Discovering a Blind SSRF Vulnerability in a PHP RSS Feed Parser

Hemant Raj Bhati Follow 3 min read · Mar 14, 2026

Introduction

During security testing and code review of a PHP-based RSS Feed Parser, I discovered a **Blind Server-Side Request Forgery (SSRF)** vulnerability.

The issue occurs because the application fetches user-supplied URLs directly on the server without proper validation. This allows an attacker to make the server send requests to arbitrary destinations.

In this article, I will explain how the vulnerability works and demonstrate how the issue can be identified during a security assessment.

. . .

Application Interface

The application allows users to input an RSS feed URL which is then fetched and parsed by the server.

Example interface:

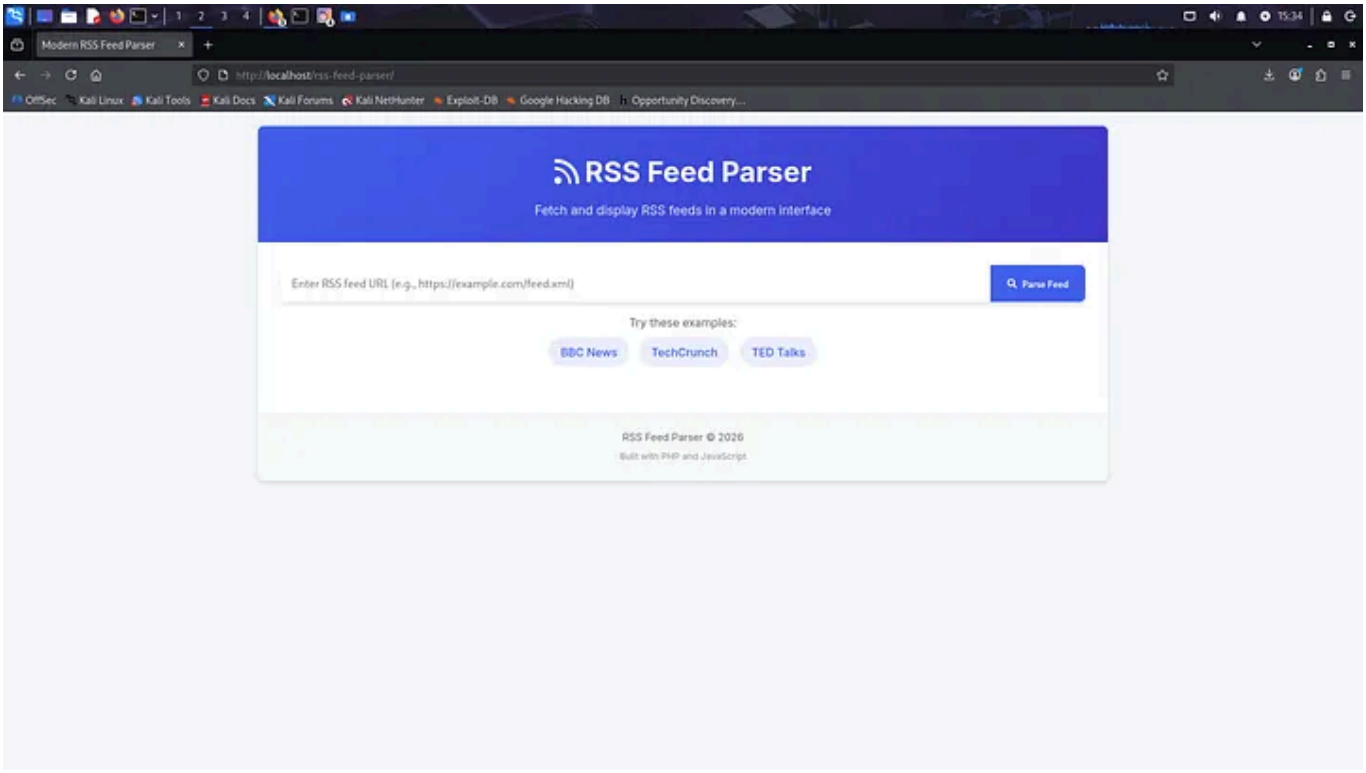


Figure: The RSS Feed Parser interface where users can submit an RSS feed URL for parsing.

