



OWASP Top 10 - Web (2021 and 2025)

OWASP Top 10:2021 (A01-A10) explained with CWE names, detection coverage, 2017 and 2025 deltas, secure configuration

#OWASPTop10

#WebSecurity

#AppSec

#Vulnerabilities

What the list is

OWASP Top 10:2021

OWASP (Open Worldwide Application Security Project, a non-profit foundation) publishes an **awareness document, not a standard**: the ten most critical **web application risk categories**. Eight categories are ranked from vulnerability data on 500k+ applications; two (A04, A10) come from a community survey of practitioners, to capture risks the data cannot see yet. Each category groups several CWEs (Common Weakness Enumeration: MITRE's catalogue of weakness types, e.g. CWE-89 = SQL injection). For a **testable** requirement list use ASVS (Application Security Verification Standard); for **how to test** use WSTG (Web Security Testing Guide).

Incidence rate — Share of tested applications that had at least one CWE of the category (not the number of findings).

CWE — Common Weakness Enumeration: MITRE catalogue of software weakness types, each with a number and a name.

Root-cause category — Category named after why the flaw exists (e.g. Insecure Design) rather than its symptom (e.g. a specific attack).

IDOR — Insecure Direct Object Reference: the app uses an id from the request without checking the caller may access that object.

SSRF — Server-Side Request Forgery: the server is made to send a request to a URL chosen by the attacker.

XXE — XML External Entity: an XML parser resolves an entity pointing to a file or URL, leaking data or reaching internal hosts.

94%

apps tested for some form
of A01

avg incidence 3.81%

400

CWEs analysed

vs 30 in 2017

8+2

categories from data +
survey

3

new categories in 2021

A04, A08, A10

2003 First OWASP Top 10 released

2010 Risk-rating methodology introduced (likelihood x impact)

2013 A9 Using Components with Known Vulnerabilities added

2017 Data call from 40+ organisations; XXE and Insecure Deserialization added

2021 Root-cause categories; A04 Insecure Design, A08 Integrity Failures, A10 SSRF added

2025 Release candidate (Nov 2025): Software Supply Chain Failures and Mishandling of Exceptional Conditions added, SSRF merged into A01

A01-A10 (2021): what each category checks

A01 Broken Access Control

Users act outside their permissions: IDOR (change an id, see another user's data), path traversal, forced browsing to admin URLs, privilege escalation, permissive CORS, JWT (JSON Web Token) tampering. CWEs: 22 Path Traversal, 200 Exposure of Sensitive Information, 284 Improper Access Control, 352 CSRF, 639 Authorization Bypass Through User-Controlled Key. 34 CWEs.

A03 Injection

Untrusted input reaches an interpreter as code: SQL, NoSQL, OS command, LDAP, expression languages (EL/OGNL), SSTI (server-side template injection). XSS (cross-site scripting) now lives here. CWEs: 79 XSS, 89 SQL Injection, 73 External Control of File Name or Path, 78 OS Command Injection. 33 CWEs.

A05 Security Misconfiguration

Default accounts and settings, open cloud buckets, verbose error pages, unneeded features enabled, missing security headers. XXE merged here (the fix is a parser setting). CWEs: 16 Configuration, 611 Improper Restriction of XML External Entity Reference. 20 CWEs.

A07 Identification and Authentication Failures

Was Broken Authentication. Credential stuffing, default passwords, weak recovery, missing MFA (multi-factor authentication), session fixation, session ids in URLs. CWEs: 287 Improper Authentication, 297 Improper Certificate Validation with Host Mismatch, 384 Session Fixation. 22 CWEs.

A02 Cryptographic Failures

Was Sensitive Data Exposure (renamed after the root cause). Cleartext transport, weak or home-made algorithms, hard-coded keys, weak randomness, missing HSTS (HTTP Strict Transport Security). CWEs: 259 Use of Hard-coded Password, 327 Broken or Risky Crypto Algorithm, 331 Insufficient Entropy. 29 CWEs.

A04 Insecure Design

New (survey). The control was never designed, so perfect code cannot fix it: no threat model, no abuse cases, unlimited retries, trust-boundary violations. CWEs: 209 Error Message with Sensitive Info, 256 Plaintext Storage of Password, 501 Trust Boundary Violation, 522 Insufficiently Protected Credentials. 40 CWEs.

