A Security Architecture for Query Tools used to Access Large Biomedical Databases

Shawn N. Murphy, MD, Ph.D. and Henry C. Chueh, MD, M.S.
Laboratory of Computer Science, Massachusetts General Hospital, Boston, MA.

Disseminating information from large biomedical databases can be crucial for research. Often this data will be patient-specific, and therefore require that the privacy of the patient be protected. In response to this requirement, HIPAA released regulations for the dissemination of patient data. In many cases, the regulations are so restrictive as to render data useless for many purposes. We propose in this paper a model for obfuscation of data when served to a client application, that will make it extremely unlikely that an individual will be identified. At Partners Healthcare Inc, with over 1.4 million patients and 400 research clinician users, we implemented this model. Based on the results, we believe that a web-client could be made generally available using the proposed data obfuscation scheme that could allow general usage of large biomedical databases of patient information without risk to patient privacy.

INTRODUCTION

At the core of hospital sponsored clinical research programs are services for recruiting patients for clinical research studies. At Partners Healthcare Inc. in Boston, a corporation that includes both the Massachusetts General Hospital (MGH), and the Brigham and Women's Hospital (BWH), there has been created a clinical data warehouse named the Research Patient Data Registry (RPDR). This data warehouse contains all of the inpatient and outpatient data identified as desirable for research as defined by a previous study (1). This data includes diagnoses, medications, procedures, and laboratory tests. It contains about 270 million patient-concept associations from 26 million patient encounters encompassing about 1.4 million patients.

Faculty members at the MGH or BWH are able to find cohorts for their research studies through a visual interface (query tool) supported from any networked Microsoft Windows client workstation in the hospitals. The model for the user interface consists of a hierarchical tree of items for the users to choose from, where items are dragged into a set of containers that roughly model a Venn diagram. Aggregate numbers of patients who match the query criteria are returned in a series of display widgets. The individual patients that represent the aggregates can later be identified, provided the Institutional Review Board has approved the study request.

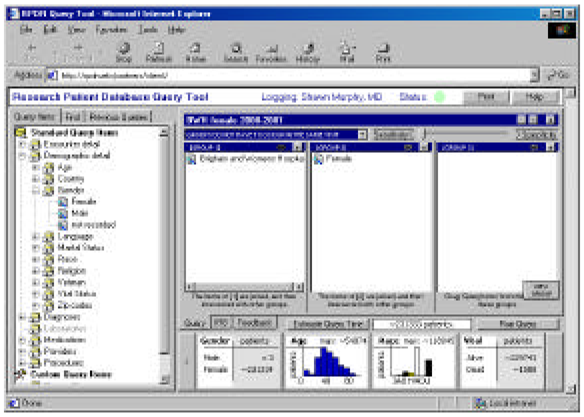

Figure 1 – Queries to the clinical data warehouse are made through this web query tool, where items arranged hierarchically in the left panel are dragged into the group panels on the right to construct a Boolean query. This query is asking for the number of BWH patients who are also female. Results are shown in the lower right displays. Approximate numbers are returned to prevent the identification of specific patients using the query tool.

A fairly general user base has evolved, as many faculty members did not want to use the tool themselves, but rather wished to delegate that task to a fellow or research associate. Another security concern is that traditional audit logs have a difficult time proving that a person is searching for a specific individual with the query tool, making it more necessary to prevent such an attack at the point of the lookup. These two factors heightened concerns about patient confidentiality using a general query tool, and it became imperative that a single individual could not be identified through the user interface.

An attack on the system would occur by someone entering criteria into the query tool that would result in only one matching person. For example, a person wishes to know if someone in their neighborhood has manic-depression, and that person knows the birthday of the son of this woman. They might start by asking the query tool for how many 30-year-old females in the 02054 area code had a normal delivery on September 3, 1988. If this resulted in only one matching person, then they would ask how many 30-year-old females in the 02054 area code had a normal delivery on September 3, 1988, plus have manic-depression. If this also resulted in one matching person then they would know that the person in their neighborhood had manic-depression.

An approach to preventing the identification of individuals in the database could be to de-identify patients in the database with a system that complies with HIPAA's 18 points for data de-identification (2). Although this would be a good start, there are two sets of problems with this. The first set of problems results from the destruction of important clinical data. This results mostly from the destruction of location information (zip codes) and the destruction of date information. Just attempting to recruit patients in the general area of the hospital for research studies relies on zip codes, not to mention more complex epidemiological studies. The destruction of date information is equally problematic. The HIPAA regulations state that dates should be rounded to the month if not to the year. However, very often a query will want to identify which event occurred first as a means of finding cause and effect. For example, in a study on drug adverse events, one may want to find all cases where a rise in liver function tests followed the initiation of a drug by 1-3 days. The second set of problems is that even with the destruction of all the above information, there is still no guarantee that someone cannot be identified in the database. As greater quantities of data are added, the chance that a set of clinical values (such as an SGOT of 367 and an SGPT of 1325) will not be able to uniquely define a patient becomes very low.

