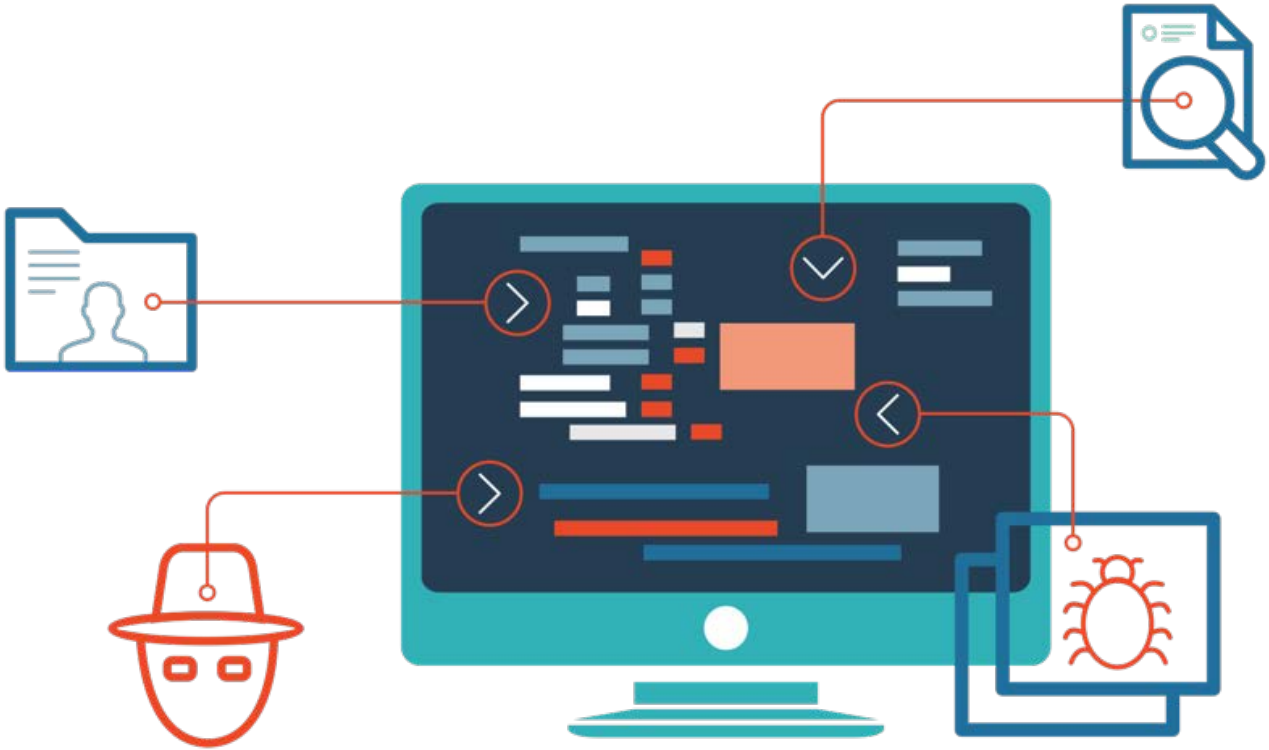


7. Application Security

Jin Hong

jin.hong@uwa.edu.au

Demo preparation

- We will use both Kali and Windows 10 VMs
- On the Windows VM:
  - Install Apache Open Office (latest version is fine).
    - ~470MBs
  - Install JRE 7 Update 72 32-bit
    - https://www.filehorse.com/download-java-runtime-32/17560/download/
    - ~120MBs

Application Security

- Nowadays almost all applications utilize the internet for functionalities
  - Getting updates
  - Checking user requests
  - Saving states of the applications etc.
- No point drawing a line between application and web security
  - We now have web application security!
- We will look at both developing software applications issues and some examples of malicious software

Secure Application Engineering Issues

- Many possible security vulnerabilities for a software system, depending on:
  - Requirements (missing, misunderstood, features not securely implemented)
  - Design
  - Implementation language/library issues
  - Subsystem interactions/issues
  - Network environment issues
  - (Lack of) testing/vulnerability analysis
  - Other factors

Software Development

- Common specifications of software development process
  - ISO 12207 – Systems and software engineering – software life cycle processes
    - Business or mission analysis
    - Stakeholder needs and requirements definition
    - Systems/Software requirements definition
    - Architecture definition
    - Design definition
    - System analysis
    - Implementation
    - Integration
    - Verification
    - Transition
    - Validation
    - Operation
    - Maintenance
    - Disposal

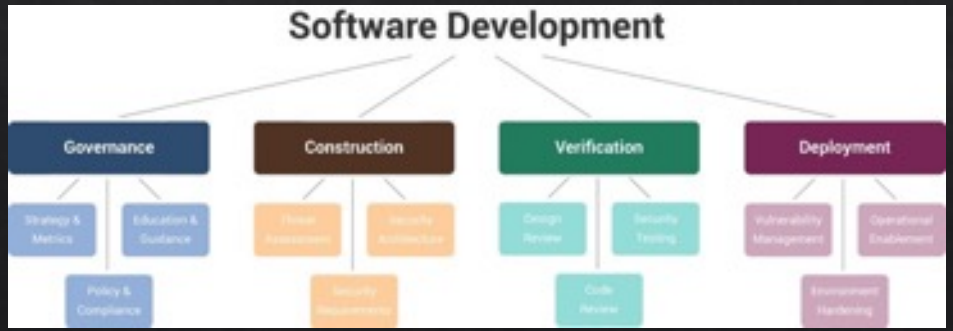
Note: Security related tasks are not explicitly specified

