Hunting the Unknown

Elisa Costante, Jerry Den Hartog, Milan Petković, Sandro Etalle, Mykola Pechenizkiy

To cite this version:

Elisa Costante, Jerry Den Hartog, Milan Petković, Sandro Etalle, Mykola Pechenizkiy. Hunting the Unknown. 28th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec), Jul 2014, Vienna, Austria. pp.243-259, 10.1007/978-3-662-43936-4_16 . hal-01284860

HAL Id: hal-01284860

https://inria.hal.science/hal-01284860

Submitted on 8 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Hunting the Unknown
White-Box Database Leakage Detection

Elisa Costante[1], Jerry den Hartog[1], Milan Petković[1,2], Sandro Etalle[1,3], and Mykola Pechenizkiy[1]

[1] Eindhoven University of Technology, The Netherlands {e.costante, j.d.hartog, m.petkovic, s.etalle, m.pechenizkiy}@tue.nl
[2] Philips Research Europe, High Tech Campus, The Netherlands
[3] University of Twente, The Netherlands

Abstract. Data leakage causes significant losses and privacy breaches worldwide. In this paper we present a *white-box* data leakage detection system to spot anomalies in database transactions. We argue that our approach represents a major leap forward w.r.t. previous work because: i) it significantly decreases the False Positive Rate (FPR) while keeping the Detection Rate (DR) high; on our experimental dataset, consisting of millions of real enterprise transactions, we measure a FPR that is orders of magnitude lower than in state-of-the-art comparable approaches; and ii) the white-box approach allows the creation of self-explanatory and easy to update profiles able to explain *why* a given query is anomalous, which further boosts the practical applicability of the system.

Keywords: Leakage Detection, Privacy, Data Security, Anomaly Detection.

1 Introduction

Data is valuable; databases, storing customer and confidential business data, represent a core asset for any organization. This makes data leakage, i.e. the unauthorized/unwanted transmission of data and information [1], a major threat. The harm caused by a data leakage includes economic loss, damage to the reputation and decrease of the customers' trust. In case the leakage involves personal or sensitive information, legal liability for not complying with data protection laws also comes into play. To reduce these enormous costs and comply with legislation, timely detection of data leakage is essential.

Insiders threats, e.g. malevolent or simply careless employees, are amongst the top sources of data leakage: because of their right to access internal resources, such as databases, insiders can cause significant damages [2]. According to [3], data leaked from database accounts for most of the records disclosed in 2012.

Access Control (AC) mechanisms [4] aim to guarantee that only users that have the rights can access certain data and as such form a first line of defense against data leakage. However, AC has some limitations. For example, AC rules might be not expressive enough and, especially in dynamic context, might require frequent and costly updates. In addition, AC might reduce data availability which is critical in emergency situations (e.g. in healthcare domains) or productivity (e.g. time loss to ask for permission to access certain documents). As a result, organisations often apply relaxed AC policies by giving users access to more information than they actually need [5], which obviously also reduces the AC effectiveness against data leakage. Beside AC, tools and methodologies exist for data leakage detection and prevention [6], which mainly differ in the location where they operate (e.g. network, workstation or database). In this paper we act at a database level: in this way we can detect leakages at a very early stage, i.e. when sensitive data is leaving its primary source.

Academic solutions to database leakage detection typically work by monitoring database traffic in terms of SQL queries. Existing solutions can be divided into *signature-based* and *behavioural-based* systems [7]. Generally, in signature-based systems a blacklist defines the set of dangerous or denied access patterns. On the other hand, behavioural-based systems automatically learn permitted access patterns by observing 'normal' activities and mark every anomaly as potential threat. The main problem of signature-based approaches is that they can only detect well-known attacks, whereas behavioural-based approaches have the great potential of detecting unknown database attacks. In addition, by automatically generating fine-grained profiles, behavioural-based solutions require less human-effort thus offering the best possible detection at the lowest cost. These advantages make behavioural-based approaches widely adopted in literature [8–15]. However, these approaches have also drawbacks. The first problem is the high False Positive Rate (FPR) they usually generate. Since each false alert has to be analyzed by the security officer to establish whether it indicates a real threat or not, false positives have a high operational cost. In network anomaly detection [16, 17] (a different yet similar field), a system starts to be "usable in practice" when it shows a FPR in the order of 0,01%, a rate by far not attained by present database anomaly detection systems. The second drawback is that current solutions provide little or no support for alert handling. Usually, when an alert is raised, it prompts an investigation process carried out by the security officer. It is important to support the security officer in making a correct and efficient decision by providing as much useful information as possible on the nature of the alert. In this respect, signature-based systems have an 'unfair' advantage: when they raise an alert, they can say exactly which signature is violated and why this violation may constitute a problem. On the other hand, behaviour-based solutions usually accompany a raised alert with a *deviation degree* or an *anomaly score* which is virtually useless to the officer as it does not clearly state "what is going on". Explaining the reason of an alert is generally more difficult for anomaly detection systems because of their *black-box* nature, i.e. the underlying engine (be it a neural network or a machine learning classifier) is difficult to understand and update, properties which are particularly important to reduce the number of false positives and to understand the meaning of an alert.

