Python Snippets (OWASP Top 10)

1. **Hardcoded Credentials (A07: Identification & Authentication Failures)**

```python
def login(username, password):
    if username == "admin" and password == "admin123":
        return "Welcome Admin!"
    return "Access Denied"
```

🔴 **Vulnerability:** Hardcoded credentials (`admin123`) can be leaked, reused, or brute-forced.
✅ **Fix:** Store hashed passwords in a DB + use authentication library.

---

2. **No Access Control on Route (Flask) (A01: Broken Access Control)**

```python
@app.route("/admin")
def admin_panel():
    return "Admin Dashboard"
```

🔴 **Vulnerability:** Anyone can access `/admin`. No authentication or authorization check.
✅ **Fix:** Implement RBAC or login-required decorators.

---

3. **Insecure System Call (A03: Injection)**

```python
user = input("Enter username: ")
os.system("echo " + user)
```

🔴 **Vulnerability:** Command injection (`username="; rm -rf /"`) possible.
✅ **Fix:** Use `subprocess.run(["echo", user])`.

---

4. **Reflected XSS in Flask (A03: Injection)**

```
@app.route("/echo")
def echo():
    msg = request.args.get("msg")
    return f"<h1>{msg}</h1>"
```

🔴 **Vulnerability:** Unsanitized input directly returned → Reflected XSS.
✅ **Fix:** Escape/encode output (`flask.escape`).

---

5. Insecure Deserialization (A08: Software & Data Integrity Failures)

```
import pickle

data = request.get("payload")
obj = pickle.loads(data)
```

🔴 **Vulnerability:** Malicious payload in `pickle.loads()` → remote code execution.
✅ **Fix:** Avoid pickle for untrusted input, use JSON.

---

6. Hardcoded API Key (A02: Cryptographic Failures)

```
print("API Key: 12345-ABCDE")
```

🔴 **Vulnerability:** Sensitive data exposed in source code.
✅ **Fix:** Use environment variables or secrets manager.

---

7. SQL Injection via Django ORM Extra (A03: Injection)

```
# Old Django < 2.2 vulnerable pattern
User.objects.extra(where=["username = '%s'" % user])
```

🔴 **Vulnerability:** Direct string formatting in query → SQL Injection.
✅ **Fix:** Use parameterized queries with ORM (`User.objects.filter(username=user)`).

---

8. Weak Authentication (Base64 Token) (A07: Identification & Authentication Failures)

```
token = base64.b64encode(b"user:admin")
```

🔴 **Vulnerability:** Base64 is reversible → attacker can decode easily.
✅ **Fix:** Use HMAC, JWT, or session tokens.

---

9. No Logging of Sensitive Actions (A09: Security Logging & Monitoring Failures)

```
def withdraw(amount):
    balance -= amount
    return balance
```

🔴 **Vulnerability:** No logging of financial transactions → fraud undetectable.
✅ **Fix:** Add structured logging with monitoring/alerts.

📌 Summary Table – Python Specific Vulnerabilities

#	Vulnerability	OWASP Top 10 (2021)
1	Hardcoded credentials (`admin123`)	A07: Identification & Authentication Failures
2	No access control on Flask `/admin`	A01: Broken Access Control
3	Command Injection via `os.system`	A03: Injection
4	Reflected XSS in Flask	A03: Injection
5	Insecure Deserialization (`pickle.loads`)	A08: Software & Data Integrity Failures
6	API Key hardcoded in source	A02: Cryptographic Failures
7	SQL Injection (Django `.extra()`)	A03: Injection
8	Weak Authentication using Base64	A07: Identification & Authentication Failures

#	Vulnerability	OWASP Top 10 (2021)
9	No logging of withdrawals	A09: Security Logging & Monitoring Failures