

API SECURITY

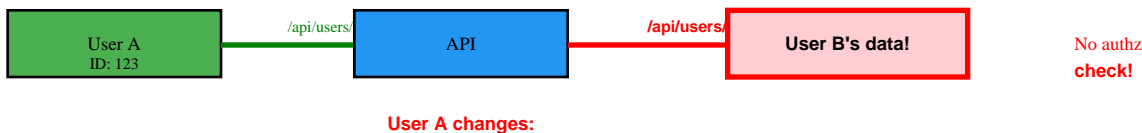
REST & GraphQL Attacks

What is API Security?

APIs (Application Programming Interfaces) power modern applications. REST and GraphQL APIs expose data and functionality to clients. Without proper security, APIs leak data, allow unauthorized access, enable mass enumeration, and can be abused for RCE. API security is critical as APIs become the backbone of web, mobile, and microservices architectures.

1. Broken Object Level Authorization (BOLA)

Description: Most common API vulnerability. API endpoints accept object IDs without authorization checks. User A accesses User B's data by changing ID in request. Same as IDOR but API-focused. Found in REST, GraphQL, and all API types.



Example Attack:

```
GET /api/users/123/profile → Change to /api/users/456/profile (unauthorized access)
```

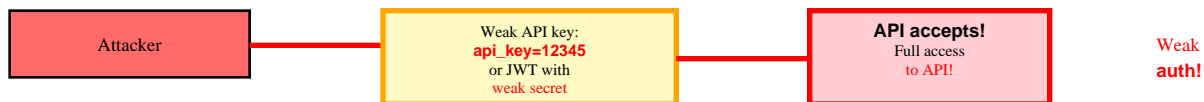
Attack Techniques:

```
ID enumeration | Sequential IDs | GUID manipulation | Parameter tampering
```

Impact: Unauthorized data access, privacy violation, mass data harvesting

2. Broken Authentication

Description: Weak or missing authentication in API. No API keys required. Predictable tokens. JWT with weak secrets. Missing expiration. No refresh token rotation. Credential stuffing at scale via API.



Example Attack:

API accepts any JWT token | Weak API key: `api_key=12345` | No rate limit on `/login`

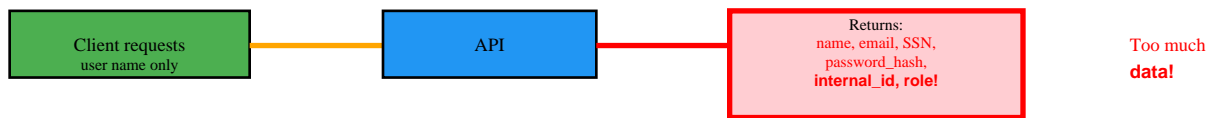
Attack Techniques:

Token theft | Weak API keys | JWT manipulation | Brute force | Token replay

Impact: Account takeover, unauthorized API access, impersonation

3. Excessive Data Exposure

Description: API returns more data than needed. Full user objects when only name needed. Sensitive fields in responses (SSN, passwords hashes, tokens). Reliance on client-side filtering. Generic `to_json()` functions expose everything.



Example Attack:

```
GET /api/users/me returns: {user, email, ssn, password_hash, internal_id, role...}
```

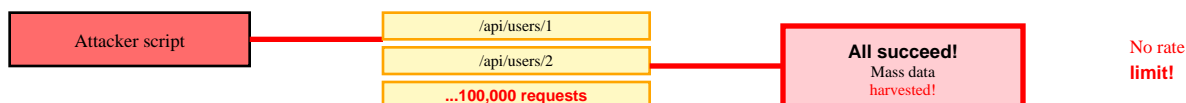
Attack Techniques:

Inspect API responses | Remove client filters | Access raw endpoints

Impact: Information disclosure, sensitive data leakage, privacy violation

4. Lack of Resources & Rate Limiting

Description: No rate limiting allows abuse. Mass data scraping via API. Brute force attacks at scale. DoS by exhausting resources. Enumeration of all objects. Automated abuse with no throttling.



Example Attack:

```
Enumerate all users: /api/users/1, /api/users/2... /api/users/1000000 (no limit)
```

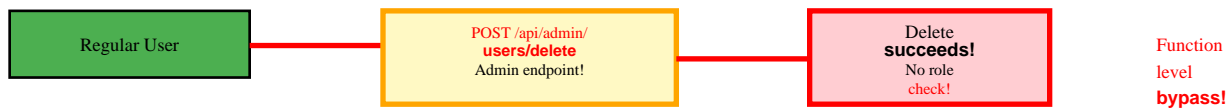
Attack Techniques:

Mass enumeration | Brute force | Resource exhaustion | Scraping automation

Impact: Data scraping, DoS, brute force success, competitive intelligence

5. Broken Function Level Authorization

Description: Regular users can access admin endpoints. No role/permission checks on API functions. Modify HTTP method (GET to PUT/DELETE). Access debug/internal endpoints. Administrative functions exposed.