To enable practical detection of unknown database leakage threats we introduce what, to the best of our knowledge, is the first *white-box* behavioural-based database leakage detection system. As opposed to existing black-box solutions,

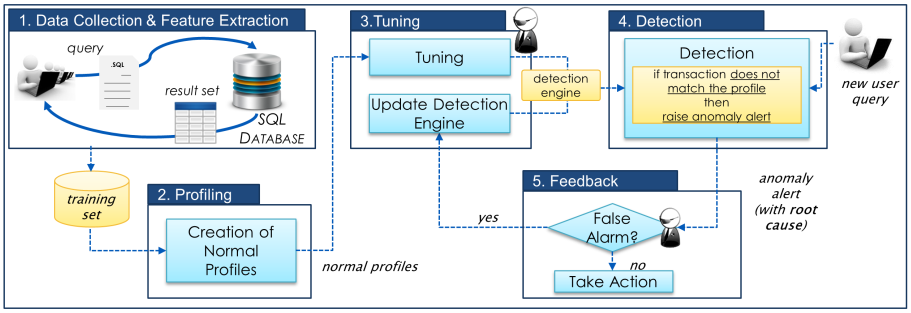


Fig. 1: Framework Overview.

which only flag a transaction as anomalous or not, the white-box approach enables the extrapolation of the root cause of an alarm. Our system creates self-explanatory and easy to update histogram-based profiles of database usage by observing SQL queries and raises an alarm when a query does not match the profiles. Finally, the system provides a feedback mechanism enabling the officer to mark an alarm as false positive, so that the system can be updated accordingly to avoid the repetition of the same mistake. Thanks to this new approach, our system strongly mitigates the aforementioned problems, namely:

- the white-box approach makes profiles and detection rules self-explanatory, so that users can easily modify them to improve the system accuracy;
- when an alarm is raised, the system explicitly indicates the alarm's root cause, facilitating the user in handling the alarms;
- the feedback mechanism permits to progressively reduce the FPR.

To validate the proposed solution, we carried out two sets of experiments, the first one using a database consisting of more than two million real transactions taken from a company database and the second one taken from a simulated scenario. In these experiments we benchmarked the systems against the ones presented in [9] and [10]. According to these tests, our system achieves a very high Detection Rate (DR) with a FPR which is one or two orders of magnitude lower (i.e. better) than the approaches used for comparison.

2 The Framework

The solution we present in this paper targets two core objectives, namely i) maximizing the DR of known and unknown threats; and ii) minimizing the FPR, i.e. the number of normal transactions erroneously marked as anomalous. To meet these objectives it is necessary to have a rich feature space, i.e. a set of features describing database transactions able to capture database usage at a fine-grain, so that even small deviations from usual behaviour can be detected. Figure 1 shows the five main phases of our framework. The first phase, the *Data Collection and Feature Extraction*, captures users (SQL) transactions (select/insert/update) and extracts the feature space. The output of this phase is the *training set*, used during *Profiling* phase to construct normal behaviour profiles. Once profiles are ready the security officer can use the *Tuning* to inspect them and verify whether they are complete, and optionally decide to extend the profiling period. The tuning generates a *detection engine* which is used during *Detection* to flag each new incoming query as normal or anomalous. In case of an anomalous query, an alert which clearly states why the detector considers it as an anomaly is raised. Finally, when an alarm is raised, the *Feedback* mechanism allows the security officer to flag it as false positive, which cause an immediate update of profiles and detection engine so that the same mistake is not repeated.

2.1 Data Collection and Feature Extraction

Definition 1 (Feature Space). *The* feature space $F = \langle f_1, f_2, ..., f_n \rangle$ *is the list of query characteristics, including its syntax, result and context (features), used to represent a query.*

The feature space determines the level of details of the profiles we build, and the kind of attacks that can be detected. Table 1 describes our feature space; features can be divided into three groups: *syntax-centric*, *context-centric*, and *result-centric*. Each type of feature helps in detecting specific threats. For example, syntax-centric features, by profiling the kind of queries users usually make, enable the detection of leakages caused by e.g. privileges misuses. By using context-centric features it is possible to detect misuses related to database usage at unusual location or time. Finally, result-centric features represent the only way to spot leakages caused by the access to illegitimate data values (see Table 1). To construct the feature space it is necessary to monitor user's activities in terms of SQL queries. Several ways to capture users' queries are discussed in [18]. To learn representative profiles it is important that during data collection all queries normally submitted to the database are captured to form a training set w.r.t. our definitions.

Definition 2 (Query). *A* query Q *is a list of* n*-query values, with one query value* $Q(f_i)$ *for each feature* f_i*.*

Definition 3 (Feature Values). *Each feature* f_i *has a type and an associated finite set of values* $V_i = \{v_{i,1}, \ldots, v_{i,m_i}\}$ *that the feature can take.*

Definition 4 (Training Set). *The set of queries collected during the monitoring period, comprises our training set* $T = \{Q_1^t, Q_2^t, ..., Q_k^t\}$*, containing the* k *queries which will be used to build normal profiles.*

2.2 Profiling

Profiles should be able to completely describe normal database usage, i.e. which queries are submitted, when, at which location, what results are retrieved and so on. Normal usage may be related to different entities, e.g., the *user* or the *role* submitting the query. Thus different profiles may be needed for different 'entities'. The first step in the profiling process is choosing the entity for which profiles are built.

Table 1: Feature classification and feature utility in detecting specific threats.

	Feature	**Detected Threats**
Syntax Centric	Query (command, tables, columns)	These features help in detecting leakages due to users accessing information out of their working scope (e.g. due to *excessive privileges granted* or *misuse of privileges*).
	Where Clause (length, special chars, columns and tables)	*SQL Injection* and *XSS attacks* usually act on the syntax of a query to inject malicious (SQL) statements which can be used to extract sensitive information from the database. Dangerous injected statements usually contain specific keywords that can be monitored to help the detection.
Context Centric	Response Code	Specific codes are given for specific database events, e.g. failing login attempts. Multiple failures might indicate a *password guess* attack, which might start a data leakage.
	Client/DBMS USERID/ROLE	Identifying which end-user and with which role or which client application is responsible of anomalous activities is helpful for *accountability reasons.*
	Timestamp, IP Address	Access from unusual location or at unusual time might indicate the credentials have been stolen e.g. someone is carrying on a *masquerade attack.*
Result Centric	N. of Records and Bytes	Retrieving a large quantity of data (e.g. copying customer list) might indicate a data leakage/misuse is taking place.
	Result Set	The results returned by a query helps in detecting misuses where the query syntax is legitimate (e.g. a doctor can access the table *disease* and *patient*) but the data retrieved are not (e.g. the doctor access records of patients she does not treat).

