CS 458 — Module 6: Database Security

1 Database Concepts

- Normalization — how to avoid having rows that are too tightly integrated when they shouldn't. For example, if you had prices of an activity tied to the users that actually used it, and you accidentally deleted all users who did that activity. So where do we find the price now?
- Some steps for normalization (informally):
  1. Is every row unique? If not, delete identical rows. This is now 1NF (first normal form).
  2. Does it take the whole key to identify a non-key field? This is now 2NF.
  3. Can any non-key field be determined knowing another non-key field? This is now 3NF.
- Some other basic vocabulary:
  - DBMS is a database management system
  - Schema is the structure of the DB

2 Database Security

- We want to maintain physical integrity and logical integrity.
- Physical problems include power failures, disk crashes, etc. Backups and redundancy are a good solution.
- Logical corruption could occur with invalid users or errors.
- Element integrity concerns itself with the correctness/accuracy of database elements.
- Some checks may be done to validate correctness (ie: within a range, certain data type, etc.)
- Keeping a change log is another good idea if space allows for it.
- Referential integrity ensures that a value in one relation that is used as a key in another relation actually *exists* in that other relation.
- Auditability — consider whether there is a need to keep an audit log of all database accesses.
- Access control — some places (like Ontario) have incredibly strong consent laws in regards to medical records. This isn't enforceable via a database — this is an open research problem!
- Note that access control is really hard — things that may not seem harmful in terms of access control may implicitly reveal information we're trying to protect!
- User authentication — the database may authenticate that a user was allowed to do something with the DB.
- Availability — what if database sharing causes availability problems?

3 Data Disclosure and Inference

- Data inference is deriving sensitive data from supposedly non-sensitive data.
- Two types of attacks:
  - Direct attack — issue a query that will give sensitive data but in an obfuscated query to fool the DMBS into allowing it.
  - Indirect attack — infer sensitive data from some statistical results.

3.1 Tracker Attacks

- "Tracker attacks" basically fool DBs into revealing information through seemingly innocent commands, and then using the data to infer the desired result that was initially directly blocked.
- For example, to get a specific user's grade, perhaps abuse features about that user to help get info about them specifically.
- Best way to defend against it is just audit logs and then deal with the attacker — it's just too hard to defend against!
- Other potential ways are:
  - Suppression — suppress sensitive data from a result.
  - Concealing — provide answers that are close to an actual value, but not actually.
  - n-item k-percent rule — for a set of records included a in result, if there is a subset of n records that are responsible for over k percent of the result, omit the n records from result. Note that the omission in itself could also leak information.
  - Combine results in the returned data, or report a set/range of possible values.
  - Return a random sample.
  - Randomly add a bit of error to each value before computing the result, and store the random error in your DB elsewhere.

3.2 Differential Privacy

- Imagine you have two datsets — one that includes restaurants Bob rates highly, and one that doesn't include any ratings from Bob.
- Differential privacy ensures that if Bob executes a query on the first dataset he gets nearly the same results as if he executed it on the second dataset.
- That is, from Bob's perspective, there is no difference.
- Basically, this matters because we don't want people to be able to tell us apart within the databases of companies that collect a lot of data.
- Typically, diff. privacy can be achieved by adding noise to the result of a query before releasing it.
- But with differential privacy, by adding noise to do so, we reduce accuracy.
- For example, a public database of Warfarin use originally leaked too much sensitive patient data, but the dosing recommendations were safer. But by improving privacy via adding noise, the dosing recommendations could kill patients!

4 Multilevel Security Databases

- MLS databases support the classification/compartmentalization of information according to its confidentiality, using two sensitive levels (sensitive and not sensitive).
- This can be done at the element level if required.
- In an MLS database, each object has a sensitivity classification and maybe a set of compartments.
- Suppose you wanted to implement the *-property (no read up, no write down) in an MLS setting.