A06 Vulnerable and Outdated Components

Unsupported or unpatched libraries, frameworks, runtimes, OS; no inventory, no SCA (software composition analysis). Highest average incidence (8.77%). CWEs: 1104 Use of Unmaintained Third-Party Components, plus 937 and 1035 (the 2013 and 2017 'known vulnerable components' categories). 3 CWEs.

A08 Software and Data Integrity Failures

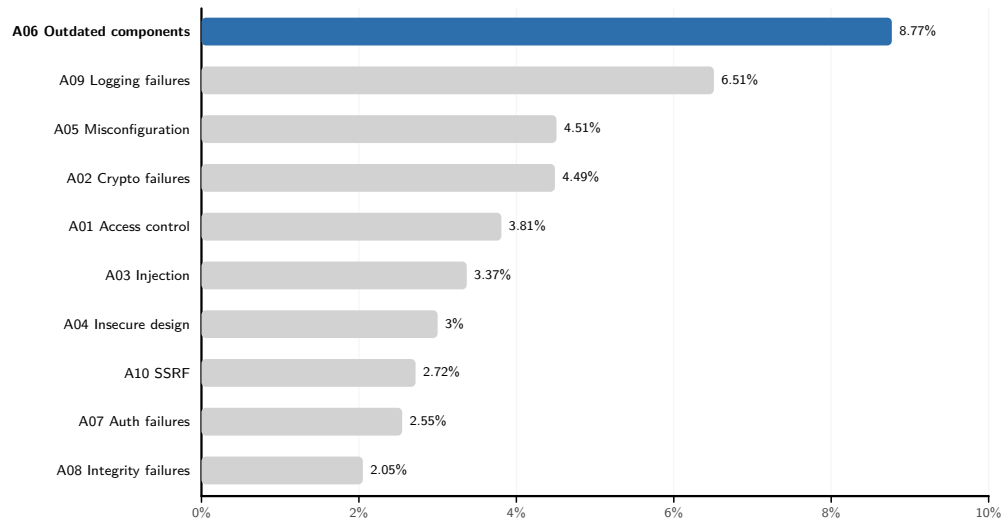
New. Code or data trusted without an integrity check: insecure CI/CD (continuous integration / delivery) pipelines, unsigned auto-updates, insecure deserialization (former A8:2017). CWEs: 502 Deserialization of Untrusted Data, 494 Download of Code Without Integrity Check, 829 Inclusion of Functionality from Untrusted Control Sphere. 10 CWEs.

A09 Security Logging and Monitoring Failures

Logins, failures and high-value transactions not logged; logs not monitored; no alerting; logs only local. CWEs: 778 Insufficient Logging, 117 Improper Output Neutralization for Logs (log injection), 223 Omission of Security-relevant Information, 532 Sensitive Information in Log File. 4 CWEs.

A10 Server-Side Request Forgery

New (survey). The server fetches a user-supplied URL without validation and reaches cloud metadata (169.254.169.254 holds temporary cloud credentials), internal services or localhost admin ports. CVE-918 SSRF. 1 CWE.

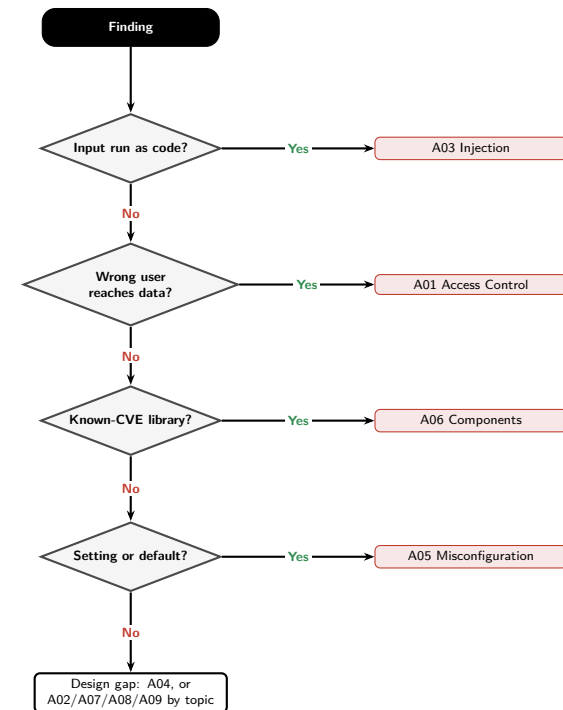


Average incidence rate per category (share of tested apps affected), OWASP Top 10:2021 data

EXAM TIP

Ranking weighs **incidence** (share of apps affected) with exploit and impact scores, not raw finding count. A01 Broken Access Control is #1 because 94% of apps were tested for it and its max incidence is 55.97%; A06 Outdated Components has the highest **average** incidence but only 3 CWEs. Exams (CEH, Security+ SY0-701 objective 2.3, PenTest+) ask you to classify an attack: IDOR is A01 Access Control, not A03 Injection; XXE is A05 Misconfiguration; deserialization is A08 Integrity Failures.

