A Survey of Database Inference Attack Prevention Methods

Michael Hylkema

12/2009

Abstract

Inference attacks are notoriously hard to mitigate due to the fact that some data always needs to be made available to legitimate sources. It's difficult to prevent a determined individual from connecting available non-sensitive data and making inferences about more sensitive data. With more databases reachable from the web, this opens numerous opportunities for hackers to gain knowledge about sensitive or confidential data which they should not have.

This paper attempts a thorough coverage of the advancements in methods of inference attack detection and prevention. There are a number of methods each with it's own advantages and disadvantages. There are design models which contribute to the mitigation of these attacks as well as functionality which can be added to database front-end and/or back-end.

This paper pulls together information on each of the methods and models of inference detection / prevention and attempts summarize them in an easy to understand manner. I'll also discuss the usefulness of each in different situations and suggest where two or more methods may compliment each other well. Lastly I'll note which currently available products support any methods for implementing these techniques and how they might be used to accomplish it effectively.

Introduction

Evolution of information security threats.

The concept of information security has been around for as long as there's been information worth securing. The classic CIA model (Confidentiality, Integrity, Availability) has been the cornerstone of information security and a number of systems have been built with this model in mind. While all vectors in the model are related, this paper will mainly focus on the concept of confidentiality. Confidentiality in information security refers to an assurance that information is shared only among authorized persons or organizations. For example, if a malicious user is able to capture and decrypt encoded information which he is not authorized to view this would be considered an attack on the confidentiality of that data. Many strategies have been developed over the years to protect against these types of attacks, both technical (cryptography, access control) and process-related (least privilege, paperwork control). More and more, however, both information security professionals and hackers (sometimes hackers first) have been shifting their focus on deriving sensitive data from available nonsensitive data. The nonsensitive data may not even be obviously related to the data being derived. There is an entire field of research focused on this method of obtaining data, and it is called inference. Inferences can be made by analyzing a number of different data sources, but the most common sources are databases so we will focus our analysis there.

What Database Inference is and is not

Database Inference Is:

Database inference is not easily categorized in any other group of information security attacks. This is due to the fact that an inference attack leverages the human mind, or similar logic systems, in order to obtain data that may be considered "secure" in the traditional sense. There are many definitions of what an inference is, but in the context of database security it is defined as the act or process of deriving sensitive information from premises known or assumed to be true. The "premises known or assumed to be true" may be freely/publicly available information or information gleaned through other methods.

Database Inference Isn't:

I have noticed in my research that there is some confusion about what constitutes "inferencing". Guessing at data without confidence is not inferencing. Inferences are made with some level of statistical confidence. SQL Injection also is not inferencing. While SQL Injection can be used to mine data or to test inferences, it is not itself inferencing. Lastly, breaking encryption is not inferencing. I believe the confusion is that you're obtaining sensitive data from something which may have been publicly available (encrypted documents) which is similar to the goals of inferencing. Cryptanalysis may be assisted by inferencing techniques to obtain keys to decrypt data, but is not itself inferencing.

Where common security models fall short

Today, most commercial MLS database systems supply functionality similar to that outlined in the Bell-Lapadula model. Mandatory and/or Discretionary access control (MAC/DAC) is common. As far as models specifically designed to mitigate database inference attacks, there are none which have yet been widely adopted. Outside of suggestions made in research papers and proprietary implementations. For the most part, these types of security features have been written into application logic rather than the DBMS itself. Organizations develop their own custom software to address their specific security needs. This is good, however doesn't solve the problem of someone directly accessing the database and bypassing the security features of the front-end software.

What to look for: Data Mining

When an attacker attempts inferencing they generally have some idea what they are looking for. They may start out with that knowledge, or they may not. How would they initially know what they should be looking for? Data Mining is a technique used to gather data and find frequent patterns, find associations between data and build rules for those associations and patterns. For example, data mining techniques may determine that there are a lot of references to particular words or phrases in a group of documents stored in a database. It may also determine that there are associations which can be predicted (if a group of data contains items a and b, it is highly likely to also contain item c). These predictions can be formed into rules which can be applied using inferencing techniques to infer missing or restricted data. Data mining can utilize any collected data, although generally publicly available (via the web) sources are used. Data may also be found accidentally, or through social engineering.

