

# Unit #4b

MIS5214

Application Security

# Agenda

- Team Project Guidance
- Distributed Systems
  - File Server Architecture
  - Client/Server Architecture
  - N-Tier Architecture
  - Cloud Architecture
  - Service Oriented Architecture (SOA)
- Example Cloud-based N-Tier SOA Application Development System
- Control Stages, Objectives, Application Security Testing
- Additional Best Practices

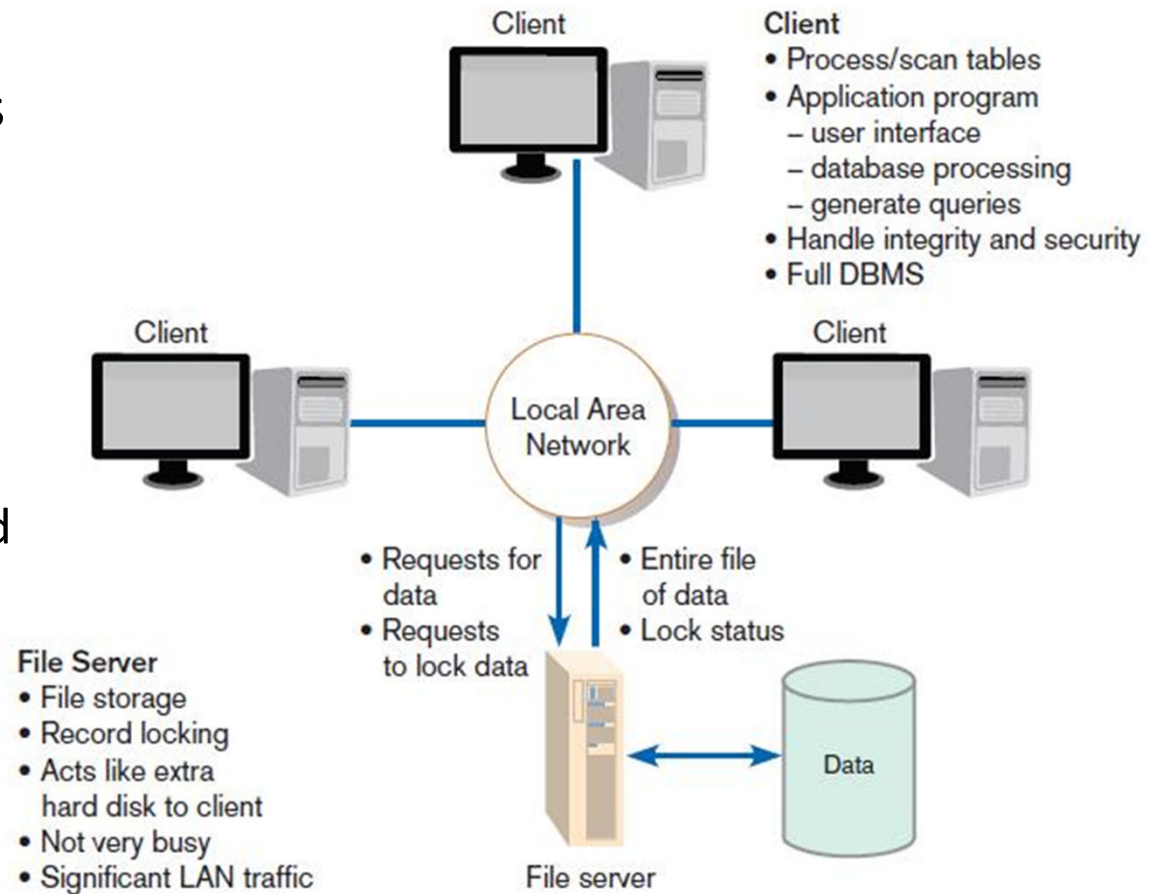
# Agenda

- Distributed Systems
  - File Server Architecture
  - Client/Server Architecture
  - N-Tier Architecture
  - Cloud Architecture
  - Service Oriented Architecture (SOA)
- Example Cloud-based N-Tier SOA Application Development System
- Control Stages, Objectives, Application Security Testing
- Additional Best Practices

# File Server architecture

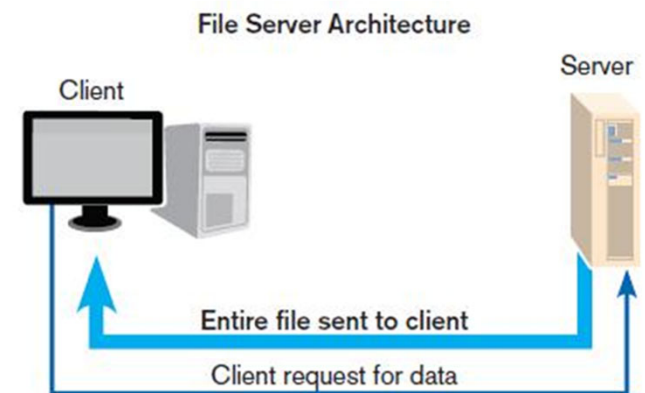
**File server:** a device that manages file operations and is shared by each client PC attached to a LAN

- The simplest configuration
  - Applications and data control take place on the client computers.
  - The file server simply holds shared data



# Limitations of File Server Architecture

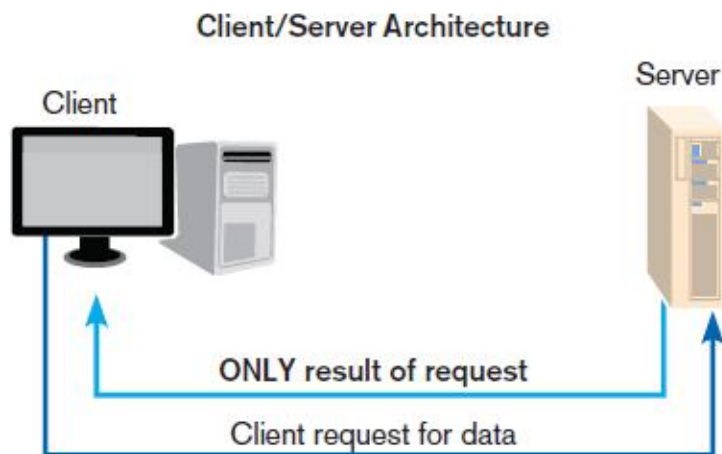
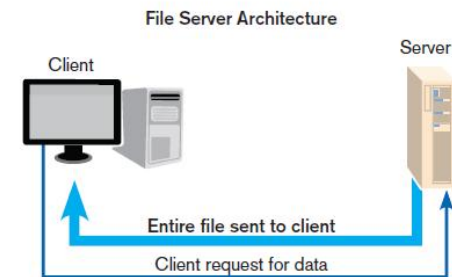
- Excessive data movement
  - Entire dataset must be transferred, instead of individual data records
- Need for powerful client workstations
  - Each client workstation must devote memory and computational resources to run a complete standalone application
- Decentralized data control
  - Data file concurrency control, recovery, and security are complicated



# Client-Server Architecture

LAN-based computing environment in which

- A central database server or engine performs all database commands sent to it from client workstations
- Application programs on each client concentrate on user interface functions



Application processing is divided between client and server

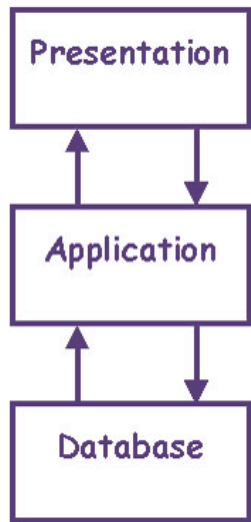
Client manages the user interface

Database server is responsible for data storage and query processing

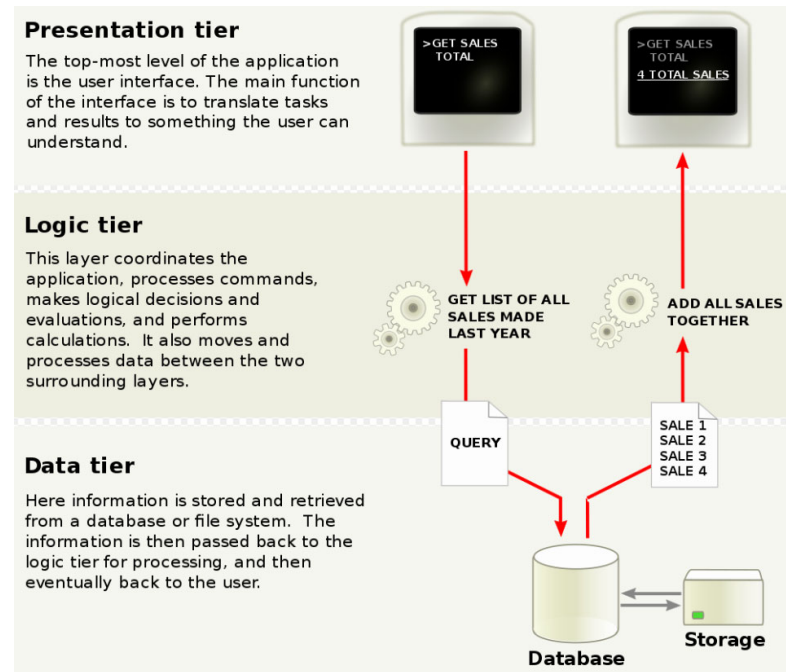
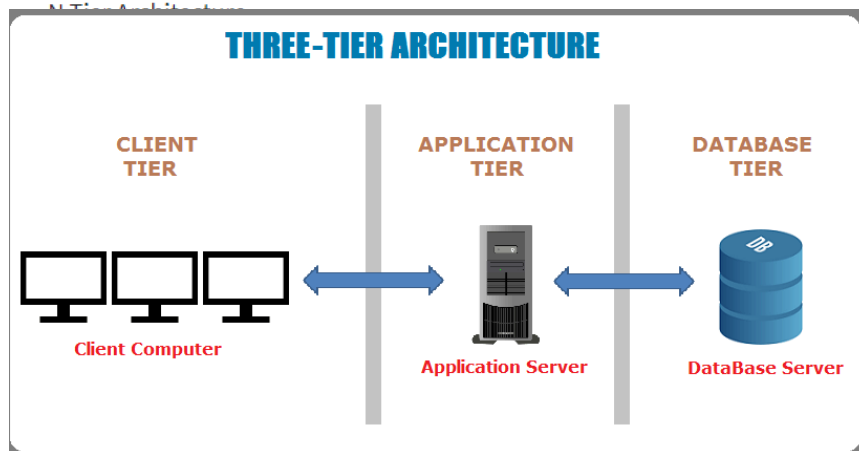
Increased efficiency and control over File server

- Server only sends specific data, not entire files, which saves on network bandwidth
- Computing load is carried out by the server
  - Increasing security
  - Decreasing computing demand on the clients

# N-Tier Architecture



N-tier architecture (also called multi-tier architecture) is a software architecture pattern that divides an application into multiple layers (tiers) to separate concerns and improve scalability, security, and maintainability. Each tier is responsible for a specific function, and they communicate with each other over a network.



