in order to obtain the maximum likelihood. In practice, this would not only lead to a bad runtime of the algorithm, but also overfit the model. Thus, the maximum likelihood of the model L has to be balanced against the number of model parameters K. The Bayesian information criterion (BIC) is considered as a standard criterion for model selection of GMM because of its theoretical consistency in choosing the number of components [Ker00].

In general, the BIC can be defined as

$$\mathrm{BIC} = K \ln(N) - 2 \ln(\hat{L}), \tag{2.2}$$

which derives from the findings in [Sch78]. The BIC balances the number of model parameters K and number of data points N against the maximum likelihood function L. In the model selection, the optimal number of model parameters K minimizes the BIC, such that the BIC provides a principled way of selecting between multiple different models. More complex models almost always fit the data better, resulting in a lower value of $-2\ln(\hat{L})$. The BIC penalizes extra parameters by introducing the term $K\ln(N)$. Beyond penalizing more parameters, it furthermore assists in making a judgement as to how the additional parameters improve the model in the presence of more data.

Considering a mixture model with K components defined by

$$\begin{aligned} p(\boldsymbol{x}|\theta) &= \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\theta), \\ 0 \leq \pi_k \leq 1, &\sum_{k=1}^{K} \pi_k = 1, \\ \theta &:= \{\overline{\boldsymbol{x}}_k, \mathbf{C}_k, \pi_k : k = 1, \ldots, K\}, \end{aligned} \tag{2.3}$$

the parameters of K multivariate normal distributions are to learn, with d dimensions, these are d values in the mean vector. Since a covariance matrix is symmetric, only $d(d+1)/2$ entries in a full covariance matrix have to be computed. Additionaly, K mixture weights have to be determined. Since these sum to one, it is sufficient to determine $K-1$ weights, leading to $Kd+K(d(d+1)/2)+K-1$ parameters. Thus, the BIC for a dataset D with N datapoints of dimensionality d and a GMM M as defined above is stated as

$$\mathrm{BIC}(M|D) = (Kd + K(d(d+1)/2) + K - 1)\ln(N) - 2\ln(\hat{L}). \tag{2.4}$$

A model selection algorithm could look like the following.

Algorithm 1 GMM Selection with BIC

Input: Dataset D
Output: Gaussian Mixture Model M

$B \leftarrow$ new Array
GMM $\leftarrow$ new Array
for each k in $\{1, \ldots, K\}$ **do**
    $M_k \leftarrow \mathrm{fitGMM}(D)$
    $b_k \leftarrow \mathrm{BIC}(M|D)$
    $\mathrm{GMM}[k] \leftarrow M_k$
    $B[k] \leftarrow b_k$
$\hat{k} \leftarrow \mathrm{argmin}(B)$
$\hat{M} \leftarrow \mathrm{GMM}[\hat{k}]$
return $\hat{M}$

2.3 Locality Sensitive Hashing

Finding similar objects, formally known as the nearest-neighbour search problem, is a central problem in computer science in general. And besides the exact search, it is also an important topic in the field of intrusion detection. For example, exact searches allow to compare the signatures of known malware with low false positive rates. Considering polymorphic attacks, however, it is essential to also detect all objects that are similar to the known attack and thus detect modified forms of it. In this context, this section presents a well-studied approach called *Locality Sensitive Hashing* (LSH) that solves an approximate version of the nearest-neighbour search problem by partitioning the search space with a hash function. This way, the searching problem is reduced to pairs that are most likely to be similar. Besides the application for solving the NN search problem, for which LSH was originally conceived, the concept has been proven to be effective for numerous other use cases, such as dimensionality reduction, clustering or classification. As the proposed generative pattern database integrates a locality-sensitive hash function for enabling both similarity search and data parallelism, LSH and a specific variant called *Random Projection* (RP) is described in detail in this section.

Section 2.3.1 introduces the approximate version of the nearest neighbour search problem as it is a prerequisite for the general definition of LSH. After that, Section 2.3.2 begins by clarifying the core idea of LSH and subsequently explains how to construct a locality-sensitive hash function. Lastly, a specific family of LSH that uses the cosine distance for similarity calculations, also known as *Random Projection* is presented in Section 2.3.3.

2.3.1 The Approximate Nearest Neigbour Problem

For all definitions of the NN and its variants, a set of n points $P = \{p_1, \ldots, p_n\}$ in a metric space (X, d) where $P \subset X$ is considered.[1] Then, the general NN is stated as follows.

Definition 1 (Nearest Neigbour Problem)**.** Construct a data structure so as to efficiently answer the following query: Given any query point q, find some point $p \in P$ such that

$$\min_{q \in X} \; d(p, q). \tag{2.5}$$

A specific example of Definition 1 could include high-dimensional data in $P \in \mathbb{R}^k$ with $k \gg 1$ and define d as the euclidean distance. In this case, an exhaustive search would require a query time of $O(kn)$. Unfortunately, all exact algorithms that provide a better time complexity than an exhaustive search require $O(2^k)$ space [Rub18]. This tradeoff between time and space complexity is usually referred to as "curse of dimensionality" and can only be resolved by accepting approximate solutions. The c-approximate nearest neighbour problem (c-ANN) is defined as follows.

Definition 2 (c-Approximate Nearest Neighbour Problem)**.** For any given query point $q \in X$ and some approximation factor $c > 1$, find some point $p \in P$ such that

$$d(q, p) < c \cdot \min_{s \in P} \; d(s, q). \tag{2.6}$$

[1] It is assumed that d is a proper *metric*, which means that it is *symmetric*: $d(p, q) = d(q, p)$, *reflexive*: $d(p, q) \leq 0$, $d(p, q) = 0 \iff p = q$ and satisfies the *triangle inequality*: $d(p, q) \leq d(p, s) + d(s, q)$.

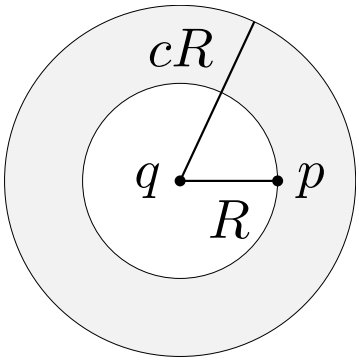


Figure 2.1: In the cR-approximate nearest neighbour problem some point within cR is accepted, if there exists a point p where $d(p, q) \leq R$.

Thus, the distance from the query point q to the approximate nearest neighbour p is at most c times the distance to the true nearest neighbour s. Strictly speaking, LSH does not solve the c-ANN directly. Instead, Indyk and Motwani relaxed the problem by introducing the cR-approximate nearest neighbour problem (cR-ANN) as follows.

Definition 3 (cR-Approximate Nearest Neighbour Problem)**.** For any given query point $q \in X$, some approximation factor $c > 1$ and some target distance $R > 0$, if there exists a point $p \in P$ where $d(p, q) \leq R$, then return a point $p' \in P$ where

$$d(p', q) \leq cR. \tag{2.7}$$

Figure 2.1 illustrates the cR-ANN. The target distance R represents the distance of the query object from its nearest neighbour. If there is such a point, the algorithm returns points within cR distance from the query object. Otherwise it returns nothing. It is shown that LSH can solve the c-ANN by solving the cR-ANN for different settings of R [IM98].

2.3.2 Locality-Sensitive Hash Functions

Introduced by Indyck and Motwani in 1998 [IM98] as an algorithm that solves the approximate nearest neighbour problem (ANN), locality-sensitive hashing (LSH) has since been extensively researched and is now considered among the state of the art for approximate searches in high-dimensional spaces.[2] The basic idea of the approach is to partition the input data using a hash function that is sensitive to the location of the input within the metric space. This way, similar inputs collide with a higher probability than inputs that are far apart. Thus, LSH exhibits fundamental differences to conventional hash functions [3], although the most general definition applies to both.

A hash function is a function that maps a large input set to a smaller target set. The elements of the input set are called *messages* or *keys* and may be of arbitrary different lengths. The elements of the target set are called *digests* or *hash values* and are of fixed size length. More specifically, we define a hash function as $h : A \to B : \boldsymbol{p} \mapsto \boldsymbol{u}$ where $A \subset \mathbb{R}^d$ with $d \in \mathbb{N}$ is the input set and $B = \{0, 1\}^k$ with $k \in \mathbb{N}$ the target set of all bit

[2] See [NBJ21] for an exhaustive survey of NN-Search Techniques.

[3] In the following, cryptographic and non-cryptograhpic hash functions are referred to as conventional hashing.

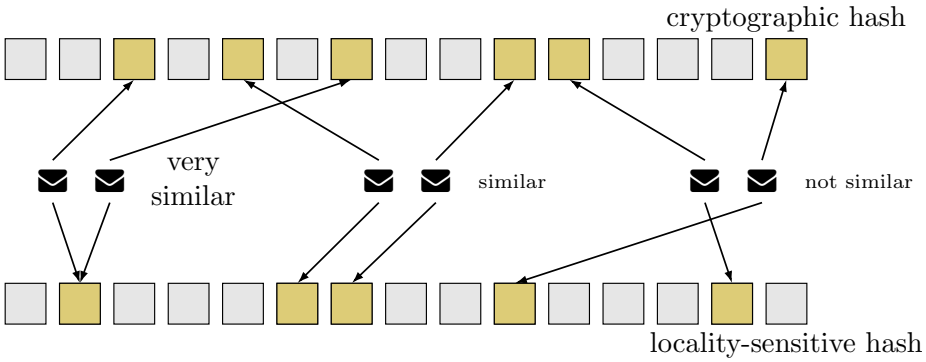


Figure 2.2: Cryptographic hash function and locality-sensitive hash function exhibit different properties for the probability of collisions.

sequences of fixed size k, with $k < d$. Furthermore, a hash table is defined by a hash function h, that maps the keyspace K into the slots of a hash table $T[0\,..\,S-1], S \in \mathbb{N}$, i.e. $h : K \to \{0, 1, \ldots, S-1\}$ with $S \ll |K|$. Thus, a message with key $k \in K$ hashes to slot $h(k)$. Additionally, $h(k)$ is the hash value of the key k [Cor+22, p. 256].