Vulnerabilities

There are a number of vulnerabilities which can lead to making inferences much more easily. A good database administrator should be aware of these and make every effort to minimize them.

Inconsistent security classifications for replicated data

Most databases will unavoidably have some amount of replication. It is vital to ensure that all replicated data is classified uniformly. If even small amounts of sensitive data can be collected elsewhere with lesser privileges the opportunities to build association rules through data mining significantly increase the chances of successful inferencing.

Insufficiently restricted data

Similar to the previous vulnerability, not sufficiently restricting data to begin with is of course a problem as well. Understanding the data and what should be restricted is crucial. Insufficiently

restricted data are prime candidates for data mining and again significantly increase the chances of successful inferencing.

N-item k-percent rule violations

This rule applies to statistical data sets where only aggregate queries have been allowed. It says that whenever a query is made some number of items (N) should not represent greater than a certain percentage (k) of the result reported. This is to ensure that a where clause isn't added to an aggregate query which reduces the rows calculated to few enough to to infer specific data items. The most obvious case is where 1-item represents 100-percent of the result. In other words a where clause has been tailored to return only a single row value which represents the entire result, therefor defeating the requirement that only aggregate queries be allowed.

Unencrypted Index

While secure databases are often encrypted, the indexes frequently remain unencrypted for quicker access. Indexes are used to make searches and certain queries run more quickly. Encrypting them defeats this purpose to some extent and therefore frequently the index are left in plain text. Data from unencrypted indexes can be used to piece together closely related data just by noting table names and keys.

What's at Stake?

Database inference is an information security issue. Whenever we talk about information security we think about the CIA model (Confidentiality, Integrity, Availability). Database inferencing is all about compromising confidentiality. The end result of a successful inference attack is equivalent to a leak of sensitive information. Even if actual information is not leaked, certain statistics about that information can provide enough information to make inferences which may still constitute a legitimate breach.

Methods of Attack

There are several different methods used for effecting inference attacks. They can be used individually, or more commonly (and most effectively) in conjunction with each other.

Out of Channel Attacks

"Out of Channel" refers to using information from outside sources to attack the target database. Most inference attacks are effected using at least some out of channel data, but it's not necessary. Extensive data mining of numerous publicly accessible information sources and using that data to infer data in a secured database is a good example. Out of channel attacks are extremely difficult to guard against as frequently the data is out of the control of the target. The web makes all types of information easily available and search-able. It's not always possible or feasible to remove these sources of information.

Direct Attacks

These are attacks directly on the target database. They seek to find sensitive information directly with queries that yield only a few records. These are the easiest to detect and deny. MAC and DAC methods can mitigate these types of attacks by ensuring data is properly classified. Similarly, triggers can be written to ensure that queries conform to security policy standards. Direct attacks are most effective when database security is lax or systems have been misconfigured.

Indirect Attacks

Indirect attacks seek to infer the final result based on a number of intermediate results. Intermediate result may be obtained by aggregate (Sum, Count, Median, etc) or set theory. A number of complex and surprisingly effective techniques can be used. Intersections of sets can be examined. With statistical databases linear systems of equations can be utilized to solve for missing (sensitive) data values.

Inferencing Categories

Logical Inferences

Uses association rules such as those gleaned from data mining to make logical assumptions about data. If a,b,c $\rightarrow$ d and a,b,e $\rightarrow$ d then probably a,b,f $\rightarrow$ d. Logical inferences are most commonly used to make associations between textual data. Techniques borrowed from data mining such as apriori and clustering. These generally fall under the category of direct attacks, but can also be considered indirect when more complex methods are used.

