HengHa: Data Harvesting Detection on Hidden Databases

Shiyuan Wang
Department of Computer Science, UC Santa Barbara
Santa Barbara, CA, USA
sywang@cs.ucsb.edu

Divyakant Agrawal
Department of Computer Science, UC Santa Barbara
Santa Barbara, CA, USA
agrawal@cs.ucsb.edu

Amr El Abbadi
Department of Computer Science, UC Santa Barbara
Santa Barbara, CA, USA
amr@cs.ucsb.edu

ABSTRACT

The back-end databases of web-based applications are a major data security concern to enterprises. The problem becomes more critical with the proliferation of enterprises hosted web applications in the cloud. While much prior work has concentrated on the malicious attacks that try to break into the database by using vulnerabilities of web applications, little work has focused on the threat of *data harvesting* through web form interfaces, in which large collections of the underlying data can be harvested and sensitive information can be learnt by iteratively submitting legitimate queries and analyzing the returned results for designing new queries. Although the individual data items in the database are public, data harvesting aims at accumulating large subsets of the underlying data, thus potentially revealing competitive information.

To defend against data harvesting, traditional prevention approaches such as inference control could be used, but unfortunately they hurt usability. Thus a detection approach should be used either as an alternative or complement to prevention approaches. In this paper, we summarize the characteristics of data harvesting, and propose the notions of *query correlation* and *result coverage* for data harvesting detection. We design a detection system called *HengHa*, in which Heng examines the correlation among queries in a session, and Ha evaluates the data coverage of the results of queries in the same session. Our experimental results verify the effectiveness and efficiency of HengHa for data harvesting detection.

1. INTRODUCTION

With the success of the cloud-computing paradigm due to its desirable features of scalability, elasticity, fault-tolerance, self-management, and pay-per-use, an increasing number of enterprises are migrating their applications to the cloud environments. Web-based data-intensive applications are the majority of those applications. However, data security remains the main obstacle for enterprises to host their web applications in the cloud. The vulnerability and security of enterprise data is still a significant research challenge.

A high-level classification of vulnerabilities and risks in data-rich web applications reveals two potential threats to enterprise data. The first threat is unauthorized access to the machines in the cloud that host the back-end databases. The attackers usually try to break into the back-end database by looking for vulnerabilities in a web application (e.g. insufficient input checking), submitting malicious requests, and then compromising the underlying data. A typical example is SQL injection, which has been extensively studied [35, 19, 34] and can be mitigated by solid coding practices.

The other threat arises due to the permissive intent of web-based applications which allows users from all over the Internet to interact with the underlying back-end databases through well-defined form-based query interfaces. Clearly, the goal of such web-based applications is to make this interface as usable as possible by providing informative results to the queries from their respective visitors with the hope that many of the casual visitors will be converted to legitimate clients. This is indeed the underlying business model that is prevalent amongst almost all E-Commerce web sites. Even if the database is properly protected against malicious attacks, and the application is carefully developed, we may still not be able to avoid unwanted information leakage and abuse. It is extremely difficult to prevent the adversaries from disguising themselves as ordinary visitors to a web site by submitting sequences of legitimate queries, but analyzing the returned results for designing new queries in order to dig sensitive information or gradually harvest the data inventory. Ironically, in the past few years, there has been a large body of research that addresses the issue of extracting information from such back-end databases referred to as the *hidden databases*, or *hidden web* [23].

Although data extraction from the hidden web is in general useful for public good, since it enables web-like search, it poses potential problems for the data provider in an enterprise landscape. For example, a competitor of company X, say company Y, can launch a deep web crawler to crawl product sale information from X's web site. Although the information on X's web site is searchable to the public, revealing a large number of data records may enable company Y to infer sensitive business information of X (e.g. inventory of a popular product). This type of attack launched by company Y is referred as a *crawling attack*. Occasionally, even web content crawling without malicious intent can be perceived as adverse by content owners. For example, some large internet search companies were sued for copyright infringement because they crawled book contents from library web sites [1], and news content from newspaper web sites [3] for commercialization.

Many web applications adopt a simple solution to mitigate data crawling attacks by setting a limit on the number of

searches that a client can perform per day or per session. However, in the presence of such a restriction, company Y could still perform a uniform random sampling on the data results through smart adaptive sequences of queries, resulting in company Y being able to infer approximations of sensitive aggregates of X's inventory (e.g. a popular product is in short supply) [6, 8, 7], and to adjust its own business strategy against X based on that information. This second type of attack performed by Y is referred as a *sampling attack* [8]. In addition to the above crawling and sampling attacks, unknown attacks with the purpose of harvesting data and learning sensitive information could also exist. We refer collectively to such attack activities as *data harvesting*.

Although similar problems of data inference were extensively studied in statistical and general databases [29, 9, 12], little research efforts have been put in data harvesting on hidden databases. Traditional data inference can be controlled using access control, query sets restriction or data perturbation. However, there are two problems when applying these same prevention approaches to control data harvesting. First, query sets restriction is not effective on sampling attacks, as shown in the above example. Second, query sets restriction and data perturbation hurt usability. Therefore, we propose using lightweight detection approaches as alternative or complement to prevention approaches.

Technical Challenge. Detecting data harvesting on a hidden database is a non-trivial task. Traditional analysis based on individual queries [34, 19, 20] does not work in this context, since each query is legitimate by design. A more appropriate choice would be to analyze user sessions, including web robot detection. However, such techniques typically rely on identifying special characteristics of HTTP traffic [33] and they may have difficulty in finding a deep web crawler, because a deep web crawler can be easily camouflaged as a normal user by submitting HTTP requests similar to those of normal users.

Observation. We observe that when regular users use web form interfaces for searching, in most cases they have a specific task in mind. For example, to find clothes for a particular occasion on a clothing store web site; to find bargain flight tickets and hotel rooms for a trip on a flight ticket booking web site such as priceline.com; or to find beauty and skin care products in a department store web site. In contrast, data harvesting attackers are interested in finding as broad and as diverse information as possible from the underlying database. This observation is consistent with the findings of web crawler behavior in prior research investigations [10].