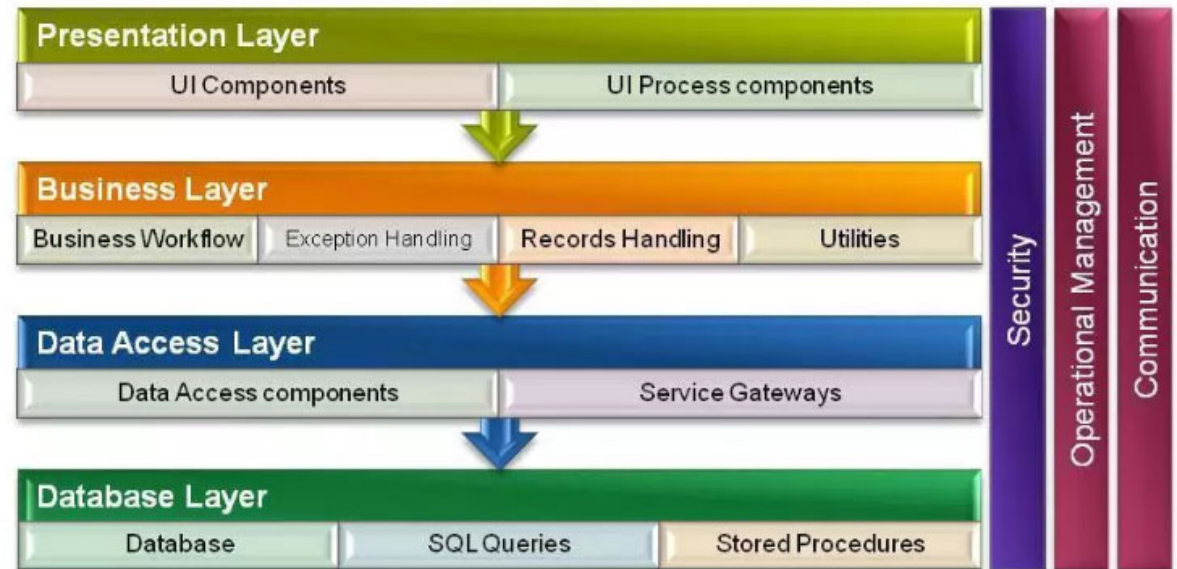
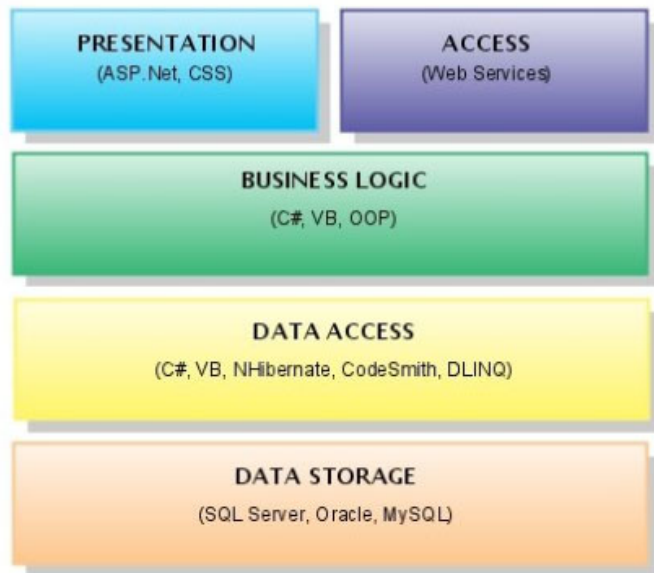
## Comparison of N-Tier Architectures

N-Tier Model	Number of Layers	Components	Use Cases	Security Measures
<b>2-Tier Architecture</b>	2	<ul style="list-style-type: none"> <li>- Client (Presentation)</li> <li>- Database (Data Tier)</li> </ul>	Small applications, standalone desktop apps, lightweight web apps.	Basic Authentication, Database Encryption, Network Firewalls.
<b>3-Tier Architecture</b>	3	<ul style="list-style-type: none"> <li>- Presentation (Client UI)</li> <li>- Application (Business Logic, API)</li> <li>- Data (Database)</li> </ul>	Web applications, SaaS solutions, enterprise applications.	Web Application Firewall (WAF), IAM (OAuth2, MFA), Database Encryption (AES-256).
<b>4-Tier Architecture</b>	4	<ul style="list-style-type: none"> <li>- Presentation (Client UI)</li> <li>- Application (Business Logic, API)</li> <li>- Data (Database)</li> <li>- Security &amp; Compliance Layer (IAM, SIEM, IDS)</li> </ul>	Highly secure enterprise SaaS, fintech, healthcare apps.	SIEM Logging, Threat Intelligence, Privileged Access Management (PAM), IDS/IPS.
<b>5-Tier Architecture</b>	5	<ul style="list-style-type: none"> <li>- Presentation (Client UI)</li> <li>- Application (Business Logic, API)</li> <li>- Data (Database)</li> <li>- Security &amp; Compliance Layer</li> <li>- Infrastructure Layer (Cloud &amp; Network Security)</li> </ul>	Mission-critical SaaS, government, military, large-scale cloud applications.	Cloud Security Posture Management (CSPM), Zero Trust Network Access (ZTNA), Secure Cloud VPN, Network ACLs, Compliance Controls (SOC 2, HIPAA).



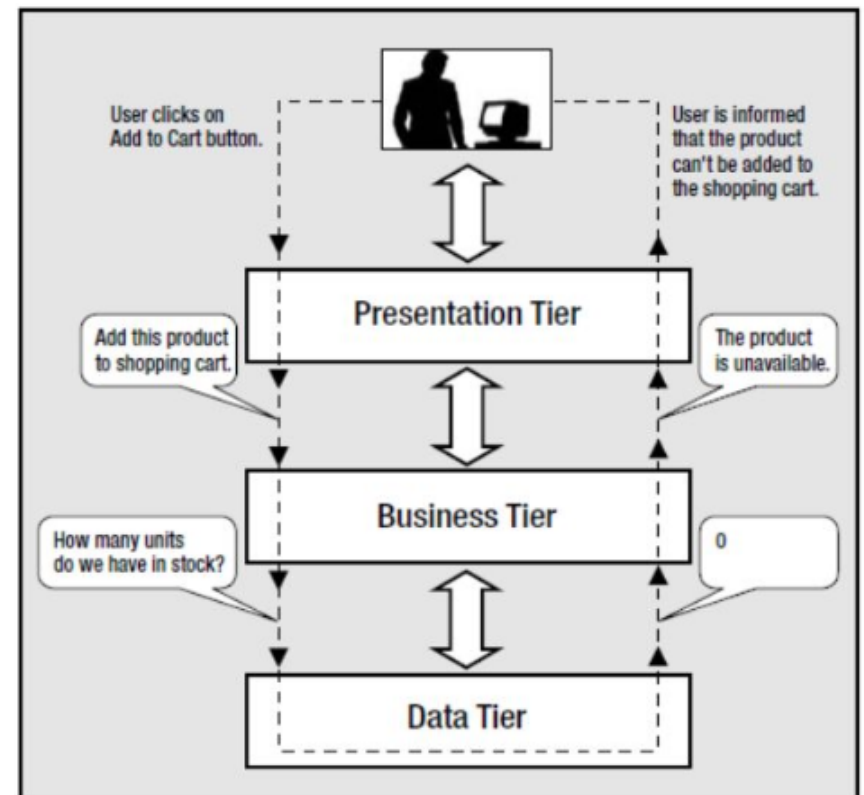
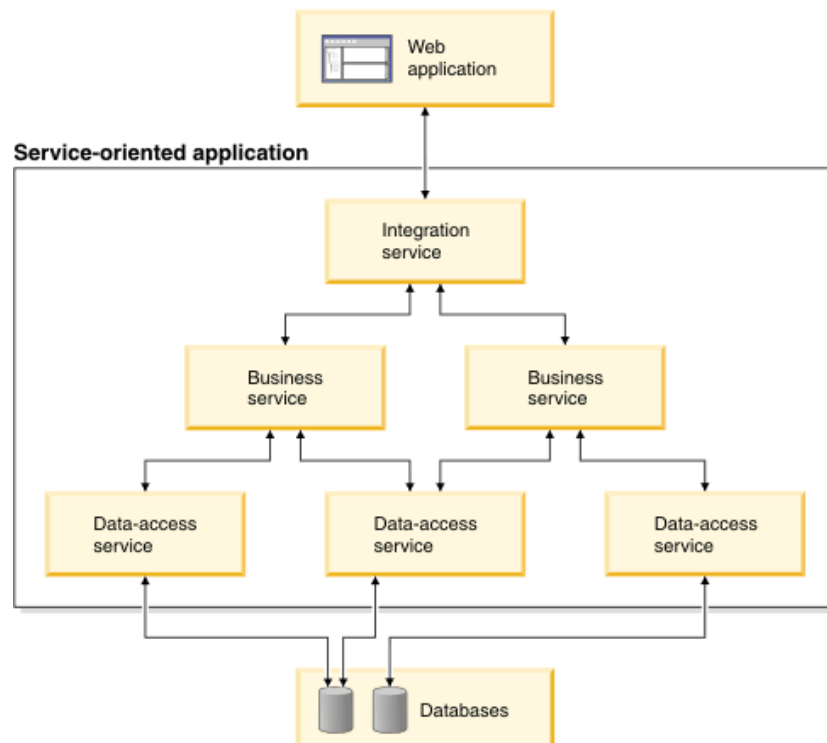
Layer	Definition	Role in System Architecture	Example Technologies
<b>Application Layer (UI/Client Layer)</b>	The user-facing interface where users interact with the system.	Displays content, collects user inputs, and sends requests to the Business Layer.	Web Browsers (Chrome, Edge), Mobile Apps (iOS, Android), Web Apps (React, Angular, Vue.js).
<b>Business Layer (Logic &amp; Processing Layer)</b>	The core processing engine of the application that applies business rules.	Manages authentication, processes user requests, enforces policies, and interacts with the Data Access Layer.	Java, .NET, Python (Django, Flask), Node.js, Spring Boot.
<b>Data Access Layer (Middleware, API, Security Layer)</b>	Acts as an intermediary between the Business Layer and the Database Layer, ensuring secure and optimized data exchange.	Handles API communication, enforces security measures (IAM, RBAC), and optimizes data transactions.	API Gateway (Kong, Apigee, AWS API Gateway), OAuth2, JWT, GraphQL, REST APIs, ORM (Hibernate, SQLAlchemy).
<b>Database Layer (Storage &amp; Data Management Layer)</b>	Stores, retrieves, and manages structured/unstructured data.	Ensures data integrity, security, and availability with backup, encryption, and replication strategies.	SQL Databases (PostgreSQL, MySQL, SQL Server), NoSQL (MongoDB, DynamoDB, Cassandra), Cloud Storage (AWS S3, Azure Blob).

# N-Tier Applications



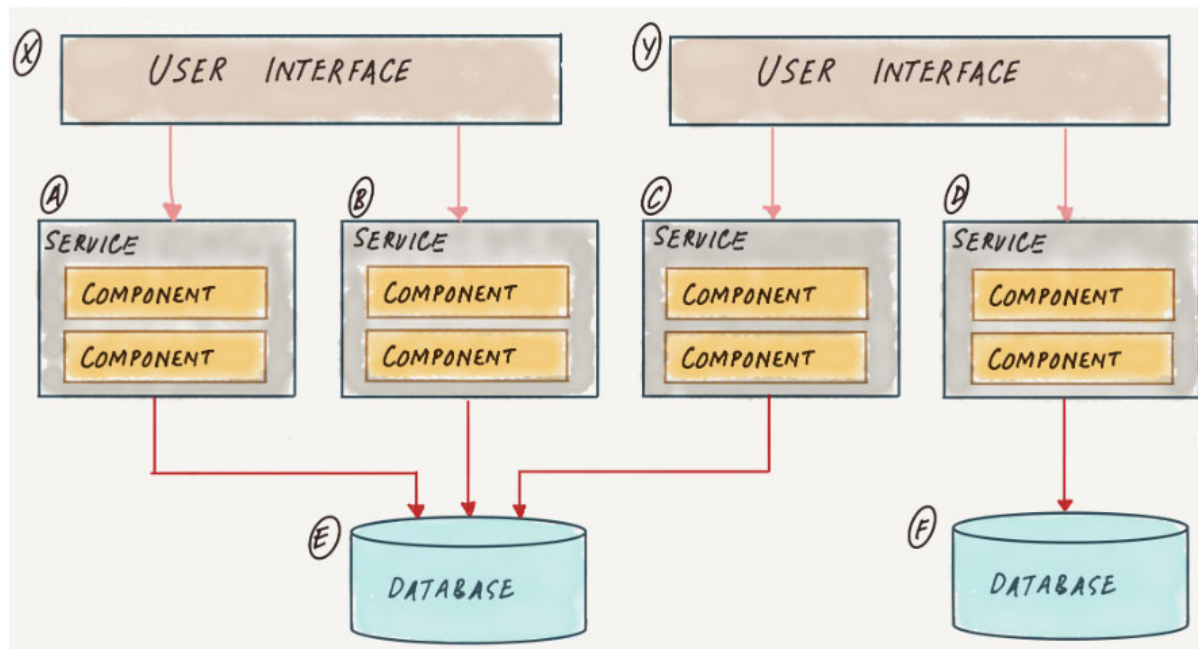
*Where's the programming code?*

# N-Tier Applications



# Service Oriented Architecture (SOA)

**Service-Oriented Architecture (SOA)** is a software design pattern where applications are built using **loosely coupled services** that communicate over a network. Each service is self-contained, providing specific business functionalities, and interacts with other services through standardized protocols such as **SOAP, REST, or GraphQL**

