AUBURN

Open Web Application Security Project (OWASP)

OWASP Top Ten

AUBURN

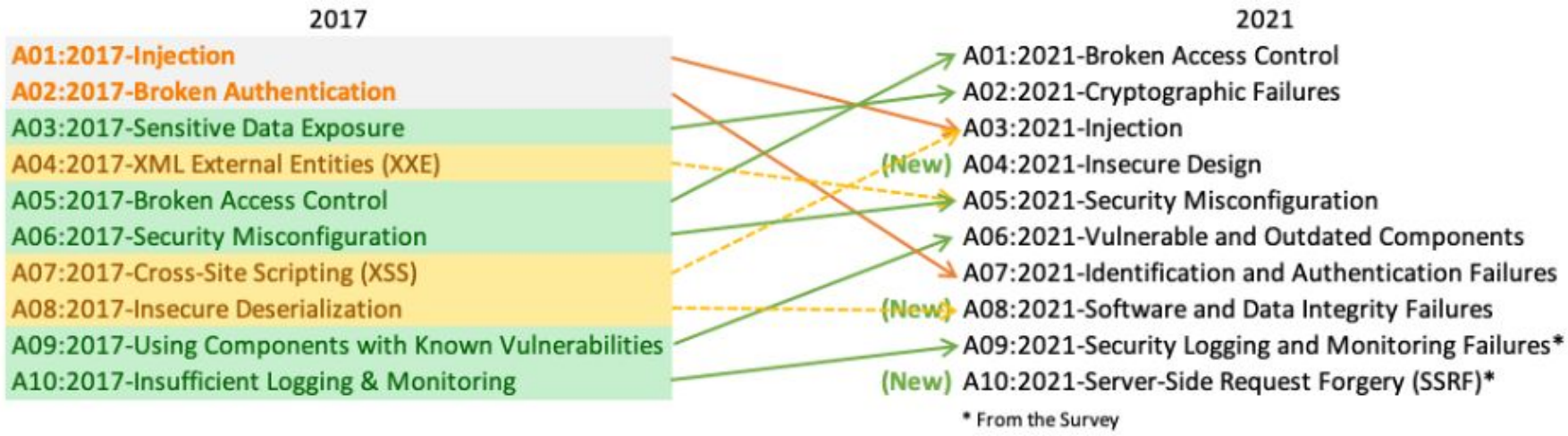
- Current list of the most common web security vulnerabilities
  - Updated almost annually as threats and vulnerabilities are determined to be prevalent in web technology
  - Changes also due to prevention methods build into applications technology stacks and browsers.
- Objectives
  - Build awareness with among key stakeholders
    - Developers, Architects, Software Testers, etc.
  - Avoids setting any type of standards
  - Re-enforce best practices during software development lifecycle on addressing vulnerabilities
- OWASP is frequently sited throughout the industry, websites, books, etc.

2021 OWASP Top Ten

AUBURN

- Broken Access Control
- Cryptographic Failure - Module 3
- Injection (Javascript, SQL, RCE, etc.) - Module 3
- Insecure Design
- Security Misconfiguration
- Vulnerable and Outdate Components – Module 4
- Identification and Authentication Failures
- Software and Data Integrity Values
- Security Logging and Monitoring Failure - Module 2
- Server Side Request Forgery (CSRF)

Changes Since 2017



Broken Access Control

AUBURN

- Access control enforces policy such that users cannot act outside of their intended permissions.
- Access Control defects cause:
  - Unauthorized information disclosure (Confidentiality Principle)
  - Modification or destruction data (Integrity Principle)
  - Performing a business function outside the user's limits (Integrity Principle)
  - Ability to bring down system components (Availability Principle)
- Common Mistakes:
  - Least privilege principle by not giving new users lowest possible permissions
  - HTTP URL requests not checking user permissions
  - Accessing API with missing access controls for POST, PUT and DELETE
  - Impersonating as an administrative user

Cryptographic Failures

AUBURN

- Protection needs of data in transit and at rest.
  - Passwords, credit card numbers, health records, personal information, and business secrets require extra protection
  - Privacy Regulations - EU's General Data Protection Regulation (GDPR), PCI Data Security Standard (PCI DSS), US HIPAA
- Common Issues
  - Is any data transmitted in clear text?
  - Cryptographic algorithms not up to date?
  - Encryption not enforced (HTTP vs. HTTPs)
  - Are weak crypto keys generated or re-used, or is proper key management or rotation missing?
  - Are secrets checked into source code repositories?

