1. Team secure code

- Assume that anything that comes from outside the server, can be *anything*
- Assume that anything that leaves the server, can go *anywhere*, to *anyone*
- Keep all security/authentication-related functions on the server, and have the client program only do actions that give no sensitive knowledge of the underlying system
    - If you don't want your user to see the code you are writing, don't write it!
- Limit user input to safe characters
- Use a library escape *all* input strings (assume the input string can be *absolutely anything*, treat it ***only*** as a sequence of characters, ***always***.
    - If you need to evaluate a search query, safely parse it yourself or using a trustworthy library, so that there could be no unintended consequences
    - Nothing from the outside should be able to influence your server *without your permission and guidance*
- Set up server using Node.JS, which is a very popular platform that allows you to exactly specify your response to any request, and is very cost-effective due to its modularity, customization, quick start-up and wide usage in the field

2. Code style

- Limit the set of functions that edit a specific piece of data, such that there are fewer chances for buffer overflow related bugs
    - Intrinsically link and centralize your data such that there is less of a chance for discrepancies to appear within your code across your data structures
    - Helps by organizing the code and reducing duplication of the same data, which will prevent problems later on and save costs of refactorization
- Use **bounded** data structures that throw an error when out of bounds
    - and handle this error quietly so as to not give the hacker a chance to gain information
    - Easier for debugging, trusted data structures - saves debugging costs
- **Refrain** from using pointers, and when they must be used, make sure that their entire lifespan and related data is completely contained within a blackbox data structure (does not apply to javascript)
    - And when you *must* use pointers, ***always*** use unique_ptr or shared_ptr
- Use dynamic buffer that expand when adding new elements

3. Code style

- Make sure all web requests are secured using HTTPS with trusted certificates
    - The money that could be lost by stolen information through insecure HTTP communications could be much greater than the cost of the certificate and its setup, which is now widespread
- Every time the user sends a command, they must provide their one-way authentication info with that command
    - This authentication info must be encoded in such a way no information about the user can be deciphered by anyone except the bank, with their own ***very*** private keys and salted hashes

    - There are many trusted libraries, like OAuth, that accomplish this, so it would be cheap to use one of those libraries
- Store authentication info using ciphertext, not plaintext
    - Storing a salted hash on the server is ultimately more secure than keeping raw data on the server which can be stolen, like the Equifax data breach!

ZKP