1.) Initial Review + Analysis

Things to review:

- System architecture pattern (client/server, peer-to-peer, ???)
- Software requirements
- Tools / technology to be used
  - ex: third party libraries?
  - using dependencies?
  - database?
  - are you using 3rd party APIs?
  - are you building your own API?
- regulations
  - Health data: HIPAA
  - Card transactions/data: PCI DSS
  - Financial: GLBA
  - Privacy: EU privacy act
- what are your data sources?
- what are your data sinks?
- are you storing intermediate data?
- who/what might your application connect to?
  - what data is being transmitted over these connections?
  - How sensitive is this data?

PAUSE. BREATH.

PRIORITIZE

- what security concerns are most relevant?
- what is the level of risk or harm if your data is exposed, your server goes down, or service is interrupted?

greater risk → more security areas are applicable

more stringent security controls

- you could make a case that all security areas are applicable, but that's rarely feasible.

You have to balance

- time
- resources/budget
- risk level

- So, w/that in mind,

Which vulnerability areas are most relevant/applicable?

Encapsulation

- object-oriented paradigms
- keep implementation separate from interface
- limit access to internal object data (or internal methods)

- argument could be made that this is also a sub-set of Code Quality

Code Quality

- secure coding pratices
  - ex: credential configuration
- secure patterns
  - ex: validating access/authorization
    session management

subsets of

Code Error

- secure error handling

Input Validation

Secure API Interactions

Cryptography

- encryption use
  - keeping data secret
    when it's stored and/or
    when in transit

(secret = can't be understood without the cipher key

→ not the same as securing access to data via passwords, permissions, etc

Client/Server

- secure distributed computing

(distributed computing is when processing & data storage is distributed across multiple devices and systems