Typically,conventional hashing is used for the realization of, e.g. hash tables, data integrity checks, error correction methods or database indexes. Depending on the application, different requirements are imposed on the utilized hash function. In this context, the most important property of a hash function is the probability of a *collision*. A collision occurs when two keys $\boldsymbol{p}_1 \neq \boldsymbol{p}_2$ are projected onto the same hash value $\boldsymbol{u} = h(\boldsymbol{p}_1) = h(\boldsymbol{p}_2)$.

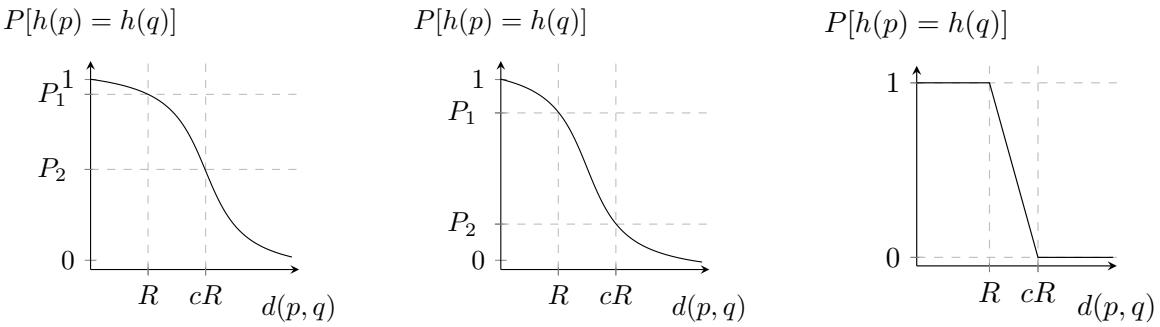
For example, cryptographic hash functions are used in systems, where adversaries try to break these systems. Thus, different security requirements are defined [Wil, p. 349]. In particular, these hash functions are designed to be resistant against collisions, which is the key difference to LSH. Applications that do not require the hash function to be resistant against adversaries, e.g. hash tables, caches or de-duplication, are usually implemented by using a hash function that exhibits relaxed guarantees on the security properties in exchange for significant performance improvements. Nevertheless, such non-cryptographic hash functions share the same idea with cryptographic hash functions.

Figure 2.2 illustrates this difference. LSH projects similar inputs onto the same or near elements. In contrast, conventional hashing tries to distribute projections onto the elements of the target as randomly as possible.

As a locality-sensitive hashing function can be constructed as a general concept, specific families of functions can be derived. We refer to a *family* of hash functions $\mathcal{H} : A \to B$ as a collection of hash functions that have the same domain and range, share a basic structure and are only differentiated by constants. Three basic requirements are demanded for such a family of functions [LRU14, p. 99].

1. close pairs are to be hashed in to the same bucket with higher probability than distant pairs

2. the functions need to be statistically independent, such that the product rule can be applied on the results of two or more functions; It is said, that these functions



(a) Single LSH Function. (b) Amplified LSH Function. (c) Exact Search

Figure 2.3: The behaviour of a (R, cR, P_1, P_2)-sensitive function in (a) and (b) (adapted from [LRU14, p. 100]) approaching the ideal probability gap in (c) resembling the behaviour of an exact search.

are combinable, such that functions can be build/combined/constructed that provide less fals positives and false negatives than single functions

3. they need to be efficient, i.e. able to identify similar pairs in much less time than the time it takes to make a linear scan through all pairs (faster than exhaustive search)

The first step is to define LSH generally. Applied on the cR-ANN, the first requirement states more specifically with high probability, two points p and q should hash to the same hash value if their distance at most R, i.e. $d(p, q) \leq R$. And if their distance is at least cR, the points should hash to different hash values, i.e. $d(p, q) > cR$. Thus, A formal definition of a locality-sensitive hash function is given as follows [AI06].

Definition 4 (Locality-Sesitive Hash Function)**.** Given a threshold $R \in \mathbb{R}^{>0}$, an approximation factor $c \in \mathbb{R}^{>1}$ and probabilities $P_1, P_2 \in \mathbb{R}^{\geq 0}$, a family $\mathcal{H} = \{h : A \to B\}$ is called (R, cR, P_1, P_2)-sensitive if for any two points $p, q \in A$ and any hash function h chosen uniformly at random from $\mathcal{H}$ the following conditions are satisfied:

$$d(p, q) \leq R \implies P[h(p) = h(q)] \geq P_1,$$
$$d(p, q) \geq cR \implies P[h(p) = h(q)] \leq P_2.$$

Ideally, the gap between P_1 and P_2 should be as big as possible as depicted in Figure 2.3c, which in fact represents an exact search, which is, as already discussed, no option due to its time and space requirements. Considering a single locality-sensitive function as shown in Figure 2.3a, where the probability gap between P_1 and P_2 is relatively close, the false negative rate would be relatively high. Increasing the gap would require to increase c and lead to a high number of false positives. Therefore, a single function would provide only a tradeoff. But it is possible to increase P_1 close to 1 and decrease P_2 close to $1/n$ while keeping R and cR fixed as shown in Figure 2.3b by introducing a process called *amplification*.

First reduce P_2 by applying logical AND (AND-construction on $\mathcal{H}$): sample k hash functions indepedently from $\mathcal{H}$ and hash each point $p \in P$ to a k-dimensional vector with a new constructed function $g \in \mathcal{H}^k$:

$$g(p) = [h_1(p), h_2(p), \ldots, h_k(p)] \tag{2.8}$$

Then by the independence of $h_1, \ldots, h_k$ the product rule applies and for any two points p and q, a collision occurs if and only if $h_i(p) = h_i(q)$ for all $i = \{1, \ldots, k\}$. The probabilities are as follows:

$$P[h_i(p) = h_i(q)] \geq P_1 \implies P[g(p) = g(q)] \geq P_1^k \tag{2.9}$$
$$P[h_i(p) = h_i(q)] \leq P_2 \implies P[g(p) = g(q)] \leq P_2^k \tag{2.10}$$

By increasing k, P_2 can be arbitrarily decreased approaching 0. However, such an AND-construction lowers both P_1 and P_2.There is another construction in order to improve P_1, the OR-construction: Using multiple families; specifically define $l \in \mathbb{N}$ functions $g_1, \ldots, g_l$.

What is the probability that a collision occurs when hashing a point $p \in P$ witch each $g_j(p)$ for $j \in \{1, \ldots, l\}$? Considering, that the algorithm is successful, when p, q collide at least once for some g_j, the probability is

$$P[\exists j, g_j(p) = g_j(q)] = 1 - P[\forall i, g_j(p) \neq g_j(q)] \tag{2.11}$$
$$= 1 - P[g_j(p) \neq g_k(q)]^l \tag{2.12}$$
$$\geq 1 - (1 - P_1^k)^l \tag{2.13}$$

As the AND-construction lowered both P_1 and P_2, similarly the OR-construction rises both probablities. By choosing k and l judiciously, P_2 can be approached close to 0 while P_1 approaches 1. Both constructions may be concatenated in any order to manipulate P_1 and P_2. Of course, the more construction are used and the higher the values for the parameters k and l are picket, the larger the final function will be and it also takes longer to apply such a function.

The algorithm is as follows. The construction of the data-structure:

- construct $g_1, \ldots, g_l$, each of length k
- hash each point $p \in P$ with each g_j and store them in l hash tables accordingly

Answering a query q:

- compute $g_1(q), \ldots, g_l(q)$. For each $g_j(q)$ identify those p, such that $g_j(p) = g_j(q)$. For each identified p check if it answers cR-ANN with yes, i.e. $d(p, q) \leq cR$.

Naive betrachtungsweise des algorithmus: Space complexity is $O(lnk)$, since there are l hash tables, each with n points, and for each point storing a $k - dim$ vector.

Lower bounds have been proven in [MNP06]

Theorem 5. Let (X, d) be a metric on a subset of $\mathbb{R}^d$. Given a (R, cR, P_1, P_2)-locality sensitive hash family $\mathcal{H}$ and write $\rho = \frac{\log(1/P_1)}{\log(1/P_2)}$. Then for $n = |X|$ and for any $n \geq \frac{1}{P_2}$ there exists a solution to the cR-ANN with space complexity $O(dn + n^{1+\rho})$ and query time of $O(n^\rho)$

2.3.3 Random Projection

cosine distance makes sense in spaces that have dimensions, including euclidean spaces and discrete versions of euclidean spaces [LRU14, p. 95]

calculate the cosine distance by first computing the cosine of the angle, and then applying the arc-cosine function to translate an angle in the 0-180 degree range [LRU14, p. 95] cosine distance between two points is the angle that the angle of that vectors to those points make; this angle is in the range 0 to 180 degrees, regardless of the dimensionality of the Space

Definition 6 (Cosine Distance)**.** Given two vectors p_1 and p_2, the cosine distance $\theta(p_1, p_2)$ is the dot product of p_1 and p_2 divided by their euclidean distances from the origin (L_2-norm):

$$\theta(\boldsymbol{p}_1, \boldsymbol{p}_2) = \cos^{-1}\left(\frac{\boldsymbol{x}_1 \cdot \boldsymbol{x}_2}{||\boldsymbol{p}_1||\ ||\boldsymbol{p}_2||}\right). \tag{2.14}$$

θ can be divided by π to have a distance in the range $[0, 1]$.

