The OWASP Top 10

August 4, 2022

TLP: WHITE, ID# *202208041300*

Agenda

- OWASP and the OWASP Top 10
- Understanding the Top 10
- Data Factors
- The OWASP Top 10: 2021
- The OWASP Top 10 as a Standard

Slides Key:

Non-Technical: Managerial, strategic and high-level (general audience)

Technical: Tactical / IOCs; requiring in-depth knowledge (sysadmins, IRT)

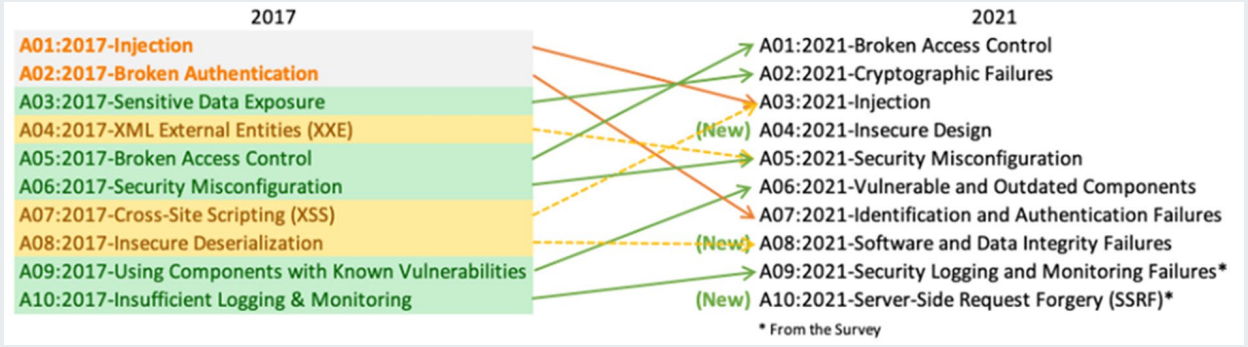
OWASP and the OWASP Top 10

- Threat Brief: Web Application Attacks in Healthcare
- Open Web Application Security Project (OWASP)
  - Nonprofit foundation dedicated to improving software security
  - Operates under an “open community” model, meaning that anyone can participate in and contribute to OWASP-related online chats, projects, and more
- OWASP Top 10
  - A standard awareness document for developers and web application security
  - Represents a broad consensus about the most critical security risks to web applications

Source: OWASP

Understanding the Top 10

- The 10 Categories
  - 8 of 10 categories are contributed data
  - 2 of the categories come from the Top 10 community survey
    - These two categories reflect what AppSec researchers see as the highest risks that may not be in the data (and may never be expressed in data)
- Common Weakness Enumerations (CWEs)
  - A community-developed list of software and hardware weakness types
  - Serves as a common language, a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts

