2 | Intro Secure Programming

What is Secure Programming?

- art of making sure
- that software we develop
- does what it is expected to do
- and that is all it does
- namely it can't be misused in ways foreseen at development

Why Secure Programming?

- Software is in our lives more and more (sensitive information, etc.)

How do we develop in a secure way?

- Security requirements
- Secure designs, policies, governance
- But someone has to implement this all

Why are creating designs, policies, governance difficult?

- Someone has to implement the designs, policies, governance
- If we are lucky they work from the beginning
- Even then, all design and policy guarantees can easily be broken by implementation flaws

What are often the weakest link?

- Implementation details

TODO Example Summary Apple goto fail ff.

Neccessities to master Secure Programming

- Grasp nature of software vulnerabilities (common design and implementatino mistakes, etc.)
- Understand common vulnerability classes
- Appreciate mitigation principles and techniques
- Understand the software development process and modern SDLC practices
- Being able to devise a **Threat Model**

Vulnerabilities and IT Risk

- $$\text{Risk} = \text{Impact} \cdot \text{Probability}$$
- $$\text{IT Risk} = \text{Impact} \cdot \text{Threat Level} \cdot \text{Vulnerability Level}$$
- Threat Level: Capabilities of malicious agent (skills and resources)
- Vulnerability Level: Presence of flaw and ease of exploitability

  If a vulnerability in our system is easy to exploit an attacker does not have to be especially skilled or have plentiful resources (money) to do us big harm.

Microsoft Vulnerability Mitigation Strategy

Strategy: Make it difficult and costly to find, exploit and leverage vulnerabilities

1. Eliminate

- Software vulnerability

2. Break exploitation techniques

- Class specific exploitation primitive
- Generic exploitation primitive

3. Contain

- Payload

(TODO maybe add rest of table)

Security in Software Development Process

- Requirements
- what do protect
- threat model
- Specification and design
- how to make it happen
- how to make it usable
- Testing
- e.g. penetration testing
- Maintenance
- regular review, means for quick response, ...

From Design to Implementation

- Details appear that were ignored at design time, so
- Concretize input / output
- Fields, formats, encodings, protocols
- Adopt assumptions on context
- Libraries, APIs, consumed services
- Compute environment, resources
- Concurrency model
- Attack surface grows
- Vulnerabilities arise
- Threat model refines

Attack surfaces

- Sum of the different points (the “attack vectors”)
- where an unauthorized user (the “attacker”) can try to
- enter data to an environment
- extract data from an environment
- Through an attack surface, conceptionally, an adversary can
- Modify data or behavior of a system, and/or
- Observe it

What Attack Surfaces exist?

- User interfaces
- Network connections
- System interfaces / IPC interfaces
- Database
- Storage
- Log

  Which are Modification and which are Observation Surface?

From Design to Implementation (Cont.)

- aspects were omitted from the design on purpose
- to manage complexity systems are composed of high level components
- otherwise we would be writing web apps in assembly
- This leads to a problem, because
- well, these abstractions are only present at a conceptual level
- and attackers do not respect abstraction boundaries either

For a good threat model we need to take these implementation details into account too

One Time Pad (XOR CMOS Gate)

TODO

Abstractions with Interfaces

- Interface: shared boundary across two or more components to
- exchange information
- expose functionality
- Interfaces usually
- defines data types, structures and operations
- hides implementation details, local or remote
- Local, implementation details, i.e.
- concurrency model
- logging
- file system operations
- dependencies
- (libraries, frameworks, i.e. JEE)
- Remote
- All the above + system architecture (storage, compute, location, downstream services)
- Questions of privacy and trust arise
- (SOAP/REST, WSDL /XSD/OpenAPI)

REST

- Remote Interface

TODO

TODO find a way to implement lengthy examples

Basic Threats to REST Web Service

- STRIDE
- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service

- Elevation of Priveledge

4 STRIDE Examples

- SQL Injection (T, I)
- Logging of sensitive information (I)
- Error message delegated to user (I)
- Resource not released on error, a.k.a. connection leak (D)

Access Control in Interfaces

- The interface definition does not mention access control (as it is often not part of it, expect policies for SOAP/WSDL)
- Some servers simply forward HEAD requests as GET but and omit response body; are HEAD request authorized too?
- JWT standard defines signing algorithm NONE for testing purposes; is it disabled in production?
- Any more vulnerabilities in the JWT library? https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/

(TODO)

Common Reasons why Vulnerabilities arise

- Developer negligence or ignorance (e.g. SQL Injection)
- Ignoring implementation details of used libraries (e.g. write to tmp files)
- Not knowing about possible side effects of productivity features `@SneakThrows` letting all exceptions escape unhandled
- Sloppy interface design

How to prevent Vulnerabilities?

- Educate developers
- Apply robust design techniques (i.e. Domain Driven)
- Scan source code
- Conduct code reviews
- Use tools (SAST)
- Test system components
- Unit tests
- Fuzz tests
- Penetration tests