Statistical Inferences

Takes aggregate data and uses math/statistics to derive data pertaining to individuals in the data set. Statistical inferencing is generally applied to numerical data sets but can be extended to use with textual data. Textual data can easily be enumerated or represented as frequencies or counts. The same statistical methods can then be used to derive associations. These generally fall under the category of indirect attacks since result are based on a combination (sometimes quite complex) of intermediate results frequently based on aggregate data.

Statistical Inference Example

Statistical inferencing can be extremely powerful as we'll see in this example. Consider the following:

- We have 5 people who have associations within three groups.
- Our database only allows aggregate queries, for example averages by group.
- Our group averages are as follows: {P1,P2,P3} , {P2,P3,P4,P5} , and {P4,P5}
- Our goal is to find P1
- The average of {P2,P3,P4,P5} is equal to ({P2,P3} + {P4,P5}) / 2

- We know {P4,P5} so we can compute {P2,P3}= ({P2,P3,P4,P5}*2)-{P4,P5}
- Since there are only two items in this average, {P2,P3} multiplied by 2 gives us (P2 + P3).
- We can now solve the equation for the average of {P1,P2,P3}, which is (P1+P2+P3) / 3.
- P1={P1,P2,P3}*3-(P2+P3) : We've found the salary of Employee 1!

Keep in mind this is a very simple system of equations. Much more intricate systems are possible with larger more complex data sets.

Mitigation Controls

Inference attacks on the whole are quite difficult to defend against as we live in an extremely data rich world and it's nearly impossible to completely control how information is accessed and distributed. There are a number of methods which have proven at least somewhat effective in controlling this, and we'll describe them here. Simply denying data to unauthorized users is the most obvious way to mitigate an inference type attack. There are various controls which can be put into place to deny information to the querying user. The two major categories of controls are data item controls which are applied to individual database items and query controls which are applied to queries run against the database.

Data Item Controls

Data item conrols focus on denying certain information to the querying user by applying some policy which states whether they are authorized to have the data. Information may be completely denied or modified in some way to "sanitize" it. Data item controls are primarily useful against direct attacks.

Suppression

Suppression is just what it sounds like, not providing some data to the querying user. Particular combinations of rows and columns can be used to limit result sets. For statistical databases we could provide only combinations of results to conceal underlying information. This is accomplished by rounding, presenting a range of results, or by presenting a random sample of results.

Concealing

Another method of inference prevention is to modify the data being returned to certain queries. There are many variations of this including combining, approximation or rounding, presenting a range of results, presenting a random sampling of results, and adding some level of error to the returned data.

Random Data Perturbation

Random data perturbation (RDP) is often used in statistical databases to prevent inference of sensitive information about individuals from legitimate aggregate queries. It operates by randomly adding some error to the result set(s) returned. The error may be + or - some degree or percentage, or it may be totally fictitious depending on the need or circumstances of the query.

Partitioning

Partitioning means simply to separate our data, often times physically, depending on classification of the information. It is also referred to as content dependent access control. It can be done at the table level or the database level. The idea is to create multiple "fragments" of the database for different access levels. This is similar to what many government agencies do. It ensure security, however adds considerable redundancy as much data will live in several "fragments" of the database. This also adds complexity to administration and resources needed to manage and store the information.

Polyinstantiation

Polyinstantiation can be used in databases with MAC support to avoid inference. It is a basic mechanism in databases with multiple levels of security. It allows administrators to classify every row in a table, ensuring access to sensitive data is restricted to users with the appropriate clearance level. In other words there could be several different records containing information about a subject, each of which contain slightly different data. Each record would be tagged by a security level so that it could be accessed only by those who have clearance sufficient to access it. This technique also allows for the creation of "cover stories". This means that every query on any subject would return some information, however depending on the querying users clearance level, the data returned may be less accurate, more generalized or entirely false. This solves some of the issues with suppression or obvious concealing since data is returned, but it may or may not be factual.