Definition 5 (Profiling Feature). *The* profiling feature $\bar{f} \in F$ *represents the entity we want to profile. We use* $\bar{x}$ *for the value of* x *for the profiling feature, e.g.* $\bar{i}$ *is the index of the profiling feature,* $\bar{f} = f_{\bar{i}}$, $\bar{V} = V_{\bar{i}}$ *is the set of* profiling values, $\bar{m} = m_{\bar{i}}$ *is the size of this set, etc.*

Definition 6 (Histogram). *A* threshold t *is a number* $\in [0, 1]$ *or a label in* $\{anomalous, normal\}$. *We define* $anomalous < 0$ *and* $1 < normal$. *A* bin *is a frequency* $freq \in N$ *together with a threshold* t. *A* histogram h_i for feature f_i *has a size* $|h_i|$ *representing the number of queries it contains, and assigns a bin* $bin(v, h_i)$ *to each value* $v \in V_i$ *that* f_i *can take. Given a value* $v \in V_i$ *and a histogram* h_i *we write:*

- $freq(v, h_i)$ *for the frequency in* $bin(v, h_i)$*;*
- $t(v, h_i)$ *for the threshold in* $bin(v, h_i)$*;*
- $prob(v, h_i)$ *for* $\frac{freq(v,h_i)}{|h_i|}$*, the probability of* v *in* h_i*.*

To build the histograms the training set T is first divided into $\bar{m}$ subsets $S_1, S_2, ..., S_{\bar{m}}$. In each subset S_j, the queries have the same value on the profiling feature (namely $v_{\bar{i},j}$). For example, if we choose the *userid* as profiling feature $\bar{f}$ and in the training set $\bar{f}$ only takes 2 different values e.g., $\bar{v}_1 = rob$ and $\bar{v}_2 = sally$, then the training set will be divided into $\bar{m} = 2$ subsets, containing respectively the queries executed by *rob* and *sally*. At this point the creation of the profiles can start. Note that when building the profiles (and during detection), we make a restriction in assuming that features are *independent* of each other (the implications of this assumption are discussed in Section 4).

Definition 7 (Profile). *A* profile H *is a list of histograms for each feature except* $\bar{f}$. *The* profile set P *gives a profile for each profiling value* $\bar{v} \in \bar{V}$. *We write* $H^{\bar{v}}$ *to refer to the profile for the profiling value* $\bar{v}$ *(e.g.* H^{rob} *or* H^{sally}*). We write* $h_i^{\bar{v}}$ *to refer to the histogram for feature* f_i *in the profile* $H^{\bar{v}}$*.*

Note that different features are of different data types (*nominal*, *numeric*, *time* and *set*) and could have wide ranges. To create meaningful profiles we need to create group (or bins) of query values, e.g. 'Timestamp' can be grouped according to the work shift. A key challenge in the creation of the histograms is the definition of an optimal size of its bins –*bin width*– which depends on the feature data type. For *nominal* features, we can simply take a bin for each different value encountered. This approach does not work for *numeric* features as it would lead to an explosion in number of bins. Instead ranges of values are used. The size of the ranges has to be a right balance between narrow (many bins with low frequency, risk of increasing false positives) and wide (few bins with high frequency, risk of missing anomalous queries). For *time* features, bins can be defined as e.g. hours, days or work shift according to the domain. With a *set* feature, e.g. the tables used in a query, one could consider two different approaches to define the bins; i) create a different bin for each set; or ii) create a different bin for each element of the set. In the first approach we care about the exact combination of values. In the second only about which values occur. In addition, in the second approach a single query could count towards multiple bins. In this paper we apply the second approach (we omit a deeper discussion of the differences between the two approaches for reason of space).

To finalize the profiling we normalize the histograms into a probability distribution. The transformation is done by dividing the frequency of each bin by the size of the histogram which (right after training) is equal to the size of the related subset S_j, thus giving $prob(v_{i,j}, h_i) = \frac{freq(v_{i,j})}{|S_j|}$. Normalization is useful to deal with different size of different subsets, e.g. a user more active than another.

Once the histograms have been built, they will represent the normal behaviour: during detection if all the values of a query fall in bins with high frequency, the query will be considered normal, while it will be considered anomalous if at least one value falls in a bin with low or zero frequency.

This way of profiling allows great flexibility. First of all, by setting the profiling feature it is possible to create different profiles, e.g. a profile per *userid* and a profile per *role* to check whether users' activities match with their correspondent role. Furthermore, the usage of histograms allows 'online learning': the profile is built one instance at a time which means it is not necessary to retrain the complete model (as it happens e.g. with clustering) in case a new transaction has to be added to the system.

2.3 Tuning

In existing solutions the training set is usually assumed to be *attack-free* and *exhaustive*, i.e. it is representative w.r.t. normal behaviour. Drawbacks of these assumptions are: i) if an attacker is already active during the data collection, the *attack-free* assumption can lead to failing to detect some leakages and misuses; and ii) the *exhaustive* assumption can contribute to the explosion of false positives in case normal behaviour is not fully represented. To detach from these assumptions, we provide a tuning mechanism to allow a security expert to inspect and adapt normal profiles.

During the tuning, the expert can set a *global threshold*, representing the minimum probability each bin of the training set must satisfy to be considered normal. The global threshold is used to globally label all bins of all histograms of all profiles: if the bin probability is lower than the threshold, the bin is marked as *anomalous*, it is marked as *normal* otherwise. The global threshold can be seen as a measure of the level of tolerance towards rare values: a threshold of zero means that every query in the training set is considered legitimate (attack-free assumption), while a threshold higher than zero means that *rare* values (i.e. value which probability is below threshold) are considered anomalous, hence they will cause an alarm. A zero threshold also means that all the alarms generated during detection are caused by *new values*, thus a high number of false positives in this setting implies the training does not exhaustively represent normal behaviour.