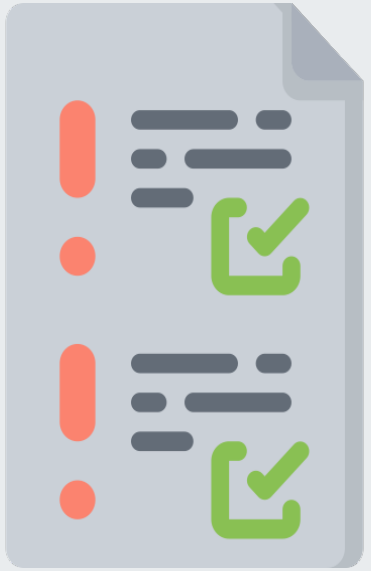


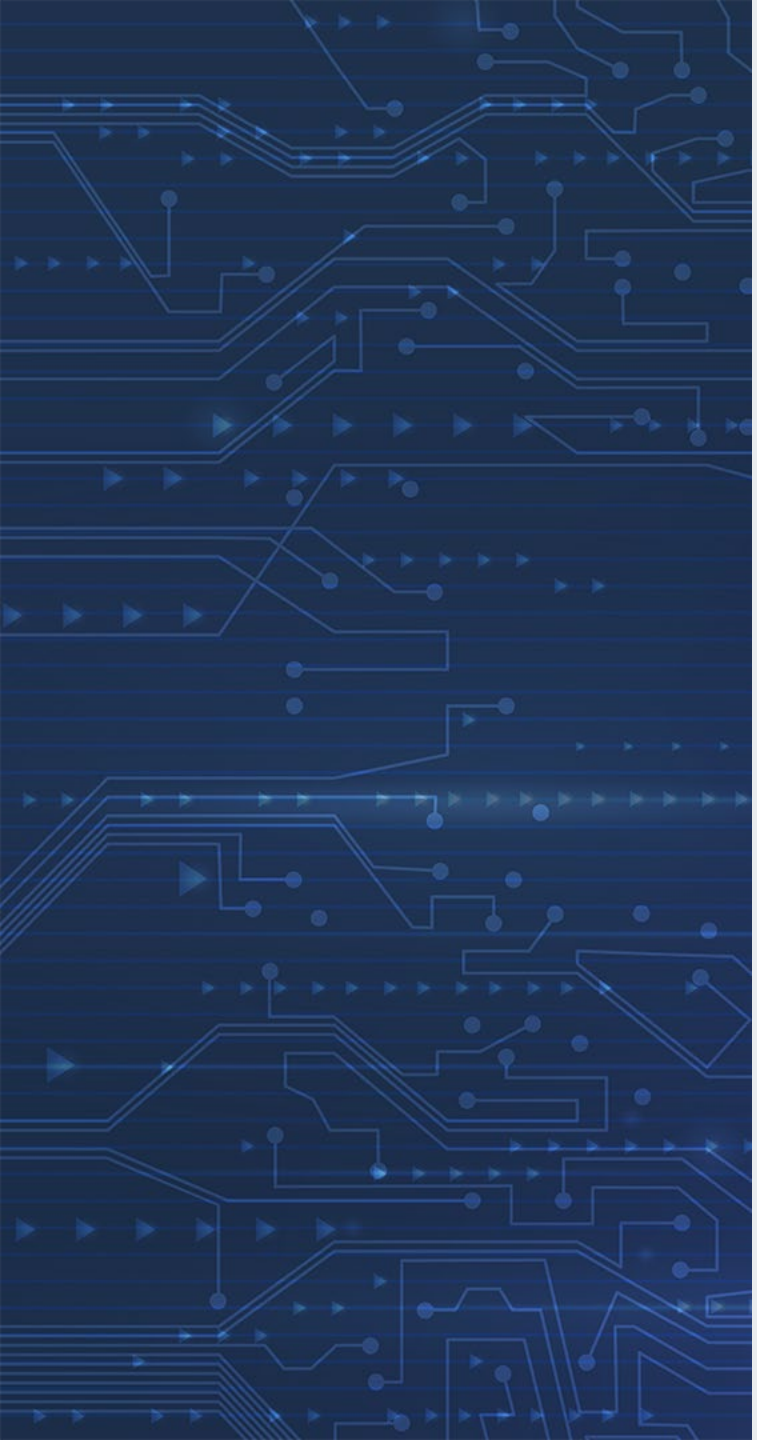
Source: OWASP

Data Factors

Data factors are listed for each of the Top 10 categories:

- **CWEs Mapped:** The number of CWEs mapped to a category by the Top 10 team
- **Incidence Rate:** The percentage of applications vulnerable to that CWE from the population tested by that organization for that year
- **Weighted Exploit:** The Exploit sub-score from CVSSv2 and CVSSv3 scores assigned to CVEs mapped to CWEs, normalized, and placed on a 10pt scale
- **Weighted Impact:** The Impact sub-score from CVSSv2 and CVSSv3 scores assigned to CVEs mapped to CWEs, normalized, and placed on a 10pt scale
- **Total Occurrences:** Total number of applications found to have the CWEs mapped to a category
- **Total CVEs:** Total number of CVEs in the National Vulnerability Database (NVD DB) that were mapped to the CWEs mapped to a category