Proposed Approach. We identify data harvesting attackers by examining if their search behaviors in a session show relatively significant *broadness* and *diversity*. Broadness is measured by the data that the attacker obtains from the search results compared to the entire data in the hidden database, which we formally refer to as *result coverage*. Diversity is measured by the degree of correlation among the attacker's queries in a single session, which we refer to as *query correlation*. Both approaches are needed since a sampling attack may not achieve significant broadness, and still obtains distinct pieces of data. A session that shows significant diversity in its visit might be just of an absent-minded user.

We refer to our system for detecting data harvesting as *HengHa*, which is derived from two icons *Heng* and *Ha* from Eastern mythology and who guard and protect Buddhist temples from all evil [1]. HengHa consists of two main subsystems: the *query correlation observer* (corresponding to Heng) and the *result coverage monitor* (corresponding to Ha). The Heng subsystem treats each query as an item set consisting of the predicates of the query (values of the sequentially ordered fields on the web form). It finds the query correlation of a session by mining frequent patterns [16] on the queries within the session. The Ha subsystem efficiently represents result coverage of a session with a *coverage bit vector*. It employs a model derived from the clusters of the coverage bit vectors of historical user sessions in order to find groups of sessions with different data access patterns. During online detection, the detector calculates the probability of a session being a data harvesting session based on result coverage and query correlation, and makes a decision for the suspicious session.

Contributions. We formally define data harvesting and summarize their characteristics (Section 3). We propose *HengHa*, a system for detecting data harvesting in the session level (Section 4), which includes two novel approaches of frequent pattern mining for finding query correlation (Section 5) and coverage bit vector for representing result coverage (Section 6). The approaches we propose do not require good sanitization of the training set, which makes them practical. They are able to perform online detection in close to real time with low false positive rates and 0% false negative rates (Section 8).

2. RELATED WORK

Databases are popular attack targets by both *insiders* who have direct access to the databases and *outsiders* who can only access the data in the databases through some application interfaces. Insider attackers typically perform malicious data modifications that may be prohibited by their roles. Outsider attackers typically use the vulnerabilities of the database management systems or the logical flaws in the front end applications, such as buffer overflow and insufficient input validation, to compromise the data in the underlying database. The data harvesting we consider in this paper are outsider attacks through web form interfaces.

To detect insider attacks, a rich body of work has focused on finding anomalous data access operations (*anomaly detection*) [20, 31, 24] or misuse of regular data operations (*misuse detection*) [5, 32, 18]. Anomaly detection typically builds a model of normal data access behaviors on attack free logs offline, e.g. by clustering or classification. During online detection of attacks, it compares a user's current operation with learnt normal models and raises an alarm if it finds significant deviation. Misuse detection builds rules to capture normal data operations or known attack operations. Anomaly detection and misuse detection techniques are also applicable to detecting outsider attacks. HengHa,

[1]For more interesting details, please refer to http://en.wikipedia.org/wiki/Nio

especially the Ha subsystem, corresponds to anomaly detection. Similar to [31, 24], our result coverage proposal also exploits data accessed by users for finding anomalous behaviors. However, different from these two proposals, we do not have to calculate statistics for every single query, which is expensive.

Outsider attacks, especially attacks based on web application interfaces, range from one step SQL injections to multi-step intrusions. For detecting SQL injection attacks, Valeur et al. [34] extract the structures of SQL queries and apply anomaly detection. Kruegel and Vigna [22] perform statistical analysis on the query attributes and values of HTTP requests. Huang et al. [19] apply software testing techniques to find vulnerabilities in web applications. For detecting multi-step intrusions, Vigna et al. [35] build state transition diagrams to model attacking process. To identify business logic violations in multi-request sessions, Roichman and Gudes [30] model queries in a session by a binary vector, each bit of which indicates the existence of a SQL fingerprint. This is similar to our idea of a coverage bit vector, but we model results of queries instead of the structures of queries, because data harvesting changes query submissions based on the results returned by previous queries.

In contrast to the common intrusive web based database attacks, data harvesting attackers do not exploit vulnerabilities or weak logics in web applications. Instead, they exploit the relationships between queries submitted and results returned to maximize the coverage of the underlying data or to infer sensitive aggregate information. Recent proposals on crawling the deep web [23] and sampling hidden databases [6] raise the concern of data harvesting on hidden database. The corresponding crawling and sampling attacks are discussed in more details in Section 3.

Similar problems of inferring sensitive individual information from multiple aggregation queries exists in statistical databases, in which restrictions on query sets or data perturbation are often used for inference control [29, 9]. Unlike statistical databases, web databases contain individual data records instead of statistical aggregates. Restriction or perturbation, though suitable for statistical databases, violates the open nature of E-commerce web applications. Similarly, access control that prevents data disclosure in traditional databases [12], if applied on web interfaces, would complicate normal data access of a large number of users. A recent proposal mitigates uniform random sampling attacks on hidden databases by inserting a large number of dummy tuples into the database [8]. This way of data perturbation breaks the data integrity of the database, and brings inconvenience to potential clients, because every time they search, they have to visually identify the real results from a large number of dummy results. Hence, we believe a lightweight detection approach should be used as an alternative to prevention approaches.

A closely related work to data harvesting detection is web robot detection. Tan and Kumar [33] use various characteristics of HTTP requests, such as referred page, request interval, type of requested resources as features to build a classification model for differentiating web robot sessions from normal user sessions. However, data harvesting attackers can camouflage as normal users by using similar HTTP requests to those of normal users, e.g. not leaving referred pages blank, and having longer waits between query submissions. Park et al. [26] design a smart javascript to capture user keyboard and mouse operations for differentiating humans from robots. Unfortunately, it is possible to mimic keyboard strokes by programming, which has been used in game programming. Moreover, some data harvesting activities can be semi-automatic, e.g. humans submitting queries while robots analyzing the query results and designing queries.

3. MODEL

3.1 Data and Session Model

In the following discussion, without loss of generality, we assume a web application with only one search form interface and an underlying database with only one data table for the application to search. The search form has d fields, $a_1, a_2, ..., a_d$, for a user to fill in (text box) or select a value (drop down list, radio box, etc). A field is *bound* if it is not empty, otherwise it is *free* or *unbound*. The d fields are sequentially ordered according to their natural display order on the web interface, and are mapped to d attributes in the underlying data table, D, through a one-to-one mapping. We assume D has N tuples with d attributes (data with more than d attributes can be grouped by exactly d attributes by performing a SQL GROUP BY query).