Definition 7 (Cosine Similarity)**.** Opposed to the cosine distance, the cosine similarity is defined as

$$1 - \theta(\boldsymbol{p}_1, \boldsymbol{p}_2) \tag{2.15}$$

Introduced in [Cha02] Given a message $\boldsymbol{x} \in A = \{x \in \mathbb{R}^d : 0 \leq x \leq 1\}$ and a randomly selected hyperplane defined as $\boldsymbol{M} = (a_{ij}) \in \mathbb{R}^{d \times k}$ where $a \sim \mathcal{N}(0, I)$, a *gaussian random projection (GRP)* aims to (I) reduce the dimensionality from d to l dimensions and (II) provide a binary encoding by first projecting $\boldsymbol{x}$ onto $\boldsymbol{M}$ and subsequently applying the sign function to each element of the result, i.e. :

$$h(\boldsymbol{x}) = [h(\boldsymbol{x}, a_1), \ldots, h(\boldsymbol{x}, a_k)] \text{ with } h(\boldsymbol{x}, a) = sign(\boldsymbol{x}^T a)$$
$$\text{with } sign(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}.$$

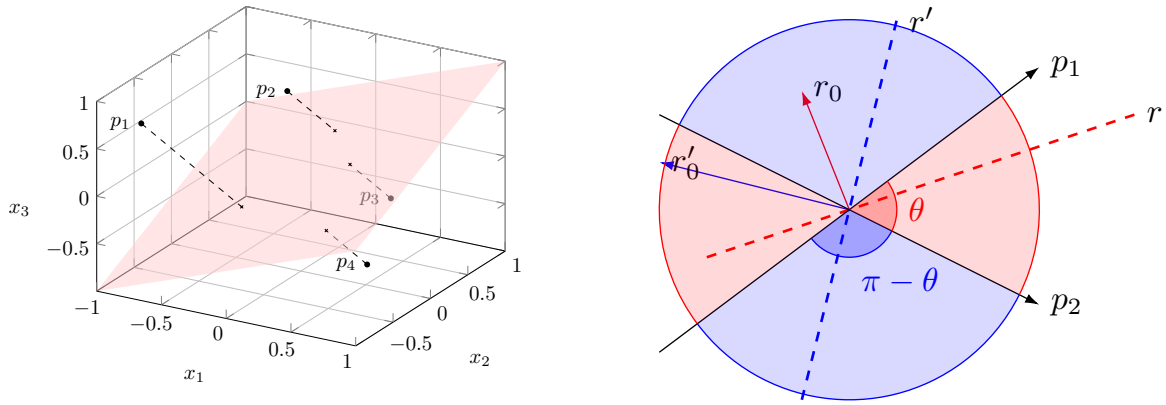
The resulting digest is a binary vector $h(\boldsymbol{x}) = \boldsymbol{u} \in B = \{0, 1\}^l$ that is commonly used as bucket index for storing $\boldsymbol{x}$ in a hash table. For any two messages $\boldsymbol{x}_1, \boldsymbol{x}_2$, the probability of being hashed to the same bucket increases with a decreasing distance, which is given by the angular distance as

$$P[h(\boldsymbol{x}_1) = h(\boldsymbol{x}_2)] = 1 - \frac{\theta(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\pi} \tag{2.16}$$

Consider Figure 2.4b, where two vectors p_1 and p_2, regardless of their dimensionality, define a plane and an angle θ in this plane.

Pick a hyperplane (actually the normal vector to the hyperplane; hyperplane v is the set of points whose dot product with v is 0)

Hyperplane intersects the plane that is spanned by two vectors p_1 and p_2 in a line



(a) Illustration of a random hyperplane (red) partitioning the space.

(b) Visual Proof of claim in equation 2.16

Figure 2.4

vector r_0 that is normal to the hyperplane represented by the red dashed line; p_1 and p_2 are on different sides of the hyperplane, thus the projections given by $\langle p_1, r_0 \rangle$ and $\langle p_2, r_0 \rangle$ will have different signs

vector r'_0 that is normal to the hyperplane represented by the blue dashed line; both $\langle p_1, r'_0 \rangle$ and $\langle p_2, r'_0 \rangle$ will have the same sign

All angles between the intersection line of the random hyperplane and the plane spanned by p_1 and p_2 are equally likely. Thus, the probability that the hyperplane looks like the red line is θ/π and like the blue line otherwise.

random projections is a $(R, cR, (1 - R/\pi), (1 - cR/\pi))$-sensitive family for any R and cR. As already explained in Section 2.3.2, such a family can be amplified as desired.

Chapter 3

State of the Art

3.1 Data Dissemination in Collaborative Intrusion Detection

The key components for designing a CIDS architecture are described in [Vas16, p. 34]. Among them, data dissemination is particularly noteworthy as one of the fundamental components for the communication between members. A central aspect is the communication *overhead* that is introduced with the dissemination of alerts and knowledge within a CIDS, which in turn is heavily influenced by the CIDS architecture [Vas16, p.39], that can be primarily categorized into centralized, hierarchical and decentralized approaches [ZLK10] (see Figure **??**). Centralized architectures, consisting of multiple monitoring units and a central analysis unit [CM02][MI03], suffer from a Single Point of Failure (SPoF) and are limited in their scalability due to a bottleneck, which is introduced by the central analysis unit. However, the SPoF problem can be mitigated if individual components in such a system are implemented redundantly and form a centralized architecture only at the logical level. A bottleneck, on the other hand, is usually avoided by a distributed implementation, which can also logically be regarded as a central data repository. Hierarchical designs exhibit multiple monitoring and analysis units organized in a tree-based topology [PN97; Zha+01; Ngu+19]. These systems are restricted in their scalability by the respective instances on higher levels, whose failure results in a malfunction of the respective sub-trees [ZLK10]. Again, such disadvantages only play a major role in non-redundantly implemented systems. Additionally, there exist approaches that are inherently distributed. Distributed architectures [Vas+15; Gil+13; BA08; Fun+08; JWQ03], wherein each participating system has a monitoring and analysis feature and where the communication is based on some form of data distribution protocol, are considered as scalable by design. However, they depend on effective and consistent data dissemination and the data attribute selected for correlation may affect load distribution among participants [ZLK10].

The flow of information in centralized and hierarchical architectures is governed by the logical arrangement of components and their specific role. In centralized architectures, there is usually a central entity that controls communication. In hierarchical systems, the communication paths are mainly governed by the topological structure. In contrast to that, different techniques exist in distributed approaches. Whereas flooding refers to unfiltered data distribution to all members, selective techniques reduce the overhead by

using, e.g., random walks [VF06], gossipping approaches [Das+06][GKM03] or publish-subscribe mechanisms. The latter provide flexible organization options and guarantee delivery. For example, subscriptions utilized in [JWQ03] are based on special attack forms.

The data dissemination strategy of a CIDS not only defines the communication paths as described above, but also influences what kind of data is exchanged between the CIDS members while also specifying format and level of granularity. This also includes the central aspects of *data privacy*, realized by utilization of, e.g., bloom filters [Vas+15][Loc+05], and *interoperability* that can be obtained by standardized formats, such as the Intrusion Detection Message Exchange Format [CM02][Dum+06].

In particular, there are two approaches to mention, that directly address the problems of communication overhead an privacy in this context. In [Loc+05], the authors present an approach for the efficient and privacy aware distribution of alert data in a distributed CIDS. Before exchanging information, the respective data is compressed by the utilization of a bloom filter data structure. Upon an alert, the relevant information, e.g. IP address, is inserted ino the bloom filter. Since the bloom filter contains information on suspicious hosts, it is called *watchlist*. The watchlist is shared among peers, which are selected by a network scheduling algorithm called *whirlpool*, that dynamically creates an overlay that defines peer relationships.

The authors state, that within data dissemination, there two challenges. First, the disclosure of sensitive data, e.g. IP addresses, to collaborative entities renders participation in the collaboration not an option. Second, the tradeoff between latency and accuracy of the information exchange and the required bandwidth. Centralized approaches disseminate information reliable and predictable, but they constitute a bottleneck. While distributed approaches scale well, information can be lost or delayed by partitioning the data among the peers.

In the context of reducing communication overhead, this approach addresses this challenge twofold. First, alert data is compressed when inserted into the bloom filter by hashing. Second, only peers exchange data, which reduces overhead significantly, when compared to a complete distribution. However, bloom filters are probabilistic data structure, which exhibit increasing false positive matches with increasing filling degree. Thus, when scaling this approach, the probability that innocent hosts are considered as malicious, increases.

Also, the authors in [Vas+15] state, that a CIDS needs to provide *scalability*, minimal message *overhead*, *privacy* of the exchanged alert data, an *domain awareness*, which describes the ability to constrain alert disseminatation to specific sub-domains of a network. Instead of randomly creating sub-domains as in [Loc+05], the authors suggest to incorporate network traffic similarities into account for this process.

data dissemination: extract specific features from alerts and add data into bloom filter; then utilize data disseminatation technique (e.g. flooding, partial flooding, gossipping) for sending bloom filters to other nodes (peers)

similarity (alert) correlation: when nodes receive data from other nodes, they compute their similarity value by performing logical operations on the bloom filters

similarity (alert) correlation: when nodes receive data from other nodes, they compute

their similarity value by performing logical operations on the bloom filters, after calculating the similarity value, nodes will make use of a threshold value t to determine wheteher the similarity value is enough for joining a group (community creation)

Community formation: After the successful dissemination and correlation of the alert data, each sensor creates a matrix with its local knowledge of other sensors. Based on this knowledge and along with the utilized threshold, sensors can identify others and form a community with them to, afterwards, exchange more fine-grained alert data.

The problem of finding an optimal threshold value (golden standard) heavily depends on the network that is to be monitored.

To sum up, the following challenges in the context of data dissemination are found to be critical for the success of the overall system.

Minimal Overhead Detection latency can be affected by various factors, some of which are encountered in conventional IDS and others that are specifically relevant in the CIDS context. With regards to the data dissemination in CIDS, the computational overhead introduced by the communication of multiple monitors in the CIDS needs to be minimized, such that potential knowledge gains are available fast enough.

Privacy Members may not want to disclose data that contains information on system- and network states of their infrastructure, as it constitutes a privacy and security problem. This includes, among other things, legal aspects when it comes to sharing log and network data. Nonetheless, the exchange of this information is crucial for the effective operation of a CIDS.

Interoperability The individual components of the overall system, which were deployed in different system and network environments, should be able to interact with each other in the context of the CIDS. In addition to system-wide standards for data collection, processing and exchange, there exists a trade-off between interoperability and privacy.

Domain Awareness t.b.d. (a feature that increases scalability by reducing overhead and contributes to privacy by partially constraining communication; also increases accuracy, because only those alerts are shared that are relevant for w.r.t. to similarity of data etc.)