Cryptographic Failures

AUBURN

- Prevention Techniques
  - Classify data processed, stored, or transmitted by an application.
  - Make sure to encrypt all sensitive data at rest.
  - Ensure up-to-date and strong standard algorithms, protocols, and keys are in place
  - Use proper key management.
  - Encrypt all data in transit with secure protocols (HTTPS)
  - Avoid legacy protocols: Telnet (use ssh), FTP (use sFTP) and SMTP for transporting sensitive data.
  - Store passwords using salted hashing functions with a work factor (work, but not too much work)

Injection

AUBURN

- Injection defined as tricking an application into including unintended commands in the data sent to an interpreter (SQL, Javascript, Java, etc.)
- Frameworks or Components take strings and interpret them as commands
  - Database – SQL Statements
  - Operating System – Shell Commands/scripts
  - XML – XPath Statements
  - LDAP - Authentication
  - Logging – Remote Code Execution
- Many applications still don't protect against in spite of being easy to protect against
- Possible Severe Impact
  - Return sensitive data from databases
  - Gain system access or execute remote code

Injection

AUBURN

- Prevention
  - The preferred option is to use a safe API, which avoids using the interpreter entirely.
    - Example: Use Object Relational Mapping Tools (ORMs) instead of direct SQL statements.
  - Input validation.
    - However many applications require special characters
  - Escape special characters using the specific escape syntax for that interpreter.
  - Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

Insecure Design

AUBURN

- New in 2021 OWASP Top Ten
- Defined as missing or ineffective control design. Differentiates between insecure design and insecure implementation.
  - Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods. Threat modeling should be integrated into refinement sessions (or similar activities); look for changes in data flows and access control or other security controls.
  - Secure software requires a secure development lifecycle, some form of secure design pattern, paved road methodology, secured component library, tooling, and threat modeling.

Insecure Design

AUBURN

- Prevention
  - Establish and use a secure development lifecycle with AppSec professionals to help evaluate and design security and privacy-related controls
  - Establish and use a library of secure design patterns or paved road ready to use components
  - Use threat modeling for critical authentication, access control, business logic, and key flows
  - Integrate security language and controls into user stories
  - Write unit and integration tests to validate that all critical flows are resistant to the threat model. Compile use-cases and misuse-cases for each tier of your application.
  - Limit resource consumption by user or service

Security Misconfiguration

AUBURN

- Any configuration along the application stack that exposes a vulnerability.
- Common Vulnerabilities:
  - Improperly configured permissions on cloud services.
  - Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
  - Default accounts and their passwords are still enabled and unchanged.
  - Error handling reveals stack traces or other overly informative error messages to users.
  - For upgraded systems, the latest security features are disabled or not configured securely.
  - The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
  - The software is out of date or vulnerable

Security Misconfiguration

AUBURN

- Prevention
  - A repeatable hardening process. Regular penetration and vulnerability scans.
  - A pre production environments configured identically as production with different credentials.
  - Automated environment setup, configuration, and deployment to make sure it is setup exactly the same.
  - A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
  - Quality Assurance performing scheduled tasks to review and update the configurations appropriate to all security notes, updates, and patches
  - A segmented application architecture provides effective and secure separation between components or tenants, with segmentation, containerization, or cloud security groups

Vulnerable and Outdated Components

AUBURN

- Any external dependent software that is not tracked or out of date.
- Examples:
  - Unknown component versions both directly and indirectly.
  - Vulnerable, unsupported, or out of date including OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
  - Regularly conduct vulnerability scans and subscribe to security bulletins related to the components you use.
  - No process to upgrade or fix underlying platform, frameworks, and dependencies in a risk-based, timely fashion.
  - No security or access control on component configurations or to software build process

Vulnerable and Outdated Components

AUBURN

- Prevention
  - Remove unused dependencies, unnecessary features, components, files, and documentation.
  - Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies using scanning tools
  - Only obtain components from official sources over secure links.
  - Check file signatures and prefer signed packages to reduce the chance of including a modified, malicious component
  - Monitor for libraries and components that are unmaintained or do not create security patches for older versions.