SLC - Organisation

- Many different SLC methodologies:
  - **Waterfall** – single series of steps
  - **Iterative** – multiple iterations, one feature at a time
  - **Spiral** – requirements or functions analysed for risks, developed as prototype, integrated into whole
  - **MS SDL** (MS Security Development Lifecycle) – Microsoft methodology
  - **Team Software Process** (TSP) – framework with set of processes, used with Personal Software Process (PSP) and CMMI
  - **Agile** methodologies – e.g. Test-driven development (TDD): gather user stories, generate reasonable and prioritized set of features, then cycle (per feature) of develop tests, implement, refactor design if needed
  - **Touchpoints** – from Gary McGraw
    - http://www.swsec.com/resources/touchpoints/

Other Approaches

- OWASP Software Assurance Maturity Model (SAMM)
- Building Security In Maturity Model (BSIMM)



SAMM

BSIMM

https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-approach-secure-software-development/

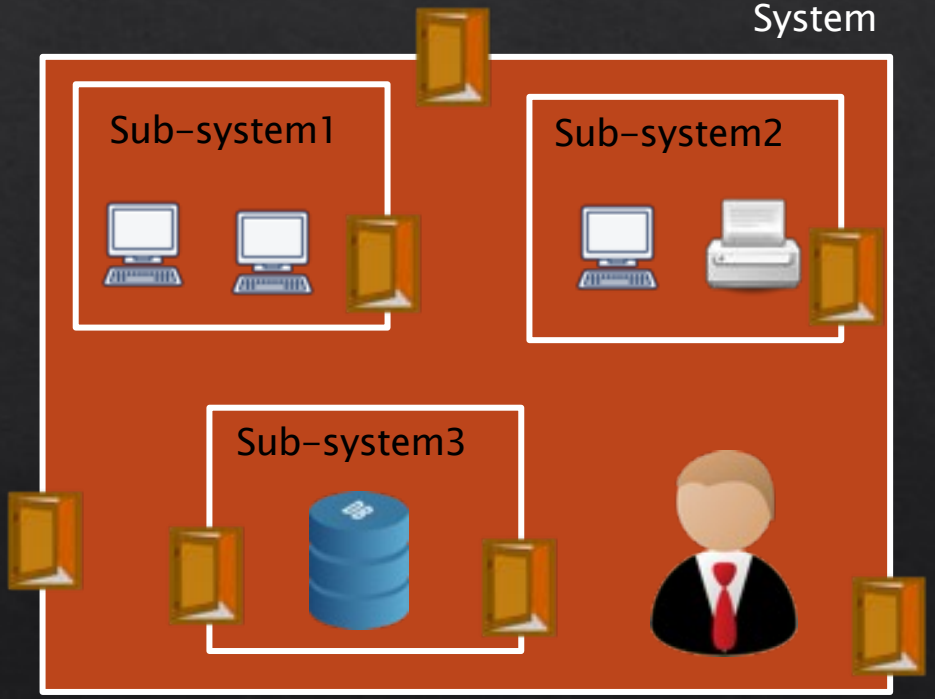
Security in Design – Architectural Level (1)

- Security Architectural Patterns
  - Single-Access Point Pattern
    - single point of entry into system
  - Check Point Pattern
    - centralised enforcement of authentication and authorisation
  - Role Pattern
    - disassociation of users and privileges

Security in Design – Architectural Level (2)

- Security Architectural Patterns
  - Distrustful Composition
    - decompose multi-function systems to reduce attack surface
  - Privilege Separation
    - Reduce the amount of code that works with increased privilege level
  - Defer to Kernel
    - Use existing user verification functionality within kernel to avoid reinventing decision control