3.2 Similarity Hashing in Malware Detection

Searching for similar objects in large data sets is a fundamental challenge that has found important applications in many fields. An important subclass of similarity search is the nearest neighbour search problem, which becomes hard in the high dimensional case, when relying on exact algorithms like a linear scan ($O(n)$) as the best solution. However, many applications do not require an exact solution, such that in these cases a randomized algorithm can be used, which provides the correct solution with high probability in sublinear time [Dat+04]. Therefore, approximate and randomized solution algorithms are widely used in practice. There is popular approach known as locality-sensitive hashing (LSH) that allows searching in large databases for similar items in a randomized manner. This technique hashes similar input onto the same hash code with high

probability for the purpose of populating a hash table. Since similar items are placed into the same buckets, this approach is suitable for data clustering or nearest neighbour search.

Also the field of cyber security has adopted methods suitable for similarity search, mainly for the analysis of polymorphic malware. This type of malware poses a significant challenge, since it changes its appearance over time in order to stay undetectable from antivirus software [WM18, p.91]. Traditional antivirus software uses cryptographic hash functions (SHA-256) to create file signatures. Such signatures are well suited to search for identical files in a knowledge database. However, due to the property of cryptographic diffusion, even minimal changes to the malware result in large differences in the resulting hash value. With the rapid evolution and proliferation of polymorphic malware, detection based on unique signatures no longer seems effective.

Based on these developments, detection schemes based on approximate matching algorithms have initially become the focus of research. In contrast to LSH, approximate matching functions are designed for producing digests of objects and a subsequent comparison of such digests, which yields a confidence value reflecting the similarity of two objects [MH17]. Popular approaches in this area are for example *ssdeep*, which implements context triggered piecewise hashing (CTPH) [Kor06] or *sdhash*, that makes use of bloom filters for the digest creation and comparison [Rou10].

3.3 Generative Algorithms and Intrusion Detection

Chapter 4

Generative Pattern Database

Three main challenges in the context of data dissemination in CIDS were identified. First, intrusion related data is usually of sensitive nature. Thus, the exchange mechanism must not compromise any policies and regulations related to data *privacy*. At the same time, the usability of the data has to be preserved. Second, the data that is subject of the exchange may exhibit large volumes. That constitutes a challenge, since the dissemination is desired to be executed with *minimal overhead* in a timely and scalable fashion. Lastly, the *interoperability* of the CIDS with existing local IDS is an important aspect that influences the practical adoption into security architectures.

In summary, existing approaches for data dissemination mainly provide mechanisms for exchanging alert data or single attributes, e.g. IP addresses, as they focus on the correlation of intrusion detection incidents that originate from different sensors. The exchange of actual training data is neglected, possibly due to high data volumes. Thus, these systems lack of mechanisms for the extraction and global persistence of novel attack patterns, e.g. from zero day exploits, that can be used for the training of an intrusion detection sensor.

The approach that is presented in this chapter exchanges attack patterns by sharing generative machine learning models that have been trained on partitions of similar data points. Such a model-based dissemination enables the receiving side to sample a synthetic dataset that enhances existing local datasets. This provides two main advantages. First, no original data leaves a local network and therefore does not violate any privacy restrictions. Second, the data is compressed considerably by representing it in form of a generative model. In order to make that mechanism scalable, the monitored data is clustered using random projections. This way, similar data points are partitioned into globally common clusters, which is exploited as a data parallelism mechanism. Given that, bursty workloads can be served effectively in a cloud deployment. Furthermore, this mechanism enables a similarity-based correlation of distributed intrusion events. The integration of both a similarity based correlation of intrusion incidents and a mechanism for sharing attack knowledge makes it possible to extract novel patterns of distributed attacks and provide them globally within the CIDS, resulting in an improved attack detection.

While Section 4.2 gives a high-level overview of the proposed architecture and its main processing primitives, details on the specific algorithms and strategies of the main

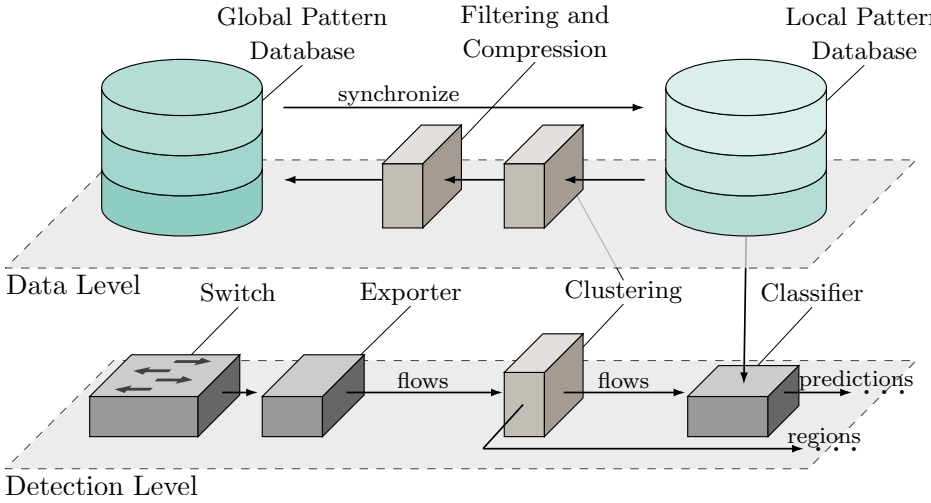


Figure 4.1: Integration of the approach into a generic NIDS.

services are described in Sections 4.3 to 4.6.

4.1 High Level Overview

Several members exist in the CIDS, each of which manages an isolated IDS. Every IDS operates according to a specific set of rules, that is essentially based on the content of a local database. The goal of the generative pattern database is to close the knowledge gaps of local databases and thus increase the detection rate of associated IDSs. By providing a *global view* on all local databases, individual IDSs can benefit from the collective knowledge of the CIDS.

Section 4.1.1 starts with a reference example to illustrate the idea that is described above and discusses the integration of the CIDS into existing infrastructures. Subsequently, Section 4.1.2 and Section 4.1.3 show the key concepts that enable data distribution and correlation under the given requirements. Finally, Section 4.1.4 combines the individual elements to present the strategy for the creation and usage of the global view.

4.1.1 Example Integration

An exemplary integration of the CIDS into a generic NIDS is shown in Figure 4.1. The shown NIDS consists of three components, which can be found on the detection layer. A flow exporter computes statistical flow features based on the network packets of a switch. A discriminative model serves as a classifier that operates on the specific feature set that the exporter extracts. After completing the training of a classifier instance on a given dataset, it is deployed within the detection pipeline. There, the classifier receives a stream of network flows and predicts them.

The CIDS mainly integrates at the data level, where the training data for the classifier is provided by the *local pattern database*. At this point, the local pattern database only contains the *local view* of the intrusion detection data from that particular member. In order to provide a global view for that member, a synchronization process between its local pattern database and the *global pattern database* has to be initiated. Before the data is transferred to the global pattern database, is subject to a set of clustering,

filtering and compression operations. This way, data privacy is maintained and the overhead is minimized by reducing the data volume. On the receiving side, the global pattern database combines data from all members to a global view.

Upon each update of the global pattern database, the new state of the global view is synchronized to each local pattern database and subsequently enhances the classifier on the detection level by extending the local intrusion detection dataset. Furthermore, an identical clustering operation as on the data level is applied on the detection level. Each incoming flow is assigned to a certain cluster, which is referred to as *region*. While the predictions from the classifier are suitable for detecting attacks that are known to the CIDS, regions are leveraged for the detection of novel data patterns and similarity-based correlations that uncover stealthy attacks, which are executed on the resources of multiple CIDS members simultanously.

4.1.2 Clustering

Since the results of the clustering operation should be consistent, while its execution is distributed among all members of the CIDS, an unsupervised algorithm with few parameters for initialization has to be selected. Additionally, the algorithm should be scalable, since it is to be applied on whole databases on the data level and on streams of live data on the detection level. Given these requirements, random projection is a good choice.

By using random projection, real data points are clustered according to their angular distance. Furthermore, the projection result is a binary string, which can be used for storing similar data points into a common bucket of a hash table by using the binary string as an index. In the context of the generative pattern database, the combination of random projections and hash tables is exploited as the main controlling primitive for data persistence and retrieval. Instead of using randomly selected projection planes, a shared seed results in the application of a common projection function among all members of the collaboration. In other words, similar data points from different datasets are indexed to a common global bucket, i.e. region. Each region is subject to the transformations individually, which is utilized as a data parallelism mechanism. Thus, this approach is natively suited for cloud deployments where bursty workloads can be served effectively. Lastly, this mechanism enables a similarity-based correlation of distributed intrusion events. As incoming data is monitored on the detection level, the clustering is applied, which results in a pattern that can be used for novelty checks or global occurrences within the CIDS.

4.1.3 Filtering and Compression

Two types of data are extracted within individual regions. First, metadata of local datasets is collected by counting label occurrences, which serve as indicator for determining if the respective region needs to be subject to the second extraction type. Second, models that are trained with generative algorithms on local attack data are the exchange medium for disseminating information within the CIDS. This provides two main advantages. For one, no original data leaves a local network and therefore does not violate any privacy restrictions. For another, the data is compressed considerably by representing it in form of a generative model.

4.1.4 Global View

4.1.5 Creation

As shown in Figure 4.2, each view is partitioned by a common projection function into an identical set of regions. Each local view contains original datapoints from its respective dataset. In this example, there exist three different classes globally, which occur differently in each local dataset. Examining a specific region, the combination of unique classes within all local views determines its complexity on a global level. If a region contains more than one class, it potentially exhibits a non-linear decision boundary, hence it is called complex. Otherwise, a region is called simple. Attack data within a complex region, seperated by its label, is used as training data for a generative algorithm. Subsequently, the resulting model is transferred to the global view.

4.1.6 Usage

A synchronization process disseminates the region complexity estimations and the generative models to all local views. By sampling data from multiple generative models, a synthetic dataset is assembled, which is blended into the respective local datasets for enhancing the subsequent training of a discriminative model.