Example Attack:

```
Regular user: POST /api/admin/users/delete (succeeds!) or GET /api/admin/logs
```

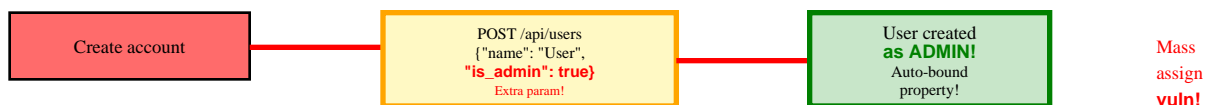
Attack Techniques:

```
Admin endpoint enumeration | HTTP method manipulation | Function access testing
```

Impact: Privilege escalation, administrative access, system compromise

6. Mass Assignment

Description: API automatically binds request parameters to object properties. Attacker adds extra parameters (is_admin=true, role=admin). No parameter whitelist. ORM/framework auto-assigns all fields.



Example Attack:

```
POST /api/users {name: 'User', is_admin: true} → User created as admin!
```

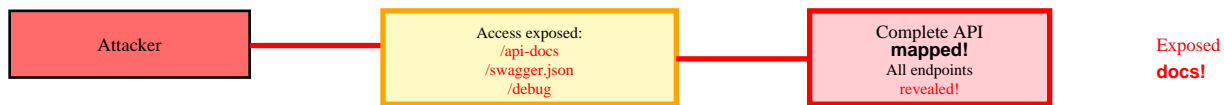
Attack Techniques:

```
Add hidden parameters | Role manipulation | Property injection
```

Impact: Privilege escalation, unauthorized modification, account takeover

7. Security Misconfiguration

Description: Exposed API documentation. Debug mode enabled in production. Verbose error messages reveal structure. CORS misconfiguration allows any origin. HTTP instead of HTTPS. Stack traces in responses.



Example Attack:

```
Access /api/docs, /swagger.json, /api-docs exposes all endpoints and parameters
```

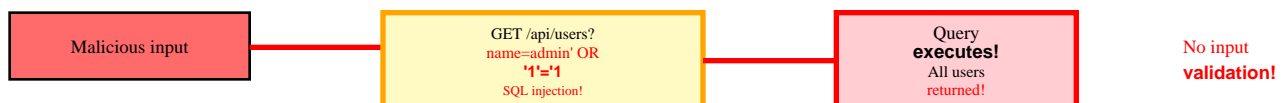
Attack Techniques:

```
Documentation discovery | Debug endpoint access | Error message analysis
```

Impact: Information disclosure, attack surface expansion, credential exposure

8. Injection (SQL, NoSQL, Command)

Description: API vulnerable to injection attacks. SQL injection in API parameters. NoSQL injection in MongoDB queries. Command injection in API that executes system commands. GraphQL injection.



Example Attack:

```
GET /api/users?name=admin' OR '1'='1 | {"$where": "this.credits == 1000"}
```

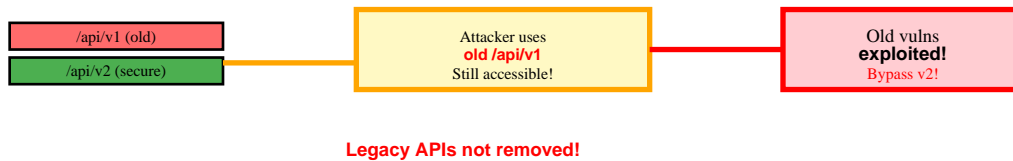
Attack Techniques:

```
SQLi payloads | NoSQL operators | Command injection | GraphQL injection
```

Impact: Database compromise, RCE, data extraction, authentication bypass

9. Improper Assets Management

Description: Old API versions still accessible. /api/v1 has vulnerabilities, /api/v2 doesn't. Deprecated endpoints not removed. Undocumented API paths. Beta/test APIs in production. Shadow APIs.



Example Attack:

```
/api/v1/users (vulnerable) still works while /api/v2/users is secure
```

Attack Techniques:

```
Version enumeration | Old endpoint testing | Documentation analysis
```

Impact: Access via outdated APIs, bypass modern security controls

10. Insufficient Logging & Monitoring

Description: No logging of API access. Failed authentication attempts not tracked. No anomaly detection. Attacks go unnoticed. No audit trail for sensitive operations. Enables long-term compromise.