  - This is really hard — if the user is doing a write-up, and cannot read the data, you are doing a blind write.
  - Write downs would need a sanitization mechanism.
  - The problem is DBMSs must have R/W access at all levels to answer user queries, perform backups, etc.
  - For example, how could you record the salary for a secret agent if you aren't allowed to know who the agent is?
  - We are left needing to trust the DMBS — so the *-property doesn't work well here.
- SQL supports polyinstantiation as a method of enforcing confidentiality. In polyinstantiation, more than one row in the same relation can have the same *key* — this violates normalization principles though!
- So, depending on the user's clearance, they would get different answers for a query.
- However, the *existence* of the record itself could be confidential!
- A more common method of implementing this style of confidentiality is partitioning.
  - Each classification level has a separate *database*, similar to the Bell LaPadula approach.
  - This is simple!
  - However, this may lead to redundant data storage in multiple databases.
  - This also does not address a problem with high-level users needing to access low-level data combined with high-level data.
- Yet another approach is encryption, with a key unique to its classification level.
  - As per usual, one must use the encryption scheme in the right way — for example, encrypting the same value in different records with the same key should lead to different ciphertexts!
  - Processing queries may now become expensive.
  - There is some research in doing processing directly on the encrypted data (homomorphic encryption).
- Yet another idea — the integrity lock.
  - This provides both integrity and access control!
  - Each data item consists of:
    1. The data item
    2. A (concealed) integrity level
    3. A MAC/signature covering the above + the item's attribute name and record number
  - The signature protects attacks on the above fields, such as attacks trying to modify the sensitivity level or move/copy the item in the database.
  - This doesn't protect against replay attacks though — an attacker could do something and replace/restore the original value with the correct MAC lock.

5 Design of Secure Databases

- We usually have a front-end that authenticates a user and forwards their queries to a DMBS.
- The front-end gets the results from the DMBS and removes data items that a user should not be able to see before passing the results back to the user.
- This allows for the use of existing DMBS and databases, but could be inefficient if the DMBS returns many items and most of them get dropped by the front-end!

- One way to tackle this is using commutative filters — the front-end should re-write the user's query according to the user's security classification.
- this means we can discard forbidden data items well before even hitting the DB.
- Another idea is using distributed/federated databases. This is based on the partitioning concept; the FE only forwards the user query to DBs that the user can access. But it may have to combine the results from multiple DBs.
- And remember, you have to trust the FE as it may see all the data!
- A third idea is views — restrict parts of the DB a user can see.
- Trueman Semantics — we will always give an answer, even if it's wrong, as *not* answering reveals too much!
- Meanwhile, Non-Trueman Semantics will reject queries that access data the user cannot access; this would mean the user is aware that this data is sensitive.

6 Data Mining

- The combination of storing sensitive information, in multiple databases, has raised concerns and attention about data mining.
- With so much data and from so many possible locations, data inference is now a very real issue.
- Data mining tries to automatically find patterns in data — there is useful/valuable uses for it, but it can also be used maliciously.
- The main security critiques against data mining are:
  - Confidentiality — derivation of sensitive info
  - Integrity — mistakes in data
  - Availability — different databases may not be compatible
- A good practice is to anonymise data records before releasing them — but often, they don't do it correctly.
- k-anonymity — anonymizing a dataset.
  - Remove key attributes AND candidate key attributes before release — things like name, SIN, address, phone number, etc.
  - Next, quais-identifiers — non-key attributes but that could beu sed to identify records. For example, 5-digit ZIP code + birth date + gender could udentiy about 87% of the US population.
  - Lastly, identify sensitive attributes — these are what we want to keep for researchers/analysis/whatever.
  - Another idea is "blurring" the quasi-identifiers via generalization, so we get semantically consistent values until we have k identical records.
- Unfortunately, k-anonmyity will not solve everything.
  - Sensitive values in an equivalence class will lack diversity.
  - The attacker having background knowledge can also help.
- Another strategy is l-diversity:
  - Ensure sensitive attributes are diverse in each quasi-identifier equivalence class.

  - The most common strategy is using distinct l-diversity. For each equivalence class, ensure there are at least l well-represented sensitive values.
  - This doesn't prevent probabilistic inference attacks, though.

- One more approach is adding noise. The accuracy will go down but the privacy will go up — but if you do it evenly it won't sacrifice too much.