The search form can be considered as a SQL select query template with at most d predicates: SELECT $*$ FROM D WHERE $a_1 = v_1$ AND $a_2 = v_2$ AND ... AND $a_d = v_d$ LIMIT K, in which $v_i (1 \leq i \leq d)$ is an attribute value of a_i or $*$, K is a limit on the number of results returned. The purpose of this limit is either for neatness of display on the first page of results, or to prevent the entire data from being released. Once a search form is submitted to the application, a SQL select query is instantiated from the above query template by keeping only the predicates corresponding to the bound fields and filling v_i with the bound values. Although different web applications have different ranking policies on the returned answers, here we simply return the first K answer tuples that are found in D. Since the order of the fields and predicates is fixed, we abbreviate a query Q as $(v_1, v_2, ..., v_d)$, in which v_i is $*$ for an unbound field a_i.

We detect data harvesting in a single session level. Each session consists of multiple queries. We assume user sessions are clearly identified. In the special case that multiple users are behind a NAT server or they use Tor [11] to hide their IPs, the web applications can still maintain session IDs by either generating a session ID when a user first visits the web site and then appending the session ID in the urls visited by the user, or forcing users' browsers to cache the session ID [2]. In another special case where a botnet is used, existing botnet detection techniques [15] could be applied. This problem and the similar problem of detecting multi-session colluding harvesting will be further explored in our future work.

3.2 Attack Model

We assume the search service functions well and is not under any Daniel-of-Service attack. We are not concerned about attacks utilizing the vulnerabilities of a web application, such as SQL injection, cross site scripting and buffer

overflow, etc. We consider the attacks that harvest data from D and possibly learn sensitive information derived from the data. Such an attack is done by iteratively submitting legitimate queries with valid field values, analyzing the results and then designing a new sequence of legitimate queries based on the analysis outcome. We summarize two characteristics of the search behavior in such an attack session as follows:

1. *Broadness*, the data that the attacker obtains from the results of the queries covers a broad scope of the underlying data in the hidden database.
2. *Diversity*, the queries submitted by the attacker are not concentrated and localized, and they reflect very distinct intents.

We call such an attack *data harvesting*. The direct outcome of data harvesting is to compromise the ownership and privacy of the data owner. The indirect outcome could be loss of business profits, distraction of usual business activities, distraction of business analysis or web campaign based on statistics of user search activities (e.g. multivariate testing for optimizing a landing page with a search form).

Note that data harvesting is different from simply flooding queries in a very short period of time. The latter is easy to detect and block, but data harvesting is not, because it uses relatively small numbers of queries to maximize its information gain. We focus on the following two types of data harvesting attacks.

3.2.1 Crawling Attack

Crawling attacks perform deep web crawling [23] and data extraction. Assume the attacker has domain knowledge of the appropriate input values to fill in the fields of a search form. To improve the efficiency of an attack, the attacker focuses on *informative query templates*, the query templates that can produce many distinct result sets with a small number of instantiated queries. The attacker starts by submitting queries instantiated from the query templates with only one bound field, and then chooses informative query templates of one bound field and generates new trial query templates with two bound fields. The attacker keeps submitting queries, testing if a query template is informative, and generating new query templates based on previous informative query templates iteratively, until all fields are bound and no more informative query templates can be generated.

Broadness and diversity can be easily seen from the above attack process. By trying different values and binding different numbers of fields, the footprints of the attacker cover a broad scope. The informative criterion itself indicates that the queries instantiated from informative query templates are diverse.

3.2.2 Sampling Attack

Similar to crawling attacks, sampling attacks try to reduce the number of queries needed to be submitted for sampling a reasonable size of data [6, 8]. However, the focus of sampling attacks is to find the queries whose results are not truncated by the limit K, meaning that the result size is in $[1, K]$. To ensure uniformity of sampling, the attacker can only sample from the result set of these queries, because sampling from the top K results of other queries will be biased. To identify the above type of queries in the sampling process, the attacker binds more fields to a query that produces more than K results, or changes the bound field value of a query that produces more than 1 but less than K results.

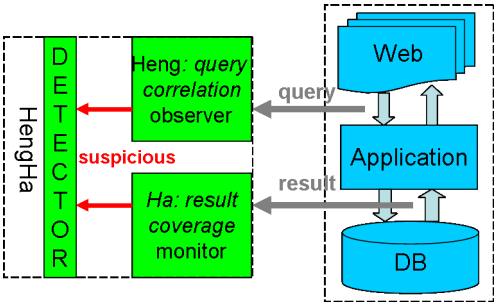

Figure 1: HengHa Detection System Architecture

Due to the uniformity of sampling, the footprints of the attacker exhibit diversity. Whether the footprints of the attacker cover a broad scope depends on the sample size and the intermediate search results for sensitive aggregates. If the expected sample size is small (this is rare since a small sample size does not provide a good estimate of the underlying data), or if the attacker successfully finds sensitive information of interest and terminates early, the footprints of the attacker may not show broadness.

4. SYSTEM ARCHITECTURE

We propose a system called *HengHa* for data harvesting detection. HengHa consists of two subsystems: a *Query Correlation Observer*, also called *Heng*, and a *Result Coverage Monitor*, also called *Ha*. Heng and Ha collect *query correlation* (Section 5) and *result coverage* (Section 6) as evidence for detecting data harvesting, and pass them to the core detector respectively. We assume a typical three layer web application, as illustrated in Fig. 1.

The directions of arrows in Fig. 1 show the data flow. The Heng subsystem is in charge of examining incoming queries submitted through the web form interface. The Ha subsystem is in charge of monitoring the outgoing results of the queries returned by the underlying database. The two subsystems are equipped with learnt behaviors of normal users sessions as models. Although they make initial decisions themselves, they report the evidence that they collect, such as the query correlation score and data coverage pattern, to the detector. The detector calculates the attack probabilities of these sessions based on the evidence and then makes a final judgement on whether these sessions are attacks.

