Got it 👍 — let’s cover **Testing for Weak Password Policy** from a **WSTG (Web Security Testing Guide)** perspective.

**🧾 Testing for Weak Password Policy**

**1. Why It Matters**

A **weak password policy** makes it easier for attackers to compromise accounts via:

* **Brute force attacks** (trying many combinations)
* **Credential stuffing** (using leaked passwords from other sites)
* **Dictionary attacks** (guessing common passwords like admin123)

If the application **doesn’t enforce strong password rules**, users may choose insecure passwords, leading to account takeover.

**2. What to Test**

**🔹 Password Strength Requirements**

Check if the application enforces:

* Minimum length (e.g., ≥ 8–12 characters).
* Complexity (upper, lower, numbers, symbols).
* Prevents dictionary words (e.g., password, qwerty).
* Prevents reused passwords.

⚠️ Vulnerable case: Application allows 12345 or password as a valid password.

**🔹 Client-Side vs Server-Side Validation**

* **Inspect with Burp Suite**: Modify password in the registration or change-password request.
* If client-side JS enforces strong password, but server **accepts weak one when bypassed** → policy is broken.

**🔹 Password Reuse**

* Try setting the same password after change.
* If allowed → attacker with leaked creds can re-exploit easily.

**🔹 Common Patterns & Default Passwords**

* Test if app allows passwords like:
  + admin, welcome123, changeme, P@ssw0rd.
* Check if default credentials remain active after setup.

**3. Tools & Methods**

**🛠 Burp Suite**

* Intercept **signup** or **password change** requests.
* Modify password to 123456 or password1.
* If server accepts → weak policy confirmed.

**🛠 Hydra / Medusa / Burp Intruder**

* Simulate brute force attack with a weak password list (only in lab!).
* Observe if app allows too many weak attempts.

**🛠 SecLists Wordlists**

* Use SecLists/Passwords/Common-Credentials to test common weak passwords.

**4. Example Test Cases**

✅ Secure

* Password: Myp@ssword2025! → Accepted.
* Password: 12345 → Rejected.

❌ Insecure

* Password: password → Accepted.
* Password: qwerty → Accepted.

**5. Mitigations**

* Enforce **minimum length ≥ 12**.
* Require **complexity** (mix of letters, numbers, symbols).
* Use **deny-lists** (block password123, qwerty).
* Enforce **password history** (no reuse of last N).
* Implement **rate limiting** + MFA to prevent brute force.

✅ **Summary**:  
Testing for weak password policy means checking if the application allows **short, simple, or common passwords**. Use **Burp Suite** to bypass client-side checks and confirm if the backend enforces strong rules. If the app accepts 12345, it’s vulnerable.

👉 Do you want me to also create a **Burp Suite step-by-step lab exercise** where you’ll actually test weak password acceptance?