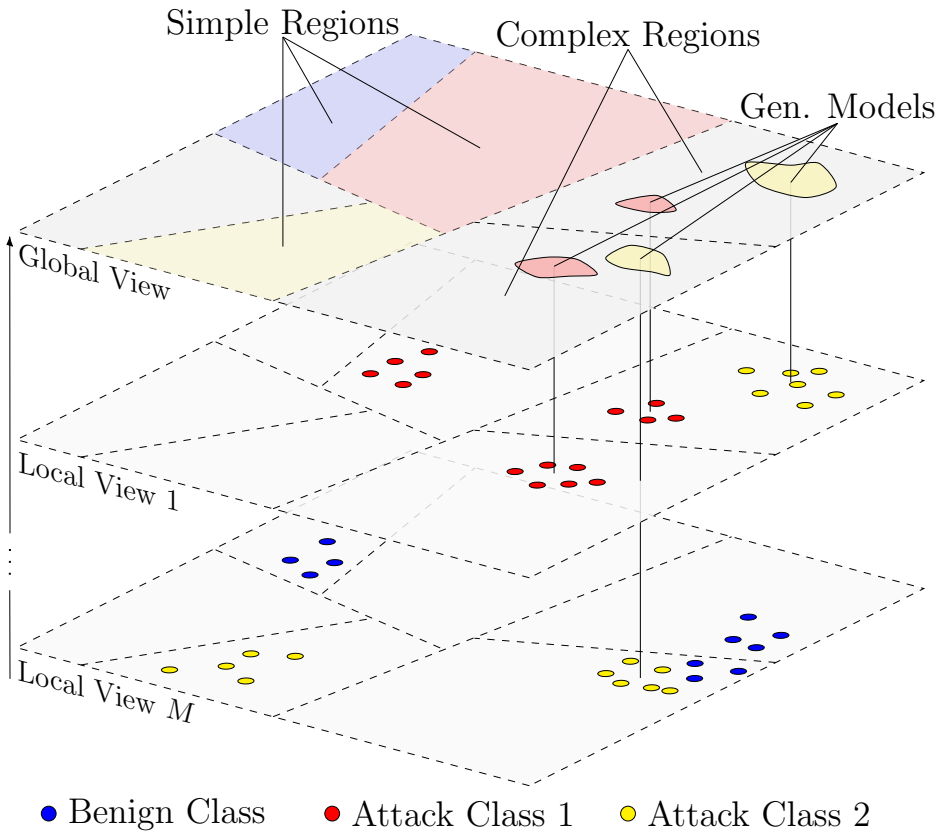


Figure 4.2: Building the global view by combining M local views.

4.2 System Architecture

Logically, the proposed CIDS exhibits a hierarchical architecture (see Figure 4.1). For one, the global infrastructure G represents the collection of $M \in \mathbb{N}$ CIDS participants and their knowledge on an abstract level. For another, it provides specific services, that are globally available to each local infrastructure $L_m, m \in \{1, \ldots, M\}$ that includes all CIDS components and processes within the IT infrastructure boundaries of a corresponding CIDS member. Each local infrastructure L_m agrees to a specified feature extraction process that provides the monitoring data $\boldsymbol{X} \subset \mathbb{R}^d$ for the attack detection. Single data points of the monitoring data are referred to as $\boldsymbol{x} \in \boldsymbol{X}$ with a total number of features $d = |\boldsymbol{x}| \in \mathbb{N}$. In addition, the set of targets $Y \subset \mathbb{N}$ with instances $y \in Y$ is known and registered by every local infrastructure L_m. Furthermore, every L_m provides an individual training dataset $D_m = \{(\boldsymbol{x}_n, y_n) : 1 \leq n \leq N_m\}$ of size $N_m = |D_m| \in \mathbb{N}$. CIDS communication across local boundaries occurs exclusively in a vertical direction. Thus, the exchange of information between individual L_m takes place indirectly via the global pattern database (PDB_G) and the global event channel (C_G). Each L_m includes a local pattern database (PDB_{L_m}), a local event channel (C_{L_m}) and an event-based data processing pipeline that consists of four services.

4.2.1 Pattern Database

depending on the scope (either local or global), different tasks are considered local pdbs store intrusion detection datasets and corresponding metadata of the respective member; global pdbs store global metadata (combined information of local datasets) and the generative models

Each instance of a pattern database (PDB) is realized as a key-value store. For the following algorithm descriptions, a PDB is treated as a hash table as defined in Section **??**, that is referred to by replacing its function variable with the identifier of the corresponding database, e.g. $PDB_G(k)$ being a specific slot or hash value related to a

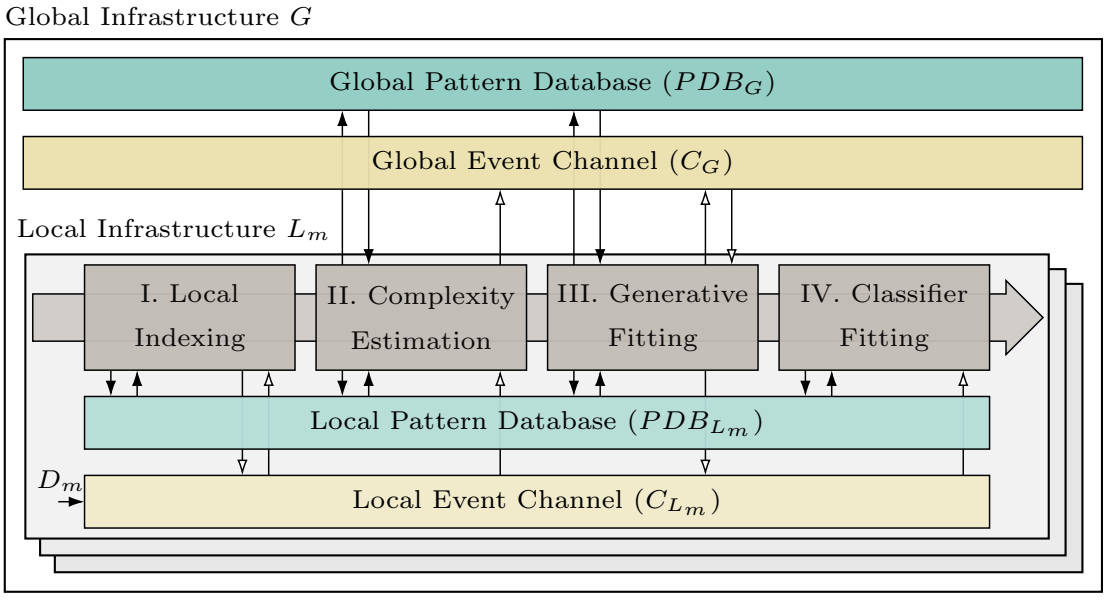


Figure 4.3: High level CIDS architecture.

key k in the global pattern database. Note that if a specific hash function is already used to construct a key (e.g. Random Projection), the hash table will internally apply a distinct hash function on the key to ensure even distribution across the slots.

4.2.2 Event Channel

Event channels provide a topic-based publish-subscribe messaging mechanism that is mainly used to distribute workloads among the service instances in the processing pipeline. Via the messaging system, service instances receive and emit events, on which upon the respective operations are triggered. Changes in a pattern database result in responses that in turn are leveraged as the respective events. In this fashion, updates are propagated throughout the processing pipeline, ensuring a timely consistency among the pattern databases.

4.2.3 Processing Pipeline

Local and global pattern databases serve exclusively as data sources and sinks for operations. The only exception is the initial import of datasets D_m into the *Local Indexing* service via the messaging system.

Notation	**Description**
G	Global Infrastructure
L_m	Local Infrastructure m
PDB_G, PDB_{L_m}	Global Pattern Database, Local Pattern Database of L_m
C_G, C_{L_m}	Global Event Channel, Local Event Channel of L_m
$M \in \mathbb{N}$	Total number of CIDS participants
$m \in \{1, \ldots, M\}$	Local Infrastructure Identifier

Table 4.1: Summary of the architecture notation.

4.3 Local Indexing

The local indexing service is responsible for the preprocessing and local storage of intrusion detection datasets. As already described in Section 4.1.2, the data is organized in regions. This means that individual data points are first assigned to a region using a locality-sensitive hash function. Based on the generated hash value, a key is constructed that is used to persist the data point in the respective local pattern database. According to the properties of a locality-sensitive hash function, similar data points are assigned to a common region and thus form a closed processing unit for subsequent operation steps. If a region is formed or an update is made to an existing region, e.g., due to the occurrence of new data, events are emitted to inform the subsequent service.

Algorithm 2 Preprocessing and inserting $B \subset D_m$ into PDB_{L_m}

Input: Pairs of datapoints and labels $B \leftarrow [(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_b, y_b)]$
Output: Regions R

$R \leftarrow$ new Set
for each $(\boldsymbol{x}, y)$ in B **do**
    $\boldsymbol{x}' \leftarrow$ normalize($\boldsymbol{x}$) // see Equation 4.1
    $r \leftarrow h(\boldsymbol{x}')$
    $k_\alpha \leftarrow$ concatenate(p_x, r, y)
    $k_\beta \leftarrow g(\boldsymbol{x}')$
    $k_\gamma \leftarrow$ concatenate(p_y, r)
    if $PDB_{L_m}[k_\alpha]$ is None **then**
        $PDB_{L_m}[k_\alpha] \leftarrow$ new Hashtable H
    if $PDB_{L_m}[k_\gamma]$ is None **then**
        $PDB_{L_m}[k_\gamma] \leftarrow$ new Set S
    if $PDB_{L_m}[k_\alpha][k_\beta]$ is None **then**
        $PDB_{L_m}[k_\alpha][k_\beta] \leftarrow (\boldsymbol{x}', y)$
        insert y into Set at $PDB_{L_m}[k_\gamma]$
        insert r into R
return R

First, an intrusion detection dataset D_m is sent to one or more service processors via the C_{L_m}. Second, the incoming stream of pairs of data points and labels $(\boldsymbol{x}_n, y_n) \in D_m$ is ingested and buffered until a batch B has been accumulated. Then, the batch is preprocessed and inserted into the PDB_{L_m} as described in Algorithm 2. In words, the datapoint $\boldsymbol{x}$ is scaled to the range $[-1, 1]$ by applying a feature-wise min-max normalization, given by

$$\boldsymbol{x}' = \frac{\boldsymbol{x} - min(\boldsymbol{X}_B)}{max(\boldsymbol{X}_B) - min(\boldsymbol{X}_B)} \cdot (b - a) + a, \tag{4.1}$$

where $a = -1$, $b = 1$ and $\boldsymbol{X}_B$ is the set of data points within the batch B. After that, the scaled data point $\boldsymbol{x}'$ is subject to both a locality-sensitive hashing function h and a non-cryptographic hashing function g. In this particular architecture, h is a gaussian random projection with a global seed for the initialization of the projection plane $\boldsymbol{M}$ (see Section 2.3.3), such that regions $r = h(\boldsymbol{x}')$ across local infrastructures are comparable.

