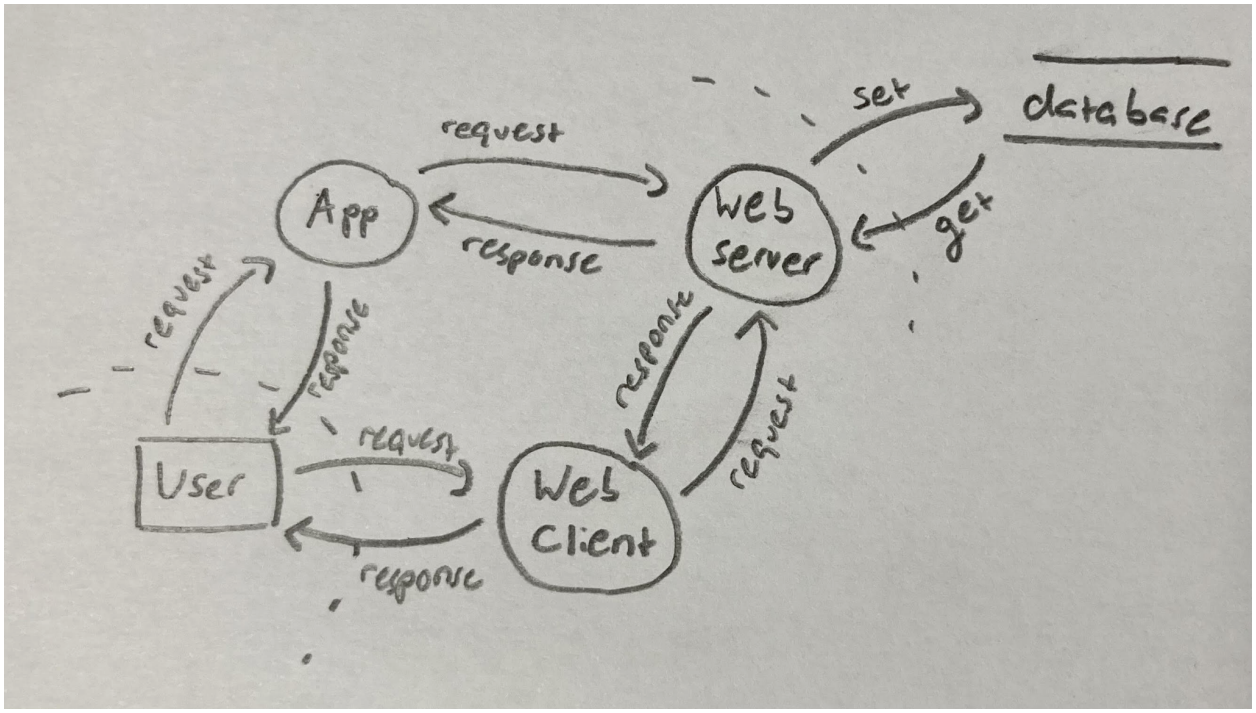
File name: misc/stride.pdf
File owner: Lysander Miller

Threat Analysis Using Stride

Data Flow Diagram:



Key:

- Rectangle: external entity
- Circle: process
- Two parallel lines/the top and bottom lines of a rectangle: data store
- Solid lines: data flow
- Dashed lines: trust boundary

STRIDE Analysis:

- **Spoofing:**
  - Threat:
    - An attacker might somehow get access to the TU database server where users' usernames and passwords are stored.[1] If usernames and passwords

[1]One way of doing this is through an SQL injection. Let's say someone fills out a username and password entry form with the following line:

" or "="

are stored in plaintext, an attacker could proceed to log into other individuals' accounts using the usernames and passwords they find.

- Solution:
  - One way to solve this problem is by storing a hash of users' passwords rather than the plaintext password. However, we know that a hash is deterministic (the same input leads to the same hash) and, thus, if two people have the same password, they'll have the same hashed password. This means that, if an attacker can decipher one user's password, they can decipher more individuals' passwords. Therefore, for each user, we actually want to store a username, a salt (a random, unique integer), and a hash of the following: the salt concatenated with the password.

    When a user inputs their username and password to login, the TU server would then go to the entry associated with the username, get the salt, and then create a hash of the following: the salt and the password the user inputted. If this new hash matches the hash stored in the database, the server "logs-in" the user.

    With this system, even if an attacker got access to the TU database server where users' usernames and passwords are stored, the storage system for usernames and passwords would make it incredibly difficult for an attacker to actually use the stored usernames and passwords (as de-hashing something is pretty difficult if not impossible).

- **Tampering:**
  - Threat:
    - Let's assume the TU database stores usernames and passwords in a SQL table called Users. Now, let's say an attacker fills out the TU username and password entry form with the following line:

      "; DROP TABLE Users

      This could result in something like the following SQL command being run:

      SELECT * FROM Users WHERE Name =""; DROP TABLE Users

      This would delete the Users table and thus, all of the data stored in that table.
  - Solution:
    - I remember in CS 257: Software Design, we primarily used SQL sanitization libraries to prevent SQL injection. While I don't remember

---

This could result in something like the following SQL command being run:

SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""

Because "=" is always true, the above command will return all usernames and passwords.