Query Controls

Query controls parse either the incoming query, the results from the execution of the query or both. If the query does not conform to a set of standards or it is deemed that an inference could have been made from a combination of records requested, the query may be denied. Likewise, if the results of a query are such that an inference may be made, the query may be denied. Complex statistical methods are used to determine the likelihood that an inference is made.The two fundamental ways to pre-process are to either check the query before it is executed or after. Ideally of course you would do both. Processing before ensures that rules regarding allowed structure are followed. Processing after ensures that the results returned don't give away uncleared secrets. For example (and the example that my demo shows) the query can be "sanitized" by adding or removing columns for easier analysis later. Query controls are primarily useful against indirect attacks.

Preprocessing

Query preprocessing is used to catch suspect queries before they are executed. For example, it may be that you want to restrict queries to only aggregates of data to public users. You wouldn't want anyone to search for an individuals salary, but it may

be OK to request the average salary of a particular group of employees.

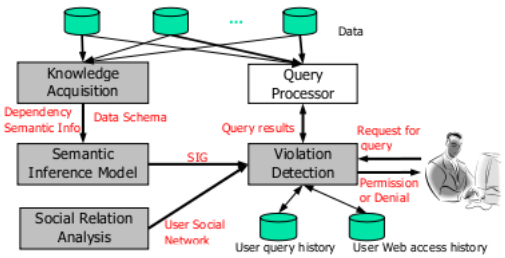
Result Analysis

Query result analysis after execution is used to catch result sets which are too specific. For example, if someone requests an aggregate which would be allowed by query preprocessing however adds a where clause which narrows the query down to a few, or even one employee then the preprocessing failed. Analysis of the result would show that results from only one record were shown and the query can be denied before data is returned to the user.

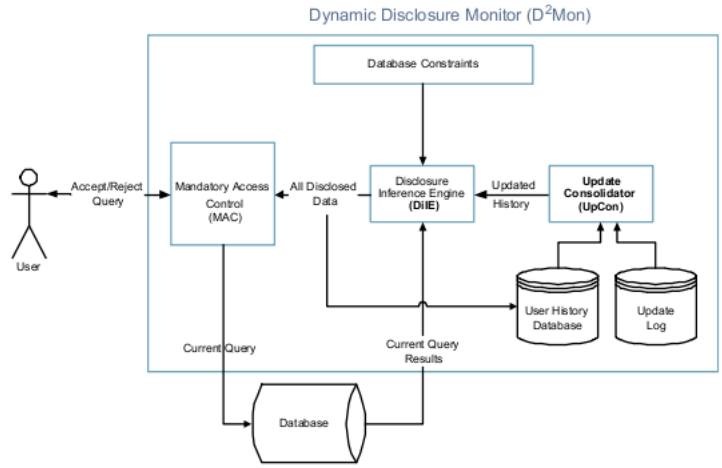
Query History Retention

This refers to storing historical queries by a user or groups of users in order to ascertain if multiple queries are being used to make inferences about sensitive data. The queries are stored in a database and analyzed along with new queries to detect inference attacks. There are a number of methods which can be used to consider a "group" of users. Generally some type of clustering algorithm is used to group user based on similar characteristics such as web browsing history and other resource useage. Collecting information on groups can assist with mitigating collaborative inferencing however requires more resources and can result in troublesome false positives.