To allow profile manipulation, after setting the global threshold, it is possible to manually flag each bin as *normal* or *anomalous*. For example, if the *delete* command is always anomalous then it can be flagged as such. In this way the expert can have a general rule of thumb (the global threshold) for initially uniformly labeling all the bins, and the fine-grain mechanism to add exceptions to the general rule. The set of profiles with labeled bins forms the *detection engine*.

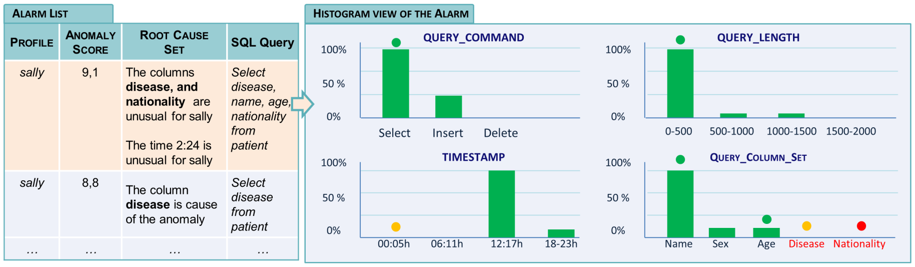
2.4 Detection & Feedback Loop

After training, the detection engine analyzes incoming queries w.r.t. normal behaviour profiles to determine whether they are anomalous or not.

Definition 8 (Anomalous Query). *Consider a query Q by entity $\bar{v} = Q(\bar{f})$. Writing $v_i = Q(f_i)$ for the value Q assumes for feature f_i and $h_i^{\bar{v}}$ for the histogram for f_i in profile $H^{\bar{v}}$ we define: i) $R(Q) = \{f_i : prob(v_i, h_i^{\bar{v}}) <= t(v_i, h_i^{\bar{v}})\}$, the* root cause set *containing the features which values are anomalous; ii) $Anomalous(Q) = true \iff R(Q) \neq \emptyset$, which says whether a query is anomalous or not; iii)$AnomalyScore(Q) = \prod_{f_i \in R(Q)} \frac{1}{prob(v_i, h_i^{\bar{v}})}$, a score which quantifies how likely anomalous Q is (but not how harmful it is).*

In case Q is anomalous an alarm is raised, as shown in Figure 2. Alarms are listed together with the related profile (*sally* in the example), the anomaly score, the root cause set and the anomalous query text. The *anomaly score* is bigger than zero for anomalous queries (rarer values contribute more to the score). The *root cause set* contains the features which values have no correspondent bin, or fall in one labeled as anomalous, thus causing the alarm. Knowing the root causes of an anomaly helps the security officer in handling the alarm. For a query to be anomalous, it is sufficient that a single feature has a value falling in an anomalous/not existent bin. Of course, several features can cause the anomaly, in which case the anomaly score will be higher and the root cause list wider. When a value has no correspondent bin, i.e. a previously unseen value, we use a minimum probability (rather than zero) to avoid division by zero.

When an alarm is raised the security officer has to handle it. Consider the first alarm shown on the left side of Figure 2, caused by *sally* using the columns



ALARM LIST

PROFILE	ANOMALY SCORE	ROOT CAUSE SET	SQL Query
sally	9,1	The columns **disease, and nationality** are unusual for sally The time 2:24 is unusual for sally	*Select disease, name, age, nationality from patient*
sally	8,8	The column **disease** is cause of the anomaly	*Select disease from patient*
…	…	…	…

Fig. 2: Detection Results: how alarms can be presented to the security officer.

disease and *nationality*, and submitting the query at *2:24 am*. When the officer selects the alarm, the values of the query (circles) are represented w.r.t. the related profile (histograms) as shown on the right side of the figure. A circle over a bin (bar) means that the query value for that feature falls in that bin. The officer can provide feedback about every alarm, i.e. she can mark each root cause as *true positive* or *false positive*. A true positive means that a specific value represent an actual anomaly and that the officer wants to be warned every time it occurs. A false positive instead means that one of the causes of an alarm is benign and it should not cause an alarm any longer. Circles can be green (for normal values), yellow (for values which have still to be evaluated by the officer), and red (for values which have been previously marked as *true positive*, e.g. *nationality*). For example, assuming that *sally* is entitled to use the column *disease*, the officer will mark such value as a false positive, which will cause a refinement of our profiles: the profile *sally* will be updated by adding a new bin with the value *disease* to the corresponding histogram; the bin frequency will be incremented of a unit, and the threshold will be set to *normal*. This implies that if *sally* uses the column *disease* again, no alarm will be raised. In addition, the other alarms related to *sally* will be updated accordingly, meaning that if an alarm is caused by the use of *disease*, e.g. the second alarm in Figure 2, it will be deleted from the list. In this way a single officer's feedback can result in the deletion of multiple alarms, hence reducing the officer's working load.

3 Evaluation

In this section, we validate our system by addressing the following questions:

- How effective is our framework. To this end, we made an experimental evaluation of its Detection Rate (DR) and its False Positive Rate (FPR).
- How does it compare with existing solutions, in particular with Kamra et al. [9] and Wu et al. [10]. To help the comparison we use Receiver Operating Characteristics (ROC) curves [19] and Area Under the Curve (AUC) values.
- What is the added value of the feedback mechanism? We test this by measuring the impact a feedback operation has in reducing the FPR.