Our System

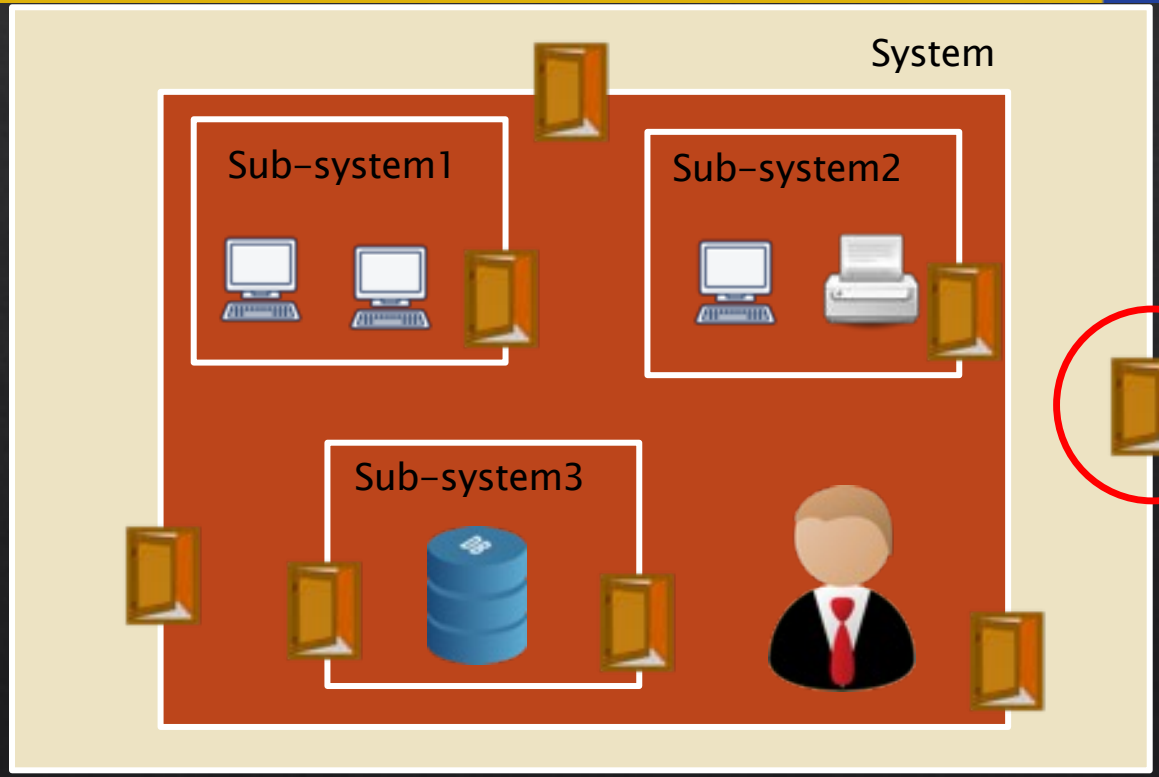


Single-Access Point Pattern

- System should have single point of entry
  - Common to all incoming requests
  - **All** users must pass through it
  - Any unauthorised requests of any other component of the system are directed to this point

- Example: Java Enterprise Edition (EE) – supports with declarative security implemented through web container

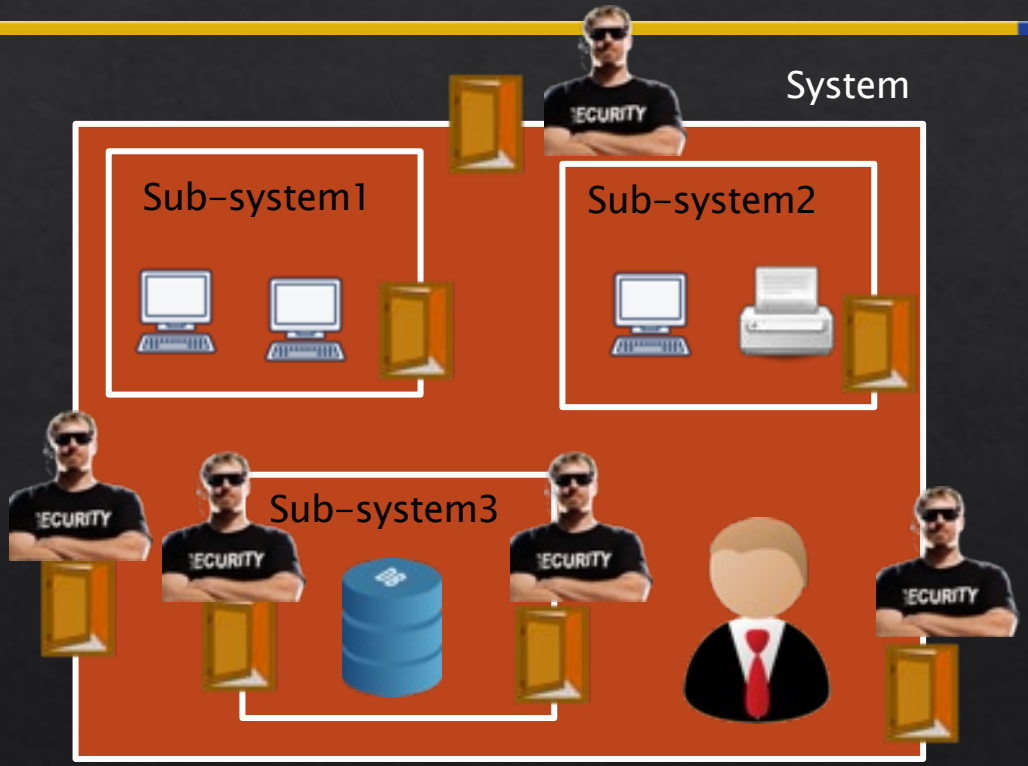
Single-Access Point Pattern



Check Point Pattern

- Centralises and enforces authentication and authorisation
- Single mechanism determines if access (or ability to grant access) to particular resource is proper
- Matches user with **level of security** allowed
- May be multiple check points
- Java EE supports with Java Authentication and Authorisation Service (JAAS)
- Pluggable Authentication Module (PAM) also supports this pattern