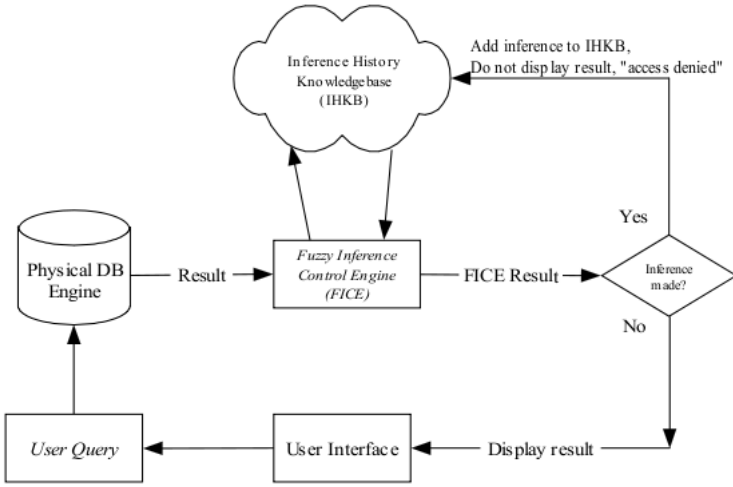
A number of proposals have been made for systems which collect information about a user or group of users' queries and attempt to detect when inferences are likely to have been made. Complex Bayesian networks are used to compute the probabilities and query results are withheld if the probability grows beyond a certain limit. A few of these that I found in my research are diagramed below:



Chen & Chu's Semantic Information Model (SIM)[1, 7]



Toland, Farkas & Eastman's D^{2}MON Architecture[3]



Huebner's Fuzzy Inference Control Engine (FICE) [9]

In general all these systems look at query results and determine if an inference was made. If so, the query result is denied otherwise it is allowed to be returned to the user. While some have been implemented, others are simply design ideas left to be implemented by others.

In my opinion this is where the serious work is being done these days. Query controls have lagged behind other methods of inference prevention/detection but Moore's law has been our friend here. Cheaper and faster processors and storage mediums have made these sorts of controls much more feasible. New research into applying Bayesian networks to figuring out probabilities of inferences has also advanced the field. New and faster algorithms in the field of data mining are allowing us to collect and make sense of huge amounts of data. Both these factors should play a part in making it feasible to more closely track queries on a database to detect inference attacks.

Query Preprocessor in Python

To illustrate both query preprocessing and result analysis I wrote a database front-end in Python which interacts with a MySQL database. A simple command line interface is used for entering queries to the database. For query preprocessing the front-end restricts queries to aggregate results only. Each item being "selected" is checked to ensure that only a supported aggregate function is used for each. Additionally, a count is added for each of the selected items to assist with later analysis. Result analysis is applied by enforcing N-item k-percent, specifically to withhold output if fewer than 2 items make up 100% of the result. To accomplish this, the added count fields are each analyzed to see if any of them are less than 2. If so, the query is denied before data is presented to the user. I arbitrarily set the minimum record to 2 for the purposes of this example, however any constant can be used, or possibly more useful a percentage of a whole can be used. The code for this example will be included with this paper. It can be executed on any machine implementing the Python interpreter and a MySQL database. Variables for configuring the username, password, and database are provided for easier configuration.

Other Novel Controls and Methods

Design time changes to the structure of the database can contribute greatly to mitigating inference attacks. Reducing repli-

cation of data, creating easier to manage table structures and implementing security right from the start are just the basics. Some have suggested we go further and completely change the way we structure the database in order to further mitigate inference type attacks. Some of these methods I stumbled across in my research are worth noting, even if implementation would be complex and difficult.

Extreme schema normalization

Biskup, Embley, and Lochner suggest an interesting concept which attempts to reduce inference control to simply access control. This requires that the relational database schema be reduced to object normal form (ONF). ONF is an extension of Boyce-Codd normal form. It then defines a confidentiality policy to the protect certain parts or combinations of tuples which they denote as facts or "potential secrets". The process of determining what exactly a fact is is done through a complex analytical algorithm. [2].