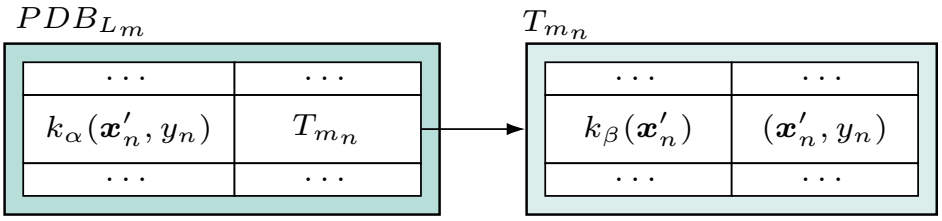


Figure 4.4: Nested indexing in a local pattern database.

Since the data is organized in regions, a nested scheme is applied for the insertion of pairs of datapoints and labels as depicted in Figure 4.4. In fact, the pairs within a region are further partitioned into disjoint subsets according to the label y. This means that for each subset of the data of a particular label within a region, a separate hashtable is initialized and inserted into the PDB_{L_m} as a second level.

Thus, for the persistence of a pair $(\boldsymbol{x}', y)$, two keys $k_\alpha, k_\beta \in K$ are constructed (see Lines 5-6 in Algorithm 2). The key k_α is a concatenation of the prefix constant p_x, the bit-string r and the label y. This way, the data is partitioned primarily by its region and secondarily by its label as described above. The key k_β is the result of the non-cryptographic hashing function $g(\boldsymbol{x}')$, which serves as a mechanism for deduplicating identical $\boldsymbol{x}'$.

Additionally, a third key k_γ is constructed by concatenating the prefix constant p_y and the bit-string r. As it is important to retrieve all existing labels within a region efficiently in a processing step of the subsequent service, k_γ is used for storing the set of labels within a region as auxiliary metadata.

Next, if not already present, a hash table is initialized and inserted into the slot $PDB_{L_m}[k_\alpha]$. Likewise, if the slot $PDB_{L_m}[k_\gamma]$ is empty, a new set[1] is initialited and inserted. After that, it is checked if the slot $H[k_\beta]$, which in turn is placed in $PDB_{L_m}[k_\alpha]$, is empty. In the positive case, the pair $(\boldsymbol{x}', y)$ is inserted into that slot and the region r is inserted into the set R. Otherwise, no data is inserted into the local pattern database and no region is added to R. After processing a batch, the set of updated regions R is emitted as events into C_{L_m}.

Since data duplicates are filtered before the insert operation in this algorithm, events are also not emitted unnecessarily multiple times if, for example, the same dataset is sent to the service repeatedly. In other words, the inserts are idempotent, which is an important property in this architecture. Since there are operations in subsequent services that are relatively computationally intensive, emitting events is expensive. For this reason, such a streaming application might in practice implement buffers at regular intervals to collect and aggregate the events of several successive batches.

[1] A set describes a data structure which is essentially an unordered collection with no duplicate elements.

4.4 Complexity Estimation

A region is said to be complex, if it contains more than one unique label. Otherwise, a region is simple. Since the data within a region already represents a cluster, the existence of multiple classes indicates a more complex decision boundary. On that basis we differentiate how a region is processed in the subsequent services of the pipeline. Furthermore, the complexity state of a region may vary, depending on the scope it is observed. Note that since the projection matrix $\boldsymbol{M}$ is initialized with the same values in every L_m, all hashes that were computed by h are globally comparable. This means that similar data points from different datasets, e.g. $\boldsymbol{x}_i \in D_1$ and $\boldsymbol{x}_j^* \in D_2$ may be hashed to the same region $h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j^*)$. However, it is also possible that that the corresponding labels $y_i \in D_1$ and $y_j^* \in D_2$ are not equal and therefore lead to a different global view on that region's complexity state. Given that, the complexity estimation module acts as a bridge between the local and global components and answers the question, which regions are considered to be complex in a global context.

Algorithm 3 Creating a global complexity state by combining local complexity states

Input: Regions R_{in}
Output: Regions R_{out}

1: $R_{\text{out}} \leftarrow$ new Set
2: $m \leftarrow$ getID() // current local infrastructure identifier
3: **for each** r in R **do**
4: $k_\gamma \leftarrow$ concatenate(p_y, r)
5: $Y_r \leftarrow PDB_{L_m}[k_\gamma]$
6: $k_\delta \leftarrow$ concatenate(p_y, r, m)
7: $PDB_G[k_\delta] \leftarrow Y_r$
8: $S \leftarrow$ new Set
9: **for each** m in $\{1, \ldots, M\}$ **do**
10: $k_\delta \leftarrow$ concatenate(p_y, r, m)
11: $Y_r \leftarrow PDB_G[k_\delta]$
12: insert Y_r into S
13: **if** $|S| > 1$ **then**
14: $c_r \leftarrow 1$
15: **else**
16: $c_r \leftarrow 0$
17: $k_\kappa \leftarrow$ concatenate(p_c, r)
18: **if** $PDB_G[k_\kappa] \neq c_r$ **then**
19: $PDB_G[k_\kappa] \leftarrow c_r$
20: insert r into R_{out}
21: **return** R_{out}

First, a set of regions R is received. Then, for each region $r \in R$ the following operations are defined. The set of unique labels Y_r for a particular region has been stored in Algorithm 2 as auxiliary metadata, which is now retrieved by constructing the correspoding key k_γ. Subsequently, the slot $PDB_{L_m}[k_\gamma]$ is accessed and Y_r is retrieved. In the next step, Y_r has to be stored in the PDB_G. Therefore, another key k_δ is constructed by concatenating the prefix constant p_y, the region r and the current local infrastructure identifier m, which prevents the collision of information from different infrastructures.

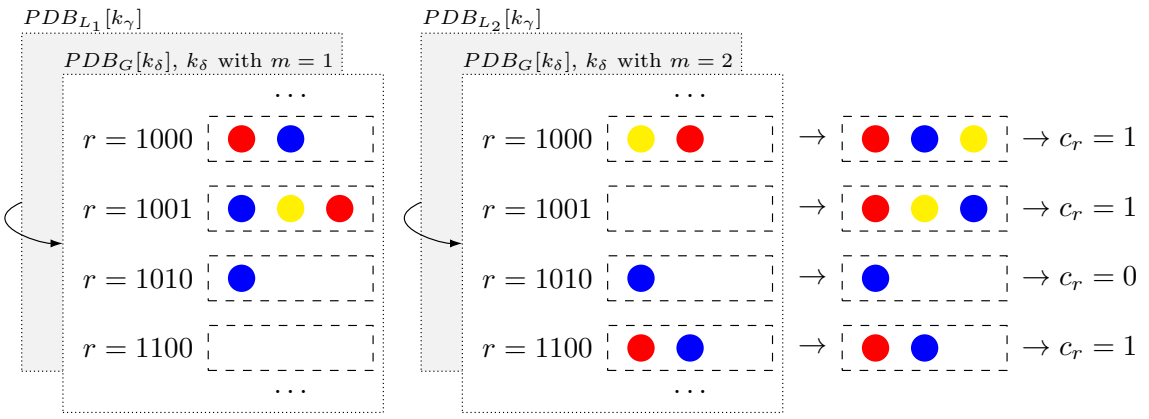


Figure 4.5: Illustration of the complexity estimation algorithm with two local infrastructures. Each local label set at $PDB_{L_m}[k_\kappa]$ is synchronized into the global pattern database at $PDB_G[k\delta]$, where k_δ is constructed with m. Per region, the union operation is applied on the global label sets, whereupon the complexity is derived from.

After Y_r is stored on a global level at $PDB_G[k_\delta]$, the label set information for that region from all members in the CIDS is aggregated. That aggregated view is essentially the global complexity state for that region. By iterating over all member identifiers in the CIDS, multiple keys k_δ are constructed. Each key retrieves the specific label set Y_r of a member and inserts its content into the set S.

After collecting all label sets in S, the complexity state is obtained by simply evaluating the cardinality $|S|$. If there is more than one class in a region on a global scope, that is $|S| > 1$, then assign a true value to the global complexity variable c_r. Otherwise, assign a false value. In order to store c_r, the key k_κ is created by concatenating the prefix constant p_c and the region r. Note, that $PDB_G[k_\kappa]$ is only updated, if storing c_r changes the state that is already persisted. This is because, if an update is executed, this information has to be propagated to the next service. Thus, in that case, the region r is inserted into R_{out}, which is subsequently sent into the global event channel C_G in order to inform services in all local infrastructures about the update.

4.5 Generative Fitting

The generative fitting service is the most demanding procedure in the context of processing resources. The service represents the filtering and compression operations presented in Section 4.1. There are two scenarios based on the region's complexity. If the region is not complex, no further actions are taken except it formally was complex. Then, existing models have to be deleted, since they are not longer used. And if the region is complex, generative models are provided, which represent the exchange medium for information. More specifically, multiple *Gaussian Mixture Models* (GMMs) are fitted on each label-subset of a region's data, which was stored in the indexing step in Section 4.3. According to a model selection process that evaluates the efficacy of each GMM, the best model is stored in the global pattern database, accessible to every member in the CIDS. The purpose of that elaborate process is that synthetic data can be sampled from these models. This way, every member has access to the global knowledge from all local infrastructures in order to enhance the local dataset that is used for fitting a classifier. Thus, this service is the key for providing *privacy* and *minimal overhead* while exchanging information.