© Classify a finding



Which 2021 category does a finding belong to? (first match wins)

🔍 Detection coverage: category x method

Category				
	SAST	DAST	SCA	Manual / review
A01 Access control	low	medium	low	critical
A02 Crypto	high	medium	low	high
A03 Injection	critical	critical	low	medium
A04 Design	low	low	low	critical
A05 Misconfig	medium	critical	low	high
A06 Components	low	medium	critical	low
A07 Auth	medium	high	low	high
A08 Integrity	medium	low	high	high
A09 Logging	medium	low	low	critical
A10 SSRF	high	medium	low	high

Detection method →

Darker = the method finds most issues of that category. SAST = static analysis of source code; DAST = dynamic testing of the running app; SCA = inventory of third-party components against CVE feeds. A01, A04, A09 need humans; A06 needs SCA and an SBOM.

Tool	What it covers
<code>semgrep --config p/owasp-top-ten .</code>	SAST rule pack mapped to the Top 10 (crypto, injection, deserialization patterns)
<code>zap-baseline.py -t https://app.example -r report.html</code>	DAST passive baseline on your own app: headers, cookies, information disclosure (A05)
<code>dependency-check --project app --scan ./ -f HTML</code>	OWASP SCA: known CVEs in dependencies (A06)
<code>trivy fs --scanners vuln,secret,misconfig .</code>	Vulnerable packages, leaked secrets, IaC (infrastructure as code) misconfig (A02, A05, A06)
<code>gitleaks detect --source . --report-path leaks.json</code>	Hard-coded keys and tokens in git history (A02)

Automated scanning vs manual review

✓ Scanners (SAST, DAST, SCA)	✗ What only humans catch
+ Cheap to run on every commit or build	– Business-logic abuse (negative quantity, skipped workflow step)
+ Good at pattern flaws: SQLi, XSS, known CVEs, missing headers	– Authorization per object and per role (A01)
+ Repeatable evidence for audits	– Missing controls (A04) and missing logs (A09): nothing to match

🕒 From 2017 to 2021 to 2025

2017	2021	2025 (RC1, Nov 2025)
A1 Injection	A03 Injection (XSS merged)	A05 Injection
A2 Broken Authentication	A07 Identification & Auth Failures	A07 Authentication Failures
A3 Sensitive Data Exposure	A02 Cryptographic Failures	A04 Cryptographic Failures
A4 XXE	merged into A05 Misconfiguration	A02 Security Misconfiguration
A5 Broken Access Control	A01 Broken Access Control	A01 Broken Access Control (+ SSRF)
A6 Security Misconfiguration	A05 Security Misconfiguration	A02 (rose to #2)
A7 XSS	merged into A03 Injection	in A05 Injection
A8 Insecure Deserialization	A08 Software & Data Integrity Failures	A08 Software or Data Integrity Failures
A9 Components with Known Vulns	A06 Vulnerable & Outdated Components	A03 Software Supply Chain Failures (broadened)
A10 Insufficient Logging	A09 Logging & Monitoring Failures	A09 Logging & Alerting Failures
-	A04 Insecure Design (new)	A06 Insecure Design
-	A10 SSRF (new)	A10 Mishandling of Exceptional Conditions (new)

What moved in the 2025 release candidate

- **A03:2025 Software Supply Chain Failures** new scope: not only outdated libraries but compromised build systems, registries, maintainers and CI/CD (xz-utils style)
- **A10:2025 Mishandling of Exceptional Conditions** new: errors and edge cases that leave the app fail-open or leak data
- **A02:2025 Security Misconfiguration** rose from #5 to #2: more software is configuration-driven (cloud, containers)
- **SSRF** no longer its own entry; folded into A01 Broken Access Control
- **A09:2025 Logging & Alerting Failures** renamed to stress that logs without alerts do not help