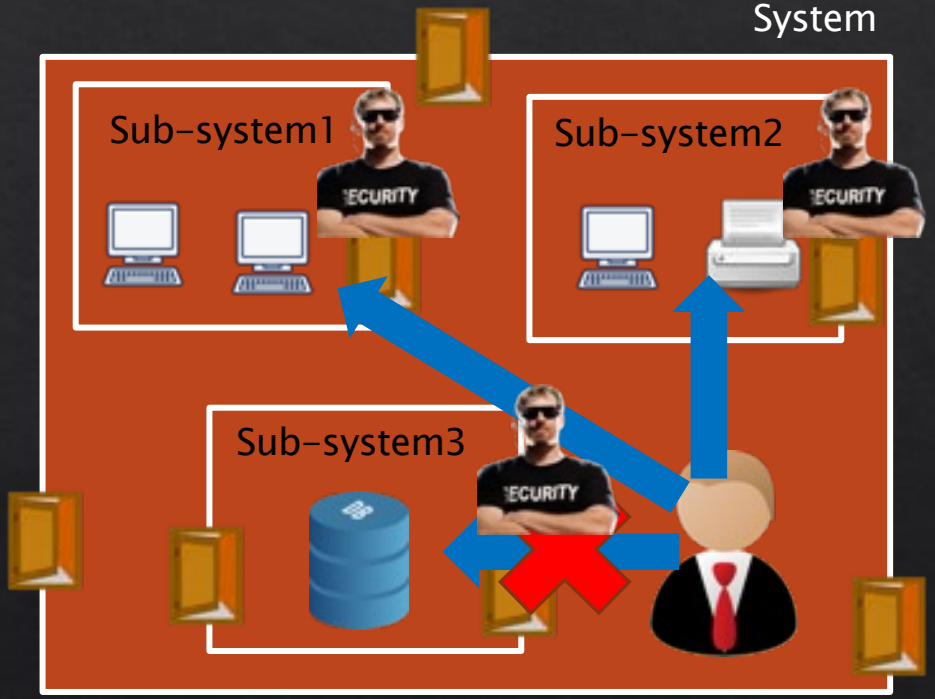
Check Point Pattern



Role Pattern

- Disassociation of users and their privileges
- Avoid an un-manageable rats' nest of combinations
- User assigned one or more roles
- Role assigned one or more privileges
- Supports easier management
- Java EE supports through declarative security model
  - Can specify in XML deployment descriptors
- Found in other large software systems (e.g. Oracle)

Role Pattern

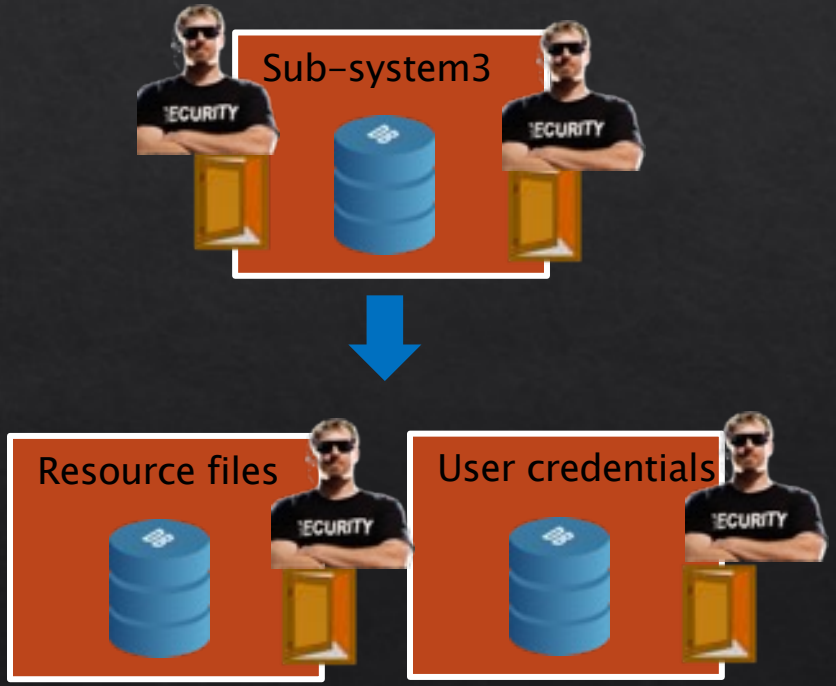


Distrustful Composition

- Normally, functionality accumulates within modules, leaving large modules with multiple functionality and possibly multiple interfaces and/or attack surfaces

- Pattern: decompose multi-function systems to **reduce** the attack surface
  - Small cohesive units have less vulnerabilities by providing a smaller attack surface
  - Small, modular units can be better isolated and tested

