Secure Software Programming and Vulnerability Analysis

Christopher Kruegel chris@auto.tuwien.ac.at
http://www.auto.tuwien.ac.at/~chris

Architecture

Overview

- Security issues at various stages of application life-cycle
  - mistakes, vulnerabilities, and exploits
  - avoidance, detection, and defense
- Architecture
  - security considerations when designing the application
- Implementation
  - security considerations when writing the application
- Operation
  - security considerations when the application is in production

Architecture

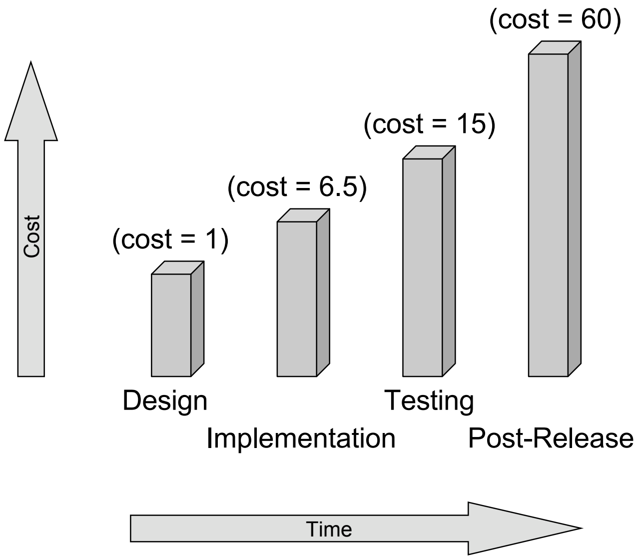
- What is security architecture?

A *body of* high-level *design principles* and decisions that allow a programmer to say "Yes" with confidence and "No" with certainty.

A *framework* for *secure design*, which embodies in microcosm the four classic stages of information security: protect, deter, detect, and react.

Architecture is Important

Cost of fixing security flaws during different development phases



Design Principles

- Design is a complex, creative process
- No standard technique to make design secure
- But general rules derived from experience

- 8 principles according to Saltzer and Schroeder (1975)
  - Economy of Mechanism
  - Fail-safe defaults
  - Complete mediation
  - Open design
  - Separation of privilege
  - Least privilege
  - Least common mechanism
  - Psychological acceptability

Economy of Mechanism

- Design should be as simple as possible
  - KISS -- keep it simple, stupid
  - Brian W. Kernighan

    Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

- Black-box / functional testing
  - exercises desired code / access paths
  - usually does not discover security problems
  - makes white-box testing / code auditing necessary

- For successful white-box testing, simple design necessary

Fail-safe Defaults

- Allow as default action
  - grant access when not explicitly forbidden
  - in case of mistake, access allowed (often not noticed)
  - improves ease-of-use
  - wrong psychological model

- Deny as default action
  - grant access only on explicit permission
  - in case of mistake, access denied (noticed quickly)
  - improves security
  - important for firewall configurations
    and input validation tasks

Fail-safe Defaults

- Configuration
  - secure initial configuration
  - easy (re)configuration
- Secure initial configuration
  - no default passwords
  - no sample users
  - files are write-protected, owned by root
- Error messages
  - should be very generic
  - additional information in log files

Complete Mediation

- Complete access control
  - check every access to every object
  - include all aspects (normal operation, initialization, maintenance, ..)
  - caching of checks is dangerous
  - identification of source of action (authentication) is crucial
- Trusted path
  - make sure that user is talking to authentication program
  - important for safe login (thwart fake logins)
  - Windows “control-alt-delete” sequence

Complete Mediation

- Secure interface
  - minimal
  - narrow
  - non-bypassable (e.g., check at server, not client)

- Input validation

- Trust input only from trustworthy channels
  - any value that can be influence by user cannot be trusted
    - do not authenticate based on IP source addresses / ports
    - email sender can be forged
    - hidden fields or client side checks are inappropriate
    - reverse DNS lookup
  - safely load initialization (configuration)

Open Design

- Design must not be secret
  - security mechanisms must be known
  - allows review
  - establishes trust
  - unrealistic to keep mechanism secret in widely distributed systems

- Security depends on secrecy of few, small tokens
  - keys
  - passwords

- Many violations of this principle
  - especially proprietary cryptography

Separation of Privilege

- Access depends on more than one condition
  - for example, two keys are required to access a resource
  - two privileges can be (physically) distributed
  - more robust and flexible
- Classic examples
  - launch of nuclear weapons requires two people
  - bank safe
- Related principle
  - compartmentalization

Separation of Privilege

- Compartmentalization
  - break system in different, isolated parts and
  - minimize privileges in each part
  - don’t implement all-or-nothing model
    - Unix root model

  → minimizes possible damage
- Sandbox
  - traditional compartmentalization technique
  - examples
    - Java sandbox (bytecode verifier, class loader, security manager)
    - virtual machines, system calls
    - paper -- Goldberg et al.,
      A secure environment for untrusted helper applications.