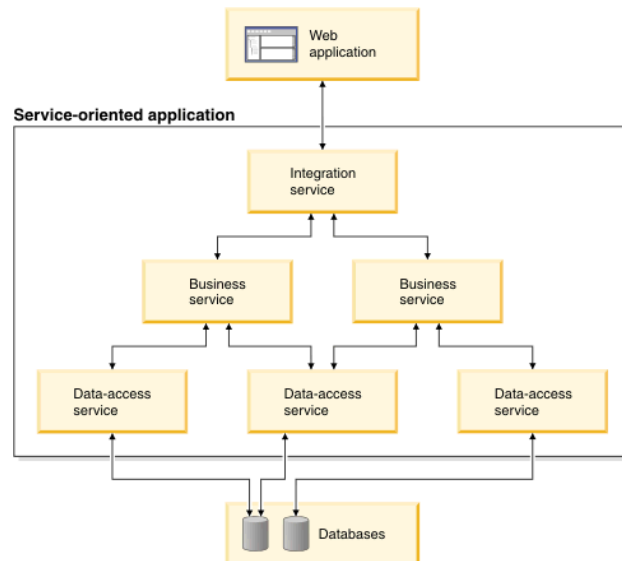


## Principles:

- Reusability
- Interoperability
- Componentization

[Reference: Service Oriented Architecture · Umair's Blog](#)

# N-Tier Applications using SOA in the cloud



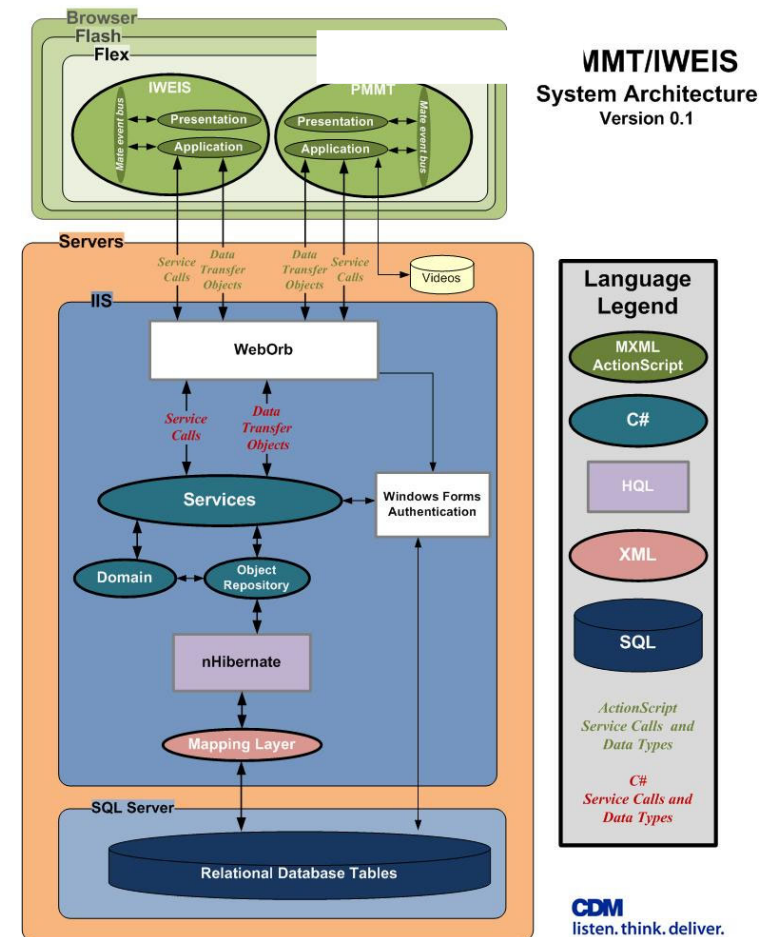
## SOA vs Traditional N-Tier Architecture

Feature	Traditional N-Tier	Service-Oriented Architecture (SOA)
Architecture Type	Layered (Tightly Coupled)	Distributed (Loosely Coupled)
Scalability	Limited to vertical scaling	High, supports horizontal scaling
Reusability	Low (Each application has its own logic)	High (Reusable services across applications)
Integration	Difficult (Requires changes to the entire system)	Easy (Uses APIs and standardized protocols)
Security	Centralized IAM	Granular security per service (OAuth, API Keys, JWT)

## Service Oriented Architecture (Application Architecture)

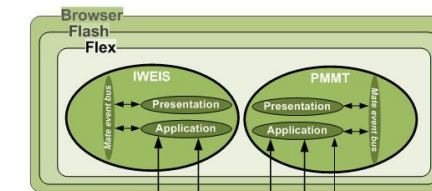
Example: for cloud-based Software as a Service  
(no security architecture in example)

1. User Interacts via the Presentation Layer  
(UI sends events to the Application Layer)
2. Application Layer processes user interactions  
(Calls the Service Layer for business operations & data exchange)
3. Service Layer manages API calls & Data Transfer Objects (DTOs)  
(DTOs send structured data between Client & Server)
4. Domain / Repository Layer handles business logic  
(Manages object creation, retrieval, and updates)
5. Mapping Layer ensures seamless object-relational mapping  
(Maps database tables to objects via nHibernate ORM)
6. Database Layer stores & retrieves structured data  
(Data persistence via SQL Server & queries using HQL/T-SQL)

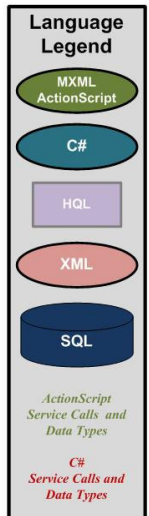
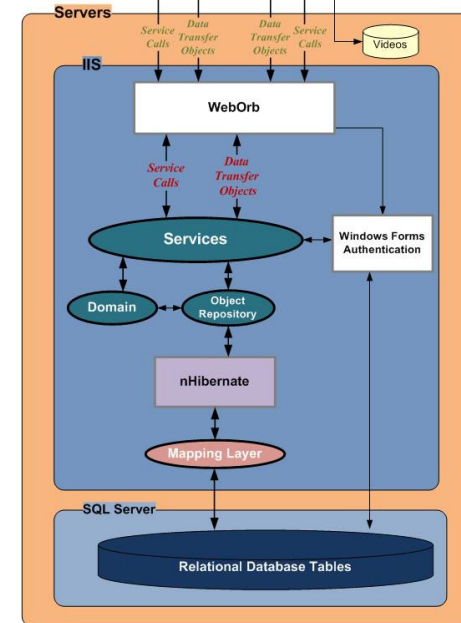


## Categorized SOA Layers with Descriptions

SOA Layer	Role & Description	Components & Functions	Technologies Used
Presentation Layer (UI / Client Layer)	Defines the visual aspects of the UI and how users interact with the system.	<ul style="list-style-type: none"> <li>- Defines layouts, colors, fonts, sizing.</li> <li>- Uses views, components, renderers, and controls.</li> <li>- Organizes interactive events (clicks, inputs).</li> </ul>	HTML, CSS, JavaScript, MXML, ActionScript, Flash Flex.
Application Layer (Client-Side Logic & Event Handling)	Defines the underlying application logic that runs within the browser (client-side).	<ul style="list-style-type: none"> <li>- Contains Client-Side Object Model &amp; Object Managers.</li> <li>- Handles user interactions &amp; UI events.</li> <li>- Makes service calls to exchange data with the server.</li> </ul>	JavaScript, ActionScript, Angular, React, Vue.js.
Service Layer (API & Communication Layer)	Exposes business functionalities through service operations and DTOs (Data Transfer Objects).	<ul style="list-style-type: none"> <li>- Defines service operations.</li> <li>- DTOs package data for communication.</li> <li>- Supports multiple DTO versions (rich vs. lite data transfers).</li> </ul>	Web Services (REST, SOAP), API Gateway, XML, JSON, WebOrb, IIS.
Domain / Repository Layer (Business Logic & Data Access)	Contains logic for creating, retrieving, and updating objects exchanged with the database and client application.	<ul style="list-style-type: none"> <li>- Loosely coupled to client apps via the Service Layer.</li> <li>- Implements repositories for querying objects.</li> <li>- Business rules and logic are enforced here.</li> </ul>	C#, HQL (Hibernate Query Language), Business Objects.
Mapping Layer (ORM & Data Mapping Layer)	Maps domain objects to database tables, ensuring a structured data flow.	<ul style="list-style-type: none"> <li>- Bidirectional mapping between domain objects and database records.</li> <li>- Uses nHibernate ORM for object-relational mapping.</li> <li>- Exposes data via repositories with HQL queries.</li> </ul>	nHibernate, XML, HQL, ORM Mappers.
Database Layer (Storage & Data Management)	Provides permanent storage of data using relational database models.	<ul style="list-style-type: none"> <li>- Implemented using SQL Server.</li> <li>- Stores data in tables, indexes, views, and stored procedures.</li> <li>- Ensures data integrity and backup policies.</li> </ul>	Microsoft SQL Server, T-SQL, Relational Databases.



## PMMT/IWEIS System Architecture Version 0.1

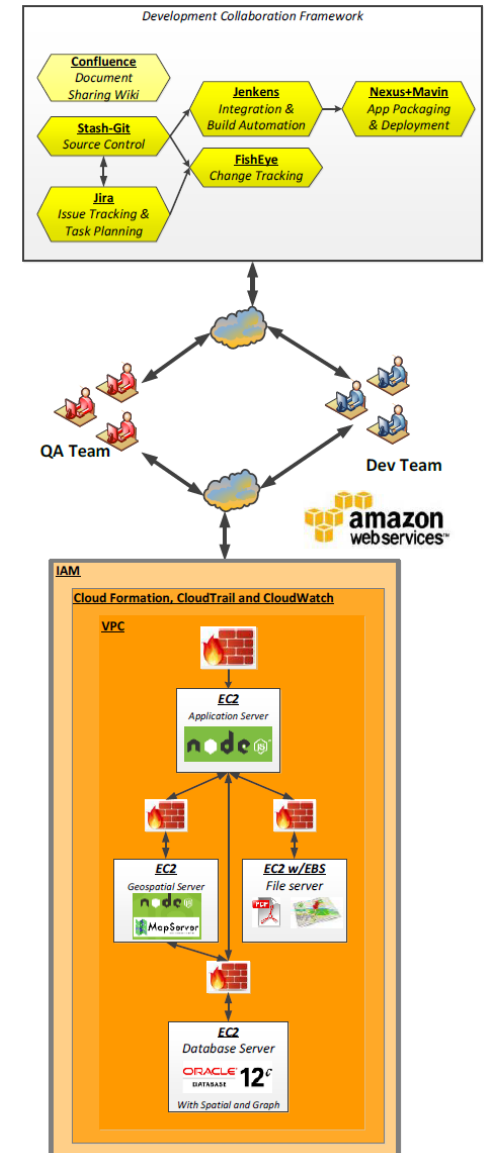




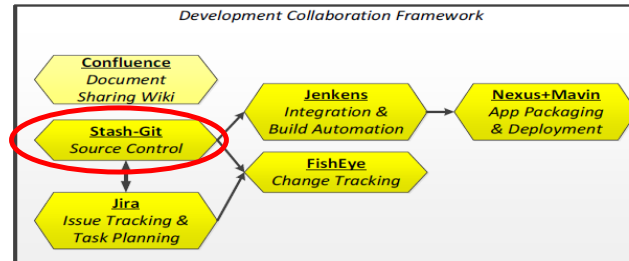
# Development Infrastructure Example...