Identification and Authentication Failures

AUBURN

- Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks.
- Examples:
  - Permits automated attacks where the attacker has a list of valid usernames and passwords.
  - Permits brute force or other automated attacks.
  - Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
  - Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe.
  - Uses plain text, encrypted, or weakly hashed passwords data stores
  - Has missing or ineffective multi-factor authentication.
  - Exposes session identifier in the URL.
  - Reuse session identifier after successful login.
  - Does not correctly invalidate Session IDs. User sessions or authentication tokens period of inactivity.

Identification and Authentication Failures

AUBURN

- Prevention
  - Implement multi-factor authentication
    - Prevents automated credential stuffing, brute force, and stolen credential reuse attacks.
  - Do not ship or deploy with any default credentials, particularly for admin users.
  - Implement weak password checks
  - Enforce password complexity with length, complexity, and rotation policies
  - Ensure registration, credential recovery, and API pathways are hardened
  - Limit or increasingly delay failed login attempts
  - Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
  - Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login.
  - Session identifier should not be in the URL, be securely stored, and invalidated after logout, idle, and absolute timeouts.

Software and Data Integrity Failures

AUBURN

- Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations.
- Examples:
  - Application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs).
  - Insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise.
    - Internal Bad Actor
  - Auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application.
    - Solarwinds SunBurst attack used this threat vector

Software and Data Integrity Failures

AUBURN

- Prevention
  - Use digital signatures or similar mechanisms to verify the software or data is from the expected source and has not been altered.
  - Ensure libraries and dependencies, such as npm or Maven, are consuming trusted repositories.
    - Consider hosting an internal known-good repository with controls
  - Ensure that a software supply chain security tool, is used to verify that components do not contain known vulnerabilities
  - Ensure that there is a review process for code and configuration changes to your software build pipeline
  - Ensure that your CI/CD pipeline has proper segregation, configuration, and access control to ensure the integrity of the code flowing through the build and deploy processes.

Security Logging and Monitoring Failures

AUBURN

- Logging is critical to help detect, escalate, and respond to active breaches.
  - Without logging and monitoring, breaches cannot be detected.
  - Logging and monitoring can be challenging to test, often involving interviews or asking if attacks were detected during a penetration test.
- Examples:
  - No logging of auditable events, such as logins, failed logins, and high-value transactions.
  - Warnings and errors generate no, inadequate, or unclear log messages.
  - Logs of applications and APIs are not monitored for suspicious activity.
  - Logs are only stored locally and propagated to a SEIM tool
  - Appropriate alerting thresholds and response escalation processes are not in place or effective.
  - Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts.
  - The application cannot detect, escalate, or alert for active attacks in real-time or near real-time.