The OWASP Top 10: 2021

A01:2021 – Broken Access Control

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
34	55.97%	3.81%	6.92	5.93	318,487	19,013

Example Scenario:

An application uses unverified data in a structured query language (SQL) call that is accessing account information:

pstmt.setString(1, request.getParameter("acct"));

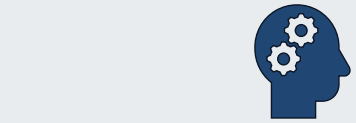
ResultSet results = pstmt.executeQuery( );

An attacker simply modifies the browser's 'acct' parameter to send whatever account number they want. If not correctly verified, the attacker can access any user's account.

https://example.com/app/accountInfo?acct=notmyacct

Notable Common Weakness Enumerations (CWEs):

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
- CWE-201: Insertion of Sensitive Information Into Sent Data
- CWE-352: Cross-Site Request Forgery



Broken Access Control Vulnerabilities

- Violation of the principle of least privilege or deny by default, where access should only be granted for particular capabilities, roles, or users, but is available to anyone.
- Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests.
- Permitting viewing or editing someone else's account, by providing its unique identifier (insecure direct object references).
- Accessing API with missing access controls for POST, PUT and DELETE.
- Elevation of privilege; acting as a user without being logged in or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token, or a cookie or hidden field manipulated to elevate privileges or abusing JWT invalidation.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

Broken Access Control Prevention

- Access control is only effective in trusted server-side code or server-less application programming interface (API), where the attacker cannot modify the access control check or metadata.
- Except for public resources, deny by default.
- Implement access control mechanisms once, and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
- Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g., repeated failures).
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- Stateful session identifiers should be invalidated on the server after logout.

A02:2021 – Cryptographic Failures

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
29	46.44%	4.49%	7.29	6.81	233,788	3,075

Example Scenario:

An application encrypts credit card numbers in a database using automatic database encryption, but this data is automatically decrypted when retrieved, allowing a structured query language (SQL) injection flaw to retrieve credit card numbers in clear text.

Notable Common Weakness Enumerations (CWEs):

- CWE-259: Use of Hard-coded Password
- CWE-327: Broken or Risky Crypto Algorithm
- CWE-331 Insufficient Entropy

Cryptographic Failure Vulnerabilities

- Data is transmitted in clear text.
- Old or weak cryptographic algorithms or protocols are used either by default or in older code.
- Use of default crypto keys, weak crypto keys are generated or re-used, or a proper key management or rotation is missing.
- Encryption is not enforced, e.g., any HTTP headers (browser) security directives or headers are missing.
- Received server certificate and the trust chain are not properly validated.
- Passwords are being used as cryptographic keys in absence of a password base key derivation function.
- Deprecated hash functions such as MD5 or SHA1 in use, or non-cryptographic hash functions are used when cryptographic hash functions are needed.
- Deprecated cryptographic padding methods such as public key cryptography standard (PKCS) number 1 v1.5 are in use.
- Cryptographic error messages or side channel information are exploitable, for example in the form of padding oracle attacks.

Cryptographic Failure Prevention

- Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Do not store sensitive data unnecessarily. Discard it as soon as possible or use Payment Card Industry Data Security Standard (PCI DSS) compliant tokenization or even truncation.
- Make sure to encrypt all sensitive data at rest.
- Encrypt all data in transit with secure protocols such as TLS with forward secrecy (FS) ciphers, cipher prioritization by the server, and secure parameters; enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Apply required security controls as per the data classification.
- Do not use legacy protocols such as file transfer protocol (FTP) and simple mail transfer protocol (SMTP) for transporting sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor).
- Keys should be generated randomly with cryptography and stored in memory as byte arrays.
- Ensure that cryptographic randomness is used where appropriate, and that it has not been seeded in a predictable way or with low entropy.
- Avoid deprecated cryptographic functions and padding schemes, such as MD5, SHA1, PKCS number 1 v1.5.
- Verify independently the effectiveness of configuration and settings.