## SOA Workflow & Data Flow in Development Collaboration Framework

1. Presentation Layer (UI)  
Jira, Confluence, FishEye
  - Users collaborate, create tasks, and track changes.
  - Interacts with Application Layer for development workflows.
2. Application Layer (Development & CI/CD)  
Git (Stash), Jenkins, Nexus+Maven
  - Developers commit code.
  - Jenkins automates build & integration.
  - Nexus+Maven prepares packaged applications.
3. Service Layer (Infrastructure & Monitoring)  
AWS IAM, CloudFormation, CloudTrail, CloudWatch
  - Automates infrastructure deployment.
  - Monitors application servers, database, and cloud services.
4. Domain / Repository Layer (Application Services)  
Node.js (Application & Geospatial Server), MapServer
  - Runs the application logic & APIs.
  - Interacts with Mapping Layer for data retrieval.
5. Mapping Layer (ORM & Data Processing)  
nHibernate ORM, Oracle Spatial & Graph
  - Translates queries into database transactions.
  - Exposes processed data to the Domain Layer.
6. Database Layer (Persistent Storage)  
Oracle Database 12c, AWS EC2 with EBS
  - Stores application data & geospatial records.
  - Provides long-term data retention.







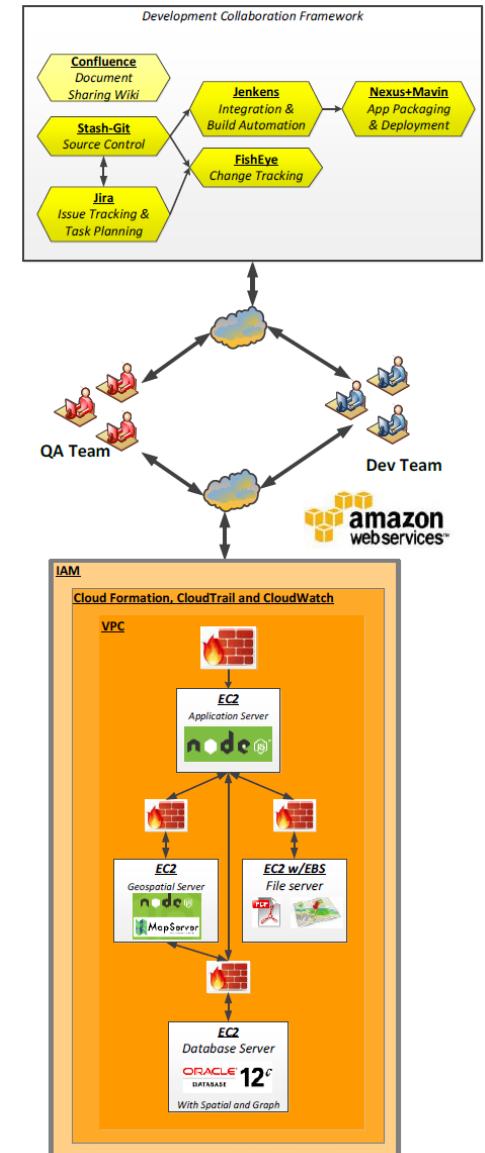
**Table: SOA Layers & Their Role in the Development Collaboration Framework**

SOA Layer	Role & Description	Components / Tools	Relationships
Presentation Layer (UI & Collaboration Layer)	Provides a user-friendly interface for documentation, issue tracking, and team collaboration.	<ul style="list-style-type: none"> <li>- Confluence (Document Sharing Wiki)</li> <li>- Jira (Issue Tracking, Task Planning)</li> <li>- FishEye (Change Tracking)</li> </ul>	- Interacts with the Application Layer for source control and automation.
Application Layer (Development & Build Automation Layer)	Manages code versioning, integration, build automation, and deployment packaging.	<ul style="list-style-type: none"> <li>- Stash-Git (Source Control)</li> <li>- Jenkins (CI/CD &amp; Build Automation)</li> <li>- Nexus+Maven (App Packaging &amp; Deployment)</li> </ul>	<ul style="list-style-type: none"> <li>- Fetches code from Presentation Layer (Developers' IDEs).</li> <li>- Sends build artifacts to Service Layer (Cloud Infrastructure).</li> </ul>
Service Layer (Infrastructure & Cloud Management Layer)	Automates cloud provisioning, monitoring, and resource allocation.	<ul style="list-style-type: none"> <li>- AWS IAM (Identity &amp; Access Management)</li> <li>- CloudFormation (Infrastructure as Code)</li> <li>- CloudTrail, CloudWatch (Logging &amp; Monitoring)</li> </ul>	<ul style="list-style-type: none"> <li>- Deploys applications to the Domain Layer (App &amp; Geospatial Servers).</li> <li>- Monitors infrastructure health.</li> </ul>
Domain / Repository Layer (Application Services & Business Logic Layer)	Implements application services and geospatial logic.	<ul style="list-style-type: none"> <li>- Application Server: Node.js</li> <li>- Geospatial Server: Node.js, MapServer</li> </ul>	<ul style="list-style-type: none"> <li>- Uses Mapping Layer to fetch/store geospatial data.</li> <li>- Exposes APIs to Presentation &amp; Application Layers.</li> </ul>
Mapping Layer (ORM & Data Handling Layer)	Translates data objects into database transactions.	<ul style="list-style-type: none"> <li>- nHibernate ORM (if used)</li> <li>- Oracle Database 12c (Spatial &amp; Graph)</li> </ul>	<ul style="list-style-type: none"> <li>- Retrieves data from Database Layer.</li> <li>- Exposes processed data to Domain Layer.</li> </ul>
Database Layer (Storage & File Management Layer)	Provides data storage for application and geospatial data.	<ul style="list-style-type: none"> <li>- Oracle Database 12c (Spatial &amp; Graph)</li> <li>- AWS EC2 with EBS (File Storage)</li> </ul>	- Serves as the backend storage for Mapping & Domain Layers.

# Continuous Integration & Continuous Deployment

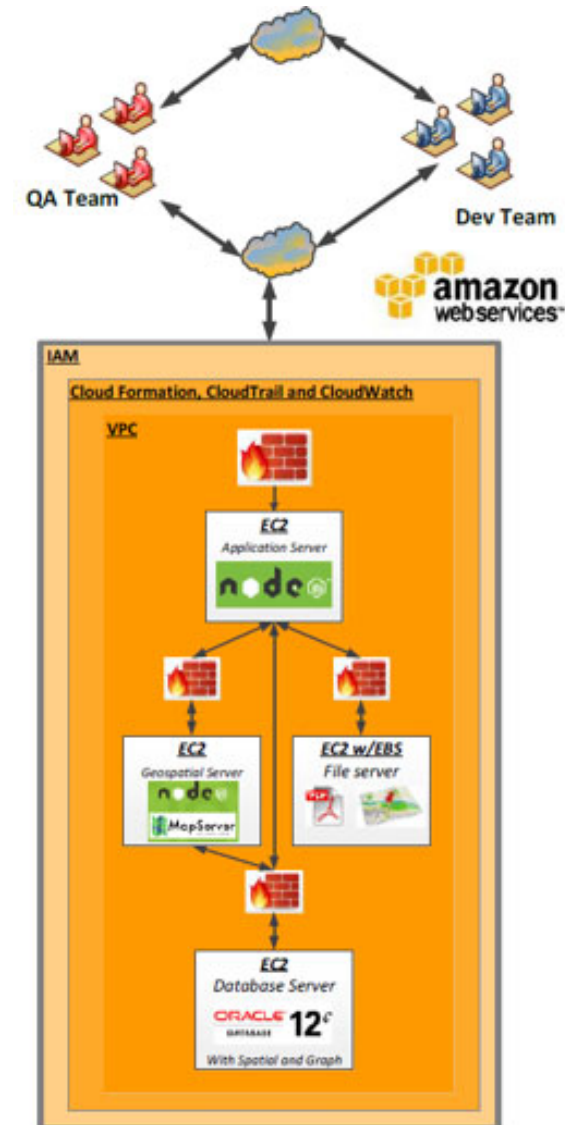
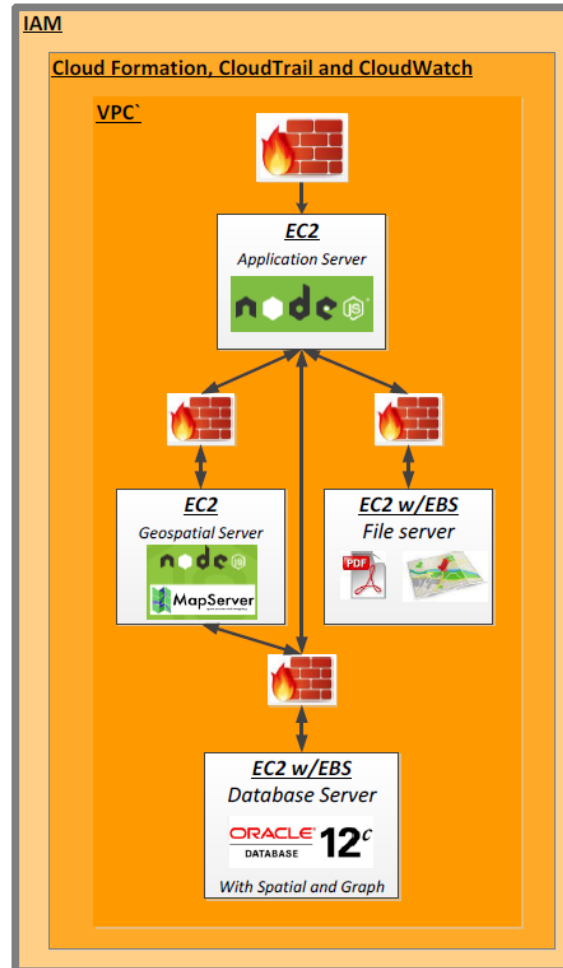
Helps development team make system builds, triggered by either

- A commit of updated source code to the version control system
- Scheduling directive
- A dependency on the completion of another component's build
- Developer kicking off the build using a URL to make the request