Algorithm 4 Retrieve Dataset from Region (Main Procedure)

Input: Regions R_{in}
Output: Regions R_{out}

```
1:  m ← getID()
2:  for each r in R_in do
3:      k_κ ← concatenate(p_c, r)
4:      k_δ ← concatenate(p_y, r, m)
5:      c_r ← PDB_G[k_κ]
6:      Y_r ← PDB_G[k_δ]
7:      if c_r = 0 then
8:          for each y in Y_r do
9:              k_ω ← concatenate(p_d, r, y, m)
10:             delete model in PDB_G[k_ω]
11:     else
12:         L ← new List
13:         for each y in Y_r do
14:             k_α ← concatenate(p_x, r, y)
15:             H ← PDB_Lm[k_α]
16:             append H to L
17:         D ← preprocessing(L)
18:         for each y in Y_r do
19:             GMM ← modelSelection(D, y)
20:             k_ω ← concatenate(p_d, r, y, m)
21:             PDB_G[k_ω] ← GMM
```

First, regions that have been updated are received as events. As the data is further organized per label within a region, the labels for a region are retrieved. Then, if the region is not complex, no model fitting is executed. Instead, potentially existing models are deleted from storage. This is the case, if the complexity status of the has been changed from complex to simple.

But if the region is complex, the generative model fitting procedure is triggered. Even if there are already models for the corresponding combination of region and label, an update of these models is initialized. For that, every hash table within a region, each containing data with a common label, is collected and added to a list. Subsequently, preprocessing operations prepare the collected region data for the model fitting. Details on the data preparation are described in Algorithm 5.

After that, the Model Selection process is started sequentially for each available label in the region. That way, the models are only fitted on data with the label in focus but evaluated with the complete region data. Specifics on the model selection are elaborated in Algorithm 7. Finally, the best fitted model is stored in the global pattern database. So far, the main procedure has been outlined. Next, the details on the data preprocessing and the model selection are elaborated.

Algorithm 5 Preprocess Data

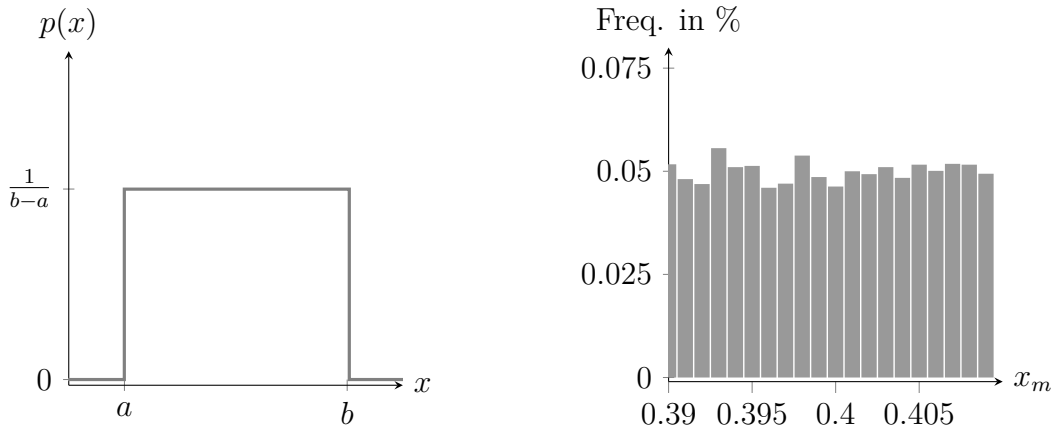
Input: List of HashMaps L
Output: Dataset D

```
D ← getValues(L)
for each label in D do
    if label ≠ 0 then
        // split into binary
        (X_a, y_a) ← (X, y) where y = label
        (X_b, y_b) ← (X, y) where y ≠ label
        if X.shape[0] < X.shape[1] then
            X_a, y_a ← upsample(X_a, y_a)
return (X_a, y_a), (X_b, y_b)
```

Starting with the preprocessing.

Splitting the dataset into a binary problem, such that the label that is currently in focus is the normal class and all other classes are attack classes. That way, the generative model can be evaluated using the machine learning efficacy method in the later course.

The upsampling process ensures that the fitting algorithm for the GMM is able to work. In practice, the number of samples has to be at least equal to the number of components. As some label subsets of a region may exhibit a low number of samples, in extreme cases only a single sample, an upsampling process is implemented. In particular, a set of nearest neighbours is generated per sample. First, the number of samples to generate is determined by the difference of the dimensionality of the data and the number of data points. Then, in order to generate from each data point equivalently, the number of nearest neighbours to sample is determined per data points. That is, the complete number of points to resample divided by the number of samples with an interger division (resulting in an integer). In case, the result of the interger division is zero, one is added to the result. Then for each data point x in the set X, nearest neighbours are generated by sampling from a uniform distribution. This is done by considering each feature value of x individually, such that the nearest neighbour is the concatenation of the sampled feature values as $x^* = [p(x_1), p(x_2), \ldots, p(x_M)]$, where $p(x_m) = \frac{1}{b-a}$ within the interval $[x_m - \delta, x_m + \delta)$. The value δ controls the interval of the uniform distribution. The larger the value for δ, the further the newly generated values deviate from the orginal feature values. In Figure 4.6b the value for a single feature x_m



(a) The pdf of a uniform distribution is $p(x) = \frac{1}{b-a}$ within the interval $[a, b)$, and zero elsewhere.

(b) Histogram of 1 000 samples drawn uniformly over the interval $[x - \delta, x + \delta)$ where $x = 0.4$ and $\delta = 1 \cdot 10^{-2}$.

Figure 4.6: Nearest neighbours are generated by sampling each feature value from a the uniform distribution.

is drawn uniformly at random. The original value of the feature was $x_m = 4$, which was extended by $\delta = 1 \cdot 10^{-2}$. By choosing a relatively small value for δ, it is ensured that the generated nearest neighbours do not alter the original data distribution significantly, while enabling the subsequent fitting of the GMM for that set of data points.

Algorithm 6 Upsampling

Input: Collection of datapoints X
Output: Upsampled collection of datapoints X_{up}

```
nResample ← X.shape[1] − X.shape[0] // Difference of dim(X) and num(X) to fill
nResamplePerX ← nResample //X.shape[0] + 1
X* ← ∅
for each x in X do
    x* ← new Array
    for each x_m in x do
        sample a new x*_m from p(x_m) and insert into x*
    add x* to X*
X_up ← X ∪ X*
return X_up
```

In the next phase of the algorithm, one or more GMMS are selected for a region's data, depending on the number of unique labels within. For each label subset, multiple models are fitted within a selection process. Since the resource demands of the EM algorithm for fitting a GMM are relatively complex, the data's dimensionality is reduced by applying a principal component analysis. Later in the course, when sampling data, the inverse operation using the same PCA parameters is applied on the synthetic data and bring it back into the original dimenions. Therefore, for each label subset of a region, both the parameters of a GMM and a PCA model is stored in the global pattern database.

Apply PCA on X_a, such that 99.9% of the variance of the data is preserved, then fit multiple GMM with the same data but with different parameters by the following rules; collect a list of different parameters for the number of components $C = \{2k + 1 : k \in \mathbb{N}, 1 \leq k \leq K\}$ with $K = \lfloor M/2 \rfloor$, the different number of components is heuristically determined; in general, there is no exact method to determine the optimal number of components for a given dataset before fitting the model; thus, different parameters have to be tried out in a model selection method; Moreover, as stated in Section (section of GMM), the runtime of a single step of the EM algorithm in this setting is asymptotically $O(NKd^3)$ or $O(NKd^2)$ by using the incremental algorithm proposed in [PE15] (N data points, K components and d dimensions); therefore, this curse of dimensionality that is encountered in calculating the covariance matrix while fitting the GMM is coped by approximation; using the diagonal covariance matrix as approximation to the regular covariance matrix; moreover, the full covariance matrix can result in overfitting, especially on small datasets, whereas the diagonal approximation acts as a type of regularization to the model. Since the focus of this algorithm is mainly focused on providing the best model, the runtime is treated as a secondary factor; thus, both types of covariances are used within the model selection process cov= {"full", "diagonal"}.

After fitting a model on $X'_a = \text{PCA}(X_a)$, the first metric for the selection is calculated. Precisely, the bayesian information criterion (BIC) is calculated as in Equation(X). Note, that for the case of a diagonal covariance matrix, the number of parameters to estimate change from $\frac{d(d+1)}{2}$ to d, such that the BIC is given as

$$\text{BIC}(M|D) = (Kd + d + K - 1)\ln(N) - 2\ln(\hat{L}). \tag{4.2}$$

Algorithm 7 Model Selection

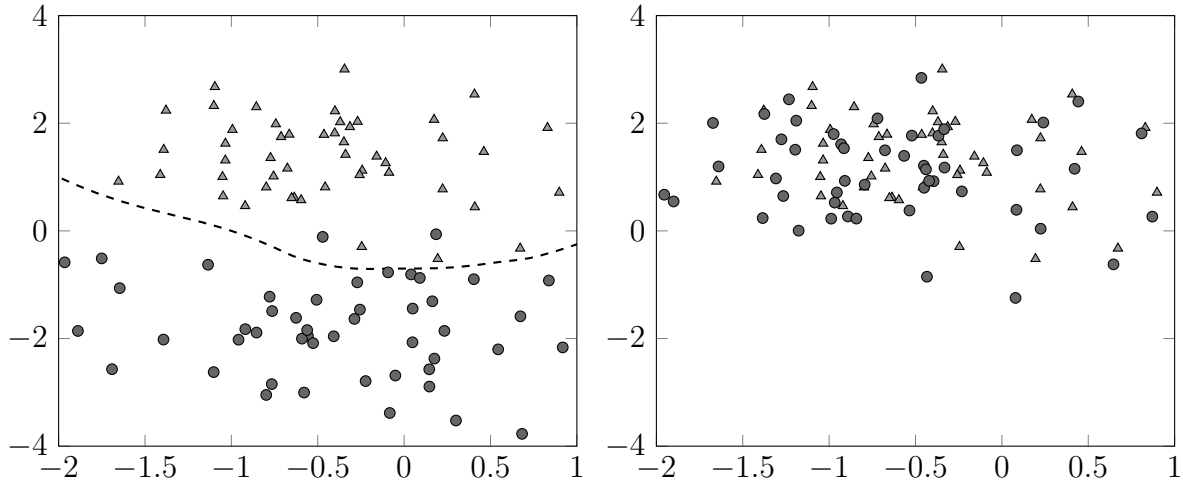
Input: Dataset D with X_a, y_a, X_b, y_b
Output: Tuple (GMM, PCA) containing model parameters

