4. INSECURE DESIGN

📘 MODULE 4 — A04: INSECURE DESIGN

Web Application Penetration Testing Course — Xsploit Hackademy

---

🔷 1. What Is Insecure Design? (The True Meaning)

"Insecure Design" means **the system was built wrong from the start**, not misconfigured after deployment.

It refers to flaws **in architecture, logic, workflows, permissions, rules, and business processes**.

Unlike misconfiguration, insecure design is:

- **Hard to patch**
- **Deeply structural**
- **Exists before code is written**
- **Exists even if code is "secure"**

Example:

A shopping website allows users to change the price of an item in the frontend.

Even if inputs are sanitized, the flaw is in the **business logic** → price should never be client-controlled.

🔷 2. Why It's Dangerous

Insecure design leads to:

- Authentication bypasses
- Broken authorization
- Logic flaws
- Business workflow abuse

- Financial loss
- Data corruption
- Privilege escalation
- Unlimited actions (rate-limit bypass, resource exhaustion)

This is how real hackers (and bug bounty hunters) get **high-severity payouts**.

🔷 3. Categories of Insecure Design

OWASP lists multiple areas where design-level vulnerabilities appear:

3.1 Business Logic Flaws

Flaws created by **wrong assumptions** in workflows.

Examples:

- Cancel order but refund does not cancel the shipment
- Apply coupon unlimited times
- Cart amount becomes negative
- Payment confirmed without verifying gateway callback

3.2 Insecure Authorization Design

Not checking authorization at every request.

Examples:

- Changing `user_id` in API request reveals another user's data
- Admin endpoints accessible via predictable URLs

3.3 Insecure Authentication Flow

Faulty login design such as:

- No brute-force protection
- OTP reuse allowed
- Password reset without verifying identity

3.4 Insecure Object Lifecycle Design

Objects flow through multiple states incorrectly.

Example:

A "pending" transaction can be modified even after “approved”.

3.5 Insecure Input/Output State Machine

System cannot properly handle:

- Race conditions
- Concurrent actions
- State transitions

🔷 4. Hacker Mindset: How Attackers Exploit Insecure Design

A pentester identifies insecure design by:

4.1 Asking Logic Questions

- What happens if I replay the same request again?
- What happens if I skip a step?
- What happens if I change the sequence?
- What happens if I modify a parameter?

4.2 Exploring All States

- Pending
- Confirmed
- Completed
- Rejected

Attackers test if transitions are enforced.

4.3 Manipulating Client-Controlled Data

- Prices
- User roles
- Quantities
- Permissions
- Status flags

4.4 Bypassing Client-Side Validation

JavaScript validation is completely useless from a security perspective.

Tools:

- Burp Suite
- Postman
- cURL

🔷 5. Real-World Case Studies

Case Study 1 — Flipkart Coupon Abuse

A coupon meant for **one-time use** worked multiple times because:

- Only frontend checked "coupon_used = false"
- Backend allowed unlimited requests

Loss: crores of rupees

Root cause: Insecure design (missing backend verification)

Case Study 2 — PayTM Payment Confirmation Flaw

Old systems trusted **client-side "payment_success=true"**.

Attackers simply modified request → free recharge.

Fixed by: server-side callback verification.

Case Study 3 — Race Condition in Banking

Two withdrawal requests sent simultaneously → both approved.

Caused balance manipulation.

Attack tool:

```
ffuf -w payloads.txt -u https://bank.com/withdraw -t 40
```

🔷 6. Penetration Testing Methodology for Insecure Design

Unlike a normal vulnerability, insecure design testing is **logic-based**, not payload-based.

⭐ 6.1 Step-by-Step Testing Plan

Step 1 — Understand the Application

Map:

- Business rules
- User roles
- Workflow state machine

Tools:

- Burp Suite
- OWASP Amass for recon
- Postman for API flow

Step 2 — Identify Critical Functions

Look for:

- Payment
- Order approval

- Account changes
- Privileged operations

Step 3 — Try Abusing the Workflow

Common tests:

- Skip step → go directly to “complete”
- Replay request → see if duplicate action is allowed
- Reverse order → trigger step 3 before step 1
- Modify hidden fields

Step 4 — Check Authorization at Every Request

Use Burp Repeater.

Example request:

```
GET /api/user/42/profile HTTP/1.1
Authorization: Bearer <token>
```

Modify user ID:

```
GET /api/user/41/profile
```

If accessible → Insecure design.

Step 5 — Check Rate Limits

Run:

```
for i in {1..1000}; do
  curl -X POST https://example.com/api/login \
  -d "user=test&pass=1234" &
```

```
done
```

If no lockout → insecure authentication design.

Step 6 — Check Predictable IDs

If system uses:

- `/invoice/1001`
- `/invoice/1002`

Test enumeration.

Command:

```
ffuf -u https://example.com/invoice/FUZZ -w numbers.txt
```

🔷 7. Tools Used in Testing Insecure Design (with Commands)

7.1 Burp Suite (Most Important Tool)

Actions for Insecure Design Testing:

- Manipulate workflow
- Modify hidden parameters
- Replay privileged requests
- Change request order
- Trigger state transitions
- Fuzz parameters

Burp Repeater Example

```
POST /api/cart/checkout HTTP/1.1
Content-Type: application/json

{"price": 1}
```

Modify price manually → backend shouldn’t trust it.

7.2 cURL

Used for:

- Step skipping
- Replay attacks
- Race condition testing

Example: Replaying payment confirmation

```
curl -X POST https://example.com/api/pay/confirm \
-d '{"txn":"123","status":"success"}'
```

7.3 FFUF / Intruder

Used to brute-force parameters or workflow states.

Example: Finding bypass states

```
ffuf -u https://site.com/order?status=FUZZ \
-w wordlists/states.txt
```

7.4 Postman

Best for API workflow mapping.

7.5 Race The Web

For concurrency attacks.

Command:

```
race --url https://bank.com/withdraw --threads 50
```

🔷 8. Full Payload & Command Examples

8.1 Workflow Skipping

Try skipping the "cart" step:

```
curl -X POST https://example.com/api/checkout
```

If checkout works → insecure workflow.

8.2 Price Manipulation

```
POST /api/order
{
  "item_id":123,
  "price":1
}
```

Backend must validate price server-side.

8.3 Modify Hidden Fields

Change role:

```
"role":"admin"
```

If accepted → insecure authorization design.

8.4 Unlimited Coupon Use

```
POST /apply-coupon
{
  "coupon":"FREEME"
}
```

Spam the request:

```
for i in {1..100}; do curl ... & done
```

🔷 9. How to Report Insecure Design in a Pentest Report

Title: Insecure Business Logic: Unlimited Coupon Abuse

Severity: Critical

Impact: Loss of revenue

Description:

System fails to validate coupon usage on backend. Attackers can repeatedly apply coupon.

Steps to Reproduce:

1. Add any item to cart
2. Apply coupon
3. Replay same request 10 times using Burp Repeater

4. Total amount becomes negative

Fix Recommendation:

- Enforce server-side coupon usage counter
- Bind coupon to user account + order ID
- Implement proper state machine

🔷 10. How to Fix Insecure Design (Developer Guidelines)

✔️ Build explicit state machines

✔️ Validate all transitions

✔️ Enforce backend authorization

✔️ Validate all client data on server

✔️ Use rate limiting

✔️ Build threat models

✔️ Perform design reviews

✔️ Enforce least privilege