# Development Infrastructure Example...

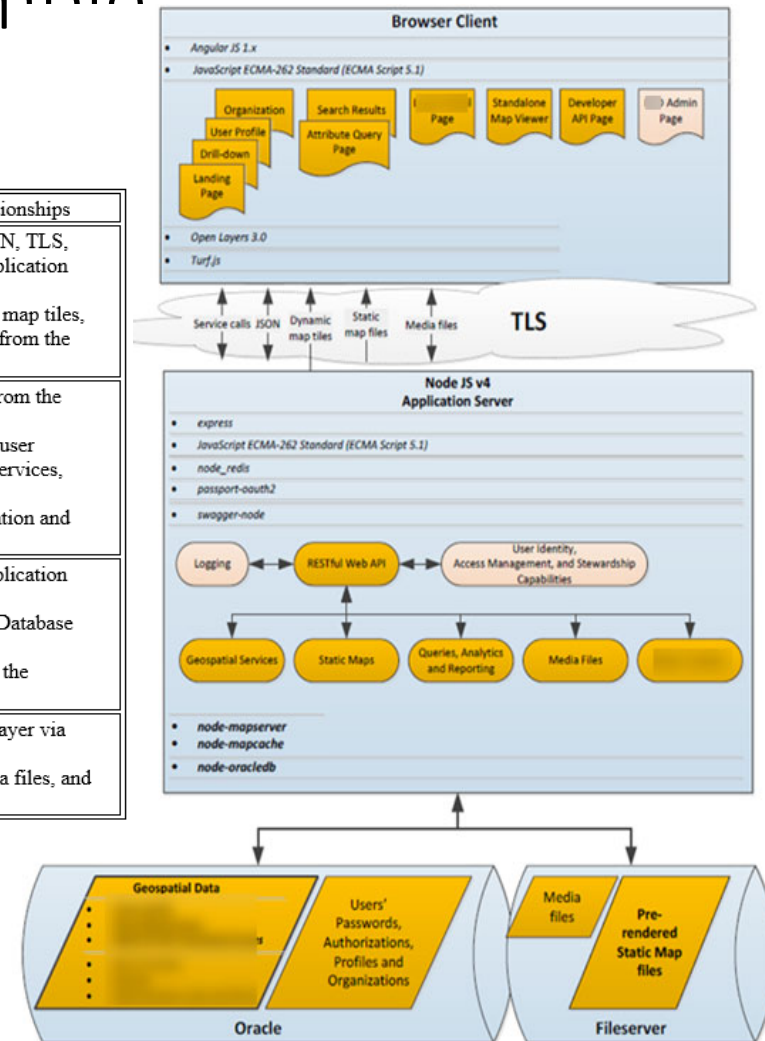
VPC = virtual private cloud



# Application 3+ Tier Architecture example

Table: SOA Layers in the 3-Tier System

SOA Layer	Role & Description	Components & Technologies	Interactions & Relationships
Presentation Layer (Client Layer)	Provides User Interface (UI) and Frontend Interactions.	<ul style="list-style-type: none"> <li>- AngularJS 1.x (UI Framework)</li> <li>- ECMAScript 5.1 (JavaScript Standard)</li> <li>- OpenLayers 3.0 (Map Rendering)</li> <li>- Turf.js (Geospatial Calculations)</li> </ul>	<ul style="list-style-type: none"> <li>- Sends service calls (JSON, TLS, RESTful APIs) to the Application Layer.</li> <li>- Receives dynamic/static map tiles, query results, media files from the Service Layer.</li> </ul>
Application Layer (Business Logic & Middleware Layer)	Processes user requests, authentication, logging, and business rules.	<ul style="list-style-type: none"> <li>- Node.js v4 Application Server</li> <li>- Express (Web Server Framework)</li> <li>- node_redis (Cache Management)</li> <li>- passport-oauth2 (Authentication)</li> <li>- swagger-node (API Documentation)</li> </ul>	<ul style="list-style-type: none"> <li>- Receives data requests from the Presentation Layer.</li> <li>- Calls RESTful APIs for user management, geospatial services, media retrieval.</li> <li>- Manages user authentication and authorization.</li> </ul>
Service Layer (API & Backend Services Layer)	Provides geospatial services, identity management, analytics, and reporting.	<ul style="list-style-type: none"> <li>- RESTful Web API (For communication)</li> <li>- User Identity &amp; Access Management</li> <li>- Geospatial Services (node-mapserver, node-mapcache)</li> <li>- Queries, Analytics, Reporting</li> </ul>	<ul style="list-style-type: none"> <li>- Communicates with Application Layer via APIs.</li> <li>- Retrieves data from the Database Layer.</li> <li>- Sends responses back to the Presentation Layer.</li> </ul>
Data Layer (Database & File Storage Layer)	Stores structured and unstructured data.	<ul style="list-style-type: none"> <li>- Oracle Database (Geospatial Data, User Profiles, Organizations, Passwords)</li> <li>- File Server (Static Media, Map Files, Pre-rendered Maps)</li> </ul>	<ul style="list-style-type: none"> <li>- Serves data to Service Layer via node-oracledb APIs.</li> <li>- Stores user details, media files, and geospatial datasets.</li> </ul>



# Agenda

- ✓ Team Project Guidance
- ✓ Distributed Systems
  - ✓ File Server Architecture
  - ✓ Client/Server Architecture
  - ✓ N-Tier Architecture
  - ✓ Cloud Architecture
  - ✓ Service Oriented Architecture (SOA)
- ✓ Example Cloud-based N-Tier SOA Application Development System
  - Control Stages, Objectives, Application Security Testing
  - Additional Best Practices
  - Team Project Guidance

# Information System Development Control Stages

When designing software, control objectives ensure that the system functions as intended while maintaining security, accuracy, reliability, and compliance. These objectives cover three primary phases: **input, processing, and output.**

Control over applications is conducted at every stage and begins at the start of the development of the information system

This takes 2 basic forms:

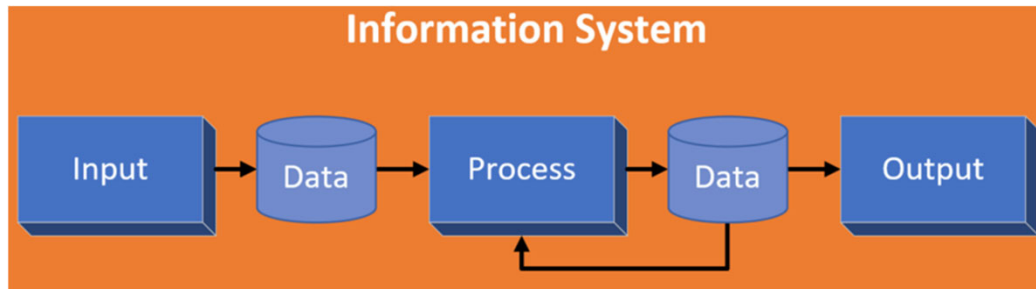
1. Control over the development process itself
2. Ensuring adequate business controls are built into the finished product

Major control stages would include:

- System design
- System development
- System operation
- System utilization

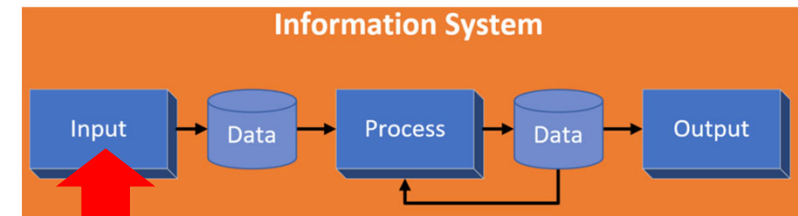


# Control Objectives for Business Information Systems



1. Input control objectives
2. Processing control objectives
3. Output control objectives

# Control Objectives for Business Information Systems



## Input Control Objectives

Input controls ensure that data entering the system is accurate, complete, and valid. Poor input validation can lead to security vulnerabilities such as SQL [injection](#), buffer overflows, and incorrect data processing.

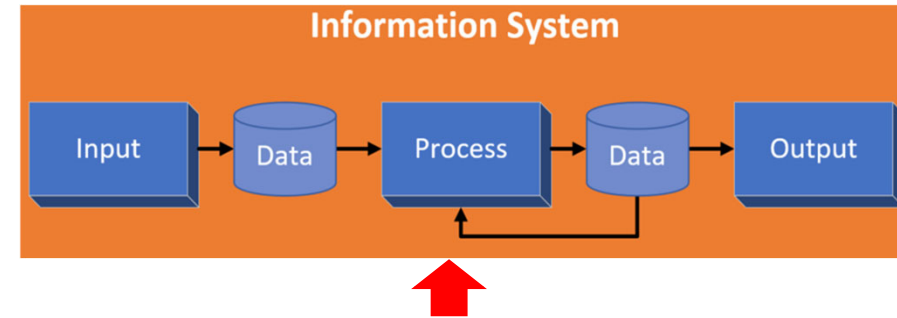
- **Validation:** Ensure that input meets predefined criteria (e.g., format, length, range).
- **Authorization:** Verify that only authorized users can enter or modify data.
- **Sanitization:** Prevent malicious input such as SQL injection, XSS (Cross-Site Scripting), and code injection.
- **Error Handling:** Provide clear error messages without exposing system details.
- **Data Integrity:** Ensure data is not altered during input.
- **Logging & Audit Trails:** Track input attempts for forensic analysis.

## Example:

- A web form validates an email field to ensure proper format before submission.
- Input sanitization prevents users from injecting malicious scripts into text fields.



# Control Objectives for Business Information Systems



## Processing Control Objectives

Processing controls ensure that the system handles input correctly, maintains integrity, and performs operations securely.

- **Completeness:** Ensure all required operations are executed (no loss of data).
- **Accuracy:** Maintain precision in computations and data transformations.
- **Concurrency & Consistency:** Prevent race conditions, data corruption, and deadlocks.
- **Security & Confidentiality:** Protect data from unauthorized modifications during processing.
- **Transaction Control:** Implement atomicity, consistency, isolation, and durability (ACID) principles.
- **Exception Handling:** Handle unexpected scenarios without system crashes.
- **Performance Monitoring:** Ensure efficient resource utilization.

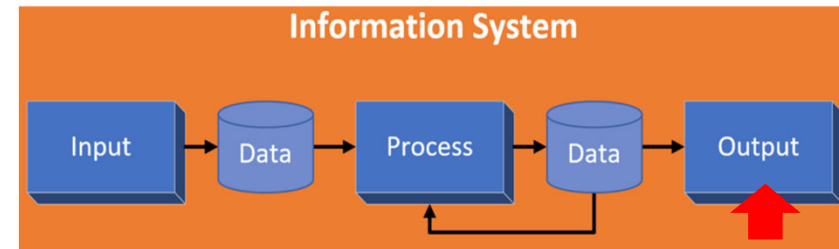
## Example:

- A banking application correctly calculates interest rates without rounding errors.
- A payroll system ensures that all employees receive the correct salary after applying deductions and bonuses.

## Controls over processing may include:

- Control totals
- Programmed balancing
- Reasonableness tests
- Segregation of duties
- Restricted access
- File labels
- Exception reports
- Error logs
- Concurrent update control

# Control Objectives for Business Information Systems



## Output Control Objectives

Output controls ensure that the final product (reports, files, user interfaces) is accurate, complete, and secure.

- **Accuracy:** Ensure output data matches the expected results.
- **Confidentiality:** Protect sensitive information (e.g., encryption of reports, access control).
- **Access Control:** Restrict output visibility based on user roles.
- **Format Consistency:** Maintain readability and compliance with standards.
- **Auditability:** Log all output transactions for review and verification.
- **Delivery Assurance:** Ensure output reaches the intended recipient securely.

### Example:

- A financial report is generated with properly formatted data and sent securely via encrypted email.
- A receipt is only shown to an authenticated user after an online transaction.

