Below is a **professionally formatted, sentence-by-sentence breakdown** of the document **“XSS and XSRF Notes.docx”**, optimized for **CompTIA A+ 220-1102 (Core 2)** exam preparation — specifically for:

* **Domain 2.0: Security**
* **Objective 2.5: Threats, vulnerabilities, and attacks**

This layout is designed so you can **copy and paste directly into Word without needing reformatting**.

**🛡️ XSS vs XSRF – Study Notes (CompTIA A+ 220-1102 Aligned)**

**✅ 1. Concept Overview**

**Cross-site scripting (XSS)** and **cross-site request forgery (XSRF/CSRF)** are both **web application vulnerabilities** that allow attackers to exploit the relationship between users and trusted websites.

* **XSS**: Attacker tricks the **user’s browser** into executing malicious scripts.
* **XSRF**: Attacker tricks a **trusted website** into performing actions as if initiated by the user.

**📘 2. Exam Relevance**

* **CompTIA A+ 220-1102 – Core 2**
* **Domain 2.0: Security**
* **Objective 2.5: Common threat types**

You are expected to:

* Know the difference between XSS and XSRF
* Recognize their goals and mechanisms
* Identify basic prevention techniques

**📋 3. Sentence-by-Sentence Breakdown**

**🌐 Cross-Site Scripting (XSS)**

* XSS occurs when an attacker **embeds malicious script commands** into a **trusted website**.
* The attacker’s goal is to:
  + **Steal cookies**
  + **Access session data**
  + **Gain elevated privileges** using malicious browser scripts
* The **victim is the user**, not the server.
* The web server may have already been compromised, or it **unknowingly serves the malicious script**.
* XSS **exploits trust between the user’s browser and the web server**.
* The attacker **injects code into a page**, which the **server then delivers** to the user’s browser.
* This makes the browser **trust and execute** the attacker’s script, assuming it’s part of the legitimate site.

**🧨 Types of XSS Attacks**

1. **Stored (Persistent) XSS**
   * Attacker submits malicious input that is **saved to the server** and **served to future users**.
   * Example: Inserting a script in a blog comment field that gets displayed to other readers.
2. **Reflected XSS**
   * The malicious code is part of a **link or URL**, and is **activated when the victim clicks** it.
   * No data is stored on the server — it’s a **non-persistent attack**.
3. **DOM-Based XSS**

**DOM** stands for **Document Object Model**.

* It’s a **programming interface** used by web browsers to **represent and interact with the content of a webpage**.
* Think of it as a **live tree-like structure** that organizes all parts of a web page — like paragraphs, headers, forms, and buttons — so they can be accessed and manipulated by code (especially **JavaScript**).

🧠 In simple terms:

The **DOM is how your browser “sees” and manages a web page** behind the scenes.

An attacker can:

* + Exploits the **browser’s Document Object Model (DOM)** directly.
  + The attack happens **entirely on the client-side**, without server involvement.
  + Also known as **client-side XSS**.

**🔐 How to Prevent XSS Attacks**

* **Programmers should**:
  + Use **output encoding** to ensure HTML and script tags are treated as text, not code.
    - **Output encoding** means **converting special characters** (like <, >, &, ", ') in user input into a safe version that the browser will **display as text**, instead of **executing it as code**.
  + Implement **input validation** to block script insertion in forms.
* **Users can**:
  + Increase **cookie security settings**.
  + **Disable scripting languages** like JavaScript when possible.

**🎭 Cross-Site Request Forgery (XSRF/CSRF)**

* XSRF **exploits the trust** that a **website has in an authenticated user**.
* The attacker **forces the user’s browser** to submit a request to a site **where the user is already logged in**.
* Example: You log into your bank. The attacker tricks your browser into sending a fund transfer request **without your knowledge**.
* The request is valid to the server because it comes from an **authenticated session**.
* The attacker **cannot see the server’s response**, but can still trigger actions like:
  + Changing a password
  + Transferring money
  + Submitting a form on your behalf

**🔒 How to Prevent XSRF Attacks**

**Great question — this concept is key to understanding how developers prevent Cross-Site Request Forgery (CSRF/XSRF) attacks, which is covered in CompTIA A+ 220-1102, Objective 2.5 under Security threats and vulnerabilities.**

**🛡️ What Does “Use Specialized Tokens in Forms” Mean?**

**It means websites should embed unique, hard-to-guess codes (called tokens) into web forms (like login, password change, or payment forms) that only a legitimate user’s browser can submit.**

**This ensures that any request sent to the server is truly from the real user — not a forged request from a hacker.**

**🔑 2. What Is a CSRF Token?**

**A CSRF token is:**

* **A unique string (like a89F3e12r5) generated by the server**
* **Tied to the user’s session**
* **Included as a hidden field in web forms**

**When the user submits the form:**

* **The browser sends the token back to the server**
* **The server checks that the token matches what it originally sent**

**If a hacker tries to forge a request, they won’t have the correct token — so the server rejects it.**

**🤖 3. What About CAPTCHAs?**

**A CAPTCHA (“Completely Automated Public Turing test to tell Computers and Humans Apart”) is:**

* **A visual challenge (like selecting images or typing distorted letters)**
* **Used to verify the user is a real person, not a bot or script**

**Though not a CSRF token, it prevents automated malicious form submissions, making it harder for attackers to:**

* **Automatically submit forged requests**
* **Exploit forms through scripting**

**🔐 4. Why This Prevents XSRF (CSRF) Attacks**

**In an XSRF attack:**

* **The attacker tricks the user’s browser into sending a request to a site the user is already logged into (like a bank)**
* **But the attacker doesn’t know the unique token**
* **Without the valid CSRF token, the server knows the request is fake — and blocks it**
* **Developers should**:
  + Use **specialized tokens** in forms (e.g., CSRF tokens or CAPTCHAs)
  + Require **double cookie submission** and match verification
    - A **cookie** is a **small text file** stored on your computer by a website.
    - It contains data the website uses to **remember information about you**, such as:
      * Your **login session**
      * **Preferences** (language, theme)
      * Items in a **shopping cart**
    - Cookies help websites **remember who you are** and **what you’re doing** — across pages and visits.
  + Use **authentication and encryption protocols**
  + **Scan XML files** uploaded by users for hidden attacks

**🧩 4. Key Comparison Table**

| **Aspect** | **Cross-Site Scripting (XSS)** | **Cross-Site Request Forgery (XSRF)** |
| --- | --- | --- |
| **Target of Attack** | The **user** | The **web server** |
| **Exploits Trust Between** | User’s **browser and website** | Website’s **trust in the user** |
| **Goal** | Run malicious scripts in user’s browser, steal data | Trick the user into making unintended server requests |
| **Stored?** | Can be stored or reflected | Typically **not stored**, but auto-executed |
| **Prevention (User)** | Disable scripts, secure cookies | Log out when done, avoid suspicious links |
| **Prevention (Dev)** | Input/output sanitization | Tokens, CAPTCHA, cookie validation, server-side checks |

**🧾 5. Exam Inclusion Notification**

✅ **Yes – Both XSS and XSRF are included in the CompTIA A+ 220-1102 exam** under:

* **Objective 2.5**: Threats, vulnerabilities, and attacks

Expect to:

* Distinguish between XSS and XSRF scenarios
* Identify which one targets the user vs the server
* Know prevention strategies from both a **developer** and **end-user** perspective