ERD – Entity relationship Diagram. Visual way of looking at database structure.

BLUF – bottom line up font, Systems that intercept inputted data are susceptible to attack.

Prevent injection attacks by, 1. Parameterize query. Separate query from the data and use built-in functionality to validate user input hasn’t been compromised. 2. SQL can be tuned to limit how many rows are returned. This will only limit how much data is spilled. 3. Firewall. Watches for injection attacks, they are hit and miss. Sometimes it works well and others it doesn’t.

**What is an injection attack?**

Attacker sends malicious data as apart of a command or query to an interpreter. IF successful, it will be executed as code by the interpreter.

The client creates a string, which is sent to the DBS parses the string, when its finished, it executes the command and sends the result set back to the client.

An application is vulnerable to attack when

* User supplied data is not validated, filtered, or sanitized by the application
* Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter
* Hostile data is used within object-relational mapping (ORM) search
* Hostile data is directly used or concatenated such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.
* Some of the more common injections are SQL, NoSQL, OS command, ORM, LDAP (lightweight directory access protocol) and expression, and expression language (EL) or object graph navigation library (OGNL) injection

How to mitigate injection attacks

Filtering, validating, encoding and escaping

**Blacklisting –** the process of rejecting/stripping known ‘bad’ input has limitations. It is always an uphill battle. Not effective to use alone.

**Whitelisting –** the process of only accepting desired input. It can be difficult on web apps where you want a user to be able to create their username with a broad range of characters.

**Encoding:** The process of encoding ‘bad’ characters after they have been submitted by the user to make them harmless. Like changing ‘ or “ into @apos; &quot;

**Escaping –** the process of escaping specialized characters such as “ ’ % to make them be interpreted as string literals rather than functioning in their intended way.

The best way to prevent injection is by query parameterization.

prepareStatement – take x variable, It looks at specific variable. Takes the setstring, scrubs it for malicious code, then inserts it.

6th to last