Requirement	ASVS 5.0 chapter	WSTG 4.2 category	CWE Top 25 (2024 rank)
A01 Broken Access Control incl. SSRF since 2025	V8 Authorization; V1 (SSRF)	ATHZ; INPV-19 SSRF	22 (#5), 352 (#4), 862 (#9), 863 (#18), 918 (#19)
A02 Security Misconfiguration	V13 Configuration; V3 headers, cookies	CONF	none ranked
A03 Software Supply Chain Failures	V15.2 dependencies; V13	INFO-08/09 fingerprinting only	none ranked
A04 Cryptographic Failures	V11 Cryptography; V12 TLS	CRYP	none ranked
A05 Injection XSS included	V1 Encoding; V2 Validation	INPV; CLNT (DOM XSS)	79 (#1), 89 (#3), 78 (#7), 94 (#11), 77 (#13)
A06 Insecure Design	V2.3 business logic; V15	BUSL	434 upload (#10)
A07 Authentication Failures	V6, V7, V9, V10	ATHN; SESS; IDNT	287 (#14), 798 (#22), 306 (#25)
A08 Software or Data Integrity Failures	V1.5 deserialization; V3.6 SRI	no dedicated test	502 (#16)

A09 Logging and Alerting Fail-ures	V16.2-16.4 logging	no dedicated test	none ranked
A10 Mishandling of Exceptional Conditions	V16.5 error handling; V15.3	ERRH	476 (#21), 400 (#24)
Top 10:2025 (RC1) crosswalk: where each risk becomes a requirement (ASVS), a test (WSTG ids: ATHZ authorization, CONF configuration, INFO information gathering, CRYPT cryptography, INPV input validation, BUSL business logic, ATHN authentication, IDNT identity management, SESS session, CLNT client-side, ERRH error handling) and which ranked CVEs it holds. The injection-heavy Top 25 barely touches design and supply-chain risks: a scanner tuned to the Top 25 misses half of the Top 10 (chapter-level mapping by the author; CWE ranks from MITRE 2024 list, verify the 2025 edition)			

⚠️ WARNING

The 2025 list was announced as a release candidate in November 2025; the final OWASP Top 10:2025 on top10.owasp.org keeps the same numbering (A01 Broken Access Control to A10 Mishandling of Exceptional Conditions). 2021 ids remain the ones most exams and many tools still reference, so state the edition when quoting an id in a report.

⚙️ Mishandling of Exceptional Conditions

A10:2025. Errors, unexpected states and failed checks that leave the application insecure: fail-open logic (a failed authorization call treated as allowed), swallowed exceptions, missing time-outs, stack traces shown to users. Root-cause category, like Insecure Design.

❌ Classic mistake: fail-open exception handling (A10:2025)

try: allowed = authz.check(user, doc) except Exception: allowed = True (or an empty catch that falls through to the handler). When the policy service times out under load, every request is authorised: an outage of one dependency becomes a full access-control bypass, and nothing is logged. Fix: default allowed = False, fail closed on every exception, log the failure with context (A09) and alert on the error rate.

