Perfect 👍 Here are **precise WSTG-style notes for “API Testing” (WSTG-APIT)**.

**📝 API Testing**

**🎯 Purpose**

To assess the **security of web APIs (REST, GraphQL, SOAP, gRPC, etc.)**, which often expose critical business logic and sensitive data.  
APIs are attractive to attackers because they:

* Expose structured endpoints.
* Often bypass web UI protections.
* Are consumed by mobile, IoT, SPA, and web apps.

**⚡ Common Risks (OWASP API Top 10)**

1. **Broken Object Level Authorization (BOLA)** → Accessing others’ data (/api/user/123).
2. **Broken Authentication** → Missing/weak auth on endpoints.
3. **Excessive Data Exposure** → API leaks unnecessary fields.
4. **Lack of Rate Limiting** → DoS or brute-force attacks.
5. **Mass Assignment** → Updating unauthorized fields (e.g., role=admin).
6. **Improper Assets Management** → Old/unused API versions left open.
7. **Injection Attacks** → SQLi, NoSQLi, Command injection in API input.

**🔍 How to Test**

**1. Discovery**

* Enumerate endpoints:
  + Inspect JS files, Swagger/OpenAPI specs, Burp site map.
  + Check /api/, /v1/, /graphql.

**2. Authentication**

* Confirm tokens (JWT, OAuth2, API keys) are required.
* Test expired/invalid tokens.
* Replay tokens from another user.

**3. Authorization**

* Try **IDOR attacks**:
* GET /api/user/100
* GET /api/user/101
  + Check if you can access others’ data.

**4. Input Validation**

* Fuzz with unexpected/malicious input:
  + SQL/NoSQL injection.
  + Command injection.
  + XSS in JSON/XML payloads.

**5. Excessive Data Exposure**

* Check API responses for sensitive fields (SSNs, credit cards, internal IDs).

**6. Rate Limiting & DoS**

* Flood endpoint with requests.
* Check if brute-force login is prevented.

**7. Versioning & Error Handling**

* Look for /v1/, /v2/, deprecated APIs.
* Analyze verbose error messages.

**8. GraphQL-specific**

* Test with \_\_schema introspection query.
* Abuse query depth ({a{b{c{d...}}}}) → DoS.
* Try field injection.

**🛡️ Mitigation**

* Enforce **strong authentication & authorization**.
* Implement **least privilege access** (RBAC).
* Validate and sanitize **all inputs** server-side.
* Apply **rate limiting & quotas**.
* Avoid exposing unnecessary fields (use whitelisting).
* Disable verbose errors in production.
* Maintain strict **API documentation & version control**.
* Apply **logging & monitoring** for anomalies.

✅ **Key Takeaway:**  
APIs often **bypass traditional UI controls** → making them prime attack targets. Testing should cover **auth, access control, input validation, data exposure, and rate limiting**.

👉 Do you want me to also make a **ready-to-use Postman/Burp Suite API testing checklist** (step-by-step actions you can reuse during engagements)?