5. QUERY CORRELATION

As mentioned in Section 1, a normal user session is usually task oriented, and therefore the queries in the session are often correlated. For example, when a user located in Santa Barbara plans to go to Chicago for business, the majority of the searches she does on a trip planning web site will be about Chicago, the flight from Santa Barbara to Chicago,

the hotels and car rentals in Chicago, etc. In contrast, as we show in Section 3, the queries of data harvesting attackers reflect diversity. In terms of crawling attacks, it was experimentally shown in [10] that web crawlers visit more distinctive resources than normal users. Similarly, data harvesting attackers fill in different binding inputs on the search form and change predicate values much more frequently in their sessions. Thus we use *query correlation*, the degree to which the queries in a session are correlated to each other, as a measurement for differentiating data harvesting attackers from normal users. Our basic hypothesis is that data harvesting sessions are bound to have lower query correlation as compared to the majority of normal user sessions.

To quantify query correlation, one straightforward way would be to calculate the distances between any two queries in a session based on some distance metric on queries (represented in predicate value vectors), and then use the sum or average distance to represent query correlation. Closer distances suggest higher correlation, while farther distances indicate lower correlation. Given the number of queries in session S, denoted by $|S|$, the time complexity of this approach is $O(|S|^2)$, which could be computationally expensive for a larger size session S.

We contend that it is not necessary to measure correlation between any two queries, as long as we can capture some common characteristics of the queries. We describe our proposal for quantifying query correlation in Section 5.1, and discuss the offline learning of the query correlation threshold value and online detection in Section 5.2 and Section 5.3, respectively.

5.1 Representation of Query Correlation

To quantify query correlation, we adopt a simple and efficient approach. From the above example, we notice that some predicate values tend to appear frequently in several queries in a normal user session, e.g. "Chicago", but it is harder to find predicate values with such high frequencies in an attacker session. Therefore, we use frequent predicate value sets to quantify query correlation in a session. More specifically, given a session S that consists of a number of queries represented by their predicate value lists, $\{Q : (v_1, v_2, ..., v_d)\}$, we mine *closed frequent predicate value sets* (called *closed patterns* in the literature [16]) on the predicate value lists of these queries. We abbreviate closed frequent predicate value set as CFS. We formally define it in the following.

Definition 1. Given a session S, a *predicate value set* $PS : (v_{p1}, v_{p2}, ..., v_{pk})$ is a subset of a query $Q : (v_1, v_2, ..., v_d)$. In another word, PS occurs in Q. The frequency of PS's occurrences in S divided by the number of queries in S, $|S|$, is called *support*, denoted as $support(PS)$. Given a threshold value of support, $support_{thres}$, a PS is *frequent* iff $support(PS) > support_{thres}$. A predicate value set PS is *closed*, if PS is frequent and there exists no super-set $PS' \supset PS$ with the same support as PS.

The reason that we mine closed frequent predicate value sets instead of all frequent predicate value sets is that a larger size of predicate value set with the same support is semantically closer to the intention of a query. To mine CFS efficiently, we use one of the fastest and well known frequent pattern mining algorithms, FP-tree [17].

Table 1: Queries in A Trip Planning Session S_1

Index	Query
Q_1	(*Santa Barbara*, *Chicago*, *April 1 2010*, *April 7 2010*)
Q_2	(*Chicago*, *4-star*, *April 1 2010*, *April 6 2010*)
Q_3	(*Chicago*, *Honda*, *April 1 2010*, *April 7 2010*)

Table 2: Closed Frequent Predicate Value Sets (CFS) in S_1 ($support_{thres} = 1/3$)

Index	CFS	support
CFS_1	(*Chicago*, *April 1 2010*)	1
CFS_2	(*Chicago*, *April 1 2010*, *April 7 2010*)	2/3

Table 3: Refined Supports of CFSs in S_1

Index	CFS	rsupport
CFS_1	(*Chicago*, *April 1 2010*)	1/3
CFS_2	(*Chicago*, *April 1 2010*, *April 7 2010*)	2/3

As an example, consider a trip planning session S_1 illustrated in Table 1. S_1 consists of three queries that search for flight tickets, hotel and car rentals. Given a support threshold $support_{thres} = 1/3$, we get two CFSs, which are shown in Table 2. Note that (*Chicago*) is not a CFS, because its support is the same as its super-set (*Chicago*, *April 1 2010*), which is a CFS.

After mining CFSs for a session S, we calculate the query correlation score of S based on the mined CFSs. For two CFSs, CFS_1 and CFS_2 s.t. $CFS_1 \subset CFS_2$, we subtract the support of CFS_1 from the support of CFS_2 to account for the unique occurrences of CFS_2. We call the new support value *refined support*, abbreviated as *rsupport*. We calculate the refined supports on all CFSs of session S. For our running example S_1, $CFS_1 \subset CFS_2$, thus $rsupport(CFS_1)$ = 1 - 2/3 = 1/3, $rsupport(CFS_2) = 2/3$, which are demonstrated in Table 3.

Let the operator $||$ be the size of a set, so $|S|$ is the number of queries in a session S, $|Q|$ is the size of the predicate value list of a query Q, and $|PS|$ is the size of a predicate value set PS. Let p be the number of CFSs of session S. Then the query correlation score of a session S, $qc(S)$ is defined as

$$qc(S) = |S| \times \sum_{i=1}^{p} (rsupport(CFS_i) \times |CFS_i|) / \sum_{j=1}^{|S|} |Q_j| \quad (1)$$

For our running example S_1, $qc(S_1) = 3 \times (1/3 \times 2 + 2/3 \times 3)/(4 + 4 + 4) = 2/3$.

Intuitively, if a session has more CFSs with higher supports, and these CFSs are more similar to the queries (they are longer), the queries in this session are correlated, which leads to a high query correlation score. Note that the number of CFSs, p is often smaller than $|S|$. Mining CFSs with FP-tree takes $O(|S|)$ time. Calculating the refined support of a CFS can be done by traversing the FP-tree and subtracting

the support of a child node from the support of the node corresponding to CFS. Calculating the refined supports of all CFSs can be done in $O(p)$ time. Thus, the time complexity for calculating the query correlation score of a session is $O(|S|)$.

Finer grained query correlation may be obtained by exploring the semantics of queries. For instances, divide a predicate value into words and apply frequent pattern mining on the word set. Alternatively, identify two values or two words as semantically close if they co-occur in the same data tuple. However, finer grained calculation is computationally expensive, which prohibits the Heng subsystem to respond quickly. It may also be too application specific and not as general as the above definition of query correlation score.