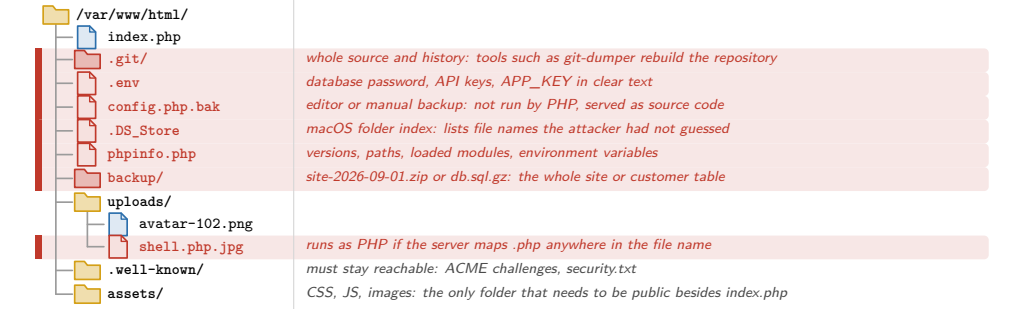
✓ Mitigation by category

Risk	Primary controls
A01 Access control	Deny by default; one central server-side authorization mechanism; ownership check on every object (WHERE owner_id = :user); log access failures; rate-limit APIs; short-lived JWT
A02 Crypto	Classify data; TLS 1.2+ and HSTS; AES-GCM for data at rest; Argon2id or bcrypt for passwords; keys in KMS/HSM (key management service / hardware security module); CSPRNG (cryptographically secure random); no MD5/SHA-1/RC4/DDES; plan post-quantum (FIPS 203 ML-KEM)
A03 Injection	Prepared statements or ORM (object-relational mapper); allow-list validation; context-aware output encoding; CSP (Content Security Policy); LIMIT on queries; never eval user data
A04 Design	Threat model (STRIDE: spoofing, tampering, repudiation, info disclosure, DoS, elevation) per feature; abuse cases; paved-road (pre-approved) architectures; business limits (retries, spend); tenant segregation
A05 Misconfig	Repeatable hardening; minimal platform; no samples or default accounts; security headers; XXE disabled in parsers; config scanning in CI

A06 Components	SBOM (software bill of materials, CycloneDX or SPDX format); SCA in CI; remove unused deps; advisory feeds; patch SLA by severity
A07 Auth	MFA; breached-password check; no default credentials; generic failure messages; new session id on login; throttling
A08 Integrity	Signed artefacts (Sigstore/cosign); SRI (Subresource Integrity hash on CDN scripts); lockfiles with hashes; reviewed pipeline config; no deserialisation of untrusted input
A09 Logging	Log logins, auth and access-control failures with user context; central SIEM; alerting; tamper-evident retention; IR (incident response) run-books
A10 SSRF	Destination allow-list; block RFC 1918 private, link-local and loopback ranges after DNS resolution; no redirects followed; egress segmentation; IMDSv2 (token-protected cloud metadata)

```
nginx security headers (A05)
add_header Strict-Transport-Security
    "max-age=63072000; includeSubDomains; preload" always;
add_header Content-Security-Policy
    "default-src 'self'; frame-ancestors 'none'" always;
add_header X-Content-Type-Options "nosniff" always;
add_header Referrer-Policy "strict-origin-when-cross-origin" always;
add_header Permissions-Policy "camera=(), geolocation=()" always;
server_tokens off;
```

HSTS max-age=63072000	2 years: browsers refuse plain HTTP to the site for that long
frame-ancestors 'none'	no site may frame the page: blocks clickjacking
nosniff	browser must not guess a file's type: stops an upload being run as script
server_tokens off	hides the nginx version from error pages and headers

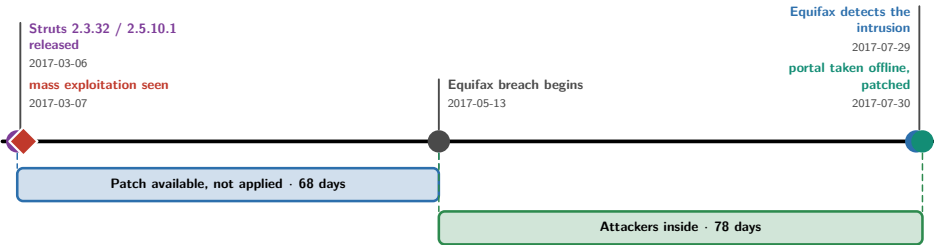


A05 in one picture: a web root where deployment left files that the server hands out as plain downloads (highlighted). Scanners and bots request these paths within hours of a site going live. Fix at deploy time (build an artefact that holds only public files, keep secrets outside the web root) and at the server: nginx location ~ /\.(!well-known) { deny all; } blocks dot-files but keeps /.well-known/ (ACME automatic-certificate challenges, security.txt); refuse to execute anything under uploads/. .env = environment file holding secrets; phpinfo() = PHP diagnostics page

```
Parameterised query (A03, Python)
# vulnerable: string concatenation
cur.execute(f"SELECT * FROM users WHERE id = {user_id}")
# safe: driver-level parameters
cur.execute("SELECT * FROM users WHERE id = %s", (user_id,))
# safe: ORM
User.objects.get(pk=user_id)
```

```
invoices.py Python (Flask)
1 # A01 VULNERABLE: any caller reads any invoice by changing the id
2 @app.get("/invoices/<int:inv_id>")
3 def get_invoice(inv_id):
4     return Invoice.query.get_or_404(inv_id).to_json()
5
6 # A01 FIXED: authenticated, and the owner is part of the query
7 @app.get("/invoices/<int:inv_id>")
8 @login_required
9 def get_invoice(inv_id):
10     inv = Invoice.query.filter_by(id=inv_id,
11                                owner_id=current_user.id).first_or_404()
12     return inv.to_json()
```

A01 Broken Access Control, vulnerable vs fixed: the fix puts the ownership condition in the database query itself, so no code path can return another user's row, and answers 404 rather than 403 so ids of other users' invoices cannot be confirmed. Test it with two accounts: user A requesting user B's id must fail.