Tree structured schema

This solution works on a tree-like semi-structured data model. Semi-structured data models are constructed without the rigid structural requirements of traditional database management systems. They can incorporate irregular, unknown, or rapidly changing structure. Their proposed formal model is called the Privacy Information Flow (PIF) Model. It is used to specify how information is transferred among participants (e.g., people and organizations) as well as to specify privacy requirements. The privacy information flow model consists of two main layers: a Privacy Information Flow (PIF) Graph that is created for every application domain and participant type, and a Privacy Contract that creates customized requirements for each individual participant. The PIF graph represents the permitted data items that can be transferred between two participants. A privacy contract specifies, for an individual participant, the information that is not allowed to be transferred to or logically entailed by the other participants. Also proposed is a privacy architecture, called Privacy Mediator (PriMe), that enforces the privacy requirements on every data transfer and provides maximal data availability. [5]

Key representation

The core idea behind this proposed solution is to convert the original database into a key representation database (KRDB). This also involves converting each new user query from string representation into a key representation query (KRQ), and storing it in a Audit Query table (AQ table). Three audit stages are proposed to repel inference type attacks utilizing three seperate algorithms. These algorithms enable the key representation auditor (KRA) to specify illegal queries which could lead to disclosing secrets. Key representation enables much simpler application of tailored algorithms to detect potential inferences. Because key representation only really works with numerical data, this solution is targeted for statistical databases only.[10]

Challenges

As with any security measure there are challenges in implementing any kind of database inference controls. The biggest problem is that any obvious restrictions or noise in query response will raise suspicions and confirm that sensitive data is available. This peek the interest of any attacker who will then focus his attacks in an effort to attempt to uncover the sensitve data he now knows must be there.

Methods such as RDP can be defeated by querying several times to determine which data has been changed. To overcome this the RDP algorithm would need to remember and replicate previous changes for each querying user. This is a complicated and resource intensive solution.

Effective query controls are rough on the DBMS and require considerable effort to set up and maintain. While systems are becoming more and more powerful, emphasis is still put on speedy response time especially for large mission critical databases. Here is another example of a balance of security of utility. Strong query controls are likely to slow down response time, however users demand high performance.

Another serious challenge is that many industry professionals still do not understand this threat and therefore are unlikely to spend money to mitigate it. If a database programmer understood the problem himself and tried to explain to upper management the need for inference control it's likely management would not understand the immediate benefit and approve funding. Addressing inference concerns can be costly and time consuming. It's difficult for many to see the benefits to allocating resources to this problem. Like with many other security related expenditures, the return on investment is not easily realized until an incident occurrs..

Beneficial Usages of Inferencing

It is important to point out that inferencing can be used for beneficial purposes. There are a number of applications of inferencing which can be extremely useful in data analysis. For example, using inferencing techniques to analyze Intrusion Detection System (IDS) log files. Inferencing techniques can help administrators perform risk assessment and harm prediction. More predictive and proactive defenses can be designed by a deeper understanding of attacker intent, objective and strategies. Specific defenses can be prepared to mitigate traditionally hard to handle attacks (ie. Distributed Denial of Service) when the probability of attack is high as well as responses to ongoing attacks.

Industry efforts to address inference security

Industry efforts have been limited as far as what I have been able to find. A number of commercial DBMS have added MLS type functionality to their software to address security concerns, but not one mentions inference as one of those concerns. Similarly, poly-instantiation is supported in some limited fashion. Nowhere in any product literature or white paper have I been able to find reference to specific functionality to address concerns of inference attacks. Limited data item controls are commonly implemented through MAC and/or DAC functionality or through virtual private database (VPD) functionality. Query controls can be implemented, to some extent, through triggers however are not well suited to the job.

In general industry has not caught on yet to the importance of database inference control at the DBMS level requiring these controls need to be implemented as an add-on solution. There are a number of query optimizers out there which preprocess queries. It is not that much of a leap to extend the functionality of these to also include inference detection. Until the problem become more widely understood it's unlikely mainstream database management systems will take the time to implement more extensive inference control measures. Below are the limited efforts I've found within the major DBMS providers to provide functionality which could be used to mitigate inference attacks.