Least Privilege

- Operate with least number of rights to complete task
  - minimize damage
  - minimize interactions between privileged programs
    - reduce unintentional, unwanted use
  - when misuse occurs, only few potential sources need auditing
- Minimize granted privileges
  - avoid setuid root programs
    - use groups and setgid (group games for high scores)
    - use special user (nobody for web server)
  - make file owner different from setuid user
    - taking control of process does not allow to modify program images

Least Privilege

- Minimize granted privileges
  - database restrictions
    - limit access to needed tables
    - use stored procedures
  - POSIX capabilities
- Minimize time that privilege can be used
  - drop privileges as soon as possible
  - make sure to clear saved ID values
- Minimize time that privilege is active
  - temporarily drop privileges
  - can often be enabled by attacker as well,
    but protects against some attacks (e.g., file access)

Least Privilege

- Minimize modules that are granted privilege
  - optimally, only single module uses privileges and drops them
  - two separate programs
    - one can be large and untrusted
    - other is small and can perform critical operations
    - important for GUI applications that require privileges

- Limit view of system
  - limit file system view by setting new root directory `chroot`
  - more complete virtual machine abstraction BSD system call `jail(2)`
  - honeypot

Least Privilege

- Do not use setuid scripts
  - kernel race condition problem
  - Linux drops setuid settings

- Minimize accessible data
  - CGI scripts
    - place data used by script outside document root

- Minimize available resources
  - quotas
  - ulimits

- Paper -- Provos et al., *Preventing Privilege Escalation.*

Least Common Mechanisms

- Minimize shared mechanisms
  - reduce potentially dangerous information flow
  - reduce possible interactions
  - more flexible

- Problems
  - beware of race conditions
  - avoid temporary files in global directories

Psychological Acceptability

- Easy-to-use human interface
  - easy to apply security mechanisms routinely
  - easy to apply security mechanisms correctly
  - interface has to support mental model
    - do what is expected intuitively

- Authentication
  - passwords
    - enforce minimum length
    - enforce frequent changes
  - PKI (public key infrastructure)
    - overhead vs. security
    - openssh did a decent job

One more Design Principle

- *Separate data and control*
  – failed separation is reason for many security vulnerabilities
    - from buffer overflows to macro viruses
  – distinction between control information and data has to be clear

- Problematic
  – with automatically executing code in data files
    - javascript in web pages
    - automatic preview of web pages in emails
    - macros in word
  – when using mobile code
    - code that is downloaded and executed locally

Retrofitting Applications

- Applying security techniques to existing applications
  – element of overall system design
  – when no source code available or
  – complete redesign too complicated

- Wrappers
  – move original application to new location and
  – replace it with small program or script that
    - checks (and perhaps sanitizes) command-line parameters,
    - prepares a restricted runtime, and
    - invokes the target application from its new location
  – can provide logging
  – can provide possibility for prologue and epilogue code

Retrofitting Applications

- Example wrappers
  - AusCERT Overflow Wrapper
    - exits when any command line argument exceeds a certain length
  - TCP Wrappers
    - replaces inetd (for telnet, ftp, finger, …)
    - access control
    - logging
  - sendmail restricted shell (smrsh)
    - sendmail known for security problems
    - smrsh restricts accessible binaries
    - vulnerable to two exploits that allow arbitrary programs to be executed

Retrofitting Applications

- Interposition
  - insert program that we control between two pieces of software that we do not control
  - filtering of data
    - add security checks and constraints
  - network proxy
    - application policy enforcement
    - SYN flood protection
  - input sanitization

Bad Practice

- Being too specific too soon
  - without having a design, solve technical problems and start implementation
- Focus only on functionality
  - security must be built in from the beginning
- Not considering economic factors
  - ignoring the cost of security features

Bad Practice

Automation Systems Group

- Not considering the human factor
  - propose solutions that users strongly dislike
    - biometric scanners instead of passwords
  - propose solutions that are annoying
    - change passwords to frequently
    - terminate idle sessions too fast
  - propose solutions that require considerable additional effort
    - producing too many alerts
    - require checking of many different log-files

Summary

- Security must be considered from the start
  - security architecture
  - keep costs to fix problems low
- Security architecture
  - process cannot be automated
  - ask important questions such as
    - what to protect (assets)
    - what to protect against (threat model)
    - how to protect (architecture)
- Most important design principles
  - least privilege
  - separate data and control