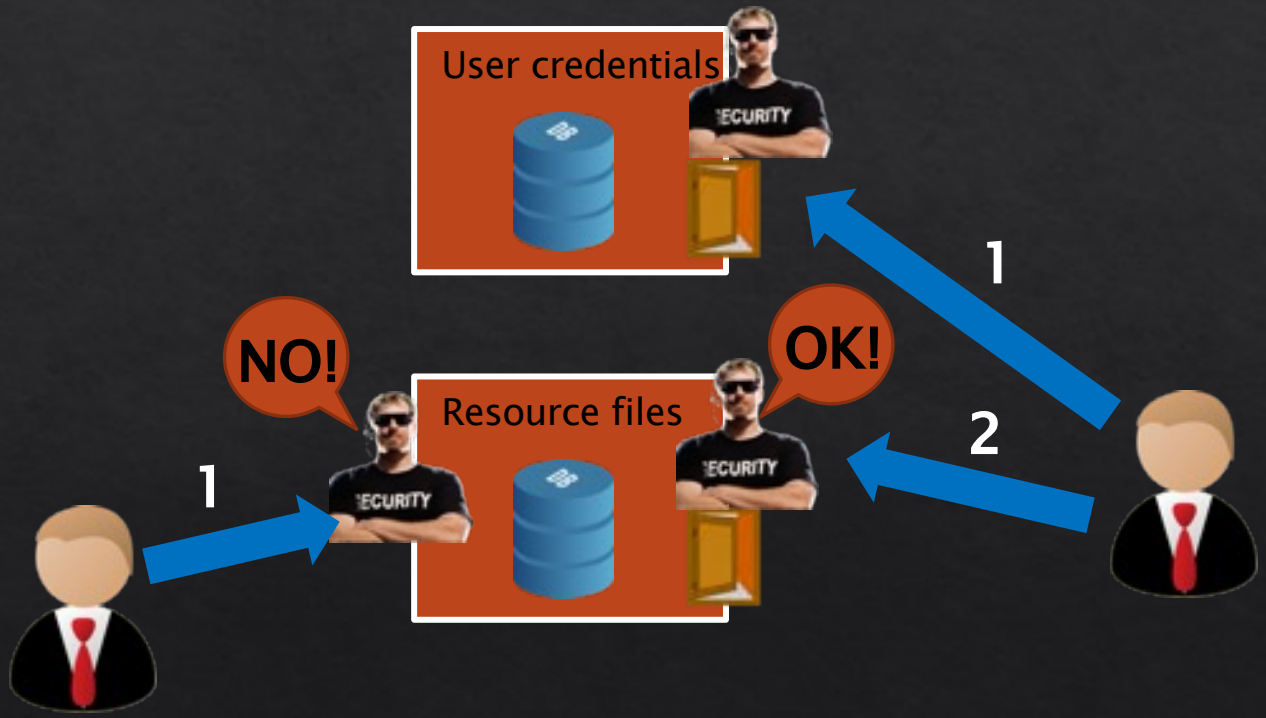
Distrustful Composition



Privilege Separation

- Reduce the amount of code that works with increased privilege level
- Specific instance of Distrustful Composition
- Uses pre-authentication and post-authentication states
  - Pre-authentication: before undergoing authentication, client or responding server process are given no privilege
  - Post-authentication, privilege is introduced, but not directly given to the client
- Uses privileged parent process as monitor and less-privileged child process as worker

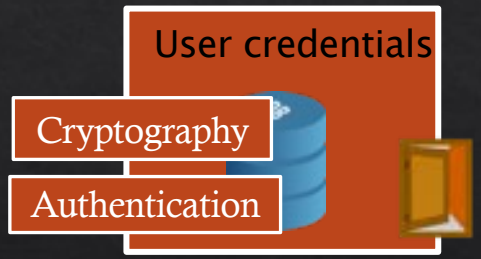
Privilege Separation



Defer to Kernel

- Use **existing** user verification functionalities within **kernel** to avoid reinventing decision control
  - Avoid re-implementing work that the kernel already can do
  - The kernel has proven, existing security features and has been analysed more thoroughly than new code

Defer to Kernel



kernel

- Cryptography
- Authentication
- Etc.

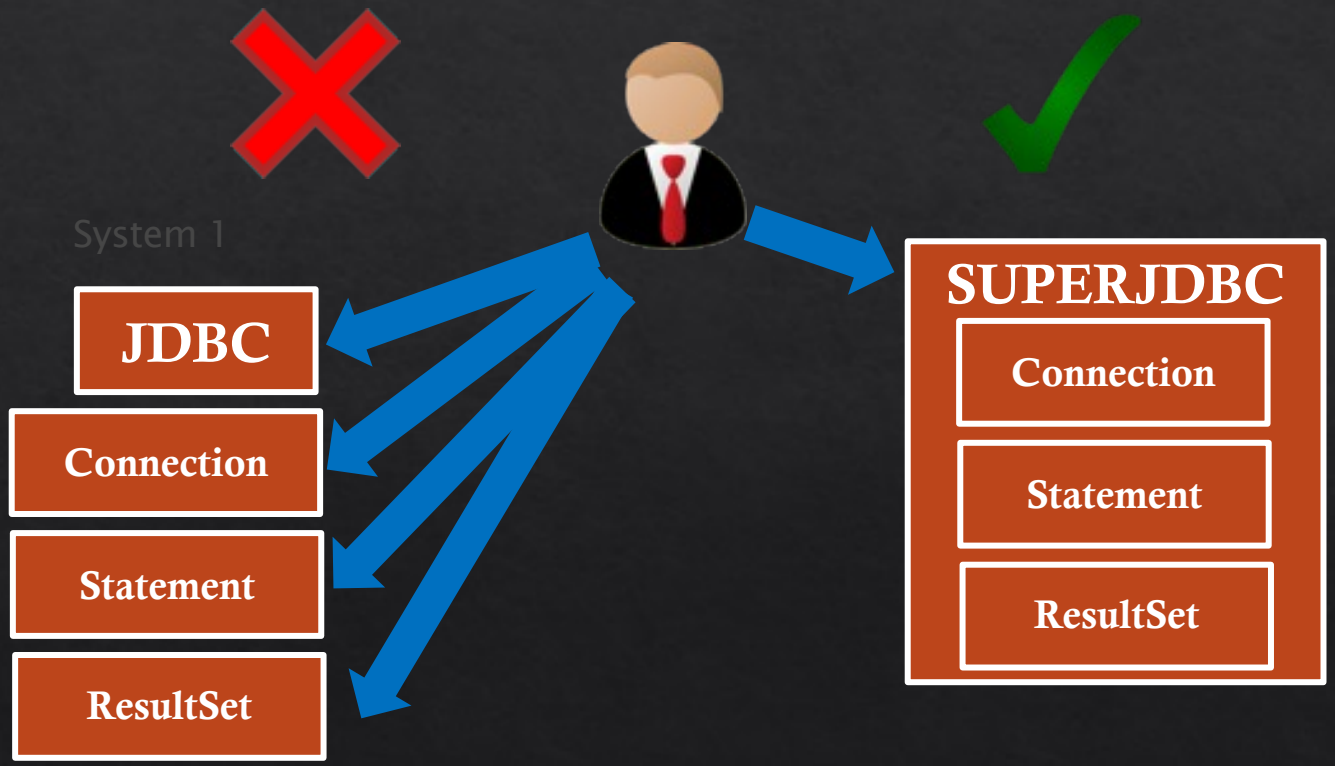
General Design Patterns Relevant to Software Security