Oracle:

Data compartmentalization features first appeared in an add-on product to Oracle7, called Trusted Oracle7, primarily driven by Oracle's clientele in the US military. Based on the Bell-LaPadula security model, Trusted Oracle7 came pre-configured with three security levels: Confidential, Secret, and Top Secret. By combining these levels with a set of compartments, say one for each project that uses the database, it was possible to create a hierarchical set of controls that limited each user to accessing only the data from their project(s) at their security level. At the top of the hierarchy, users could see data from any compartment with any security level. At the bottom, a user could be restricted to seeing only Confidential data (not Secret or Top Secret) for their one compartment (or project). The complexity of configuring and implementing the system could be quite daunting, particularly in a system hosting a dozen or more projects with millions of rows of data stored in the database. Recently many of the features of Trusted Oracle have been redesigned and reimplemented in Oracle 10. Row level security, referred to as "Label Security" allows administrators to classify every row in a table, ensuring access to sensitive data is restricted to users with the appropriate clearance level.

Sybase:

Sybase's adaptive server enterprise (ASE) product includes a Policy-Based Access Control framework that provides means for protecting data down to the row level. Administrators can define security policies that are based on the value of individual data elements. The server then transparently enforces these policies. This means that once a policy has been defined, it is automatically invoked whenever the affected data is queried, whether through an application, ad hoc query, stored procedure, or view.

IBM:

IBM's provides multilevel security support in DB2 as of version 8. This support provides row-level security labeling and is designed to meet the stringent security requirements of multi-agency access to data. The basic idea of MLS with row-level granularity is that any user reading or updating data in a DB2 table needs to be allowed to handle only the rows that his or her security label allows. Each row in a table is assigned a security label, and a user can read the row, only if his or her label dominates the label of the row. Similar rules apply for updating rows in a table with row-level security, only where updating within an MLS environment is concerned, other principles concerning write-down (that is, the declassification of data) influence the result of the update.

Microsoft:

Microsoft's SQL Server provides no built-in security measures which prevent inference. Row level security is possible through a complex configuration of security labels and views, however this does not provide the same security a standard MLS database implementing polyinstantiation would.

Conclusion

Database security is one aspect of overall information security and is in keen focus of late. Sophisticated database systems are key to any organization's operations. The security of these systems and the data kept within them is of paramount importance, especially with new legislation making it mandatory for companies to make all reasonable actions to assure certain data sets remain confidential. A number of methods have been proposed to help mitigate these attacks with limited success. What really needs to happen is to build an entire system from the ground up to be resistant to inference attacks with a layered approach much like the security of any other system. Checks need to be put into place in multiple locations to cover all possible avenues of attack. While this seems overkill, for some systems containing extremely sensitive data it may very well be necessary.

Finding a solution for the inference problem is one of the most difficult problems in the security world. This is due to the fact that inference often comes from our brains making connections between certain data sets. There is no perfect way to "defend" against someone using their brain and own intelligence to make inferences. Preventing human intelligence from doing what it is supposed to do is a task that unlikely to be solved. Due to the abstractness of inference, many security professionals are unaware of this problem and/or dismiss it since there is no quick and easy fix. We can only do our best to make it harder for such events to occur.