5.2 Offline Learning

To learn an empirical threshold for query correlation, qc_{thres}, of normal user sessions, we calculate the query correlation scores for all training sessions, and then set qc_{thres} to be the lower bound of query correlation scores of training sessions. Ideally, if the training sessions are purely normal, qc_{thres} can be set to the lowest query correlation score of the training sessions. However in practice, historical sessions as training sessions could also contain attack sessions, thus using the lowest query correlation score of the training sessions as qc_{thres} may produce a lot of false negatives.

To solve this problem, we use a statistical outliers test, Grubbs' test [14], to extract query correlation score outliers from the training sessions, and then set qc_{thres} to be the mean of the query correlation scores of all the outliers. Given a significance level α and the number of training sessions M, we categorize the query correlation of a training session, qc, as an outlier using the following Grubbs' one-sided test

$$G = \frac{\overline{qc} - qc}{s} > \frac{M-1}{\sqrt{M}} \sqrt{\frac{t^2_{(\alpha/M,M-2)}}{M-2+t^2_{(\alpha/M,M-2)}}} \qquad (2)$$

where $\overline{qc}$ and s are the mean and standard deviation of the query correlation scores of training sessions, and $t^2_{(\alpha/M,M-2)}$ is the critical value of the t-distribution with $(M-2)$ degrees of freedom and a significance level of α/M.

5.3 Online Detection

The learnt threshold qc_{thres} is used to identify suspicious sessions in online detection. To examine a recently ended new session S_{new}, Heng mines CFSs on S_{new} using the FP-tree algorithm, and then uses Formula (1) to calculate the query correlation score of S_{new}, $qc(S_{new})$. If

$$qc(S_{new}) < qc_{thres} \qquad (3)$$

Heng marks S_{new} as a suspicious attack session. The time complexity for online mining of CFSs and calculation of query correlation score is $O(|S_{new}|)$, depending on the number of queries in the session S_{new}.

Since it is very hard for real time detection to be accurate due to the lack of full footprints of the attacks' actions, a compromised approach could be to wait until the number of queries in a session reaches a predefined threshold, and start calculating query correlation scores immediately.

6. RESULT COVERAGE

Not only are the queries in data harvesting sessions less correlated than the queries in normal user sessions, but they are also more likely to cover a broader scope of the hidden data. For example, it was experimentally shown in [10] that web crawlers' visits are more exhaustive than the visits of the average normal users. This behavior applies to deep web crawlers as well. Thus we use *result coverage*, the result coverage of queries on the entire data in the hidden data table, or the proximity of result data distribution to the distribution of the entire data, as our second line of defense for differentiating data harvesting attackers from normal users. Note that result coverage does not necessarily mean big result sets, since a big result set could be obtained from certain parts of data without spanning across the entire data set. Result coverage and query correlation are also not substitutes for each other. Result coverage helps differentiate real attackers from casual browsing users, while query correlation helps catch attackers who do not want to expose themselves by visiting too much data or sampling attackers whose sample sizes are not very large.

To evaluate result coverage on the data with multiple attributes, we consider D as a d-dimensional data space, and identify the subsets, or portions of the space accessed by queries of a session. Normal users' accesses are usually localized, so the portions of space that they access are likely to be concentrated in some parts while leaving other parts untouched. In contrast, attackers' accesses are broad, so the portions of space that they access are likely to be scattered everywhere. We capture different kinds of access patterns of normal users by clustering the access patterns of training sessions, and use a small number of cluster centers to represent large numbers of access patterns in different clusters. In online detection, a new session whose access pattern deviates from the learnt access patterns of the cluster centers is reported as a suspicious attack session.

To quantify access patterns, one may consider building a spatial index tree such as a k-d tree [13] on D, and represent access patterns as accessed tree paths or tree nodes. However, this approach is not easy to apply nor efficient in online detection, since it requires a top-down tree traversal for each query. Instead, we propose an efficient approach with a much simpler representation, *coverage bit vector*, for access patterns. We describe this representation in Section 6.1, and discuss offline clustering and online detection in Section 6.2 and Section 6.3, respectively.

6.1 Representation of Result Coverage

First, we use a linearization transformation to map a multidimensional data tuple to a single integer, $f : D \rightarrow I$. If the number of attributes in D are exactly the same as the number of fields on the search form, d, f is a one-to-one mapping. Otherwise, the data tuples are grouped by the d attributes by performing a SQL GROUP BY query, and f maps each group of data tuples to a single integer. The transformed integers do not need to be consecutive, but f maps two data tuples that are close in the original multi-dimensional data space to two integers that are close (*locality-preserving*). Then, we sort the transformed integers in ascending order. We create a bit vector in which the bits correspond to the ordered transformed integers one by

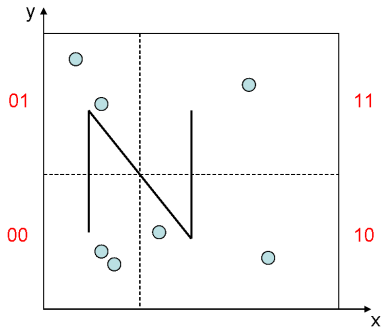

Figure 2: Example of Z-curve Transformation

one. We say a combination of attribute values is *valid*, if it appears in D. The size of the bit vector equals the number of valid combinations of the d attributes. We call this bit vector a *coverage bit vector*, abbreviated as CBV, and denote the size of CBV as $|CBV|$. Let the mapping of a transformed integer to a bit be $h : I \rightarrow B$. When a session starts, we initialize a CBV and set all its bits to 0. For any result tuple of a query, t that is mapped to a bit i in CBV by mapping $h(f(t)) = 1$, we set $CBV(i) = 1$. The resulting CBV at the end of the session captures result coverage.

Intuitively, since the data access of a normal user session is usually localized, the CBV of a normal user session has dense 1 bits in some parts, and a long consecutive range of 0 bits in most other regions of the vector. In contrast, the CBV of a data harvesting session would have many interleaving 0 and 1 bits.