- Two Examples:
  - Façade (fa-saad)
  - Proxy

Facade Design Pattern (1)

- Facade pattern provides an **unified** interface to a complex subsystem
  - Example: JDBC interface requires client to use three major Java classes: `Connection`, `Statement` and `ResultSet`
  - Could generate a **single** facade class, so that client can access that class, and that class deals with the actual detail work through the three classes, as well as providing an additional point of security

Facade Design Pattern (2)



Facade Design Pattern (3)

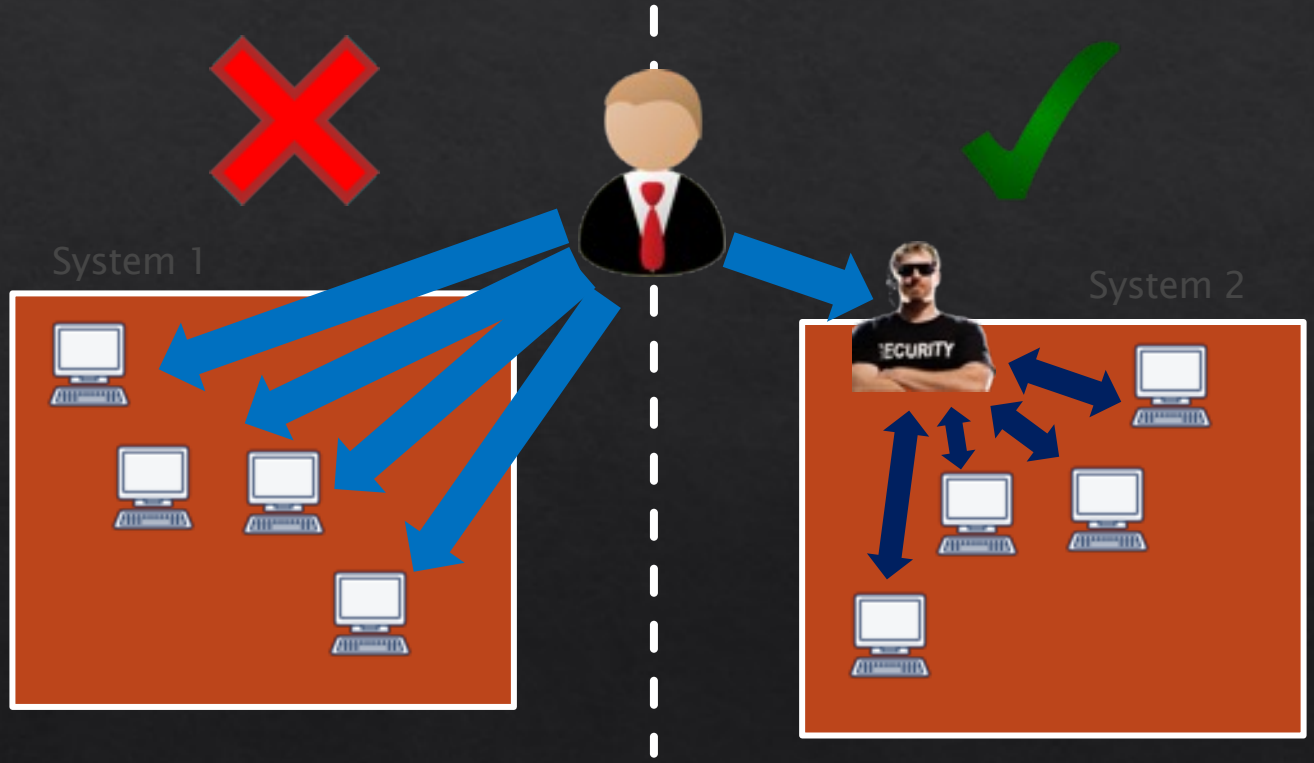
- Not only simplifies class structure (from client view) and guides user, but limits what client can do with the complex subsystem

- Facade can help implement *Single Access Point* and *Check Point* security patterns

Proxy Design Pattern (1)

- Used to provide varying degrees of access and instantiation on demand to varying clients
- Proxy acts as a *reference* to the real object
- Many networking systems use proxy classes to protect actual server classes from direct contact and security issues
- Proxy controls access to object, may be responsible for creating/deleting it

Proxy Design Pattern (2)



Proxy Design Pattern (3)

- Proxy can help implement a security layer:
  - Authentication
  - Authorisation
  - Filtering requests
  - Auditing/Logging

Security Testing Overview

- Involves good software engineering principles regarding testing
- Involves security mindset, principles, and consideration

Software Testing

- Two possible results of a test
  - Positive test – no problem found
    - ?
    - ?
  - Negative test – problem found
    - ?