A03:2021-Injection

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
33	19.09%	3.37%	7.25	7.15	274,228	32,078

Example Scenario:

An application uses untrusted data in the construction of the following vulnerable structured query language (SQL) call:

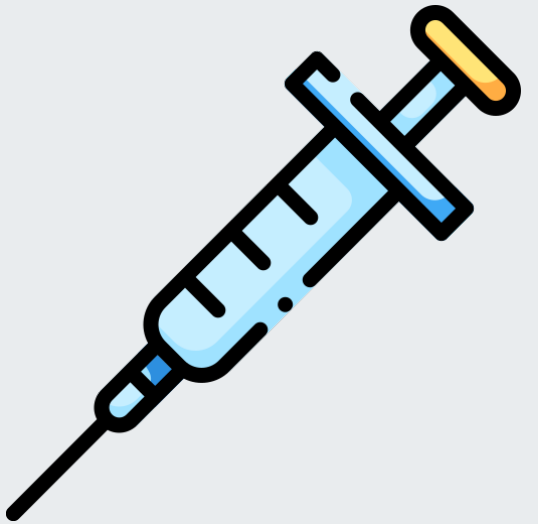
*String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "';*

Notable Common Weakness Enumerations (CWEs):

- CWE-79: Cross-site Scripting
- CWE-89: SQL Injection
- CWE-73: External Control of File Name or Path

Injection Vulnerabilities

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated; the structured query language (SQL) or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.



Injection Prevention

- The preferred option is to use a safe API, which avoids using the interpreter entirely, provides a parameterized interface, or migrates to Object Relational Mapping Tools (ORMs).
- Use positive server-side input validation.
- For any residual dynamic queries, 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.
- Source code review is the best method of detecting if applications are vulnerable to injections; automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs is strongly encouraged.

A04:2021 – Insecure Design

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
40	24.19%	3.00%	6.46	6.78	262,407	2,691

Example Scenario:

A movie theater chain that allows group booking discounts requires a deposit for groups of more than fifteen people. Attackers threat model this flow to see if they can book hundreds of seats across various theaters in the chain, thereby causing thousands of dollars in lost income.

Notable Common Weakness Enumerations (CWEs):

- CWE-209: Generation of Error Message Containing Sensitive Information
- CWE-256: Unprotected Storage of Credentials
- CWE-501: Trust Boundary Violation
- CWE-522: Insufficiently Protected Credentials

Insecure Design 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.
- Integrate plausibility checks at each tier of your application (from frontend to backend).
- 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.
- Segregate tier layers on the system and network layers depending on the exposure and protection needs.
- Segregate tenants robustly by design throughout all tiers.
- Limit resource consumption by user or service.

A05:2021 – Security Misconfiguration

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
20	19.84%	4.51%	8.12	6.56	208,387	789

Example Scenario:

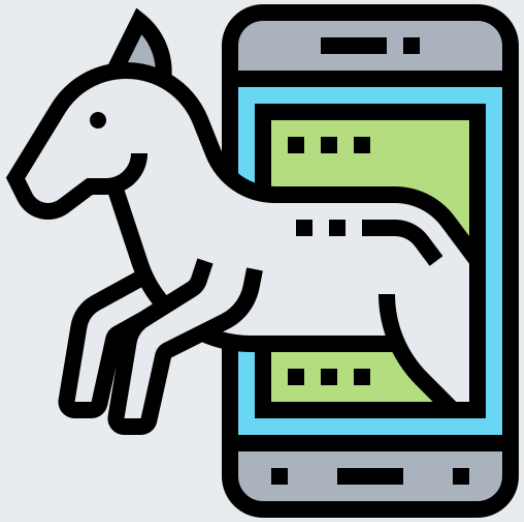
The application server comes with sample applications not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. Suppose one of these applications is the admin console, and default accounts weren't changed. In that case, the attacker logs in with default passwords and takes over.