3.1 Methodology

We implemented our framework as a RapidMiner (`http://rapidminer.com/`) extension, and made experiments with two different datasets, one with 2 Gb of real enterprise transactions, the other one with simulated queries. Both datasets were divided into a Training Set ($\sim$ 70% of the dataset) used to learn the profiles, and a Testing Set (the remaining $\sim$ 30%) used as input for the detection engine. To measure the *FPR* we assume both datasets are attack-free (the global threshold is set to zero), which is a common practice in this domain. In this way every alarm raised on the Testing Set can be considered a false positive. Beside the training and the testing set we also have distinct Attack Sets containing malicious query specifically crafted for each dataset. In these settings, we measure *FPR* and DR as:
– *FPR*=(*#alarms raised on the testing set*)/(*cardinality of the testing set*)
– *DR* = (*#alarms raised on the attack set*)/(*cardinality of the attack set*)

The Enterprise Dataset (ED) is taken from the log of an operational database of a large IT company, with about 100 users, owning 4 different roles. We collected a total of 2,349,198 transactions (1,644,437 in the Training Set and 704,761 in the Testing Set), and extracted the features listed in Table 2. Note that the Result Set (RS) and the client application *role* are not available for this dataset. In addition, we have an Attack Set composed of 107 malicious queries devised and executed by the security administrator of the database, logged in as one of the profiled users. These queries represent real threats to the enterprise since they access sensitive information normal employees do not normally access (e.g. other employees' *password* or *user ids*).

The Simulated Dataset (SD) was constructed using the healthcare management system GnuHealth (`http://health.gnu.org`). We simulated *normal behaviour* (validated by domain experts) consisting of an *admin* and different users of a hospital, where *doctors* and *nurses* take care of *patients* suffering from different *diseases.* The Simulated Dataset contains a total of 30,492 queries (21,344 in the Training Set and 9,148 in the Testing Set); in addition, we simulated two attacks:

- *Attack 1*: the admin looks at parts of the database (e.g. the table *patient*) which she should have no interest in (277 malicious queries);
- *Attack 2*: a general practitioner, who only treats patients with generic disease (e.g. *flu*, *cough*) accesses data of patients with *HIV* (26 malicious queries). Note that queries in *Attack 2* have been specifically crafted to be detected based on RS features only.

To compare our framework to competing proposals from literature, we reproduced the systems of [9, 10] and we measured their FPR and DR on both datasets. Both approaches apply Naïve Bayes to learn the *userid* (or *user role*) class: an alarm is raised if the class predicted by the classifier differs from the actual one. The main difference between the two solutions is given by the features used to learn the classifier: Kamra et al. adopt a pure syntax-centric approach while Wu et al. add context-centric features. The Kamra et al. solution has three variants – c-quiplet, m-quiplet and f-quiplet – which differs in how fine grained

Table 2: Features extracted during the experiments.

	Feature	**Type**	**White-Box without RS**	**White-Box with RS**	**Kamra et al. (c-quiplet)**	**Kamra et al. (m-quiplet)**	**Kamra et al. (f-quiplet)**	**Wu et al.**
Syntax Centric	QUERY_COMMAND	nominal	v	v	v	v	v	v
	QUERY_LENGTH	numeric	v	v				
	QUERY_COL_SET	set	v	v			v	
	QUERY_COL_NUM	numeric	v	v	v	v		v
	QUERY_TABLE_SET	set	v	v		v	v	v*
	QUERY_TABLE_NUM	numeric	v	v	v			
	QUERY_SELECT_ALL	nominal	v	v				
	WHERE_TABLE_SET	set	v	v		v	v	v*
	WHERE_TABLE_NUM	numeric	v	v	v			
	WHERE_COL_SET	set	v	v			v	
	WHERE_COL_NUM	numeric	v	v	v	v		
	WHERE_LENGTH	numeric	v	v				
	WHERE_SPEC_CHAR	numeric	v	v				
Context Centric	OS_USERNAME	nominal	v	v				
	CLIENT_APP_UID	nominal	v	v	v**	v**	v**	v**
	CLIENT_APP_ROLE	nominal	v	v	v**	v**	v**	v**
	RESPONSE_CODE	nominal	v	v				
	TIMESTAMP	time	v	v				v
	IP_ADDRESS	nominal	v	v				v
Result Centric	BYTES_NUM	numeric		v				
	ROWS_NUM	numeric		v				
	DISEASE	nominal		v				
	PATIENT_ID	nominal		v				

v - feature used by the approach
v* - the *tables* in the *from* and in the *where* clause are grouped in a single feature
v** - *role* and *userid* are mutually exclusively used as labels

their feature space is. When comparing the results, it is important to note that our approach differs from [9, 10] not only in the algorithm, but also in the following aspects. First, we consider more features: Table 2 shows the feature space used in our solution and those used by the other two approaches. Secondly, we can take into account the RS; this is however only present in the Simulated Dataset, and for this dataset we made two separate measurements: one with the the RS and the other without. Finally, our system is devised from scratch to include a feedback loop. However, to guarantee fairness, the feedback has not been used when comparing our results with those of [9, 10].

Generally, a detector performs well if it shows both a low FPR (costs) and a high DR (benefits). FPR and DR depend on the *true* and *false* alarms raised by a detector. Recall that an alarm is raised if *anomaly score* > 0 in our solution, and if *prediction* $! =$ *actual class* for the solutions in [9, 10]. To plot ROC curves we measure FPR and DR for varying values of a decision threshold t. The threshold t is used to vary the output of the detectors as follows: for our solution an alarm will be raised if *anomaly score* $> t$, while for [9, 10] an alarm will be generated if *prediction* $! =$ *actual class* and the *prediction probability* $> t$. To compare different detectors we plot their ROC curves in a single graph: the best detector is the one which ROC curve passes closer to the upper left point (0,1) (zero costs, maximum benefits). In case ROCs intersect, it might be difficult to visually spot which method performs better, thus we reduce ROC curves into a single scalar value: the AUC. Note that outside the ROC context we assume the use of the default value for t which is 0 for all the considered approaches and which leads to the best FPR-DR tradeoff.