The user simply enters a feed URL and clicks **Parse Feed.**

. . .

Code Review

During the code review process, the following PHP code was identified:

```
$content = @file_get_contents($url);
```

```
$xml = new SimpleXMLElement($content);
```

The application directly fetches content from a **user-controlled URL** without validating or restricting it.

This behavior can lead to **Server-Side Request Forgery (SSRF)**.

. . .

Why This Is a Security Issue

Since the server fetches arbitrary URLs, an attacker may force the server to send requests to unintended locations such as:

- Internal network services
- Localhost applications
- Cloud metadata endpoints
- Internal APIs

This could potentially expose sensitive information or allow internal network scanning.

. . .

Proof of Concept

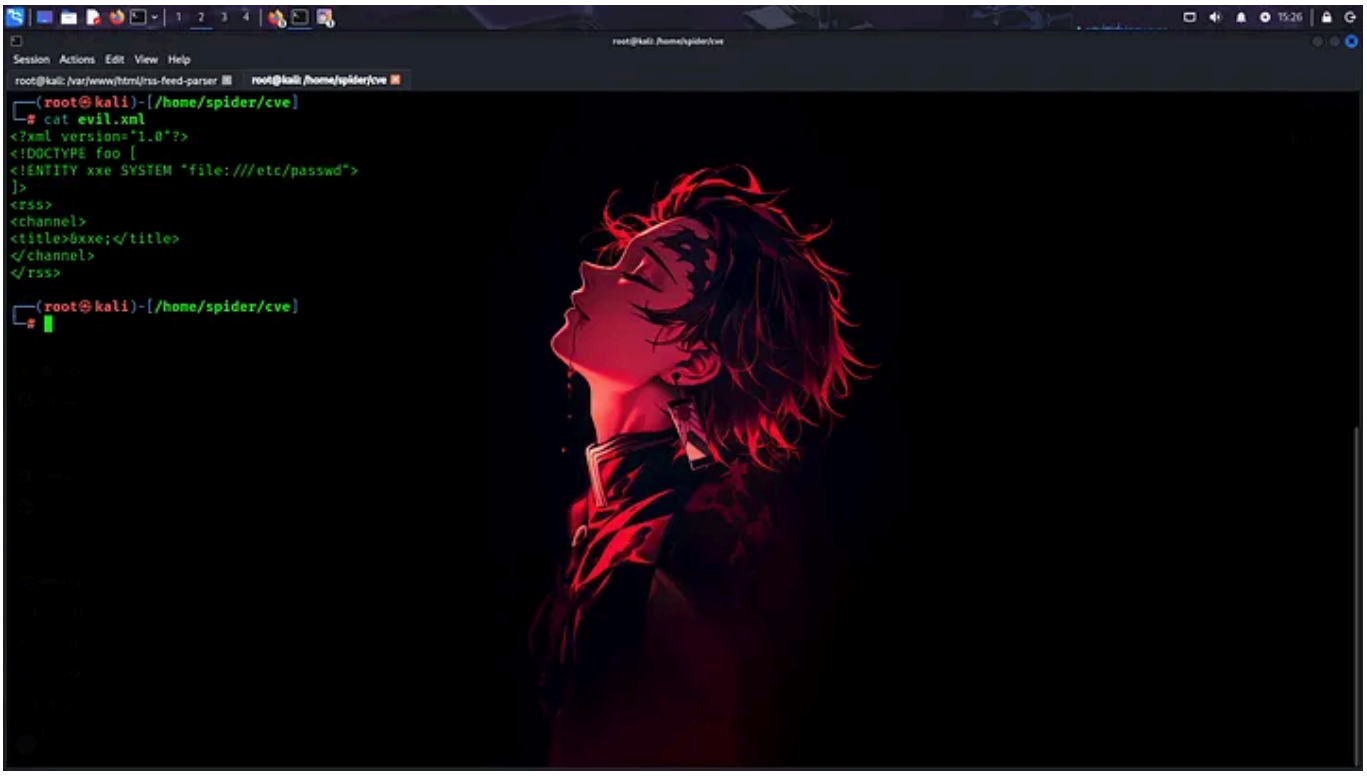
To demonstrate the issue, a test XML file was created and hosted on a local HTTP server.