$X'_a \leftarrow \text{PCA}(X_a)$
$C \leftarrow \{2k + 1 : k \in \mathbb{N}, 1 \leq k \leq K\}$ with $C = \lfloor M/2 \rfloor$
$B \leftarrow$ new Array
COV $\leftarrow$ {"full", "diagonal"}
for each k in C **do**
    for each cov in COV **do**
        GMM $\leftarrow$ fitGMM(X'_a)
        GMMs[kC] $\leftarrow$ GMM
        $b_{kC} \leftarrow \text{BIC}(\text{GMM}|X'_a)$
        acc $\leftarrow$ AdversaryEvaluation(X, y, GMM, PCA)
        $B[kC] \leftarrow (b_{kC}, \text{acc})$
sort B by b_{kC} in $B[0]$
sort B by acc in $B[1]$
$b_{kC} \leftarrow B[-1]$
return (GMMs[kC], PCA[kC])

Algorithm 8 Adversary Evaluation

Input: $X_r, y_r, \text{GMM}, \text{PCA}$
Output: acc from DecisionTree Model DT

$X'_s \leftarrow$ sample $|X_r|$ data points from GMM
$X_s \leftarrow \text{PCA}(X'_s)^{-1}$
$y_s \leftarrow \{1\}$ // as many 0s as the number of X_b
$y_r \leftarrow 0$ // setting all real labels to 0
$X, y \leftarrow \text{concatenate}(X_s, X_r, y_s, y_r)$
$X_{\text{train}}, y_{\text{train}}, X_{\text{test}}, y_{\text{test}} \leftarrow \text{shuffleSplit}(X, y)$
$\text{DT} \leftarrow \text{fitDecisionTree}(X_{\text{train}}, y_{\text{train}})$
$\tilde{y} = \{\text{DT}(x_1), \ldots, \text{DT}(x_n)\} \leftarrow$ predict X_{test} with DT
$\text{F1} \leftarrow \text{f1}(y_{\text{test}}, \tilde{y})$
return F1



(a) A poor fitted generative model: The original data (triangles) and the synthetic data (circles) can be easily separated by a classifier.

(b) A well fitted generative model: The synthetic data resemble the original data well, such that a separation is not easy.

Figure 4.7: Adversary evaluation: separation of original and synthetic data.

4.6 Classifier Fitting

Chapter 5

Experimental Evaluation

Chapter 6

Conclusion

Bibliography

[Sch78] Gideon Schwarz. "Estimating the dimension of a model". In: *The annals of statistics* (1978), pp. 461–464.

[PN97] Phillip A. Porras and Peter G. Neumann. "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances". In: *National Information Systems Security Conference* (1997).

[IM98] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *ACM Symposium on Theory of Computing* (1998).

[Ken99] Kristopher Kristopher Robert Kendall. "A database of computer attacks for the evaluation of intrusion detection systems". PhD thesis. Massachusetts Institute of Technology, 1999.

[Ker00] Christine Keribin. "Consistent estimation of the order of mixture models". In: *Sankhyā: The Indian Journal of Statistics, Series A* (2000), pp. 49–66.

[Zha+01] Zheng Zhang et al. "HIDE: a Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification". In: *IEEE Workshop on Information Assurance and Security* (2001).

[Cha02] Moses S Charikar. "Similarity estimation techniques from rounding algorithms". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 2002, pp. 380–388.

[CM02] Frédéric Cuppens and Alexandre Miege. "Alert correlation in a cooperative intrusion detection framework". In: *Proceedings 2002 IEEE symposium on security and privacy*. IEEE. 2002, pp. 202–215.

[GKM03] Ayalvadi J Ganesh, A-M Kermarrec, and Laurent Massoulié. "Peer-to-Peer Membership Management for Gossip-Based Protocols". In: *IEEE Transactions on Computers* (2003).

[JWQ03] R. Janakiraman, M. Waldvogel, and Qi Zhang. "Indra: A Peer-to-Peer Approach to Network Intrusion Detection and Prevention". In: *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (2003).

[MI03] Patrick Miller and Atsushi Inoue. "Collaborative intrusion detection system". In: *22nd International Conference of the North American Fuzzy Information Processing Society, NAFIPS 2003*. IEEE. 2003, pp. 519–524.

[Dat+04] Mayur Datar et al. "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proceedings of the twentieth annual symposium on Computational geometry - SCG '04*. Brooklyn, New York, USA: ACM Press, 2004, p. 253. ISBN: 978-1-58113-885-6. DOI: 10.1145/997817.997857. URL: http://portal.acm.org/citation.cfm?doid=997817.997857 (visited on 01/14/2022).

[Loc+05] M. E. Locasto et al. “Towards Collaborative Security and P2P Intrusion Detection”. In: *IEEE SMC Information Assurance Workshop* (2005).

[Szo05] Peter Szor. *The Art of Computer Virus Research and Defense: ART COMP VIRUS RES DEFENSE _p1*. Pearson Education, 2005.

[AI06] Alexandr Andoni and Piotr Indyk. “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions”. In: *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*. IEEE. 2006, pp. 459–468.

[Das+06] Denver Dash et al. “When Gossip is Good: Distributed Probabilistic Inference for Detection of Slow Network Intrusions”. In: *AAAI National Conference on Artificial Intelligence* (2006).

[Dum+06] Claudiu Duma et al. “A Trust-Aware, P2P-Based Overlay for Intrusion Detection”. In: *International Workshop on Database and Expert Systems Applications* (2006).

[Kor06] Jesse Kornblum. “Identifying almost identical files using context triggered piecewise hashing”. en. In: *Digital Investigation* 3 (Sept. 2006), pp. 91–97. ISSN: 17422876. DOI: 10.1016/j.diin.2006.06.015. URL: https://linkinghub.elsevier.com/retrieve/pii/S1742287606000764 (visited on 01/21/2022).

[MNP06] Rajeev Motwani, Assaf Naor, and Rina Panigrahi. “Lower bounds on locality sensitive hashing”. In: *Proceedings of the twenty-second annual symposium on Computational geometry*. 2006, pp. 154–157.

[VF06] Vivek Vishnumurthy and Paul Francis. “On Heterogeneous Overlay Construction and Random Node Selection in Unstructured P2P Networks”. In: *IEEE International Conference on Computer Communications* (2006).

[BA08] Rainer Bye and Sahin Albayrak. *CIMD - Collaborative Intrusion and Malware Detection*. 2008.

[Fun+08] Carol J. Fung et al. “Trust Management for Host-Based Collaborative Intrusion Detection”. In: *Managing Large-Scale Service Deployment* (2008).

[Rou10] Vassil Roussev. “Data Fingerprinting with Similarity Digests”. en. In: *Advances in Digital Forensics VI*. Ed. by Kam-Pui Chow and Sujeet Shenoi. Vol. 337. Series Title: IFIP Advances in Information and Communication Technology. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 207–226. DOI: 10.1007/978-3-642-15506-2_15. URL: http://link.springer.com/10.1007/978-3-642-15506-2_15 (visited on 01/21/2022).

[ZLK10] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. “A survey of coordinated attacks and collaborative intrusion detection”. In: *Computers & Security* 29.1 (2010), pp. 124–140.

[Gil+13] Manuel Gil Pérez et al. “RepCIDN: A Reputation-Based Collaborative Intrusion Detection Network to Lessen the Impact of Malicious Alarms”. In: *Journal of Network and Systems Management* (2013).

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. “Finding Similar Items”. In: *Mining of Massive Datasets*. 2nd ed. Cambridge University Press, 2014, pp. 68–122. DOI: 10.1017/CBO9781139924801.004.

[Mil+15] Aleksandar Milenkoski et al. "Evaluating computer intrusion detection systems: A survey of common practices". In: *ACM Computing Surveys (CSUR)* 48.1 (2015), pp. 1–41.

[PE15] Rafael Coimbra Pinto and Paulo Martins Engel. "A fast incremental gaussian mixture model". In: *PloS one* 10.10 (2015), e0139931.

[Vas+15] Emmanouil Vasilomanolakis et al. "SkipMon: A locality-aware Collaborative Intrusion Detection System". In: *IEEE International Performance Computing and Communications Conference* (2015).

[Vas16] Emmanouil Vasilomanolakis. "On Collaborative Intrusion Detection". PhD thesis. Darmstadt: Technische Universität Darmstadt, 2016. URL: http://tubiblio.ulb.tu-darmstadt.de/82583/.

[MH17] Vitor Hugo Galhardo Moia and Marco Aurélio Amaral Henriques. "Similarity Digest Search: A Survey and Comparative Analysis of Strategies to Perform Known File Filtering Using Approximate Matching". en. In: *Security and Communication Networks* 2017 (2017), pp. 1–17. ISSN: 1939-0114, 1939-0122. DOI: 10.1155/2017/1306802. URL: https://www.hindawi.com/journals/scn/2017/1306802/ (visited on 01/24/2022).

[Rub18] Aviad Rubinstein. "Hardness of approximate nearest neighbor search". In: *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*. 2018, pp. 1260–1268.

[WM18] Michael E. Whitman and Herbert J. Mattord. *Principles of information security*. Cengage Learning, 2018. ISBN: 978-1-337-10206-3.

[Ngu+19] Tri Gia Nguyen et al. "Search: A collaborative and intelligent nids architecture for sdn-based cloud iot networks". In: *IEEE access* 7 (2019), pp. 107678–107694.

[Hin+20] Hanan Hindy et al. "A taxonomy of network threats and the effect of current datasets on intrusion detection systems". In: *IEEE Access* 8 (2020), pp. 104650–104675.

[NBJ21] Parth Nagarkar, Arnab Bhattacharya, and Omid Jafari. "Exploring State-of-the-Art Nearest Neighbor (NN) Search Techniques". In: *8th ACM IKDD CODS and 26th COMAD*. 2021, pp. 443–446.

[Cor+22] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022.

[Wil] S. William. *Cryptography and Network Security - Principles and Practice, 7th Edition*. Pearson Education India. ISBN: 9789353942564. URL: https://books.google.de/books?id=AhDCDwAAQBAJ.