Notable Common Weakness Enumerations (CWEs):

- CWE-16 Configuration
- CWE-611 Improper Restriction of XML External Entity Reference

Secure Misconfiguration Vulnerabilities

- Missing appropriate security hardening across any part of the application stack or 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, libraries, databases, etc., are not set to secure values.
- The server does not send security headers or directives, or they are not set to secure values.
- The software is out of date or vulnerable.



Security Misconfiguration Prevention

- Development, QA, and production environments should all be configured identically, with different credentials used in each environment.
- A minimal platform without any unnecessary features, components, documentation, and samples; remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate to all security notes, updates, and patches as part of the patch management process. Review cloud storage permissions (e.g., S3 bucket permissions).
- A segmented application architecture provides effective and secure separation between components or tenants, with segmentation, containerization, or cloud security groups.
- Sending security directives to clients, e.g., Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments.

A06:2021 – Vulnerable and Outdated Components

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
3	27.96%	8.77%	51.78	22.47	30,457	0

Example Scenario:

Due to the volume of components used in development, a development team might not know or understand all the components used in their application, and some of those components might be out-of-date and therefore vulnerable to attack.

Notable Common Weakness Enumerations (CWEs):

- CWE-1104: Use of Unmaintained Third-Party Components
- CWE-937 OWASP Top 10 2013: Using Components with Known Vulnerabilities
- CWE-1035 2017 Top 10 A9: Using Components with Known Vulnerabilities

Vulnerable and Outdated Components Vulnerabilities

- Versions of components you use (both client-side and server-side) are unknown. This includes components you directly use as well as nested dependencies.
- Software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, application programing interfaces (APIs) and all components, runtime environments, and libraries.
- Vulnerabilities are not scanned for regularly.
- Fixes or upgrades are not implemented to the underlying platform, frameworks, and dependencies in a risk-based, timely fashion.
- Software developers are not testing the compatibility of updated, upgraded, or patched libraries.
- Components' configurations are not secure.

Vulnerable and Outdated Components 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 tools like versions, OWASP Dependency Check, retire.js, etc. Continuously monitor sources like Common Vulnerability and Exposures (CVE) and National Vulnerability Database (NVD) for vulnerabilities in the components.
- Only obtain components from official sources over secure links.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.
  - While the internet of things (IoT) is frequently difficult or impossible to patch, the importance of patching them can be great (e.g., biomedical devices).
- Every organization must ensure an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio

A07:2021 – Identification and Authentication Failures

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
22	14.84%	2.55%	7.40	6.50	132,195	3,897

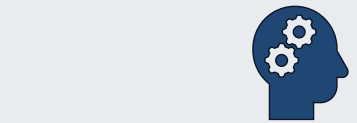
Example Scenario:

Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered the best practices, password rotation and complexity requirements encourage users to use and reuse weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.

Notable Common Weakness Enumerations (CWEs):

- CWE-297: Improper Validation of Certificate with Host Mismatch
- CWE-287: Improper Authentication
- CWE-384: Session Fixation

Identification and Authentication Failures and Vulnerabilities



- Permits automated attacks such as credential stuffing, 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 (see A02:2021-Cryptographic Failures).
- Has missing or ineffective multi-factor authentication.
- Exposes session identifier in the URL.
- Reuses session identifier after successful login.
- Does not correctly invalidate Session IDs.

Prevention of Identification and Authentication Failures

- Where possible, implement multi-factor authentication to prevent 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, such as testing new or changed passwords against the top 10,000 worst passwords list.
- Align password length, complexity, and rotation policies with National Institute of Standards and Technology (NIST) 800-63b's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence-based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts but be careful not to create a denial-of-service scenario. 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.