Software Testing

- Levels of testing
  - Unit testing
  - Integration testing
  - System testing
  - Acceptance testing

- Good testing done by multiple testers that are not developers for that code
  - Often independent quality assurance (QA) group
  - Exception: unit testing is done by developers

Software Testing

- Example Testing Strategy
  - Unit Testing (Module testing)
    - debuggers, tracers
    - done by programmers as they develop (hopefully continuously)
- Advantages
  - Tests are run very frequently - issues are identified quickly
  - Most granular form of test - high test coverage
- Disadvantages
  - Not many security vulnerabilities can be tested at this layer

Software Testing

- Example Testing Strategy (cont)
  - Integration Testing
    - communication between modules
    - start with one module, then add incrementally
- Advantages
  - Can test in the application server
  - Many security vulnerabilities can be tested (e.g., Injection, Authentication flaws and Authorisation flaws)
- Disadvantages
  - Not executed as often as unit tests
  - Overhead of starting an application server
  - Some vulnerabilities may not be easily testable, e.g.: XSS, URL filtering performed by a web server or application firewall.

Software Testing

- Example Testing Strategy (cont)
  - System Testing
    - manual procedures, restart and recovery, user interface
    - real data is used
    - users involved
- Advantages
  - Can test the security of whole system
  - All potential security vulnerabilities can be tested
- Disadvantages
  - Overhead of starting an application server
  - Some vulnerabilities may not be easily testable

Software Testing

- Example Testing Strategy (cont)
  - Acceptance Testing
    - user-prepared test data
    - Verification Testing, Validation testing, Audit Testing
- Advantages
  - Full testing of external API
  - Security consultants can use tools to script vulnerabilities
  - Documents vulnerabilities
  - Easy retesting
- Disadvantages
  - Low test coverage
  - Developers aren't involved in testing

Software Testing

- Two testing types
  - White Box Testing
    - Assume access to source code
    - Performed by examining logic flow through code
  - Black Box Testing
    - Assume no access to source code
    - Performed by testing with expected and unexpected input, examining output and side effects

Software Testing

- In general:
  1. Decide how much security is needed
  2. Carry out automated testing
  3. Test at every stage
  4. Make a test plan
  5. All system components are tested

Software Testing

- Two aspects of security testing:
  - Testing ***security mechanisms*** in system to ensure that their functionality is properly implemented
    - Part of testing general system functionality
    - Test authentication, authorization, access control, etc.
  - Performing ***risk-based security testing***, motivated by understanding and simulating the approach of a likely attacker
    - Not considered part of general functionality testing
    - Testing misuse cases to see if they have been prevented

- Question: Who is qualified/best suited to do these two types of security testing?

Ransomware

- Type of a malware
- Encrypts the system files using a cryptosystem
  - Can be either symmetric or asymmetric, as long as the key is kept secret by the attacker
- The attacker can decrypt the files using the private key
  - In order to get your files decrypted, attackers ask for "ransom"
  - Normally paid through cryptocurrency for anonymity
  - Attackers often do not decrypt files after receiving the money
  - Ransomware-as-a-service can be purchased online

Ransomware

- Ransomwares can affect the system in many ways
  - Software availability
  - Compromise security configurations
  - Tampering data
  - Loss of sensitive information
  - Breach of confidentiality