- Controls over output may include:
  - Assurance that the results of input and processing are output
  - Output is available to only authorized personnel
  - Complete audit trail
  - Output distribution logs

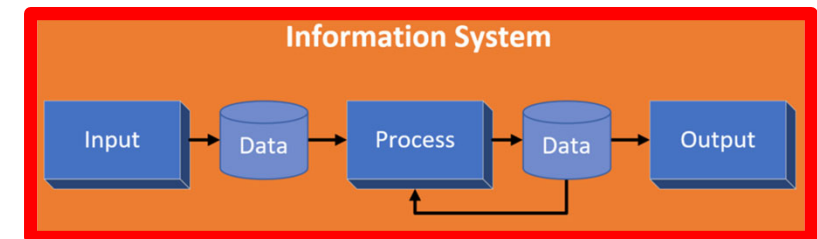
# Control Objectives for Business Information Systems

## Computer program control objectives focus on

- Integrity of programs and processing
- Prevention of unwanted changes

Typical computer program controls include:

- Ensuring adequate design and development
- Ensuring adequate testing
- Controlled transfer of programs (among machines, from version control, ...)
- Ongoing maintainability of systems
- Use of formal SDLC
- User involvement
- Adequate documentation
- Formalized testing plan
- Planned conversion
- Use of post-implementation reviews (see CISA chapter)
- Establishment of a quality assurance (QA) function
- Involvement of internal auditors



***Testing of these controls require IT auditors to seek evidence regarding their adequacy and effectiveness....***

# PPTM - People, Processes, Tools, and Measures

*A brainstorming framework for examining security of an application from the macro-level, based on*

**People** – describes every aspect of the application that deals with a human

- Make sure the right people are involved in planning, design, implementation or operations, and the right stakeholders are involved
- E.g. If the application involves end users, ensure:
  - The application has controls around providing and removing access
  - End users have been involved with the planning and design of components they will (to ensure usability)

**Process** – Describes every aspect of the application that is involved in a policy, procedure, method, or course of action

- Review the interaction of the application with interfacing systems and verify compliance with security models
  - E.g. Ensure that firewalls are in place to protect the application from external applications, users, business partners, ...
  - Policies and procedures should be written to support how the application is intended to be used
  - Adequate documentation should exist to support technicians who need to maintain the application

**Tools** – Describe every aspect of the application that deals with concrete technology or product

- Ensure appropriate hardware and environment exist to support the application
- Ensure the application interfaces with recommended technologies appropriate for your intended policies and procedures
- Verify that the application and infrastructure are tested and audited appropriately

**Measures** – Describe every aspect of the application that is quantifiable conceptually, such as the business purpose or application performance

- E.g. verify that the application meets well-documented and well-thought out acceptance criteria
- E.g. if the application is intended to solve a quantifiable business problem verify that it does indeed solve the problem
- Verify that the logs are meaningful and that you can measure the performance of the application

# STRIDE

A “simplified threat-risk model” which is easy to remember

## Spoofing Identity

- Is a key risk for applications with many users and a single execution context at the application and database tiers
- Users should not be able to become any other user or assume the attributes of another user

## Tampering with Data

- Data should be stored in a secure location, with access appropriately controlled
- The application should carefully check data received from the user and validate that it is “sane” (i.e. relevant and valid) and applicable before storing or using it
- Data entered in the client (e.g. browser) should be checked and validated on the server and not in the client where the validation checks might be tampered with
- Application should not send and calculate data in the client where the user can manipulate the data, but in the server-side code

## Repudiation

- Determine if the application requires nonrepudiation controls, such as web access logs, audit trails at each tier, or the same user context from top to bottom
- Users may dispute transactions if there is insufficient auditing or record-keeping of their activity

## Denial of Service

- Application designers should be aware that their applications are at risk of denial of service attacks
- Use of expensive resources (e.g. large files, heavy-duty searches, long queries) should be reserved for authenticated and authorized users and should not be available to anonymous users.
- Every facet of the application should be engineered to perform as little work as possible, to use fast and few database queries, and to avoid exposing large files or unique links per user to prevent simple denial-of-service attacks

## Elevation of Privilege

- If an application provides distinct user and administrative roles, ensure that the user cannot elevate his or her role to a more highly privileged one
- All actions should be controlled through an authorization matrix to ensure that only the permitted roles can access privileged functionality. It is not sufficient, for example, to not display privileged-role links

Threat	Desired property
Spoofing	Authenticity
Tampering	Integrity
Repudiation	Non-repudiability
Information disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

# OWASP (Open Web Application Security Project) Frameworks

## • Vulnerabilities

- ▶ API Abuse
- ▶ Authentication Vulnerability
- ▶ Authorization Vulnerability
- ▶ Availability Vulnerability
- ▶ Code Permission Vulnerability
- ▶ Code Quality Vulnerability
- ▶ Configuration Vulnerability
- ▶ Cryptographic Vulnerability
- ▶ Encoding Vulnerability
- ▶ Environmental Vulnerability
- ▶ Error Handling Vulnerability
- ▶ General Logic Error Vulnerability
- ▶ Input Validation Vulnerability
- ▶ Logging and Auditing Vulnerability
- ▶ Password Management Vulnerability
- ▶ Path Vulnerability
- ▶ Sensitive Data Protection Vulnerability
- ▶ Session Management Vulnerability
- ▶ Unsafe Mobile Code
- ▶ Use of Dangerous API

## • Principles

- Apply [defense in depth](#) (complete mediation)
- Use a [positive security model](#) (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with [least privilege](#)
- [Avoid security by obscurity](#) (open design)
- [Keep security simple](#) (verifiable, economy of mechanism)
- [Detect intrusions](#) (compromise recording)
- [Don't trust infrastructure](#)
- [Don't trust services](#)
- [Establish secure defaults](#) (psychological acceptability)

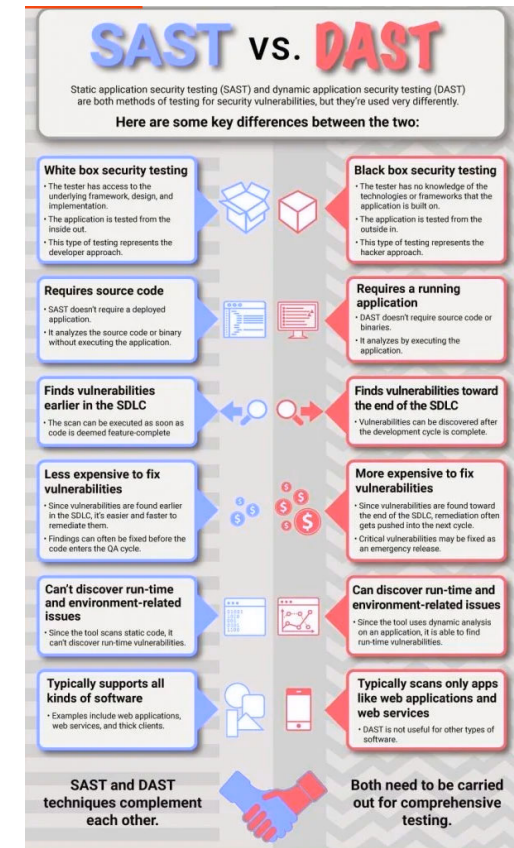
## • Top 10 Web Application Security Risks

<b>A1:2017</b> - Injection .....	<a href="#">7</a>
<b>A2:2017</b> - Broken Authentication .....	<a href="#">8</a>
<b>A3:2017</b> - Sensitive Data Exposure .....	<a href="#">9</a>
<b>A4:2017</b> - XML External Entities (XXE) .....	<a href="#">10</a>
<b>A5:2017</b> - Broken Access Control .....	<a href="#">11</a>
<b>A6:2017</b> - Security Misconfiguration .....	<a href="#">12</a>
<b>A7:2017</b> - Cross-Site Scripting (XSS) .....	<a href="#">13</a>
<b>A8:2017</b> - Insecure Deserialization .....	<a href="#">14</a>
<b>A9:2017</b> - Using Components with Known Vulnerabilities .....	<a href="#">15</a>
<b>A10:2017</b> - Insufficient Logging & Monitoring.....	<a href="#">16</a>

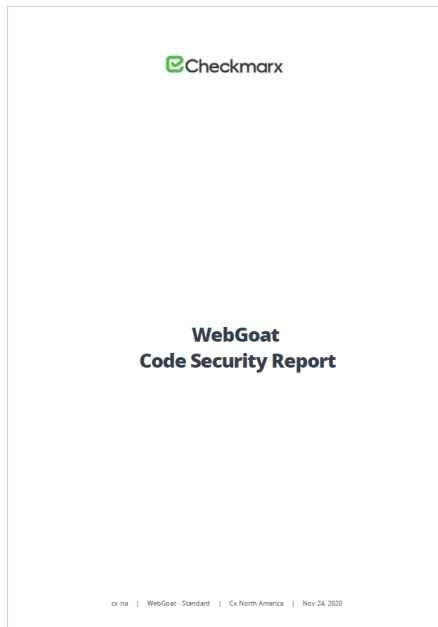
# Application Security Testing

Software testing is the process of evaluating a software application to ensure it meets its intended requirements and functions correctly, efficiently, and securely.

Testing Method	Definition	How It Works	Key Benefits	Example Tools
<b>Static Application Security Testing (SAST)</b>	<u>White-box</u> testing method that examines <b>source code, byte code, or compiled binaries</b> before execution.	Scans code for vulnerabilities such as SQL injection, XSS, and hardcoded credentials without running the application.	<ul style="list-style-type: none"> <li>- Detects vulnerabilities early in SDLC</li> <li>- Helps developers fix issues pre-deployment</li> <li>- No need for a running application</li> </ul>	SonarQube, <del>Checkmarx</del> , Fortify, Veracode
<b>Dynamic Application Security Testing (DAST)</b>	<u>Black-box</u> testing <u>method</u> that examines a <b>running application</b> from an external perspective.	Simulates real-world attacks by sending malicious inputs and analyzing responses to identify security weaknesses.	<ul style="list-style-type: none"> <li>- Finds runtime vulnerabilities</li> <li>- Does not require source code access</li> <li>- Detects authentication flaws, injections, etc.</li> </ul>	Burp Suite, OWASP ZAP, <del>Acunetix</del> , AppSpider
<b>Interactive Application Security Testing (IAST)</b>	Hybrid approach combining <b>SAST and DAST</b> techniques to monitor applications in real time.	Instruments the application and observes its runtime behavior to detect security issues dynamically.	<ul style="list-style-type: none"> <li>- Provides real-time feedback on vulnerabilities</li> <li>- More accurate than DAST due to contextual awareness</li> <li>- Integrates into CI/CD pipelines</li> </ul>	Contrast Security, HCL AppScan, Seeker
<b>Software Composition Analysis (SCA)</b>	Analyzes <b>third-party and open-source components</b> for security risks and licensing issues.	Scans dependencies to identify vulnerabilities, outdated libraries, and license compliance violations.	<ul style="list-style-type: none"> <li>- Detects security risks in software supply chains</li> <li>- Ensures compliance with open-source licenses</li> <li>- Finds outdated dependencies</li> </ul>	Black Duck, <del>Snyk</del> , WhiteSource, OWASP Dependency-Check




# Automated application security testing tools provide vulnerability reports






# SAST Compliance Report Examples

**OWASP Top 10 2013**

**Vulnerabilities: 152**

**Top 10 vulnerability types:**


- 1 Reflected\_XSS\_All\_Clients (39)
- 2 SQL\_Injection (23)
- 3 Client\_DOM\_Open\_Redirect (16)
- 4 Stored\_XSS (10)
- 5 XSRF (9)
- 6 Use\_of\_Cryptographically\_Weak\_PRNG (8)
- 7 Heap\_Inspection (8)
- 8 Use\_of\_Hard\_coded\_Cryptographic\_Key (8)
- 9 Client\_JQuery\_Deprecated\_Symbols (7)
- 10 Use\_Of\_Hardcoded\_Password (7)

**OWASP Top 10 2017**

**Vulnerabilities: 154**

**Top 10 vulnerability types:**


- 1 Reflected\_XSS\_All\_Clients (39)
- 2 SQL\_Injection (23)
- 3 Use\_Of\_Hardcoded\_Password (13)
- 4 Stored\_XSS (10)
- 5 Use\_of\_Hard\_coded\_Cryptographic\_Key (8)
- 6 Use\_of\_Cryptographically\_Weak\_PRNG (8)
- 7 Heap\_Inspection (8)
- 8 Use\_Of\_Hardcoded\_Password (7)
- 9 Log\_Forging (7)
- 10 Client\_JQuery\_Deprecated\_Symbols (7)

**PCI DSS v3.2**

**Vulnerabilities: 126**

**Top 10 vulnerability types:**

- 1 Reflected\_XSS\_All\_Clients (39)
- 2 SQL\_Injection (23)
- 3 Stored\_XSS (10)
- 4 XSRF (9)
- 5 Use\_of\_Hard\_coded\_Cryptographic\_Key (8)
- 6 Use\_of\_Cryptographically\_Weak\_PRNG (8)
- 7 Log\_Forging (7)
- 8 Use\_Of\_Hardcoded\_Password (7)
- 9 Client\_Potential\_XSS (3)
- 10 HttpOnlyCookies (3)

**OWASP Mobile Top 10 2016**

**Vulnerabilities: 66**

**Top 10 vulnerability types:**

- 1 SQL\_Injection (23)
- 2 Side\_Channel\_Data\_Leakage (17)
- 3 Use\_of\_Hard\_coded\_Cryptographic\_Key (8)
- 4 Log\_Forging (7)
- 5 Use\_Of\_Hardcoded\_Password (7)
- 6 Inadequate\_Encryption\_Strength (2)
- 7 Deserialization\_of\_Untrusted\_Data (2)

# SAST Compliance Report Examples

## FISMA 2014



Vulnerabilities: 161

### Top 10 vulnerability types:

- 1 Reflected\_XSS\_All\_Clients (39)
- 2 SQL\_Injection (23)
- 3 Client\_DOM\_Open\_Redirect (16)
- 4 Use\_Of\_Hardcoded\_Password (13)
- 5 Stored\_XSS (10)
- 6 Use\_of\_Cryptographically\_Weak\_PRNG (8)
- 7 Use\_of\_Hard\_coded\_Cryptographic\_Key (8)
- 8 Heap\_Inspection (8)
- 9 Log\_Forging (7)
- 10 Use\_Of\_Hardcoded\_Password (7)

## NIST SP 800-53




Vulnerabilities: 172

### Top 10 vulnerability types:

- 1 Reflected\_XSS\_All\_Clients (39)
- 2 SQL\_Injection (23)
- 3 Client\_DOM\_Open\_Redirect (16)
- 4 Use\_Of\_Hardcoded\_Password (13)
- 5 Stored\_XSS (10)
- 6 XSRF (9)
- 7 Use\_of\_Cryptographically\_Weak\_PRNG (8)
- 8 Use\_of\_Hard\_coded\_Cryptographic\_Key (8)
- 9 Heap\_Inspection (8)
- 10 Use\_Of\_Hardcoded\_Password (7)

# DAST Report



June 17, 2020  
<http://demo.testfire.net>

Security Analysis - June 17, 2020

SCAN SUMMARY

This site was checked for 65 classes of vulnerabilities, with up to hundreds of tests for each vulnerability class. This site is considered to be **Very Unsafe** as of June 17, 2020.