Example Attack:

```
Attacker enumerates 100K users via API, no alerts or detection
```

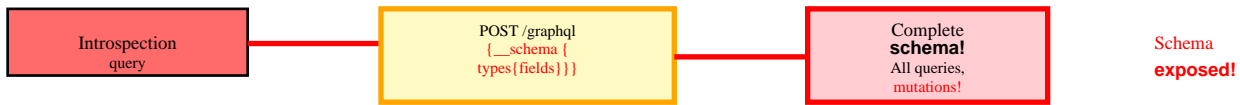
Attack Techniques:

```
Mass automated attacks | Slow credential stuffing | Long-term data theft
```

Impact: Undetected breaches, prolonged compromise, no incident response

11. GraphQL Specific: Introspection Enabled

Description: GraphQL introspection reveals entire schema. All types, fields, mutations, queries exposed. Attacker maps complete API surface. Should be disabled in production. Provides attack blueprint.



Example Attack:

```
POST /graphql {query: __schema {types {name fields {name}}}} reveals everything
```

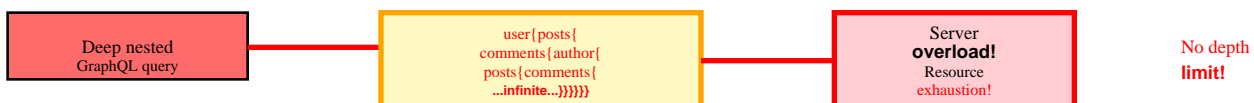
Attack Techniques:

Introspection queries | Schema enumeration | Complete API mapping

Impact: Complete API disclosure, attack surface mapping, information leakage

12. GraphQL Specific: Query Depth/Complexity

Description: No query depth limits allows deeply nested queries. Circular references cause infinite loops. Aliasing enables query multiplication. Resource exhaustion via complex queries. DoS through query complexity.



Example Attack:

```
query {user {posts {comments {author {posts {comments...}}}}} (infinite depth)
```

Attack Techniques:

Deeply nested queries | Circular references | Query batching | Aliasing abuse

Impact: Denial of Service, resource exhaustion, API unavailability

REST API TESTING GUIDE

Endpoint Discovery

- Check common documentation paths: /api-docs, /swagger, /swagger.json, /api/swagger
- Try versioned endpoints: /api/v1, /api/v2, /v1, /v2
- Look for exposed .wadl or OpenAPI specs
- Test common resource names: /api/users, /api/admin, /api/products
- Use Burp Suite to spider API endpoints
- Check JavaScript files for API URLs

Authentication Testing

- Test endpoints without authentication (401/403 vs 200)
- Try accessing with expired/invalid tokens
- Test JWT manipulation (algorithm confusion, weak secrets)
- Check for API keys in headers, query params, cookies
- Test rate limiting on authentication endpoints
- Verify proper session invalidation on logout

Authorization Testing (BOLA)

- Change user IDs in all endpoints (most critical test!)
- Test with two different user accounts
- Try accessing admin endpoints as regular user
- Enumerate resources by ID (1, 2, 3, 4...)
- Test all HTTP methods (GET, POST, PUT, PATCH, DELETE)
- Check authorization on each CRUD operation

HTTP Methods Testing

- OPTIONS request reveals allowed methods
- Try PUT/PATCH on read-only resources
- Test DELETE on protected resources
- HEAD requests might bypass some controls
- Check if GET accepts body (unusual but possible)
- Test HTTP method override headers (X-HTTP-Method-Override)

Input Validation

- Test all standard injection payloads (SQLi, XSS, command)
- Try negative values, zero, null, extremely large numbers
- Test with different content types (JSON, XML, form-data)
- Mass assignment: add unexpected parameters
- Test parameter pollution (id=1&id=2)
- Special characters and encoding bypasses

Response Analysis

- Check for excessive data in responses
- Look for sensitive data (tokens, passwords, internal IDs)
- Analyze error messages for information disclosure
- Compare authorized vs unauthorized responses
- Check response headers (CORS, security headers)
- Test if client-side filtering is enforced server-side

GRAPHQL TESTING GUIDE

GraphQL Discovery

- Common endpoints: /graphql, /graphql/console, /graphiql, /api/graphql
- Send POST request with introspection query
- Look for GraphQL Playground or GraphiQL interface
- Check for GraphQL in JavaScript files
- Try both POST and GET methods

Introspection Queries

- Full schema: `{__schema {types {name fields {name args {name}}}}}`
- List all queries: `{__schema {queryType {fields {name}}}}`
- List all mutations: `{__schema {mutationType {fields {name}}}}`
- Enumerate types and relationships
- Map entire API structure if introspection enabled

Authorization Testing

- Test direct object access: query `{user(id: 456) {email ssn}}`
- Change IDs in all queries and mutations
- Try accessing admin-only queries/mutations
- Test nested authorization (user -> posts -> author)
- Verify field-level authorization