Another approach might be to use table ambiguation by suppression of selected cells (3). However, this approach would also result in the destruction of large amounts of clinical data. Because cells that uniquely distinguish a patient from other patients are the cells to be suppressed, the most interesting clinical data would be specifically targeted for destruction. This would severely limit the usefulness of the data for many purposes, such as has been extensively analyzed by Ohno-Machado et. al. (4). Furthermore, this approach is also susceptible to re-identification of patients that have supposedly been de-identified (5).

Other approaches include allowing the patient to look at those who have accessed their records, possible spotting their neighbor from the above example. There is a high maintenance cost to such a system (issuing passwords, etc.), but even this heroic solution will not work with query tools which touch thousands if not millions of different patient records with every query.

Our solution to thwarting an attack on this kind of system was to take advantage of the following set of properties of the query tool, and inventing a solution that fit around these properties. Although the fact that these properties must exist in the query tool limits the usefulness of the solution, these are properties many query tools would have (on the web for example) where aggregate numbers of patients need to be returned. The required properties are:

1) It is sufficient that the query tool only returns numbers of distinct patients.
2) The number of distinct patients is useful even when the number is within a probable range of 2 or 3 patients. This range can be varied to obtain optimized signal to noise ratios for detecting an attack.

The requirement that the query tool return distinct patients and not some other item such as hospital encounters or events depends on the nature of the item. One may generalize and conclude that any set of independent items is a candidate to be used as the reported item. However, there are some kinds of relationships between reported items (such as if the same hospitalization is reported as multiple encounters in the database) that could be exploited to attack our data obfuscation system.

METHODS

We developed a method of data obfuscation that was generally applicable to query tools that meet the above requirements. Additionally the server application should satisfy the following technical requirements:

1) The server application that will produce the numbers for the query tool must allow the numbers to be changed in a programmatic fashion before they are sent to the query tool. An obfuscator is going to be placed between the server output and the query tool, so these numbers need to be assessable.
2) An audit trail with a minimum of user ID, date/time of query, and true number of patients returned by the queries needs to be kept. A program is going to continuously analyze this audit trail.

In an abstract sense, the key to the data obfuscation method is to make attempts to discern individuals in the database a recognizable pattern by a computer algorithm. The method works by having the server not pass true numbers of patients to the query tool, but rather numbers that would be (if the same query were performed repeatedly) distributed around the true number of patients. For example, if one queried to find out how many people carried the diagnosis of manic-depression at Partners Healthcare Inc., one might get the result of 3,487, 3,488, 3,488, 3,490, or 3,489 patients. The exact nature of this range is the subject of much discussion below, but the range would be quasi-centered on the true number of 3,489. In order to find the true number, one would need to run the same query repeatedly. It is here where a computer algorithm would recognize this attack and mark the user whose queries returned the same true (not reported) number repeatedly as a suspicious pattern.

Forcing a user to do the same query multiple times if they are trying to obtain the true number creates a detectable pattern. This is because the chance that the same number of patients will be returned from many different queries of a large database is low, except perhaps for queries that return zero patients. The mean number of attempts required to hone in on the true number will increase as the range gets larger. This suggests that the range should be large, which, however, needs to be balanced with user dissatisfaction of receiving a wide range of approximate numbers. If one studies the functions available for creating the range, one finds the optimal function is one derived from a Gaussian one as discussed in figure 2. The reason that this function is optimal is essentially that specific features of any other distributions could be exploited to allow fewer attempts for finding the true number.

As the numbers that are being returned to the query tool are obfuscated, a history table of true numbers is maintained. This history table is seen only by the system, and contains for each query the date/time, the user ID, and the number of patients truly returned by the queries. A trigger each time a query is performed queries this history table. The query looks at the numeric results of queries performed by that user to see if the exact same number of patients were returned in previous queries within a specific time period. Depending on the time range employed for obfuscation, when a certain number of queries return the exact same number of patients, the user is prevented from performing further queries. Eventually, the account is reviewed and the lock is released if it is determined that the same number of patients were returned by the queries purely by coincidence.

We did a Monte Carlo simulation (6) of an attack, where a user is repeatedly performing the same query to find the average number of patients in the repeats. When the range of the Gaussian function that randomizes the results has a standard deviation of 1.33 (giving values about the true number as shown in figure 2), we found that it would take an average of 12.3 repeats before the average of the results converged on the true number of patients. Restated, if one performs the same query repeatedly, then a series of numbers, representing the obfuscated numbers of patients, are going to be returned. If one takes the running average of these numbers, they will eventually converge on the true number. The running average will become less that 0.5 patients from the true number of patients in a consistent fashion after about 12 numbers are returned. This estimate may actually be somewhat low because it would take a few more returned numbers before the attacker was confident that convergence had indeed been obtained. The threshold of tolerance for queries returning the same number of patients depends on the risk one wants to take that a patient will be identified in the database, vs. the inconvenience to the user and hassle to the auditors of access denial based on the same numbers returned coincidentally.