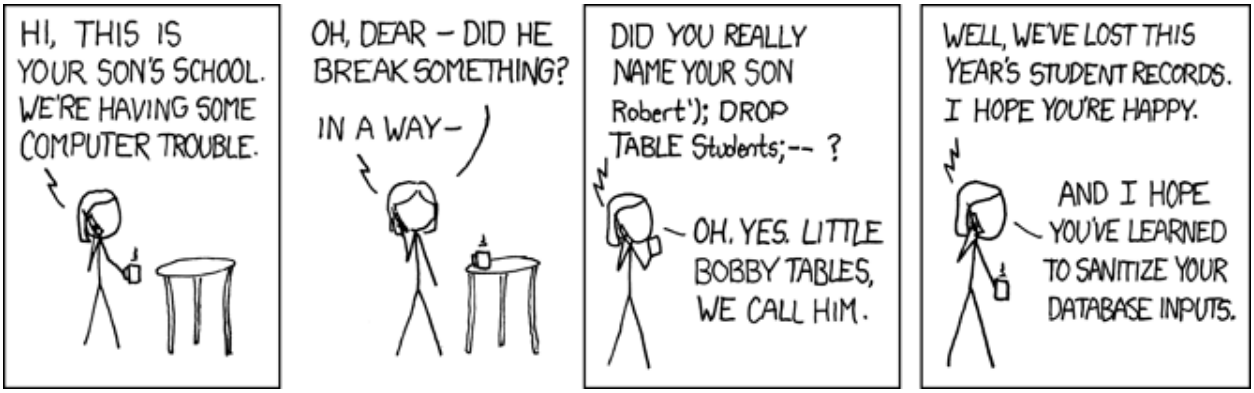
what library we used exactly, many of these libraries will do things like removing “dangerous characters” from user input. Such dangerous characters might include:

- ‘
- ;
- \--

These characters allow attackers to extend SQL queries, adding in their own requests.

While sanitization libraries are helpful, they don’t completely secure a database against SQL injection and one should also employ other techniques. One such technique might be to encrypt some of the data stored in a database. For example, hashing passwords rather than storing them in plain text (see more under the spoofing section).

- Sources:
  - https://www.codecademy.com/learn/seasp-defending-node-applications-from-sql-injection-xss-csrf-attacks/modules/seasp-preventing-sql-injection-attacks/cheatsheet
- Other:
  - A funny xkcd comic about this exact scenario:



- **Repudiation:**
  - Threat:
    - A Tapirs Unlimited user claims that they never purchased anything off of TU and TU is unfairly charging them for their “purchases.”
  - Solution:
    - Requiring digital signatures for transactions. In other words, a user would send a message detailing what they wanted to purchase along with a

digital signature. Thus, a user wouldn't be able to claim that they hadn't made a purchase.

- **Information disclosure:**
    - Threat:
        - An attacker controls a device which can intercept transmissions between a user's computer and the Tapirs Unlimited web server. The attacker then pretends to be TU when talking to the user and pretends to be the user when talking to TU. This not only threatens the integrity of the data sent back and forth between the user and TU but it also threatens the data's confidentiality. This is called an attacker in the middle attack.
    - Solution:
        - Tapirs Unlimited should make it so that all communication between users and TU is through HTTPS.
- **Denial of service:**
    - Threat:
        - In reading up on denial of services attacks, I found this website (https://www.imperva.com/learn/ddos/ping-of-death/) discussing something called the Ping of Death or a ping flood. A PoD attack is a type of denial of service attack where an attacker uses the ping command to send fragments of a malformed packet (malformed in that this packet is too large) to a web server. When the server attempts to reassemble the fragments, it ends up with a packet that's too large and results in the server's memory overflowing. This can cause large problems for the server and even result in a server crash.
    - Solution:
        - While we could use a firewall to block all ping messages to Tapirs Unlimited, that would prevent individuals who were using ping for legitimate reasons. Instead, we might want to use a firewall to block fragmented pings.
    - Sources:
        - https://www.imperva.com/learn/ddos/ping-of-death/
- **Elevation of privilege:**
    - Threat:
        - Some servers come with blank or default admin/root passwords. These passwords are not secret and can often be found in publicly available documentation. The idea behind this is that it makes it easier for the owner of the server to test things out, install stuff, and configure things before eventually setting a more secure admin/root password. However, if one forgets to change this default admin/root password, this means that an attacker could very easily log-in as an administrator and get unauthorized

access to files and system commands (such as those involved in shutting down a server).

If Jeff has forgotten/neglected to change the default admin/root passwords on the TU database server, it'd be easy to log into the server as root.

- Solution:
  - Being sure to change the default admin/root password on servers (such as the database server). Specifically, changing it to something more complex and difficult to crack.
- Sources:
  - https://www.cisa.gov/news-events/alerts/2013/06/24/risks-default-passwords-internet
  - https://www.beyondtrust.com/blog/entry/privilege-escalation-attack-defense-explained