DoS via Query Complexity

- Deeply nested queries: `user{posts{comments{user{posts...}}}}`
- Circular references for infinite loops
- Query batching: send array of 1000 queries
- Aliasing: `user1:user(id:1){name} user2:user(id:2){name}... x1000`
- Test query depth/complexity limits

Injection Testing

- SQL injection in GraphQL arguments: `user(id: "1' OR '1'='1")`
- NoSQL injection in filters
- Test all string arguments for injection
- Command injection in mutations

- XSS in fields that render in UI

Mutation Testing

- Test all mutations without authentication
- Try modifying other users' data
- Mass assignment in mutations (add extra fields)
- Test destructive operations (delete, update)
- Verify proper authorization on all mutations

COMMON API ENDPOINTS TO TEST

Endpoint Pattern	Purpose	Test For
/api/users/{id}	User profiles	BOLA, data exposure
/api/users/me	Current user	Info disclosure
/api/admin/*	Admin functions	Authorization bypass
/api/v1/* vs /api/v2/*	Versioning	Old vulnerabilities
/api/debug/*	Debug endpoints	Info disclosure
/api/internal/*	Internal APIs	Unauthorized access
/api/login	Authentication	Brute force, injection
/api/password-reset	Password reset	Enumeration, abuse
/api/orders/{id}	Orders	BOLA, price manipulation
/api/payments/*	Payments	Authorization, tampering
/api/upload	File upload	Malicious uploads
/api/export	Data export	Mass data extraction

GRAPHQL INTROSPECTION QUERY

Full Schema Discovery:

```
POST /graphql Content-Type: application/json { "query": "{ __schema { types { name fields { name args { name type { name kind } } } } queryType { name } mutationType { name } } }" }
```

API SECURITY PREVENTION

- **Implement Object-Level Authorization:** Verify user has permission to access specific object. Check authorization on every API endpoint. Never trust object IDs from client. Implement ACLs or RBAC. Most critical API security control.
- **Strong Authentication:** Use OAuth 2.0 or JWT properly. Strong API key generation (cryptographically random). Proper token expiration and rotation. MFA for sensitive operations. Rate limit authentication endpoints.
- **Function-Level Authorization:** Role/permission checks on all endpoints. Regular users cannot access admin functions. Verify authorization for each HTTP method. Separate admin and user APIs if possible.
- **Input Validation:** Validate all input server-side. Use parameterized queries (prevent injection). Whitelist allowed parameters (prevent mass assignment). Type checking and bounds validation. Reject unexpected input.
- **Rate Limiting:** Implement rate limits on all endpoints. Per-user and per-IP limits. Stricter limits on expensive operations. Exponential backoff on failed auth. Prevent brute force and scraping.

- **Minimal Data Exposure:** Return only necessary data fields. Don't expose internal IDs, hashes, tokens. Filter sensitive data server-side. Use DTOs/serializers for output. Never rely on client filtering.
- **API Versioning:** Maintain API versioning properly. Deprecate old versions securely. Remove vulnerable old versions. Document which versions are supported. Monitor usage of old APIs.
- **Disable Debug in Production:** No verbose error messages. Disable stack traces in responses. Remove debug endpoints. Disable GraphQL introspection. No exposed documentation in production.
- **Logging and Monitoring:** Log all API access with user context. Alert on anomalies (mass access, failed auth). Monitor for enumeration attempts. Track sensitive operations. SIEM integration.
- **GraphQL Specific:** Disable introspection in production. Implement query depth/complexity limits. Prevent circular queries. Limit query batching. Implement timeout on long queries. Field-level authorization.

API TESTING TOOLS

Tool	Purpose	Best For
Burp Suite	Proxy, scanner, intruder	REST API testing, BOLA
Postman	API testing platform	Manual testing, collections
Insomnia	REST/GraphQL client	Quick API exploration
GraphQL Voyager	Schema visualization	GraphQL mapping
InQL Scanner (Burp)	GraphQL testing	Introspection, injection
Arjun	Parameter discovery	Hidden parameter finding
ffuf	Fuzzing	Endpoint enumeration
Nuclei	Automated scanner	Known API vulnerabilities
curl/httpie	Command line	Quick testing

OWASP API SECURITY TOP 10 (2023)

- API1:2023 - Broken Object Level Authorization (BOLA/IDOR)
- API2:2023 - Broken Authentication
- API3:2023 - Broken Object Property Level Authorization
- API4:2023 - Unrestricted Resource Consumption
- API5:2023 - Broken Function Level Authorization (BFLA)
- API6:2023 - Unrestricted Access to Sensitive Business Flows
- API7:2023 - Server Side Request Forgery (SSRF)
- API8:2023 - Security Misconfiguration
- API9:2023 - Improper Inventory Management
- API10:2023 - Unsafe Consumption of APIs

Note: This cheat sheet is for educational and authorized security testing only. Unauthorized API access and data scraping are illegal. Always obtain written permission before testing.