Linearization transformation. We use a popular space filling curve, z-curve [25] as the mapping f. Define an order on the d attributes, $A_1, A_2, ..., A_d$. We start from the entire data space. The data space is partitioned into two halves along dimension A_i, in which the two halves have approximately an equal number of data tuples and i is the next dimension mod d. The lower half is assigned a code 0 and the upper half a 1. The newly assigned code is appended to the code of the pre-partitioned space. The partitioning of the data space continues with each subspace being assigned a binary code, until the granularity of the subspaces is acceptable, e.g. each sub space has no more than a predefined number of data points. Considering the final binary codes as binary numbers, the z-curve is a curve that traverses these sub spaces in the order of their binary codes. An example is shown in Fig. 2. Finally, the binary codes are transformed to decimal integers. We can see that the z-curve transformation tries to preserve locality of original data tuples, but it is not an exact transformation.

Categoric attribute values with hierarchical relationships can be transformed together. Fig. 3 illustrates an example of assigning binary codes to the attributes *Department* and *Category* in a clothes store database. Since attribute *Department* has three values, *Men*, *Women* and *Children*, we use two bit binary codes to represent them. In each department, the category values are assigned binary codes in the same way as the department values. Finally, the binary codes from department and category are concatenated. For example, in

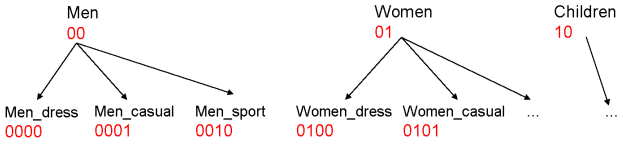

Figure 3: Example of Hierarchical Attributes Transformation

men's department, *Men_dress* is assigned a category code 00 and becomes 0000, and *Men_casual* is assigned a category code 01 and becomes 0001, etc.

6.2 Offline Learning

Given two CBVs, CBV_x and CBV_y, a natural way to measure the difference of their access patterns is to count the number of different bits, which we denote as $diff(CBV_x, CBV_y)$. To model different kinds of normal data access patterns in a systematic way, we cluster the CBVs of training sessions and use the resulting clusters to represent different categories of normal data access patterns. Since $CBV_x(i) - CBV_y(i) = 0, 1$ for any bit i $(1 \leq i \leq |CBV|)$,

$$diff(CBV_x, CBV_y) = \sum_i |CBV_x(i) - CBV_y(i)| \quad (4)$$

$$= \sum_i (CBV_x(i) - CBV_y(i))^2 \quad (5)$$

$$= d(CBV_x, CBV_y)^2 \quad (6)$$

where $d(CBV_x, CBV_y)$ is the Euclidean distance between CBV_x and CBV_y. This result suggests that we can use the Euclidean distance to measure the difference of data access patterns, and we are safe to use a Euclidean distance based clustering algorithm such as k-means for clustering. We choose k-means [16] and use an efficient variant of it [27, 28] [2]. We abbreviate *Euclidean distance* as *distance* in the following.

The CBV of a normal user session is expected to be close to one of the learnt cluster centers, $Clust_{norm}$. Although $Clust_{norm}$ is not a CBV, $|Clust_{norm}| = |CBV|$ and each value of the $Clust_{norm}$ vector is in $[0, 1]$. It is easy to see that $d(CBV_x, CBV_y) \leq \sqrt{|CBV|}$ and $d(CBV, Clust_{norm}) \leq \sqrt{|CBV|}$. During clustering, we calculate the *cluster diameters*, the farthest distance of a cluster member to its corresponding cluster center, and set the mean of cluster diameters as the upper bound distance threshold, d_{thres}, for identifying normal user sessions.

6.3 Online Detection

The learnt threshold d_{thres} is used to identify suspicious sessions in online detection. To examine a new session S_{new}, Ha initializes a coverage bit vector CBV_{new} with all bits set to 0 when S_{new} starts, and sets the bit corresponding to any returned data tuple t, $CBV_{new}(h(f(t))) = 1$

[2] k-means is not always suitable for clustering high dimensional numeric data, but it works well for CBVs. Because the value in any dimension of CBV is binary, CBV is much less affected by the curse of the dimensionality. The experimental results in Section 8 confirm that the clustering result of k-means is effective in data harvesting detection.

as S_{new} gets results to a query submitted. Then it calculates the distances of CBV_{new} to the centers of the normal clusters, and picks the smallest distance among them, $min(d(CBV_{new}, Clust_{norm}))$. If

$$min(d(CBV_{new}, Clust_{norm})) > d_{thres} \quad (7)$$

Ha marks S_{new} as a suspicious attack session. The time complexity for finding and setting a bit in CBV_{new} is O(1) by two table lookups through mapping $h(f())$. The worst case time complexity for online distance calculation and comparison is $O(N)$, when the number of attributes in D is d, and the size of the coverage bit vector equals the number of data tuples in D.

Similar to Heng, Ha does not have to wait until session S_{new} is terminated to initiate detection. It can start calculating $min(d(CBV_{new}, Clust_{norm}))$ when the number of queries in a session reaches a predefined threshold.

7. DETECTOR

During online detection, for each new session S_{new}, the detector collects the query correlation score, $qc(S_{new})$, and the smallest distance of the coverage bit vector to cluster centers, $min(d(CBV_{new}, Clust_{norm}))$, and calculates the probability of S_{new} being an attack session.

Since the Euclidean distance $d(CBV_x, CBV_y) \leq \sqrt{|CBV|}$ and $d(CBV, Clust_{norm}) \leq \sqrt{|CBV|}$, we normalize d by dividing $\sqrt{|CBV|}$ and derive a normalized distance, $nd = \frac{d}{\sqrt{|CBV|}}$. Similarly, we normalize the distance threshold d_{thres} and get $nd_{thres} = \frac{d_{thres}}{\sqrt{|CBV|}}$. Hence, the probability of a session S being an attack session is

$$P_a(S) = \frac{min(nd(CBV_S, Clust_{norm}))}{1 + qc(S)} \quad (8)$$

Since $nd(CBV_S, Clust_{norm}) \leq 1$, we guarantee $P_a(S) \leq 1$. Intuitively, the farther a session's CBV is from the learnt cluster centers, the lower the query correlation is, the higher the probability of the session being data harvesting. We define the threshold of $P_a(S)$ as

$$P_{thres} = \frac{nd_{thres}}{1 + qc_{thres}} \quad (9)$$

If $P_a(S) > P_{thres}$, the detector flags S as an attack session.