Security Logging and Monitoring Failures

AUBURN

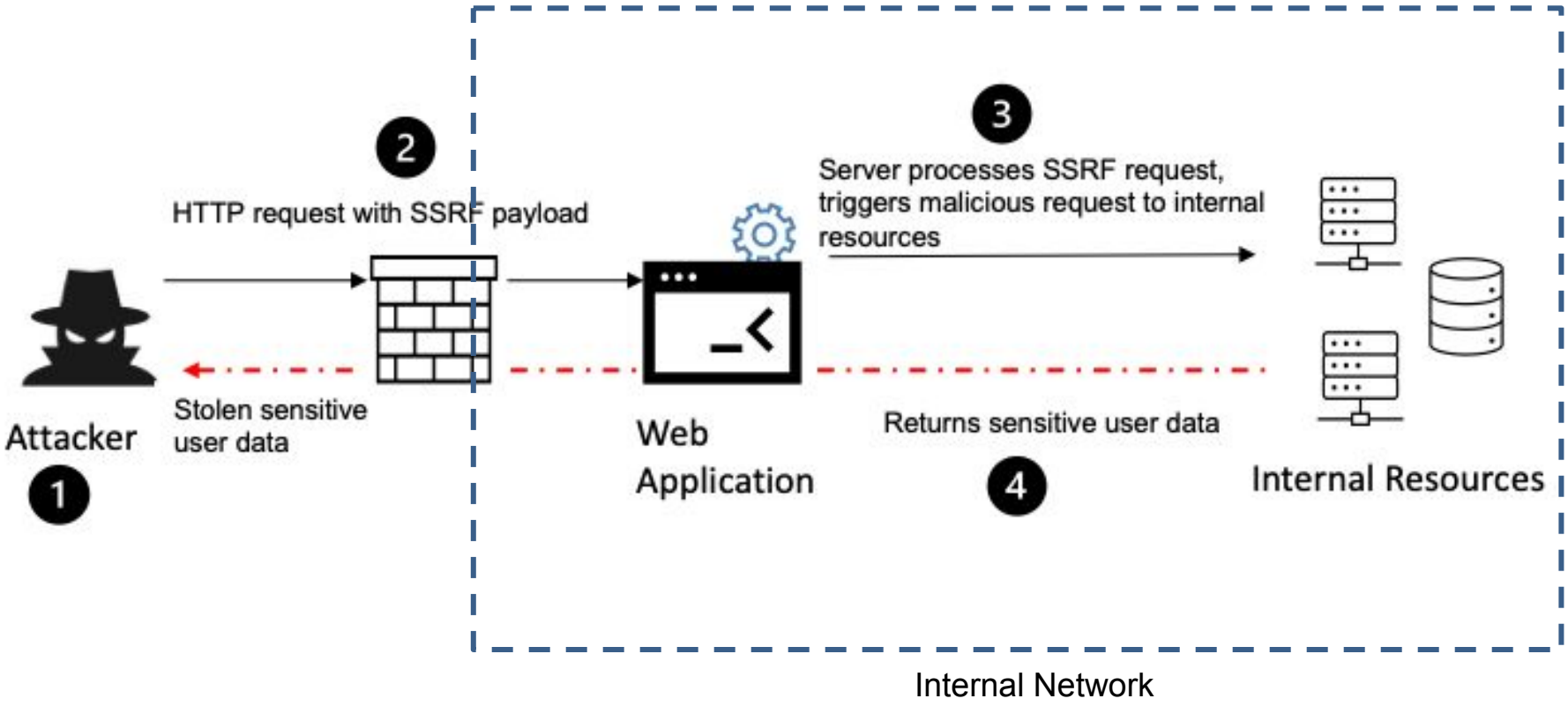
- Prevention
  - Ensure all login, access control, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for enough time to allow delayed forensic analysis.
  - Ensure that logs are generated in a format that log management solutions can easily consume.
  - Ensure log data is encoded correctly to prevent injections or attacks on the logging or monitoring systems.
    - Log4J2 vulnerability was this type of vulnerability
  - Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion
  - Operations teams should establish effective monitoring and alerting such that suspicious activities are detected and responded to quickly.
  - Establish or adopt an incident response and recovery plan

Server-Side Request Forgery (SSRF)

AUBURN

- SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL.
  - As modern web applications provide end-users with convenient features, fetching a URL, such as webhooks, becomes a common scenario
- Prevention:
  - Sanitize and validate all client-supplied input data
  - Enforce the URL schema, port, and destination with a positive allow list
  - Do not send raw responses to clients
  - Disable HTTP redirectionsegment remote resource access functionality in separate networks to reduce the impact of SSRF
  - Log all accepted and blocked network flows on firewalls to trigger alerts and incident process.

Server-Side Request Forgery (SSRF)



GET /index.php?url=http://192.0.2.100/admin/ HTTP/1.1

Server-Side Request Forgery (SSRF)

AUBURN

- Prevention
  - Sanitize and validate all client-supplied input data
  - Enforce the URL schema, port, and destination with a positive allow list
  - Do not send raw responses to clients
  - Disable HTTP redirections

AUBURN UNIVERSITY