VULNERABILITY CLASSES

The following types of vulnerabilities were looked for over the 27 URLs found during this security scan.

Allowed HTTP methods

Blind SQL Injection (timing attack)

Clickjacking

Credit card number disclosure

Cross-Site Scripting in attribute of HTML element

Cross-Site Scripting in HTML "script" tag

Cross-Site Scripting in HTML "vbscript" tag

Cross-Site Scripting (XSS) in path

Directory listing is enabled

Disclosed US Social Security Number

File Inclusion

Found an HTML object

Found Stacktrace

HTTP PUT is enabled

LDAP Injection

Missing Subresource Integrity Protection

Non HTTP-Only Cookies

Operating system command injection

Password field with autocomplete

Path Traversal

Persistent Cross-Site Scripting (XSS)

Remote file inclusion

Scriptless Cross-Site Scripting in attribute of HTML element

Session Cookie Expiration

Session ID Entropy

Spamnable contact form

SSLv3 Enabled

The TRACE HTTP method is enabled

TLS Vulnerable to POODLE

Unencrypted password form

WebDAV

XPath Injection

YAML Injection (timing)

ASP.NET DEBUG Method Enabled

Buffer Overflow

Code Injection

Cross-Site Request Forgery

Cross-Site Scripting in event attribute of HTML element

Cross-Site Scripting in HTML tag

Cross-Site Scripting (XSS)

CVS/SVN user disclosure

Disclosed e-mail address

DOM Based Cross-Site Scripting

Found a CAPTCHA protected form

Found Robots.txt

FrontPage Extensions Enabled

Insecure Cookies

Misconfiguration in LIMIT directive of .htaccess file

Mixed Resource

OpenSSL Heartbeat Extension Memory Leak (Heartbleed)

Outdated TLS Supported

Password Submission via GET

Permissive CORS Policy

Private IP address disclosure

Response splitting

Server-Side Include Injection

Session Fixation

Shellshock

SQL Injection

Struts2 (CVE-2017-5638)

TLS fallback is not supported

Unencrypted HTTP Basic Authentication


Unvalidated redirect

XML External Entity Injection

YAML Injection

SCAN OVERVIEW

STATUS ON 06/17/2020



Your website is  
**Very Unsafe**

NUMBER OF VULNERABILITIES

33

Total Vulnerabilities

WHAT'S THE WORST THAT COULD HAPPEN?

With your current vulnerabilities a hacker could potentially infiltrate your website, steal your user's cookies, log the keys they type, and pretend to be them on your website. And that's just the tip of the iceberg. Data breaches like this, once disclosed, can often lead to a 20% loss in your customer base. We highly recommend you fix these vulnerabilities quickly and with much vengeance.

LOGIN STATUS

Login Successful: Yes

SITEMAP

<http://demo.testfire.net/>

<http://demo.testfire.net/admin/admin.jsp>

<http://demo.testfire.net/bank/apply.jsp>

<http://demo.testfire.net/bank/ccApply>

<http://demo.testfire.net/bank/customize.jsp>

<http://demo.testfire.net/bank/doTransfer>

<http://demo.testfire.net/bank/main.jsp>

<http://demo.testfire.net/bank/queryxpath.jsp>

<http://demo.testfire.net/bank/showAccount>

<http://demo.testfire.net/bank/showTransactions>

<http://demo.testfire.net/bank/transaction.jsp>

<http://demo.testfire.net/bank/transfer.jsp>

<http://demo.testfire.net/default.jsp>

<http://demo.testfire.net/discclaimer.htm>

<http://demo.testfire.net/doSubscribe>

<http://demo.testfire.net/feedback.jsp>

<http://demo.testfire.net/index.jsp>

<http://demo.testfire.net/search.jsp>

<http://demo.testfire.net/sendFeedback>

[http://demo.testfire.net/status\\_check.jsp](http://demo.testfire.net/status_check.jsp)

## VULNERABILITY: CROSS-SITE REQUEST FORGERY

### DETAILS

Severity	High
URL	<a href="http://demo.testfire.net/admin/admin.jsp">http://demo.testfire.net/admin/admin.jsp</a>
Variable Element	addAccount form

### INJECTION

Matched by Regular Expression	<pre>&lt;form id="addAccount" name="addAccount" action="" method="post"&gt; &lt;tr&gt; &lt;td colspan="4"&gt; &lt;h2&gt;Add an account to an existing user&lt;/h2&gt; &lt;/td&gt; &lt;/tr&gt; &lt;tr&gt; &lt;th&gt; Users: &lt;/th&gt; &lt;th&gt; Account Types: &lt;/th&gt; &lt;th&gt; &lt;/th&gt; &lt;th&gt; &lt;/th&gt; &lt;/tr&gt; &lt;tr&gt; &lt;td&gt; &lt;select name="username" id="username" size="1"&gt; &lt;option value="admin"&gt;admin&lt;/option&gt; &lt;option value="jdoe"&gt;jdoe&lt;/option&gt; &lt;option value="jsmith"&gt;jsmith&lt;/option&gt; &lt;option value="sspeed"&gt;sspeed&lt;/option&gt; &lt;option value="tuser"&gt;tuser&lt;/option&gt; &lt;/select&gt; &lt;/td&gt; &lt;td&gt; &lt;select name="accttypes"&gt; &lt;option value="Checking"&gt;Checking&lt;/option&gt; &lt;option value="Savings"&gt;Savings&lt;/option&gt; &lt;option value="IRA"&gt;IRA&lt;/option&gt; &lt;/select&gt; &lt;/td&gt; &lt;/tr&gt; &lt;tr&gt; &lt;td&gt; &lt;input type="submit" value="Add Account"&gt;&lt;/td&gt; &lt;/tr&gt; &lt;/form&gt;</pre>
-------------------------------	--

### DESCRIPTION

Cross-Site Request Forgery (CSRF) allows an attacker to execute actions on behalf of an unwitting user who is already authenticated with your web application. If successful, user data and user actions can be compromised. If the user who is attacked with CSRF happens to be an administrator, the entire web application should be considered compromised. CSRF occurs when a user submits data to a form or input he/she did not intend; usually an attacker will accomplish this by sending them a link or convincing them to input to a different form that looks similar and posts to the same place.

### HOW TO FIX

A unique token that guarantees freshness of submitted data must be added to all web application elements that can affect business logic.

### REFERENCES

Wikipedia - [http://en.wikipedia.org/wiki/Cross-site\\_request\\_forgery](http://en.wikipedia.org/wiki/Cross-site_request_forgery)  
CGI Security - <http://www.cgisecurity.com/csrf-faq.html>  
OWASP - [https://wiki.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)](https://wiki.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))

# Application Security Assessment and Recommendations

## Issue Types 21

TOC

Issue Type	Number of Issues
H Authentication Bypass Using HTTP Verb Tampering	3
H Cross-Site Request Forgery	23
H Cross-Site Scripting	2
H Microsoft FrontPage Extensions Site Defacement	3
H Missing Secure Attribute in Encrypted Session (SSL) Cookie	5
H RC4 cipher suites were detected	1
M Alternate Version of File Detected	45
M Body Parameters Accepted in Query	9
M Browser Exploit Against SSL/TLS (a.k.a. BEAST)	1
M Cacheable SSL Page Found	67
M Direct Access to Administration Pages	1
M Drupal "keys" Path Disclosure	1
M Insecure "OPTIONS" HTTP Method Enabled	1
M Microsoft FrontPage Server Extensions Vital Information Leakage	2
M Microsoft IIS Missing Host Header Information Leakage	1
M Missing "Content-Security-Policy" header	5
M Missing Cross-Frame Scripting Defence	4
M Query Parameter in SSL Request	185
M Temporary File Download	3
M Unencrypted __VIEWSTATE Parameter	20
M Web Application Source Code Disclosure Pattern Found	1

## Fix Recommendations 19

TOC

Remediation Task	Number of Issues
H Review possible solutions for hazardous character injection	2
M Add the 'Secure' attribute to all sensitive cookies	5
M Change server's supported ciphersuites	2
M Configure your server to allow only required HTTP methods	3
M Set proper permissions to the FrontPage extension files	3
M Validate the value of the "Referer" header, and use a one-time-nonce for each submitted form	23
L Always use SSL and POST (body) parameters when sending sensitive information.	185
L Apply configuration changes according to Q218180	1
L Apply proper authorization to administration scripts	1
L Config your server to use the "Content-Security-Policy" header	5
L Config your server to use the "X-Frame-Options" header	4
L Contact the vendor of your product to see if a patch or a fix has been made available recently	1
L Disable WebDAV, or disallow unneeded HTTP methods	1
L Do not accept body parameters that are sent in the query string	9
L Modify FrontPage extension file permissions to avoid information leakage	2
L Modify your Web.Config file to encrypt the VIEWSTATE parameter	20
L Prevent caching of SSL pages by adding "Cache-Control: no-store" and "Pragma: no-cache" headers to their responses.	67
L Remove old versions of files from the virtual directory	48
L Remove source code files from your web-server and apply any relevant patches	1

This report contains the results of a web application security scan performed by IBM Security AppScan Standard.