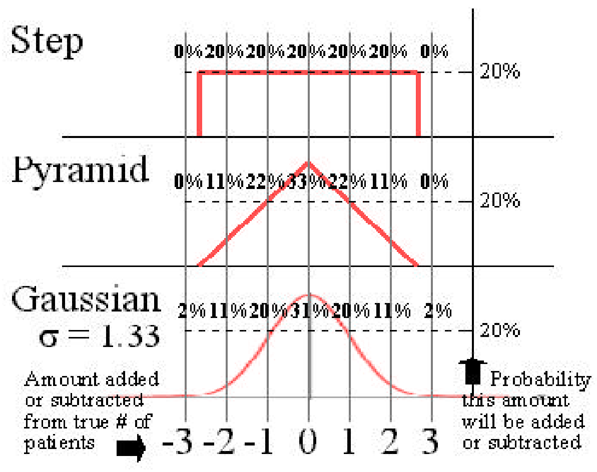

Figure 2 – An obscuration function is used to create small values that will be added or subtracted from the true number of patients to prevent patient identification. We found that functions derived from the Gaussian function were the least vulnerable to attack. Other examples of functions include the Step function and the Pyramid function. The Step function has a 4% chance of being cracked in just 2 tries (when #patients+2 and #patients–2 are obtained). The Pyramid function has a 1.2% chance of being cracked in 2 tries, but almost 6% in 4 tries (calculated via binomial probability function). A Gaussian derived function with roughly the same range of values cannot be definitively cracked in any finite number of tries.

RESULTS

Using our query tool as an example, we investigated how large, biomedical databases could be set up to be explored for aggregate patient numbers without allowing the identity of an individual to be revealed. We do not mean to indicate that an interesting finding will not boil down to a single patient. Rather, a detailed investigation of a single patient without potentially revealing the identity of that person would be an almost impossible task. Cases that require this kind of detailed investigation of a very small domain of patients are not candidates for study through our query tool, although our query tool may give approximate numbers of patients to determine if enough patients exist to fulfill a study designed for those patients.

Given that we are interested in investigating large patient samples and trends, we studied various methods to obfuscate the results. Our initial approach was to de-identify all of the patients in the database, removing obvious identifiers. This will prevent a direct attack, where someone breaks into the database itself to perform a quick lookup of a patient. However, if someone had more time they could use gender, date of birth, and zip code to identify many patients (7). This kind of attack would even be possible through the query tool client application without direct access to the database. Our next approach was to blur some of our demographic variables, such as convert birth dates to ages, and present any queries that returned 2 or fewer patients as < 3, in theory trying to enforce a bin size of 2. This approach suffered from the weakness that a clever person could create a "step" of patients from a previous query which would return a constant number of patients, 10 for example. This step could then be logically OR'd with criteria of the patient to be identified. Then, to find if our example person had manic-depression, the count would go from 10 to 11. If one implemented some kind of rounded output, an attacker would just need to determine the position and direction of the rounding and put a step exactly where the rounding occurred.

Our final solution was randomizing the output within a certain range using a Gaussian function. This would force any person to statistically search for the mean output if they wanted to find the true answer. By forcing an attacker to perform a query repeatedly in hopes that an average can be ascertained, the system will see the query returning the same number of patients repeatedly. When queries are seen returning the same number of patients repeatedly it removes the access of the user to the system.

Although this method may seem overly complex in some ways, the task of obfuscating data in a foolproof manner is extremely difficult. The method we have described also has some vulnerabilities that need to be explored. One vulnerability is that we allow a larger number of queries to be performed that return zero patients. Unfortunately, naï ve users who are performing incorrect queries will often have zero patients returned. Having the table-watching agent terminate access to these users would be too sensitive a response. However, attacks around zero are more likely to succeed. Therefore we modify our range algorithm to have a wider obfuscation at values close to zero, and indicate the greater range by returning "<3" when the obfuscated number is less that 3.