3.2 Results

Table 3 presents the performance of different solutions over the Enterprise Dataset, with *userid* as profiling feature. The results show that, while the DR is high (100%) for every solution, the White-Box approach has the lowest FPR (1.65%), which is 7 times better than the best results amongst the comparing approaches (Kamra et al., m-quiplet, FPR of 12.24%). Our solution is also the one with the best FPR-DR tradeoff, as shown by the ROC curves in Figure 3 and by the highest AUC value (0.987). This means we can provide the best benefit at the lowest cost, hence we consistently reduce the officer's work load.

On the Simulated Dataset, we could test the performance of our solution also in presence of RS. Results over this dataset, with *userid* as profiling feature, are shown in Table 4 and confirm the outstanding FPR of the White-Box solution which is orders of magnitude lower than the one of compared approaches (1.09% without RS and 1.41% with RS, versus 44.00% of Wu et al., the next in order with the lowest FPR). Interestingly, the addition of the RS features results in a slight increase of FPR, but it does allow us to detect *Attack 2*, which was devised explicitly to be detectable only in presence of RS information (therefore *Attack 2* should not be detected by syntax-based approaches). With these settings, it is not surprising that the White-Box solution without RS has DR2 = 0.0%: it misses the potential of detecting *Attack 2*. What is surprising is that DR2 is different from zero for the competing approaches: in principle, neither of them – being syntax-based – should be able to detect this (note that a random guess detector also has a DR different from zero). On the other hand, when we add RS features we obtain the highest DR (80.77%). Note that although we fail to detect all malicious queries in *Attack 2*, we actually manage to detect the attack and to warn the security officer that a specific user is acting strangely. Finally, the White-Box solution with RS is the one offering the best FPR-DR tradeoff as shown by the ROC in Figure 4 (DR1 and DR2 are combined to plot the ROC).

Table 5 shows the results of our test when the *user role* is chosen as profiling feature. The results refer to the Simulated Dataset only as role information is not available in the Enterprise Dataset. The table shows that FPR is generally lower, while DR is either less than or equal to the results obtained when *userid* is chosen as profiling feature (see Table 4). This is because, especially in large enterprises, *userid* profiles are more specific than *user role* profiles. The better FPR is due to the fact that the engine is less sensitive to small variations; the "lower" DR to the fact that profiles are less specific. As shown in the table, the White-Box solution (with and without RS) has the same DR for both *userid* and *role* profiles. In the case of *Attack 1* this is due to the fact that we have only one user with role *admin*, which makes role and user profiles identical. In the case of *Attack 2* the same DR means that none of the other doctors ever looked at disease HIV, thus if any doctor looks at it, an alarm will be raised. Note that if we add a doctor who treats patients with HIV to the normal behaviour scenarios, DR2 would likely decrease. In overall, as shown by the ROC curves in Figure 5, the White-Box solution with RS performs better than the others (AUC=0.984), with a small cost in terms of FPR (0.91%).

	FPR	**DR**	**AUC**
White-Box (without RS)	**1.65%**	100%	**0.987**
White-Box (with RS)	*na*	*na*	*na*
Kamra et al. (c-quiplet)	15.00%	100%	0.927
Kamra et al. (m-quiplet)	12.24%	100%	0.951
Kamra et al. (f-quiplet)	12.67%	100%	0.981
Wu et al. (standard)	13.03%	100%	0.957

Table 3: Results for Enterprise Dataset (*userid* profiles)

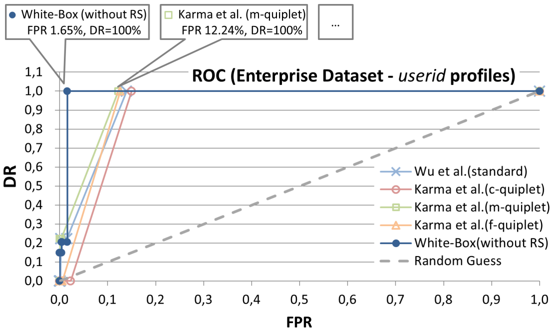


Fig. 3: ROC curves comparison - Enterprise Dataset (*userid* profiles).

	FPR	**DR1**	**DR2**	**AUC**
White-Box (without RS)	**1.09%**	100%	0.00%	0.948
White-Box (with RS)	**1.41%**	100%	**80.77%**	**0.980**
Kamra et al. (c-quiplet)	72.18%	98.92%	53.85%	0.584
Kamra et al. (m-quiplet)	71.03%	100%	76.92%	0.406
Kamra et al. (f-quiplet)	69.58%	100%	73.08%	0.422
Wu et al. (standard)	44.00%	100%	42.31%	0.824

Table 4: Results for Simulated Dataset (*userid* profiles)

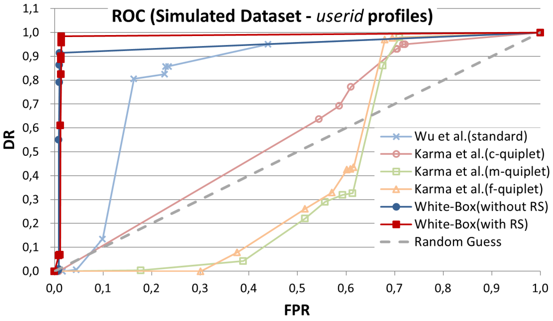


Fig. 4: ROC curves comparison - Simulated Dataset (*userid* profiles)