High severity issues:	79
Medium severity issues:	198
Total security issues included in the report:	277
Total security issues discovered in the scan:	308

# Application Security Vulnerability Assessment Report

## Issues Sorted by Issue Type

- Authentication Bypass Using SQL Injection 2
- Blind SQL Injection 4
- Cross-Site Request Forgery 24
- Cross-Site Scripting 3
- HTTP PUT Method Site Defacement 20
- Inadequate Account Lockout 1
- Microsoft FrontPage Extensions Site Defacement 3
- Missing Secure Attribute in Encrypted Session (SSL) Cookie 1
- Phishing Through URL Redirection 1
- WebDAV MKCOL Method Site Defacement 20
- Alternate Version of File Detected 50
- Cacheable SSL Page Found 26
- Hidden Directory Detected 7
- Microsoft FrontPage Configuration Information Leakage 1
- Microsoft FrontPage Server Extensions Vital Information Leakage 2
- Microsoft IIS Missing Host Header Information Leakage 1
- Query Parameter in SSL Request 66
- Temporary File Download 32
- Unencrypted \_\_VIEWSTATE Parameter 11
- Web Application Source Code Disclosure Pattern Found 237

# AppScan example

## Advisories

- Authentication Bypass Using SQL Injection
- Blind SQL Injection
- Cross-Site Request Forgery
- Cross-Site Scripting
- HTTP PUT Method Site Defacement
- Inadequate Account Lockout
- Microsoft FrontPage Extensions Site Defacement
- Missing Secure Attribute in Encrypted Session (SSL) Cookie
- Phishing Through URL Redirection
- WebDAV MKCOL Method Site Defacement
- Alternate Version of File Detected
- Cacheable SSL Page Found
- Hidden Directory Detected
- Microsoft FrontPage Configuration Information Leakage
- Microsoft FrontPage Server Extensions Vital Information Leakage
- Microsoft IIS Missing Host Header Information Leakage
- Query Parameter in SSL Request
- Temporary File Download
- Unencrypted \_\_VIEWSTATE Parameter
- Web Application Source Code Disclosure Pattern Found



**H** Authentication Bypass Using SQL Injection 2 TOC

Issue 1 of 2 TOC

**Authentication Bypass Using SQL Injection**  
Severity: **High**  
URL: https://www.r...  
Entity: UserName (Parameter)  
Risk: It may be possible to bypass the web application's authentication mechanism  
Causes: Sanitation of hazardous characters was not performed correctly on user input  
Fix: [Review possible solutions for hazardous character injection](#)

Reasoning: The test result seems to indicate a vulnerability because when four types of request were sent - a valid login, an invalid login, an SQL attack, and another invalid login - the responses to the two invalid logins were the same, while the response to the SQL attack seems similar the response to the valid login.

Issue 2 of 2 TOC

**Authentication Bypass Using SQL Injection**  
Severity: **High**  
URL: https://www...  
Entity: Password (Parameter)  
Risk: It may be possible to bypass the web application's authentication mechanism  
Causes: Sanitation of hazardous characters was not performed correctly on user input  
Fix: [Review possible solutions for hazardous character injection](#)

Reasoning: The test result seems to indicate a vulnerability because when four types of request were sent - a valid login, an invalid login, an SQL attack, and another invalid login - the responses to the two invalid logins were the same, while the response to the SQL attack seems similar the response to the valid login.



## Authentication Bypass Using SQL Injection

TOC

### Test Type:

Application-level test

### Threat Classification:

Insufficient Authentication

### Causes:

Sanitation of hazardous characters was not performed correctly on user input

### Security Risks:

It may be possible to bypass the web application's authentication mechanism

### Affected Products:

### CWE:

508

### References:

"Web Application Disassembly with ODBC Error Messages" (By David Litchfield)  
SQL Injection Training Module

### Technical Description:

The application uses a protection mechanism that relies on the existence or values of an input, but the input can be modified by an untrusted user in a way that bypasses the protection mechanism.

When security decisions such as authentication and authorization are made based on the values of user input, attackers can bypass the security of the software.

Suppose the query in question is:

```
SELECT COUNT(*) FROM accounts WHERE username='$user' AND password='$pass'
```

Where \$user and \$pass are user input (collected from the HTTP request which invoked the script that constructs the query - either from a GET request query parameters, or from a POST request body parameters). A regular usage of this query would be with values \$user=john, \$password=secret123. The query formed would be:

```
SELECT COUNT(*) FROM accounts WHERE username='john' AND password='secret123'
```

The expected query result is 0 if no such user+password pair exists in the database, and >0 if such pair exists (i.e. there is a user named 'john' in the database, whose password is 'secret123'). This would serve as a basic authentication mechanism for the application. But an attacker can bypass this mechanism by submitting the following values: \$user=john, \$password=' OR '1'=1.

### Technical Description:

The application uses a protection mechanism that relies on the existence or values of an input, but the input can be modified by an untrusted user in a way that bypasses the protection mechanism.

When security decisions such as authentication and authorization are made based on the values of user input, attackers can bypass the security of the software.

Suppose the query in question is:

```
SELECT COUNT(*) FROM accounts WHERE username='$user' AND password='$pass'
```

Where \$user and \$pass are user input (collected from the HTTP request which invoked the script that constructs the query - either from a GET request query parameters, or from a POST request body parameters). A regular usage of this query would be with values \$user=john, \$password=secret123. The query formed would be:

```
SELECT COUNT(*) FROM accounts WHERE username='john' AND password='secret123'
```

The expected query result is 0 if no such user+password pair exists in the database, and >0 if such pair exists (i.e. there is a user named 'john' in the database, whose password is 'secret123'). This would serve as a basic authentication mechanism for the application. But an attacker can bypass this mechanism by submitting the following values: \$user=john, \$password=' OR '1'='1'.

The resulting query is:

```
SELECT COUNT(*) FROM accounts WHERE username='john' AND password='' OR '1'='1'
```

This means that the query (in the SQL database) will return TRUE for the user 'john', since the expression 1=1 is always true. Therefore, the query will return a positive number, and thus the user (attacker) will be considered valid without having to know the password.



# Agenda

- ✓ Team Project Guidance
- ✓ Distributed Systems
  - ✓ File Server Architecture
  - ✓ Client/Server Architecture
  - ✓ N-Tier Architecture
  - ✓ Cloud Architecture
  - ✓ Service Oriented Architecture (SOA)
- ✓ Example Cloud-based N-Tier SOA Application Development System
- ✓ Control Stages, Objectives, Application Security Testing
- Additional Best Practices

# Additional best practices for secure application development

1. Defense-in-Depth
2. Positive Security Model
3. Fail Safely
4. Run with Least Privilege
5. Avoid Security by Obscurity
6. Keep Security Simple
7. Use Open Standards
8. Keep, manage and analyze logs to detect Intrusions
9. Never Trust External Infrastructure and Services
10. Establish Secure Defaults

*Characteristics which can help in quickly spotting common weaknesses and poor controls*

# Defense In Depth

*Layered approaches provide more security over the long term than one complicated mass of security architecture*

- **Sequences of routers, firewalls and intrusion detection/protection monitoring devices used to examine data packets, reduce undesired traffic and protect the inner information systems**
- **Access Control Lists (ACLs)**, for example, on the networking routers and firewall equipment to allow only necessary traffic to reach the different system components (i.e. web servers, application servers, file servers, database servers...)
  - *Quickly eliminating access to services, ports, and protocols significantly lowers the attack surface and overall risk of compromise to the system on which the application is running*

## What is Defense in Depth?

Defense in Depth (DiD) is a cybersecurity strategy that employs multiple layers of security controls to protect information systems and networks. The concept is rooted in military defense strategies where multiple barriers delay and prevent an adversary's advance. In cybersecurity, DiD ensures that even if one security measure fails, other mechanisms remain in place to mitigate risks.

## How is Defense in Depth Applied?

Defense in Depth is applied through a combination of administrative, technical, and physical security measures. These include:

- **Administrative Controls:** Policies, procedures, and security awareness training.
- **Technical Controls:** Firewalls, intrusion detection systems (IDS), endpoint security, multi-factor authentication (MFA).
- **Physical Controls:** Access control, surveillance, biometric authentication, locked server rooms.

For example, a corporate network may employ:

- **Firewall + IDS/IPS:** Blocking unauthorized access.
- **Antivirus + Endpoint Detection and Response (EDR):** Protecting endpoints from malware.
- **Multi-Factor Authentication (MFA):** Ensuring identity verification.
- **Data Encryption + Access Controls:** Securing sensitive data.

## Why is Defense in Depth Needed?

- **Mitigates Single Points of Failure:** If one security control is bypassed, others can still protect the system.
- **Reduces Attack Surface:** Multiple layers make exploitation difficult for attackers.
- **Delays Attack Progression:** Attackers must overcome several defenses, increasing detection and response time.
- **Enhances Security Resilience:** Provides redundancy to sustain operations even under attack.

# Positive Security Model

Positive security models use “whitelist” to allow only what is on the list, excluding everything else by default

- “Deny by default”
- In contrast with negative (blacklist) security models that allow everything by default, eliminating only the items known to be bad
  - Problems:
    - Blacklist must be kept up to date
    - Even if blacklist is updated, an unknown vulnerability can still exist
    - Attack surface is much larger than with a positive security model

# Fail Safely

- An application failure can be dealt with in one of 3 ways:
  1. Allow
  2. Block
  3. Error
- In general, application errors should all fail in the same way:
  - Disallow the operation (as viewed by the user) and provide no or minimal information on the failure
  - Do not provide the end user with additional information that may help in compromising the system
    - Put the error information in the logs, but do not provide to the user to use in compromising the system

# Run with Least Privilege

- Principle of Least Privilege mandates that accounts have the least amount of privilege possible to perform their activity
- This includes:
  - User rights: file system access permissions, application process access permissions, and database permissions
  - Resource permissions: CPU limits, memory capacity, network bandwidth

# Avoid Security by Obscurity

- Obfuscating data (hiding it) instead of encrypting it is a very weak security mechanism
  - If a human can figure out how to hide the data a human can learn how to recover the data
- Never obfuscate critical data that can be encrypted or never stored in the first place



# Keep Security Simple

- Simple security mechanisms are easy to verify and easy to implement correctly
- Avoid complex security mechanisms if possible
  - *“The quickest method to break a cryptographic algorithm is to go around it”*
- Do not confuse complexity with layers: Layers are good; complexity isn't

# Use Open Standards

- Open security standards provide increased portability and interoperability
- IT infrastructure is often a heterogeneous mix of platforms, open standards helps ensure compatibility between systems as the application grows
- Open standards are often well known and scrutinized by peers in the security industry to ensure they remain secure

# Keep, manage and analyze logs to help detect intrusions

- Applications should have built-in logging that is protected and easily read
- Logs help you troubleshoot issues, and just as important – help you to track down when or how an application might have been compromised

# Never Trust External Infrastructure and Services

- Many organizations use the processing capabilities of third-party partners that more than likely have differing security policies and postures than your organization
- It is unlikely that you can influence or control an external third party
- Implicitly trusting externally run systems is dangerous!

# Establish Secure Defaults

- New applications should arrive or be presented to users with the most secure default settings possible that still allow business to function
- This may require training end users or communications messages
- End result is a significantly reduced attack surface
  - *Especially when application is pushed out across a large population*

# Test Areas for Auditing Applications

## 1. Input Controls, Process Controls, and Output Controls

- Review and evaluate controls built into system transactions for i data
- Determine the need for error/exception reports related to data integrity and evaluate whether this need has been filled

## 2. Interface Controls

- Review and evaluate the controls in place over data feeds to and from interfacing systems
- If the same data is kept in multiple databases and/or systems, ensure that periodic sync processes are executed to detect any inconsistencies in the data

## 3. Audit Trails

- Review and evaluate the audit trails present in the system and the controls over those audit trails
- Ensure that the system provides a means of tracing a transaction or piece of data from the beginning to the end of the process enabled by the system

# Test Areas for Auditing Applications

## 4. Software Change Controls

- Ensure that the application software cannot be changed without going through a standard checkout/staging/testing/approval process after it is placed into production
- Evaluate controls regarding code checkout and versioning
- Evaluate controls regarding the testing of application code before it is placed into a production environment
- Evaluate controls regarding batch scheduling

## 5. Backup and Recovery

- Determine whether a Business Impact Analysis (BIA) has been performed on the application to establish backup and recovery needs
- Ensure that appropriate backup and recovery controls are in place
- Ensure appropriate recovery controls are in place

# Test Areas for Auditing Applications

## 6. Data Retention and User Involvement

- Evaluate controls regarding the application's data retention
- Evaluate overall user involvement and support for the Application

## 7. Identity, Authentication, and Access Controls...

## 8. Host Hardening...



# Agenda

- ✓ Midterm Exam Review
- ✓ Team Project Guidance
- ✓ Distributed Systems
  - ✓ File Server Architecture
  - ✓ Client/Server Architecture
  - ✓ N-Tier Architecture
  - ✓ Cloud Architecture
  - ✓ Service Oriented Architecture (SOA)
- ✓ Example Cloud-based N-Tier SOA Application Development System
- ✓ Control Stages, Objectives, Application Security Testing
- ✓ Additional Best Practices