Another vulnerability is that prior knowledge of data in the database may allow "steps" to be pre-conceived, and thus queries that do not return the same number of patients could be used to get the true mean from several queries. For example, if one wished to know the manic-depressive status of the 30 year old female in the 02054 area code, and one knew the manic-depressive status of 30 year old females in numerous other area codes, one could add each of these area codes to several queries. Thus if one knew that the 02055 area code had 2 manic-depressives, and the 02056 area code had 10 manic-depressives, the result of 3 and 12 when each query was performed with the 02054 area code added, might suggest that 1 manic-depressive lived in the 02054 area code (obtained once the known numbers were subtracted out). Since the queries returned different numbers of patients, the agent watching the numbers of patients returned by the queries would not catch this attack. This type of attack would require that one had extensive specific knowledge of numbers of patients in other areas regarding specifically what one wanted to discern in a given individual. Our environment at Partners Healthcare Inc. makes this an unlikely scenario. On the other hand, this may be an important scenario if the query tool were to be used on a nationwide database. Someone may know all the manic-depressives in Florida, and use these to construct "steps" to illicitly find the manic-depressive status of someone in Massachusetts. To counter such an attack, one would implement a rounding function on the results, such as rounding to the nearest 10. An attack would need to operate at the margin of the rounding, and the agent would be look for a pattern of numbers of patients returned at the margins of the rounding.

Finally, there is a potential vulnerability concerning breakdowns of patients into different categories by the query tool, such as by males and females. Our strategy for addressing this vulnerability is to add the obfuscating value for the total patient number across random breakdown numbers, and then to obfuscate the breakdown numbers by the same Gaussian function used to obfuscate the total. This approach will prevent the breakdown totals from being used to further converge on the true number of total patients. However, the effect will be to produce less and less reliable numbers as one drills into increasing detail in the breakdown presentation.

DISCUSSION

The objective of our data obfuscation is to allow a general population of users to have access to aggregate data about patients without threatening the confidentiality of the individual patient. It has been repeatedly shown that de-identified data can often be combined with other publicly available data to identify the individual (8). Attempts to obfuscate the raw data in a database can result in a deterioration of the potential knowledge that might be extracted from the database (4). As databases contain greater amounts of specific clinical information about specific patients it would become necessary to increasingly alter the clinical data to allow the patients to fit into larger "bins".

Hospital data consortiums have long been available in the form of de-identified data dumps from hospital admission records and the like. However, it has been shown that most of these data sets are prone to re-identification and therefore violate our concept of privacy (7,8). An alternative to large data dumps is to allow such data to be requested from server-side services that are more easily amenable to data obfuscation and monitoring. Designed correctly, these services could greatly decrease the risk of divulging private information.

Acceptable methods of obfuscation will be dependent on the types of data that need to be served back to the client application. The methods described in this paper apply to clients that need sets of distinct patients where exact numbers of patients are not required.

Naï ve approaches to data obfuscation are often to restrict the types of data elements that may be used in a query. Our example of looking for manic-depressives in certain zip codes may spur a naï ve response to restrict the use of zip codes. But this is a "slippery slope", because any data can potentially be used in combination with other data to identify patients, such as child delivery dates, hospital admission dates (8), lab values, etc. The richer the database is with data, the more problematic this becomes.

The data obfuscation method discussed in this paper does not limit the logic of the query, nor the data elements or values that can be used in the query. It depends exclusively on the numeric quantity of the results. In our environment, this data obfuscation allows general research queries to be performed with relatively little risk to patient confidentiality. This allows a greater domain of users and a greater comfort level from our sponsors. This model allows sophisticated data to be stored intact in the database such as gene sequences and bio-markers of various kinds. Without this obfuscation method, there would be steep reservations about including increasingly patient-specific data in large databases for the MGH and BWH research community.

This work was supported by Partners Healthcare Inc.

References

1. Murphy, S. N., Morgan, M. M., Barnett, G. O., Chueh, H. C. Optimizing Healthcare Research Data Warehouse Design through Past COSTAR Query Analysis. Proc AMIA Fall Symp. 1999; 892-6.
2. Rules and Regulations. Federal Register 65(250), Section 164.514, 82818, Dec 28, 2000.
3. Fischetti M,Salazar J.Model and algorithms for the 2-dimensional cell suppression problem in statistical disclosure control. Mathematical Programming 1999; 84:283-312.
4. Ohno-Machado, L., Dreiseitl,S., Vinterbo, S., Effects of Data Anonymization by Cell Suppression on Descriptive Statistics and Predictive Modeling Performance, Proc AMIA Fall Symp 2001; 503-7.
5. Dreiseitl,S., Vinterbo, S., Ohno-Machado, L., Disambiguation Data: Extracting Information from Anonymized Sources, Proc AMIA Fall Symp 2001; 144-8.
6. J. E. Gentle, Random Number Generation and Monte Carlo Methods (Statistics and Computing), Springer-Verlag, 1998
7. Sweeney L. Guaranteeing anonymity when sharing medical data, the DataFly system. Proc. AMIA Fall Symposium. 1997; 51-5.
8. Malin, B., and Sweeney, L. Re-Identification of DNA through an Automated Linkage Process, Proc AMIA Fall Symp 2001; 423-7.