Based on the time complexity for query correlation calculation $O(|S|)$ and the worst time complexity for result coverage analysis $O(N)$, the time complexity for the entire online detection is linear to the length of the session and the size of the database $O(max(|S|, N))$, thus detection represents little overhead.

8. EXPERIMENT EVALUATION

We evaluate the effectiveness and efficiency of HengHa, the stand-alone query correlation observer, Heng, and the stand-alone result coverage monitor, Ha, for detecting data harvesting in terms of false positive rate, false negative rate and online detection time. The evaluation of running only one of the Heng, Ha subsystems for detection is valuable, since in some applications, not both queries and results are available. Experimental results on a real user session set with synthesized attack sessions show that both HengHa and its two subsystems are able to achieve very low false positive rates and 0% false negative rates, and they are very efficient in online detection.

8.1 Experimental Setup

We implemented HengHa in Java. We used the FP-tree implementation in [4] for mining frequent predicate value sets (Section 5.1), and the k-means implementation in [27, 28] for clustering coverage bit vectors (Section 6.2). We implemented the crawling attack following [23] and the sampling attack following [6, 8]. To camouflage the attacker as a normal user, we generated waits between consecutive queries that follows a Pareto distribution.

We used a real clickstream data set from KDD Cup 2000 [21]. It contains the clickstream log from Gazelle.com, a legwear and legcare web retailer which closed their online store in 2000. To facilitate business analysis, this log does not only record HTTP requests, but also records the products and related assortments that users clicked on. Sessions are clearly separated in the log. The original clickstream data has 234,954 sessions with 777,480 HTTP requests. Note that a real data set with real query sessions is difficult to obtain for evaluation. Hence, we recovered session queries from HTTP requests and recorded product assortments. These recovered queries, though not exact due to some unknown predicates, approximate real user queries. We removed the sessions without queries and removed the HTTP requests that are not queries, e.g. editing account profile, and checkout. After this refinement, we had 99,169 sessions with 466,638 product queries or views, which we used in our experiments. Since we do not have the database of Gazelle.com, we reproduced a partial product data table by collecting the products that users clicked on, which resulted in a partial product database of 387 products. Note that this partial product data is still meaningful in evaluating result coverage, since it represents the popular subset of the data.

We set the limit on the size of results per web page, K, to 10, and just returned the first available 10 results for a query. We synthesized 1000 random attack sessions, in which 40% of them are crawling attack sessions implemented according to [23] and 60% of them are sampling attack sessions implemented based on the algorithm in [6, 8]. The informative threshold for evaluating a query template in a crawling attack is set to be 6, meaning that on the average, each of the queries instantiated from a informative query template has 6 distinct results. For synthesizing sampling attacks, we randomly generated a desired sample size between 5% and 50%.

8.2 Experiment Results

Note that the 99,169 sessions are not pure normal user sessions. By manual observation on randomly picked sessions, we found some obvious data harvesting sessions exist, e.g. there are two sessions with thousands of queries that almost cover each finest grained category of the products. To extract a session set of better quality for offline learning, we collected the average number of distinct queries per minute, *QPM*, for each of the 99,169 sessions, and performed Grubbs' test for outliers [14] to remove 605 outlier sessions

that have an excessively large number of queries per minute. The smallest QPM in an outlier sessions is 7.3. Note that this cleaning step is very coarse, since some of the manually observed obvious data harvesting sessions still exist after removing the outliers. We set the lower bound waits between queries in an attack to 10 seconds, thus our synthesized attack sessions cannot be detected by simple statistical outlier tests based on measurements on time and the number of queries, such as the QPM we used for cleaning training sessions. Since the data harvesting we considered in this paper tries to maximize the information gain while minimizing the number of queries submitted, the duration of an attack session is generally not long. A synthesized attack session that we generated for KDD CUP 2000 data typically terminates in an hour with between 78 and 158 query submissions.

After removing outliers, we performed four folds cross validation detection on the remaining 98,564 sessions. In each validation, 3/4 of the 98,564 sessions are used for training, and the rest 1/4 along with the 1000 synthesized attack sessions are used for testing. Our testbed is a Linux server with Intel 2.40GHz CPU and 3GB memory, running Federal Core 8 OS.

We evaluated the effectiveness and efficiency of HengHa, the Heng and Ha subsystems for detecting crawling and sampling attacks. Specifically, HengHa relies on the detector (Section 7) and the Heng, Ha subsystems for detecting attacks. The Heng subsystem relies on Formula (3), while the Ha subsystem relies on Formula (7) for detecting attacks.

Table 4 summarizes the false positive and false negative rates of the three systems. Here the false positive rate and false negative rate are abbreviated as FPR and FNR respectively. The results suggest that HengHa and the two proposals for query correlation and result coverage are effective for data harvesting detection. The false positive rates are very low, which means that normal user sessions are almost never suspended and we save the system administrator from checking too many potential attack sessions. The false negative rates are all zero, which means that all attacks are caught. HengHa performs better than the Heng subsystem in all validations. An interesting phenomenon is that the false positive rate of HengHa is slightly higher than the standalone Ha subsystem. The reason is because we consider the 98,564 sessions as "normal" in calculating the false positive and false negative rates, but actual attacks might exist in this "normal" training set, given that it is extremely difficult to obtain a sanitized data set and the preprocessing we have is very simple.

Table 5 summarizes the detection calculation time per session of the three systems. The time is measured in milliseconds. We can see that both HengHa and the two subsystems respond very quickly in online detection. This helps the web applications to locate the attackers and block them from further harvesting from the underlying databases. Heng spends most of its time on mining closed frequent patterns (the detection time of Heng should be less in practice, since we used a stand alone FP-tree miner and file IOs for feeding input to the miner and processing its output are also included). In contrast, Ha only maintains the coverage bit vector and calculates its distance to learnt cluster centers during online detection, so it takes much less time than Heng.