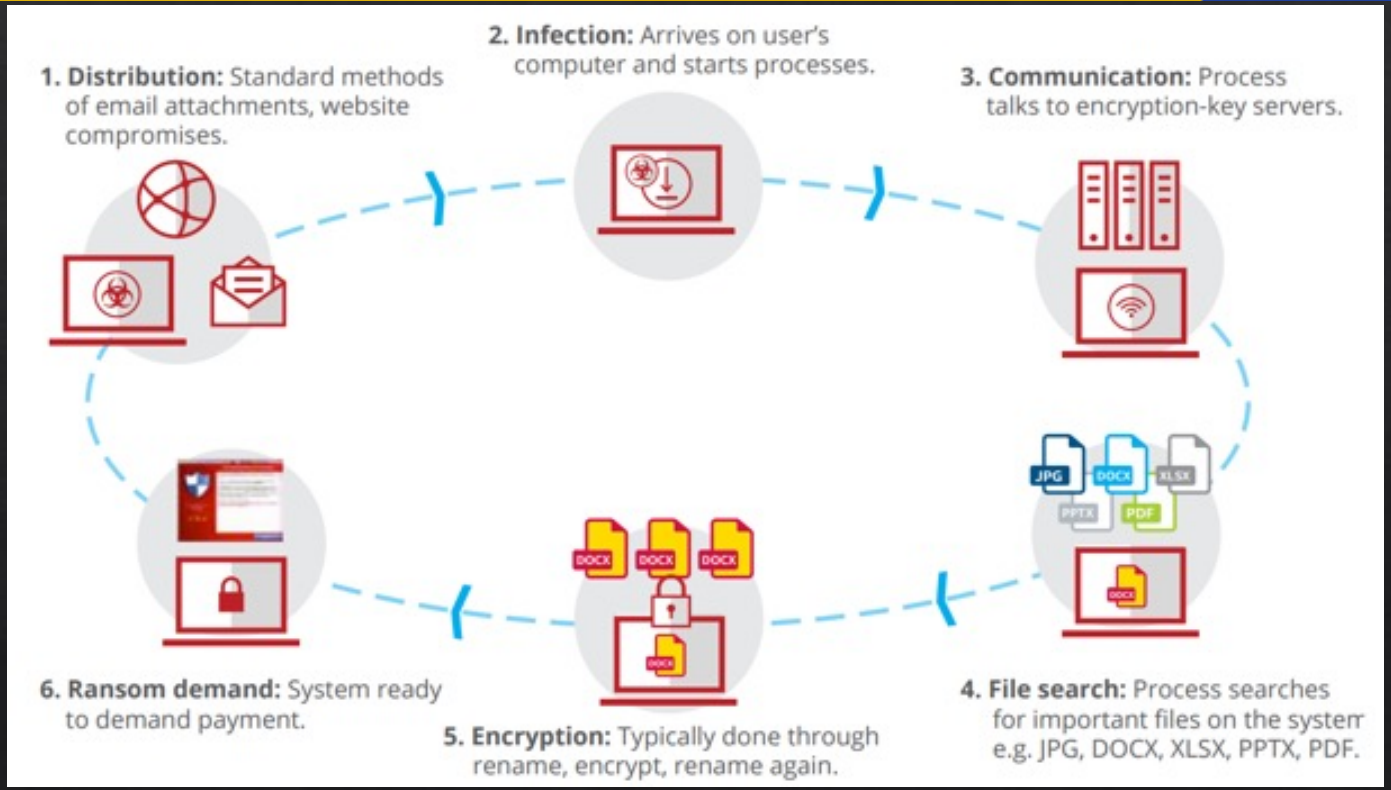
Ransomware

- Types include
  - Locker ransomware
    - Locks the computer
  - Crypto ransomware
    - Requires decryption key
  - Master boot record ransomware
    - The MBR is attacked so the message appears on boot up
  - Web server encrypting ransomware
    - Affects the web server files
  - Mobile device ransomware
    - Affects mobile devices

Ransomware

- Notable ransomwares include
  - CryptoLocker (2013) – over 500,000 infected
  - TeslaCrypt (2015) – CryptoLocker variant
  - SimpleLocker (2015) – android platform
  - WannaCry (2017) – patch available, but not implemented
  - NotPetya (2017) –cyberattack on Ukraine, global damage over $10B

Ransomware



https://www.mcafee.com/au/resources/white-papers/wp-understanding-ransomware-strategies-

Ransomware

- (1) Distribution and (2) installation
  - To convince the victim to download the executable of the ransomware
  - Uses exploit kit to lure people to click links
  - Send malicious email attachments
  - USB driver installations
  - Malvertising

Ransomware

- (3) Communication
  - First, identify the system configuration
  - Connect to the attacker server
  - Transmit victim files
  - Transmit encryption details used to lock the system

Ransomware

- (4) File search and (5) Encryption
  - The encryption process is run silently (may be visible in task manager)
  - Search files that matches the extensions starting from the local files, then removable media, then network locations
  - Malware copies are added as autorun to the registry key
    - Persist the malware when the system reboots
  - Malware can also remove backup files

Ransomware

- The ransomware malware can selectively encrypt files at various locations – why?
  - To ensure the system is still functional
  - Needs an application to display the ransom message

Ransomware

- (6) Ransom demand
  - Notify the victim of the ransomware
  - Uses anonymous currency to receive the ransom
    - Popular – Bitcoins
  - Attackers may not unlock the system even with the paid ransom
    - Many victims experienced this

Demo?

wget https://raw.githubusercontent.com/uwacyber/cits3006/2022s2/cits3006-labs/files/ransomware.zip

Ransomware Protection

- Backup files
- Keep up to date with security
- Review permissions
- Security policy for phishing
- Anti-ransomware
- Decryption tools

Malicious Applications

- We can also create some malicious applications using techniques we learned so far
  - Let's create a malicious document file
  - We actually already know all the necessary techniques to do this

Malicious Applications

Demo

- We will create the malicious doc as described before.
- You will need Apache Open Office setup to actually do the exploit.

- This demo works for MS Word doc too, but since installing MS Word on our demo Windows VM is a bit of pain, we will just exploit Open Office instead.

wget https://raw.githubusercontent.com/uwacyber/cits3006/2022s2/cits3006-labs/files/report.odt

Application Security

- We need to carefully consider how we develop applications
  - People are too rushed to create applications without considering security implications
- We need to be careful when using applications from others
  - Attackers can easily create applications that looks like legitimate applications
  - Always check the source of the application
  - If the checksum is given, good practice to confirm this

Additional Items

- Ransomware
  - https://www.safaribooksonline.com/library/view/ransomware/9781491967874/ch01.html
  - http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf
  - http://documents.trendmicro.com/assets/wp/wp-ransomware-past-present-and-future.pdf
  - https://asecuritysite.com/ransomware_new01.pdf
- Web application vulnerability scanning tools
  - W3af (http://w3af.org/)
  - wpoison (https://sourceforge.net/projects/wpoison/)
  - Wapiti (http://wapiti.sourceforge.net/)
  - OWASP ZAP (https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

Additional Items

- Agile
  - MS SDL: https://www.blackhat.com/presentations/bh-dc-10/Sullivan_Bryan/BlackHat-DC-2010-Sullivan-SDL-Agile-wp.pdf
  - SAFECODE: http://www.safecode.org/publication/SAFECode_Agile_Dev_Security0712.pdf
- US CERT: security requirements engineering
  - https://www.us-cert.gov/bsi/articles/best-practices/requirements-engineering/security-requirements-engineering
- OWASP SAMM: Software Assurance Maturity Model
  - https://www.owasp.org/index.php/OWASP_SAMM_Project