A08:2021 – Software and Data Integrity Failures

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
10	16.67%	2.05%	6.94	7.94	47,972	1,152

Example Scenario:

The SolarWinds malicious update. Nation-states have been known to attack update mechanisms, with a recent notable attack being the SolarWinds Orion attack. The company that develops the software had secure build and update integrity processes. Still, these were able to be subverted, and for several months, the firm distributed a highly targeted malicious update to more than 18,000 organizations, of which around 100 or so—including a hospital—were affected. This is one of the most far-reaching and most significant breaches of this nature in history.

Notable Common Weakness Enumerations (CWEs):

- CWE-829: Inclusion of Functionality from Untrusted Control Sphere
- CWE-494: Download of Code Without Integrity Check
- CWE-502: Deserialization of Untrusted Data

Software and Data Integrity Failures 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. If you have a higher risk profile, consider hosting an internal, known-good repository that's vetted.
- Ensure that a software supply chain security tool, such as OWASP Dependency Check or OWASP CycloneDX, is used to verify that components do not contain known vulnerabilities.
- Ensure that there is a review process for code and configuration changes to minimize the chance that malicious code or configuration could be introduced into your software pipeline.
- Ensure that your continuous integration and continuous deployment (CI/CD) pipeline has proper segregation, configuration, and access control to ensure the integrity of the code flowing through the build and deploy processes.
- Ensure that unsigned or unencrypted serialized data is not sent to untrusted clients without some form of integrity check or digital signature to detect tampering or replay of the serialized data.

A09:2021 – Security Logging and Monitoring Failures

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
4	19.23%	6.51%	6.87	4.99	53,615	242

Example Scenario:

A children's health plan provider's website operator couldn't detect a breach due to a lack of monitoring and logging. An external party informed the health plan provider that an attacker had accessed and modified thousands of sensitive health records of more than 3.5 million children. A post-incident review found that the website developers had not addressed significant vulnerabilities. As there was no logging or monitoring of the system, the data breach could have been in progress since 2013, a period of more than seven years.

Notable Common Weakness Enumerations (CWES):

- CWE-778 Insufficient Logging
- CWE-117 Improper Output Neutralization for Logs
- CWE-223 Omission of Security-relevant Information
- CWE-532 Insertion of Sensitive Information into Log File

Security Logging and Monitoring Failures Vulnerabilities

- Insufficient logging, detection, monitoring, and active response occurs any time:
  - Auditable events, such as logins, failed logins, and high-value transactions, are not logged.
  - 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.
  - 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 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.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- DevSecOps 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, such as National Institute of Standards and Technology (NIST) 800-61r2 or later.

A10:2021 – Server-Side Request Forgery (SSRF)

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences	Total CVEs
1	2.72%	2.72%	8.28	6.72	9,503	385

Example Scenario:

If a network architecture is unsegmented, attackers can use connection results or elapsed time to connect or reject server-side request forgery (SSRF) payload connections to map out internal networks and determine if ports are open or closed on internal servers.

Notable Common Weakness Enumerations (CWEs):

As new entries are likely to be a single or small cluster of CWEs for attention and awareness, the hope is that they are subject to focus and can be rolled into a larger category in a future edition.

Server-Side Request Forgery (SSRF) Prevention

- From network layer:
  - Segment remote resource access functionality in separate networks to reduce the impact of SSRF.
  - Enforce “deny by default” firewall policies or network access control rules to block all but essential intranet traffic.
- From application layer:
  - 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.
  - Be aware of the URL consistency to avoid attacks such as DNS rebinding and “time of check, time of use” (TOCTOU) race conditions.
  - Do not mitigate SSRF via the use of a deny list or regular expression.
- Additional measures to consider:
  - Don't deploy other security relevant services on front systems (e.g., OpenID). Control local traffic on these systems (e.g., localhost).
  - For frontends with dedicated and manageable user groups use network encryption (e.g., VPNs) on independent systems to consider very high protection needs.

