ShadowRecon – XSS Vulnerability Finding

1. Overview

During a web application security assessment, a custom tool named **ShadowRecon** was used to automate reconnaissance, WAF detection/bypass, fuzzing, and exploitation against the target application.
As part of this workflow, ShadowRecon identified a **Cross-Site Scripting (XSS)** vulnerability in the **Description** field on the user profile page.

2. ShadowRecon Workflow (High Level)

ShadowRecon is composed of several chained modules:

- **Recon & Endpoint Discovery**
    - Enumerates subdomains and application endpoints.
    - Aggregates URLs and parameters into centralized lists to be used for fuzzing and exploitation.
- **WAF Detection & Bypass**
    - Detects the presence and type of Web Application Firewall in front of the target.
    - Automatically tests multiple HTTP methods, headers, encodings, and path/parameter mutations to find request patterns that bypass filtering and return HTTP 200 responses.
- **Fuzzing Orchestration**
    - Parses responses and log files to identify promising endpoints and parameters (including authenticated profile functionality).
    - Generates and runs fuzzing commands (e.g., FFUF) against query parameters and body fields using predefined wordlists.
- **Exploitation Engine (XSS, SQLi, etc.)**
    - Takes the “interesting” URLs from the fuzzing stage.
    - Automatically runs external exploit tools against those URLs.
    - Logs any positive or “possibly vulnerable” results into dedicated files.

Using this pipeline, ShadowRecon isolated the **Edit Description** endpoint as a candidate for XSS and flagged it as “XSS possible”, which led to manual validation and full confirmation of the vulnerability.

---

3. Vulnerability Summary

- **Vulnerability Type:** Cross-Site Scripting (XSS), hybrid between Stored XSS and Self-XSS
- **Affected Area:** User profile page, **Description** field (Edit Description form)
- **Authentication:** Requires a valid authenticated user account
- **Discovery Method:** Automatically identified by **ShadowRecon** (dalfox module in the exploitation stage), then manually verified

The Description field accepts HTML/JavaScript content which is stored in the database and later rendered in the **Edit Description** form without proper output encoding. As a result, JavaScript code embedded in this field is executed in the user’s browser when opening the edit form.

---

4. Technical Details

4.1 Behavior

- The payload inserted into the Description field is **persisted in the database**.
- The payload **does not execute** when viewing the public profile page.
- The payload **does execute** when the user opens the **Edit Description** form (where the stored value is re-injected into the HTML context without sufficient sanitization/encoding).

This means the issue currently behaves like:

- **Self-XSS** for the profile owner (the script runs in their own browser when they open Edit Description).
- **Stored XSS potential**, because the malicious value is stored server-side and may be rendered in other internal views.

4.2 Risk Escalation Scenario

While the current impact is limited to the account owner, the fact that the payload is stored in the database creates a realistic escalation path:

- If any privileged user (e.g., **Admin, Moderator, Support, or internal reviewer**) views the Description field in a **raw or weakly sanitized** form (e.g., in an admin panel, moderation queue, CS tools, or analytics dashboard), the stored payload will also execute in **their** browser.
- This can lead to:
    - Session hijacking via cookie or token theft.
    - Arbitrary JavaScript execution in highly privileged contexts (e.g., performing admin actions on behalf of the victim).
    - Lateral movement and privilege escalation inside the application's back-office.

Depending on how and where this Description field is reused internally, the vulnerability can escalate from **Self-XSS** to a fully exploitable **Stored XSS** against privileged accounts.

---

5. Steps to Reproduce

1. **Log in** with a valid user account.
2. Navigate to the **Profile** page.
3. Click on **"Edit Description"** (or the equivalent edit profile/biography function).
4. In the **Description** field, insert a JavaScript payload (for example, a simple alert() proof-of-concept).
5. Save the changes.
6. Re-open the **Edit Description** form.
7. Observe that the injected script is executed in the browser instead of being displayed as plain text.

---

6. How ShadowRecon Was Used

1. **Endpoint Enumeration**

- ShadowRecon's recon and fuzzing modules enumerated the authenticated profile-related endpoints, including the URL corresponding to the **Edit Description** functionality.

2. **WAF Handling and Fuzzing**
    - The tool automatically adjusted HTTP headers, methods, and paths to bypass the WAF and ensure stable 200 OK responses for the profile endpoints.
    - Fuzzing against parameters and form fields indicated that the Description field accepted and reflected user-controlled data.
3. **Automated XSS Testing**
    - The exploitation module passed the discovered profile URLs to an XSS scanner as part of ShadowRecon's pipeline.
    - The tool reported the Edit Description endpoint as **"possibly vulnerable to XSS"** and logged that URL into the XSS candidates list.
4. **Manual Confirmation**
    - Using the ShadowRecon output, the specific endpoint was tested manually with a proof-of-concept payload.
    - The payload was stored in the Description field and executed when the Edit Description form was loaded, confirming the XSS vulnerability.

This shows that ShadowRecon was directly responsible for **identifying the vulnerable endpoint and indicating XSS behavior**, significantly reducing manual effort and increasing coverage.

---

7. Business Impact

- Currently:
    - XSS executes for the **profile owner** when they open Edit Description (Self-XSS behavior).
    - This alone demonstrates that the application does not properly encode user-generated content in this context.
- Potentially:

- Any internal system that reads and renders the Description as HTML (admin/ops dashboards, moderation tools, CRM, etc.) is at risk.
- If a privileged user views the malicious profile, the attacker's JavaScript runs in the context of that privileged session.
- This can lead to:
    - Account takeover of admin/support users.
    - Unauthorized administrative actions.
    - Exposure or manipulation of sensitive data belonging to other users.