Source Code for Query Control Example

```
# Mysql Database Query Preprocessor / Result Analyzer
# Michael Hylkema
# Boston University
# December, 2009

import MySQLdb
import sys

user="user"
password="password"
database="database"
host="localhost"

def Sanitize_Query(Query):
        Aggs=["avg(","corr(","count(","covar_pop(","covar_samp(","cume_dist(","dense_rank(","first(","group_id(","grouping(","gr
ouping_id(","last(","max(","min(","percentile_cont(","percentile_disc(","percent_rank(","rank(","stddev(","stddev_pop(","stddev_
samp(","sum(","var_pop(","var_samp(","variance(","collect(","median(","stats_binomial_test(","stats_crosstab(","stats_f_test(","
stats_ks_test(","stats_mode(","stats_mw_test(","stats_one_way_anova(","stats_t_test(","stats_wsr_test("]

        Offset1=7
        Offset2=0
        From_Clause=[]

        for x in Query:
                if (x==","):
                        From_Clause.append(Query[Offset1:Offset2])
                        Offset1+=((Offset2-Offset1)+1)
                Offset2+=1

        From_Clause.append(Query[Offset1:Query.find("from")-1])

        Check=0
        San_Query="select"

        for y in From_Clause:
                for z in Aggs:
                        if (z in y): Check+=1

        if (Check < len(From_Clause)):
                return 0
        else:
                for x in From_Clause:
                        San_Query+=(" "+x+","+"count"+x[x.find("("):x.find(")")]+"),")
                San_Query=San_Query[:len(San_Query)-1]+" "+Query[Query.find("from"):]
                return [San_Query,len(From_Clause)]

if __name__=='__main__':
        try:
                conn = MySQLdb.connect(host=host, user=user, passwd=password, db=database)
        except MySQLdb.Error, e:
                print "Error %d: %s" % (e.args[0], e.args[1])
                sys.exit (1)

        while (1):
                Deny=Switch=0
                Output=""
                raw_query=raw_input('Enter your query: ')
                if (raw_query=="quit"): sys.exit(1)
                san_query=Sanitize_Query(raw_query)
                if (not san_query):
                        print 'Query Denied!  Only aggregate queries are allowed on these fields.'
                        break
                print 'Sanitized query: ', san_query[0]
                try:
                        cursor= conn.cursor()
                        cursor.execute(san_query[0])
                        row=cursor.fetchone()
                        if row==None:
                                print 'No rows returned'
                        else:
                                for x in range (0,san_query[1]):
                                        if (row[(x*2)+1]<2): Deny=1
                        if (Deny):
                                print 'Query Denied!  Too few records for aggregate query.'
                        else:
                                for z in row:
                                        if (Switch==0):
                                                Output+=" "+str(z)
                                                Switch=1
                                        else: Switch=0
                                print Output
                except MySQLdb.Error, e:
                        print "Error %d: %s" % (e.args[0], e.args[1])
```

Bibliography

[1] Yu Chen, Wesley W. Chu. "Database Security Protection via Inference Detection". University of California Computer Science Department. Los Angeles, CA. March 2006.

[2] Joachim Biskup, David W. Embley, Jan-Hendrik Lochner. "Reducing inference control to access control for normalized database schemas". Information Processing Letters, Volume 106, Issue 1, 31 March 2008, Pages 8-12

[3] Tyrone S. Toland, Csilla Farkas, Caroline M. Eastman. "The inference problem: Maintaining maximal availability in the presence of database updates". Computers & Security, In Press, Corrected Proof, Available online 26 July 2009

[4] Yasunori Ishihara, Toshiyuki Morita, Hiroyuki Seki, Minoru Ito. "An equational logic based approach to the security problem against inference attacks on object-oriented databases". Journal of Computer and System Sciences, Volume 73, Issue 5, August 2007, Pages 788-817

[5] Csilla Farkas, Alexander Brodsky, Sushil Jajodia. "Unauthorized inferences in semistructured databases". Information Sciences, Volume 176, Issue 22, 22 November 2006, Pages 3269-3299

[6] Bhavani Thuraisingham. "Privacy constraint processing in a privacy-enhanced database management system". Data & Knowledge Engineering, Volume 55, Issue 2, November 2005, Pages 159-188.

[7] Yu Chen, Wesley W. Chu. "Protection of Database Security Via Collaborative Inference Detection". University of California Computer Science Department. March 2008.

[8] Paul D. Stachour, Bhavani Thuraisingham. "Design of LDV: A Multilevel Secure Relational Database Management System". IEEE Transactions of Knowledge and Data Engineering Vol. 2, No. 2. June 1990.

[9] Richard A. Huebner. "Automated Mechanisms for Controlling Inference in Database Systems". January, 2004.

[10] Asim A. Elshiekh, P.D.D Dominic. "Efficient Algorithms for the Key Representation Auditing Scheme". International Journal of Computer and Electrical Engineering, Vol. 1, No. 3, August 2009.