The OWASP Top 10 as a Standard

- Also use:
  - OWASP Application Security Verification Standard

Use Case	OWASP Top 10 2021	OWASP Application Security Verification Standard
Awareness	Yes	-
Training	Entry level	Comprehensive
Design and architecture	Occasionally	Yes
Coding standard	Bare minimum	Yes
Secure Code review	Bare minimum	Yes
Peer review checklist	Bare minimum	Yes
Unit testing	Occasionally	Yes
Integration testing	Occasionally	Yes
Penetration testing	Bare minimum	Yes
Tool support	Bare minimum	Yes
Secure Supply Chain	Occasionally	Yes

Reference Materials

References

- “Broken Access Control,” OWASP. N.d. https://owasp.org/Top10/A01_2021-Broken_Access_Control/
- “Cryptographic Failures,” OWASP. N.d. https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
- “How to use the OWASP Top 10 as a standard,” OWASP. N.d. https://owasp.org/Top10/A00_2021_How_to_use_the_OWASP_Top_10_as_a_standard/
- “Identification and Authentication Failures,” OWASP. N.d. https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/
- “Injection,” OWASP. N.d. https://owasp.org/Top10/A03_2021-Injection/
- “Insecure Design,” OWASP. N.d. https://owasp.org/Top10/A04_2021-Insecure_Design/
- “OWASP Top 10,” OWASP. N.d. https://owasp.org/www-project-top-ten/
- “OWASP Top 10 2021,” Synopsys. N.d. https://www.synopsys.com/glossary/what-is-owasp-top-10.html
- “OWASP Top Ten 2021 : Related Cheat Sheets,” OWASP. N.d. https://cheatsheetseries.owasp.org/IndexTopTen.html

References

- “Security Logging and Monitoring Failures,” OWASP. N.d. https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/
- “Security Misconfiguration,” OWASP. N.d. https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
- “Server Side Request Forgery,” OWASP. N.d. https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/
- “Software and Data Integrity Failures,” OWASP. N.d. https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/
- “Vulnerable and Outdated Components,” OWASP. N.d. https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/
- “Welcome to the OWASP Top 10 - 2021,” OWASP. N.d. https://owasp.org/Top10/A00_2021_Introduction/

Questions

FAQ

Upcoming Briefing

- 8/18 – The Impact of Social Engineering in Healthcare

Product Evaluations

Recipients of this and other Healthcare Sector Cybersecurity Coordination Center (HC3) Threat Intelligence products are **highly encouraged** to provide feedback. To provide feedback, please complete the HC3 Customer Feedback Survey.

Requests for Information

Need information on a specific cybersecurity topic? Send your request for information (RFI) to HC3@HHS.GOV.

Disclaimer

These recommendations are advisory and are not to be considered as federal directives or standards. Representatives should review and apply the guidance based on their own requirements and discretion. The HHS does not endorse any specific person, entity, product, service, or enterprise.

About HC3

The Health Sector Cybersecurity Coordination Center (HC3) works with private and public sector partners to improve cybersecurity throughout the Healthcare and Public Health (HPH) Sector. HC3 was established in response to the Cybersecurity Information Sharing Act of 2015, a federal law mandated to improve cybersecurity in the U.S. through enhanced sharing of information about cybersecurity threats.

What We Offer

Sector and Victim Notifications

Direct communications to victims or potential victims of compromises, vulnerable equipment, or PII/PHI theft, as well as general notifications to the HPH about current impacting threats via the HHS OIG.

Alerts and Analyst Notes

Documents that provide in-depth information on a cybersecurity topic to increase comprehensive situational awareness and provide risk recommendations to a wide audience.

Threat Briefings

Presentations that provide actionable information on health sector cybersecurity threats and mitigations. Analysts present current cybersecurity topics, engage in discussions with participants on current threats, and highlight best practices and mitigation tactics.

Office of
Information Security
Securing One HHS

Health Sector Cybersecurity Coordination Center

Contacts

HHS.GOV/HC3

HC3@HHS.GOV