Table 4: Effectiveness for Detecting Data Harvesting

	System	FPR(%)	FNR(%)
1	*HengHa*	0.21	0
	Heng:Query Correlation Observer	0.71	0
	Ha:Result Coverage Monitor	0.18	0
2	*HengHa*	0.06	0
	Heng:Query Correlation Observer	0.26	0
	Ha:Result Coverage Monitor	0.05	0
3	*HengHa*	0.06	0
	Heng:Query Correlation Observer	0.13	0
	Ha:Result Coverage Monitor	0.06	0
4	*HengHa*	0.24	0
	Heng:Query Correlation Observer	0.55	0
	Ha:Result Coverage Monitor	0.24	0

Table 5: Efficiency for Detecting Data Harvesting

	System	Detection Time (ms)
1	*HengHa*	8.787
	Heng:Query Correlation Observer	8.182
	Ha:Result Coverage Monitor	0.812
2	*HengHa*	6.276
	Heng:Query Correlation Observer	5.909
	Ha:Result Coverage Monitor	0.515
3	*HengHa*	7.863
	Heng:Query Correlation Observer	7.115
	Ha:Result Coverage Monitor	0.392
4	*HengHa*	6.12
	Heng:Query Correlation Observer	5.97
	Ha:Result Coverage Monitor	0.294

In summary, HengHa and its subsystems are both effective and efficient in online detection of data harvesting sessions.

9. CONCLUSION

This paper considers emerging attacks on the back-end databases of enterprise web applications, which harvest the data and learn sensitive aggregate information from the databases through web form interfaces by iteratively submitting legitimate queries and analyzing the returned results for designing new queries. As more enterprise web applications migrate to cloud, such attacks will be more prevalent. We refer to such attacks as *data harvesting*. We have identified two types of data harvesting, *crawling attack* and *sampling attack*, based on previous work, and summarized their characteristics as *broadness* and *diversity*. We have proposed a data harvesting detection system called *HengHa*, which uses *result coverage* and *query correlation* to capture broadness and diversity, respectively. The effectiveness and efficiency of HengHa for detecting data harvesting has been verified in the experiment on a real data set.

10. ACKNOWLEDGEMENT

We wish to thank Blue Martini Software for contributing the KDD Cup 2000 data.

11. REFERENCES

[1] Copyright lawsuit targets google. http://www.wired.com/techbiz/media/news/2005/09/68928, 2005.

[2] Tracking users via the browser's cache. http://yro.slashdot.org/article.pl?sid=06/09/17/2126210, 2006.

[3] Murdoch wants a google rebellion. http://www.forbes.com/2009/04/03/rupert-murdoch-google-business-media-murdoch.html, 2009.

[4] C. Borgelt. An implementation of the fp-growth algorithm. In *OSDM '05: Proceedings of the 1st international workshop on open source data mining*, pages 1–5, New York, NY, USA, 2005. ACM.

[5] C. Y. Chung, M. Gertz, and K. N. Levitt. Demids: A misuse detection system for database systems. In *IICIS*, pages 159–178, 1999.

[6] A. Dasgupta, G. Das, and H. Mannila. A random walk approach to sampling hidden databases. In *SIGMOD Conference*, pages 629–640, 2007.

[7] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das. Unbiased estimation of size and other aggregates over hidden web databases. In *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*, pages 855–866, 2010.

[8] A. Dasgupta, N. Zhang, G. Das, and S. Chaudhuri. Privacy preservation of aggregates in hidden databases: why and how? In *SIGMOD Conference*, pages 153–164, 2009.

[9] D. E. Denning and J. Schlorer. Inference controls for statistical databases. *Computer*, 16(7):69–82, 1983.

[10] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou. An investigation of web crawler behavior: characterization and metrics. *Computer Communications*, 28(8):880–897, 2005.

[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 21–21, 2004.

[12] C. Farkas and S. Jajodia. The inference problem: a survey. *SIGKDD Explor. Newsl.*, 4(2):6–11, 2002.

[13] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[14] F. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[15] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *SS'08: Proceedings of the 17th conference on Security symposium*, pages 139–154, 2008.

[16] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[17] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA, 2000. ACM.

[18] Y. Hu and B. Panda. Identification of malicious transactions in database systems. In *IDEAS*, pages 329–335, 2003.

[19] Y.-W. Huang, S.-K. Huang, T.-P. Lin, and C.-H. Tsai. Web application security assessment by fault injection and behavior monitoring. In *WWW*, pages 148–159, 2003.

[20] A. Kamra, E. Terzi, and E. Bertino. Detecting anomalous access patterns in relational databases. *VLDB J.*, 17(5):1063–1077, 2008.

[21] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000.

[22] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261, New York, NY, USA, 2003. ACM.

[23] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Y. Halevy. Google's deep web crawl. *PVLDB*, 1(2):1241–1252, 2008.

[24] S. Mathew, M. Petropoulos, H. Ngo, and S. Upadhyaya. A data-centric approach to insider attack detection in database systems. Technical report, Department of Computer Science and Engineering, University at Buffalo, 2009.

[25] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *PODS*, pages 181–190, 1984.

[26] K. Park, V. S. Pai, K.-W. Lee, and S. Calo. Securing web service by automatic robot detection. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 23–23, Berkeley, CA, USA, 2006. USENIX Association.

[27] D. Pelleg and A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 277–281, New York, NY, USA, 1999. ACM.

[28] D. Pelleg and A. W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[29] D. E. Robling Denning. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982.

[30] A. Roichman and E. Gudes. Diweda - detecting intrusions in web databases. In *DBSec*, pages 313–329, 2008.

[31] A. Spalka and J. Lehnhardt. A comprehensive approach to anomaly detection in relational databases. In *DBSec*, pages 207–221, 2005.

[32] A. Srivastava, S. Sural, and A. K. Majumdar. Database intrusion detection using weighted sequence mining. *JCP*, 1(4):8–17, 2006.

[33] P.-N. Tan and V. Kumar. Discovery of web robot sessions based on their navigational patterns. *Data Min. Knowl. Discov.*, 6(1):9–35, 2002.

[34] F. Valeur, D. Mutz, and G. Vigna. A learning-based approach to the detection of sql attacks. In *DIMVA*, pages 123–140, 2005.

[35] G. Vigna, W. Robertson, V. Kher, and R. Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003)*, pages 34–43, Las Vegas, NV, December 2003.