	FPR	**DR1**	**DR2**	**AUC**
White-Box (without RS)	**0.38%**	100%	0.00%	0.954
White-Box (with RS)	**0.91%**	100%	**80.77%**	**0.984**
Kamra et al. (c-quiplet)	40.04%	100%	3.85%	0.687
Kamra et al. (m-quiplet)	50.73%	100%	0.00%	0.517
Kamra et al. (f-quiplet)	50.87%	100%	0.00%	0.518
Wu et al. (standard)	29.34%	100%	0.00%	0.823
Wu et al. (hierarchical)	27.68%	100%	0.00%	0.839

Table 5: Results for Simulated Dataset (*role* profiles)

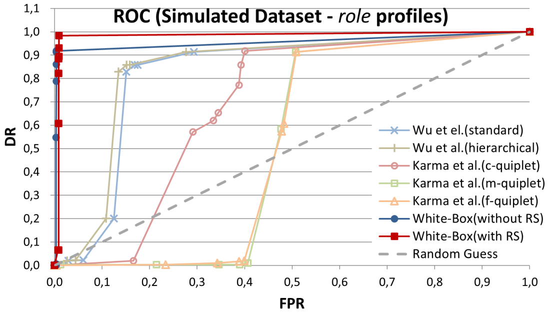


Fig. 5: ROC curves comparison - Simulated Dataset (*role* profiles).

Table 6: Feedback Impact on FPR (*userid* profiles).

	Feedback Operation	**Enterprise DS**		**Simulated DS**	
		FPR (%)	FP (#)	FPR (%)	FP (#)
White-Box (without RS)	0	1.65%	11596	1.09%	100
	1	1.17%	8234	1.08%	99
	4	0.05%	327	1.04%	95
	7	0.05%	318	1.01%	92
White-Box (with RS)	0	*na*	*na*	1.41%	129
	1	*na*	*na*	1.40%	128
	4	*na*	*na*	1.13%	104
	7	*na*	*na*	1.09%	98

Impact of the Feedback Loop. A novel component of our framework is a feedback loop mechanism that allows to iteratively decreasing the FPR. All the results shown in the previous section did not take advantage of this feedback mechanisms to ensure a more fair comparison with the other systems in the literature [9, 10]. Table 6 shows how the feedback loop mechanism actually improves the FPR. The table refers to *userid* profiles, and shows that in the enterprise dataset, only 7 feedback operations are enough to drop the FPR from 1.65% to 0.05% (from 11596 to 318 alarms). On the simulated dataset, 7 feedback operations reduce the FPR from 1.09% to 1.01% and from 1.41% to 1.09% respectively in the solution without and with RS. Note that – as expected – the impact of a single feedback operation is higher when multiple false alarms are caused by the same feature value, e.g. in the enterprise dataset a single feedback operation eliminated 3362 false alarms. It is worth mentioning that a single feedback operation consists of only few mouse clicks: those necessary to mark an alarm as false positive. Finally, note that none of the existing solutions offer a built-in feedback mechanism, mainly because it would require a complete re-training of the detection engine, which is not necessary in our framework.

4 Discussion, Limitations and Future Work

As our experimental validation has shown, our framework boosts the practical applicability of anomaly-based database leakage detection by presenting the highest DR with a FPR which is orders of magnitude lower when compared to existing solutions. In addition, the FPR can be further reduced by applying the *feedback loop* to eliminate false positives. Finally, the white-box nature of our solution clearly states the causes of an anomaly, hence helping the officer in handling the alerts. We believe the good results achieved mainly depend on our fine-grained profiles, which allow creating a more faithful model of usage behaviour. On the other hand, we think there are several reasons why the other approaches (we compared to) present a high FPR. First of all, the other solutions use Naïve Bayes which inherently depends on the user's ids (or role) distribution of activities: their performance are usually better with training sets in which actions are evenly distributed amongst users. However, relying on this assumption is a strong limitation in database leakage detection where query data flows are generally not uniformly distributed. Secondly, these solutions have a limited feature space which does not allow to creating fine-grained profiles. However, since Naïve Bayes is not designed to take advantage of each feature in a specific way as we do, we expect that a larger feature space would not significantly improve performance of the other approaches (initial tests seem to confirm this).

Although the general good results, our solution still has some limitations we need to address. The major limitation is due to the feature independency assumption used to build profiles. This assumption implies that anomalous queries for which each individual feature by itself is normal cannot be detected. For example, it might be normal that *sally delete* on column *age* and *select* on column *disease*, however the fact that she *delete* on column *disease* might represent a misuse that is currently undetected. To address this problem, we aim at applying data mining techniques (rule-mining in particular) to find combination of features that discriminate anomalies from normal behaviour. The second limitation is related to the detection per single query. In the current implementation, each query is individually analyzed and labeled by the detection engine. In this way, attacks that are split over multiple queries (or multiple sessions) might not be detected. Extending our solution to detect anomalies which involve groups of queries is another interesting path we intend to follow. Finally, several improvements can be made to the feedback mechanism e.g., by adding a signature-based approach to let the officer define fine-grained exception rules.

5 Related Work

Database Leakage Detection Solutions. Available database leakages detection solutions mainly differ for the type of features (*syntax-centric*, *context-centric*,or *result-centric*) used for representing normal behaviour.

The works presented in [8–10] are syntax-centric. In [8] the authors build normal behaviour profiles based on the assumption that SQL commands and their order in each database transaction are relevant. During detection, if an attacker executes valid queries but in an incorrect order, an alert is raised. In [9] normal profiles are built using a Naïve Bayes classifier. The system learns to predict the *userid* or the *role* based on the SQL command and on the tables and columns in the query. When a new query arrives, in case the userid (or the role) predicted by the classifier mismatch the actual value, an alarm is raised. In [10] the approach is very similar to the one in [9], with an extended feature space.

