Software security fiche

Pierre Colson

Contents

Markdown version on *github*
Compiled using *pandoc* and `gpdf` *script*

General

- **Security** is the application and enforcement of *policies* through *mechanisms* over data and resources
  - *Policies* specify what we want to enforce
  - *Mechanisms* specify how we want to enfore the policies (i.e. an implementation/instance of a *policies*)
- **Software Security** is the area of Computer Science that focuses on testing, evaluating, improving, enforcing, and proving the security of software.
- A **Software bug** is an error, flaw, failure, or fault in a computer program or sytem that causes it to produce an incorrect or unexpected result, or to behave unintented ways. Bugs arise from mistakes made by people in either a program's source code or its design, in framworks and operating systems, and by compilers

- A vulnerability is a software weakness that allows an attacker to exploit a software bug
- Ethics
  - Black hat: Attack other systems for profit (illegal)
  - Grey hat: look out for your own benfit
  - White hat: Honest security professional

Basics

Basic security principles

- CIA Security triad
  - **Confidentiality** : an attacker cannot recover protected data
  - **Integrity**: an attacker cannot modify protected data
  - **Availability**: an attacker cannot stop/hider computation
  - Accountability/non-repudiation may be used as fourth fundamental concept: it prevents denial of message transmission or receipt
- The **Threat model** defines the abilities and resources of the attcker. Threat models enable structured reasoning about the attack surface
- Security is
  - expensive to develop
  - expensive to maintain
  - may have performance overhead
  - may be inconvinient to users
- Fundamental security mechanisms
  - **Isolation**: Isolate two components from each other. One component cannot acces data/code of the other component except through a well-defined API.
  - **Least privilege**: The principle of least privilege ensures that a component has the least privilege needed to function.
    - Any further removed privilege reduces functionality
    - Any added privilege will not increase functionality
    - This property constraints an attacker in the obtainable privilege
  - **Fault compartments**: Seperate individual components into smallest functional entity possible. These unit contain faults to individual components. Allow abstraction and permission checks at boundaries
  - **Trust and correctness**: Specific components are assumed to be trusted and correct according to a specification
- **Abstraction** is the act of representing essential features without including the background details or explenations. Abstraction allow an encapsulation of ideas without having to go into implementation details
- **OS abastraction**
  - *Single domain OS*
    - A single layer, no isolation or compartimentalization
    - All code runs in the same domain: the applcation can directly call into operating system drivers
    - High performance, ofter used in embedded systems
  - *Monolithic OS*
    - Two layers: the operating sytem and applications
    - The OS manages resources and orchestrates access
    - Applications are unprivileged, must request access from the OS
    - Linux fully and Windows mostly follows this approach for performance (isolating individual components is expensive)
  - *Micro Kernel*
    - Many layers: each component is a seperate process
    - Only essential parts are privileged

    * Applications request access from different OS process
  - *Library OS*
    * Few thin layers, flat struture
    * Micro-kernel exposes bare OS services
    * Each application brings all necessary OS components
- **Hardware abstraction**
  - Virtual memory through MMU/OS
  - Only OS has access to raw physical memory
  - DMA for trusted devices
  - ISA enforces provilege abstraction
  - Hardware abstractions are fundamental for performance
- **Access Control**
  - **Authentication** : Who are you ?
  - **Authorization** : Who has access to object
  - **Audit/Provenance** : I'll check what you did
- There are three fundamental type of **identification**
  - What you know : username, password
  - What you are : biometrics
  - What you have : smartcard, phone
- **Information Flow control** : Who can see what information ?
  - Access policies are called access control models
- Type of **access control**
  - **Mandatory Access Control** (MAC)
    * Rule and lattice-based policiy
    * Centrally controlled
    * One entity controls what permissions are given
    * Users cannot change policy themselves
    * Examples : The admin sets permissions for each file
    * Bell and LaPadula : *read down and write up* => enforces *condifentiality*
    * Biba : *read up and write down* => enforces *integrity*
  - **Discretionary Access Control** (DAC)
    * Object owner specificies policy
    * MAC requires central control, DAC empowers the user
    * User has authority over her resources
    * User sets permissions for ther data if other users wanc access
    * Examples : Unix Permissions
  - **Role-Based Access Control** (RBAC)
    * Policies defined in terms or roles (sets of permissions), individuals are assigned roles, roles are authorized for tasks.

Secure software lifecycle

- **Secure software engineering**
  - Prevent loss/corruption of data
  - Prevent unauthorized access to data
  - Prevent unauthorized computation
  - Prevent escalation of privilege
  - Prevent downtime of resources
- **Secure development Cycle** (SDC)
  - **Requirement Analysis** : Define scope of a project and security/privacy boundatries. Define security specification, identify assets, asses environment, and specify use/abuse cases
    * **Threat Modeling** : threats, attack vector, and emergency plans (i.e. how to react when things go wrong)
    * **Security requirements** : privacy policy, data management plan

    * **Third party dependencies** : define third party dependencies along with their update policies, risk analysis on dependencies
- **Design** : the classic design phase focuses on functionality requirements. here we make security concerns an integral part of the analysis
    * Continuously **update threat model** as requirements change
    * **Secuity design review** : a major milestones, review security design and its interaction with functionality/requirements
    * **Design documentation** : up-to-date document of requirements and functionality with security assessments
- **Implementation** : During implementaion, the design mau be slightly refined and the security documents must be updated accordingly along with continuous reviews and analysis
    * Continuous **code review** ensure software is built according to specification and checked for bugs
    * **Static analysis** ensures high code quality and highlights flaws
    * **Vulnerability scanning** of external dependencies for exploits
    * **Unit tests** ensure functionality/security across components
    * **Accountability** : use a source code/version control system
    * **Coding Standards** : assertions and documentation
    * **Continuous integration** : run unit tests, static analysis, and linter whenver code is checked in
- **Testing** : Completed components are rigorously tested before they are finally integrated into the prototype
    * **Fuzzing** is a form of probabilistic test integration
    * **Dynamic analysis** complements fuzzing with heavy-weight tests based on symbolic execution and models
    * **Third party penetration testing** provides external validation and clean slate testing
- **Release** : Before release of the final prorotype, verify the base assumptions from the initial requirements analysis and design
    * **Security review** : check for compliance of security properties
    * **Privacy review** : check for privacy policy complicance
    * **Review all licensing agreements**
- **Maintenance** : Afte shipping software, continuously maintain security properties
    * **Track third party software** and update accordingly
    * **Provide vulnerability disclosure contacts** through, e.g. a bu bounty program or a least a public contact
    * **Regression testing** : whenever an update is deployed recheck security and functionality requirements
    * **Deploy updates securly**

Policies and Attacks

Security prolicies

Bug, a violation of a security policy

Attack vectors

Stopping Exploitation

Mitigations

Advanced mitigations

Fiding bugs

Testing

Sanitizer

Case study ?

Browser security

Web security

Modbile Security

Summaries

- **Basis Security principles**
  - Sofware security goal: allow intended use of software, prevent unintented use that may cause harm
  - Security triad : Condidentiality, Integrity, Availability
  - Security of a system depends on its thread model
  - Concepts: isolation, least privilege, fault compartments, trust
  - Security relies on abstractions to reduce complexity
- **Secure software lifecycle**
  - Secure software development enforces security principles during software development
  - Software lives and evolves
  - Security must be first class citizen
    - Secure Requirements/specification
    - Security-aware Design (Threats?)
    - Secure Implementation (Reviews?)
    - Testing
    - Updates and patching