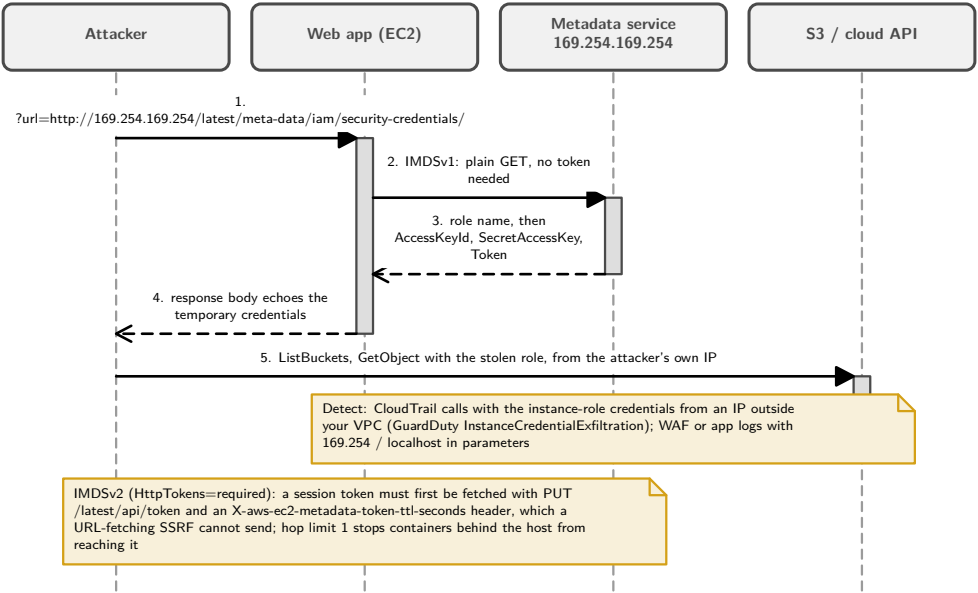


A06 (vulnerable and outdated components) in one picture: Apache Struts CVE-2017-5638 was fixed and exploited in the wild within days of disclosure in March 2017, yet Equifax's portal stayed unpatched and attackers were inside from May to late July (about 147 million people affected)

⚠ Practitioner notes and traps

📄 EXAMPLE

Log4Shell (CVE-2021-44228, Dec 2021) is both A06 (vulnerable component) and A03 (injection: a logged string triggered a JNDI remote lookup). **Capital One (2019)** was A10 SSRF: a misconfigured WAF fetched the EC2 metadata service, the stolen role credentials then read S3 buckets. **SolarWinds (2020)** and **xz-utils (2024)** are A08 integrity failures in the build and supply chain.



SSRF to cloud credentials (A01:2025, the Capital One 2019 pattern) and why IMDSv2 breaks it; detection points in the notes

Category	What the tester checks	Evidence of the flaw
A01 IDOR	Request another user's object id with your own session	Data of user B returned to user A
A01 CORS	Send a foreign Origin header	Origin reflected with Allow-Credentials: true
A03 SQLi	Add a quote, then a true/false or delay condition	DB error, or responses/timing differ
A03 SSTI	Submit a template arithmetic expression like {{7*7}}	Page shows 49: input is evaluated
A05 XXE	XML body declaring an external entity to a host you control	Parser contacts that host (out-of-band)
A07	Log in with an unknown vs known user	Different error text or timing: user enumeration
A10 SSRF	Point a URL parameter at a collaborator host	Inbound request from the app server

Q1 Which 2021 category absorbed XSS (cross-site scripting)?

A A03 Injection

Q2 Which 2021 category absorbed XXE (XML external entities)?

A A05 Security Misconfiguration: the fix is disabling external entities in the parser

Q3 Two 2021 categories added from the community survey rather than data?

A A04 Insecure Design and A10 Server-Side Request Forgery

Q4 Where does insecure deserialization sit in 2021?

A A08 Software and Data Integrity Failures

Q5 CWE number for SSRF?

A CWE-918 Server-Side Request Forgery

Q6 Which 2021 category has the most mapped CWEs?

A A04 Insecure Design (40)

Q7 What changed for SSRF in the 2025 release candidate?

A Merged into A01 Broken Access Control