Pure syntax-centric approaches have to deal with some structural limitations. For example, they fail to detect situations where the type of query submitted is normal but the data retrieved is anomalous. These considerations are at the basis of the solution proposed in [12], that suggests to profile normal behaviour based on the data users retrieve. A mixed approach combining result-centric and context-centric is used in [13] where a mining algorithm is used to define association rules between context and result set. In this way the same request may be legitimate if performed within one context but abnormal within another. Finally, in [14] syntax-centric and result-centric approaches are combined. Normal profiles are represented in terms of statistical distribution: if a new query does not match the original probability distribution, it will be considered anomalous.

Anomaly Detection Techniques. Behaviour-based solutions detect potential database leakages by identifying deviations from normal behaviour, a problem generally known as *anomaly detection*. Anomaly detection techniques can be divided into supervised, semi-supervised, and unsupervised [20]. In *supervised* approaches, it is assumed that the training set contains queries labeled either as *normal* or as *anomalous*. In the *semisupervised* approach the assumption is that the training set contains only labels for normal transactions. Finally, in the *unsupervised* approach no labels are needed: the dataset is sought for intrinsic similarities that can be modeled so that outliers can easily pop out.

In the context of database leakage detection a labeled dataset is very hard to obtain, thus unsupervised techniques are those used the most in this field. Examples of unsupervised techniques include clustering, association rules, and statistical methods. Clustering techniques group similar instances of the training set according to a distance or similarity function. Any sample that does not belong to any cluster is considered an anomaly. A major drawback of clustering techniques is that the number of clusters has to be defined a-priori. In addition, a complete re-training is necessary if a new sample has to be added to the model. Association rules have the main advantage of being self-explanatory, thus creating a white-box system. However, they have a high computational cost and high memory consumption [21]. Finally, statistical methods are based on the probabilistic model generated from the training set: if the probability of the new data instance to be generated by such probabilistic model is very low, then the instance is considered an outlier. Statistical methods includes Hidden Markov Models (HMM) and histogram analysis. HMMs are very good to model temporal relationships, however, the high computational and memory cost, together with the high number of parameters that need to be set can discourage their usage [7, 21]. Histogram-based are the simplest nonparametric statistical techniques. The profiles of normal data are based on the construction of a histogram of frequencies for each feature of the feature space. This technique is computationally inexpensive, and generates a self-explanatory model. Its simplicity and intrinsic white-box structure, together with the support to the *online learning* – no need of retraining when a model update is required– make this solution the best candidate when a model easy to understand and to update is required.

6 Conclusions

In this paper, we presented a white-box behaviour-based database leakage detection system. Our experimental validation shows that our approach provides high DR for different kinds of attacks, both syntax and result set related, by keeping a low FPR. Especially, the FPR is orders of magnitude lower than that of comparable solutions (1.65% over the Enterprise Dataset), thus boosting the practical applicability of our solution. The number of false positives can be fur-

ther reduced thanks to the feedback loop. Furthermore, the white-box nature of the solution creates self-explanatory profiles and allows the extraction of the root causes of an alarm. As future work we aim at enhancing our system to cope with current limitations and to extend the validation process to other (real) datasets.
Acknowledgment - This work has been partially funded by the Dutch program COMMIT under the THeCS project.

References

1. Gordon, P.: Data Leakage - Threats and Mitigation. Technical report, SANS Institute (2007)
2. Software Engineering Institute: 2011 CyberSecurity Watch Survey. Technical report, Software Engineering Institute, Carnegie Mellon University (2011)
3. Verizon: The 2013 Data Breach Investigations Report. Technical report (2013)
4. Samarati, P., de Vimercati, S.: Access control: Policies, models, and mechanisms. Foundations of Security Analysis and Design (2001)
5. Caputo, D., Maloof, M., Stephens, G.: Detecting insider theft of trade secrets. In: S&P, IEEE (2009)
6. Shabtai, A., Elovici, Y., Rokach, L.: A survey of data leakage detection and prevention solutions. Springer (2012)
7. Patcha, A., Park, J.: An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks (2007)
8. Fonseca, J., Vieira, M., Madeira, H.: Integrated intrusion detection in databases. In: Dependable Computing. Springer (2007)
9. Kamra, A., Terzi, E., Bertino, E.: Detecting anomalous access patterns in relational databases. The VLDB Journal (2007)
10. Wu, G., Osborn, S., Jin, X.: Database intrusion detection using role profiling with role hierarchy. In: SDM, Springer (2009)
11. Bockermann, C., Apel, M., Meier, M.: Learning sql for database intrusion detection using context-sensitive modelling. In: DIMVA, Springer (2009)
12. Mathew, S., Petropoulos, M.: A data-centric approach to insider attack detection in database systems. In: RAID, Springer (2010)
13. Gafny, M., Shabtai, A., Rokach, L., Elovici, Y.: Applying unsupervised context-based analysis for detecting unauthorized data disclosure. In: CCS, ACM (2011)
14. Santos, R., Bernardino, J., Vieira, M., Rasteiro, D.: Securing Data Warehouses from Web-Based Intrusions. In: WISE, Springer (2012)
15. Chung, C., Gertz, M., Levitt, K.: Demids: A misuse detection system for database systems. In: Integrity and internal control information systems. (2000)
16. Bolzoni, D., Etalle, S., Hartel, P.: Panacea: Automating attack classification for anomaly-based network intrusion detection systems. In: RAID, Springer (2009)
17. Hadžiosmanović, D., Simionato, L., Bolzoni, D., Zambon, E., Etalle, S.: N-Gram against the machine: on the feasibility of the n-gram network analysis for binary protocols. In: RAID, Springer (2012)
18. Jin, X., Osborn, S.: Architecture for data collection in database intrusion detection systems. In: SDM, Springer (2007)
19. Fawcett, T.: An introduction to ROC analysis. Pattern Recognition Letters (2006)
20. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection. ACM Computing Surveys (2009)
21. Mazhelis, O.: One-class classifiers: a review and analysis of suitability in the context of mobile-masquerader detection. South African Computer Journal (2006)