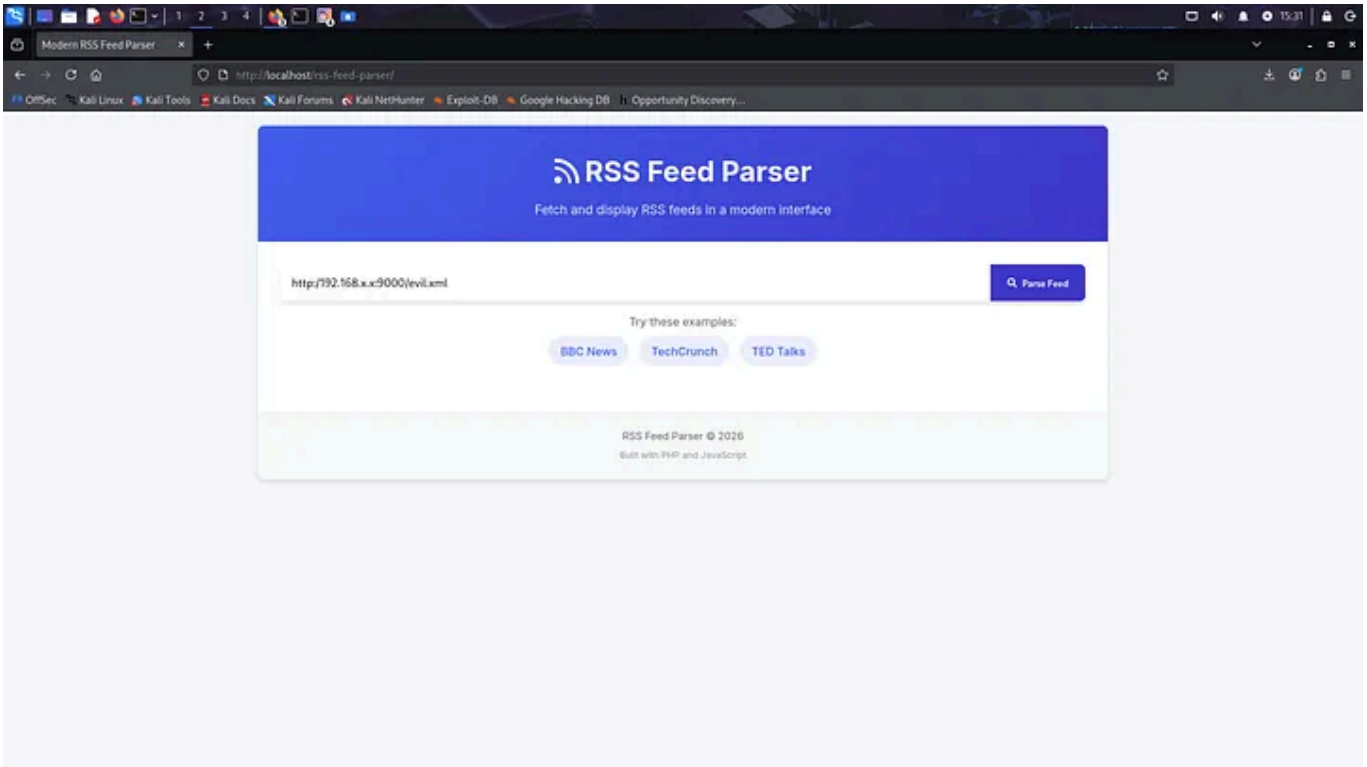
Example command used to host the file:

```
python3 -m http.server 9000
```

The following URL was then supplied to the RSS parser:

```
http://ATTACKER-IP:9000/evil.xml
```





• • •

Server Request Confirmation

The local server logs confirmed that the application server attempted to retrieve the attacker-controlled file.

Example server logs:

```
GET /evil.xml HTTP/1.1 200
```

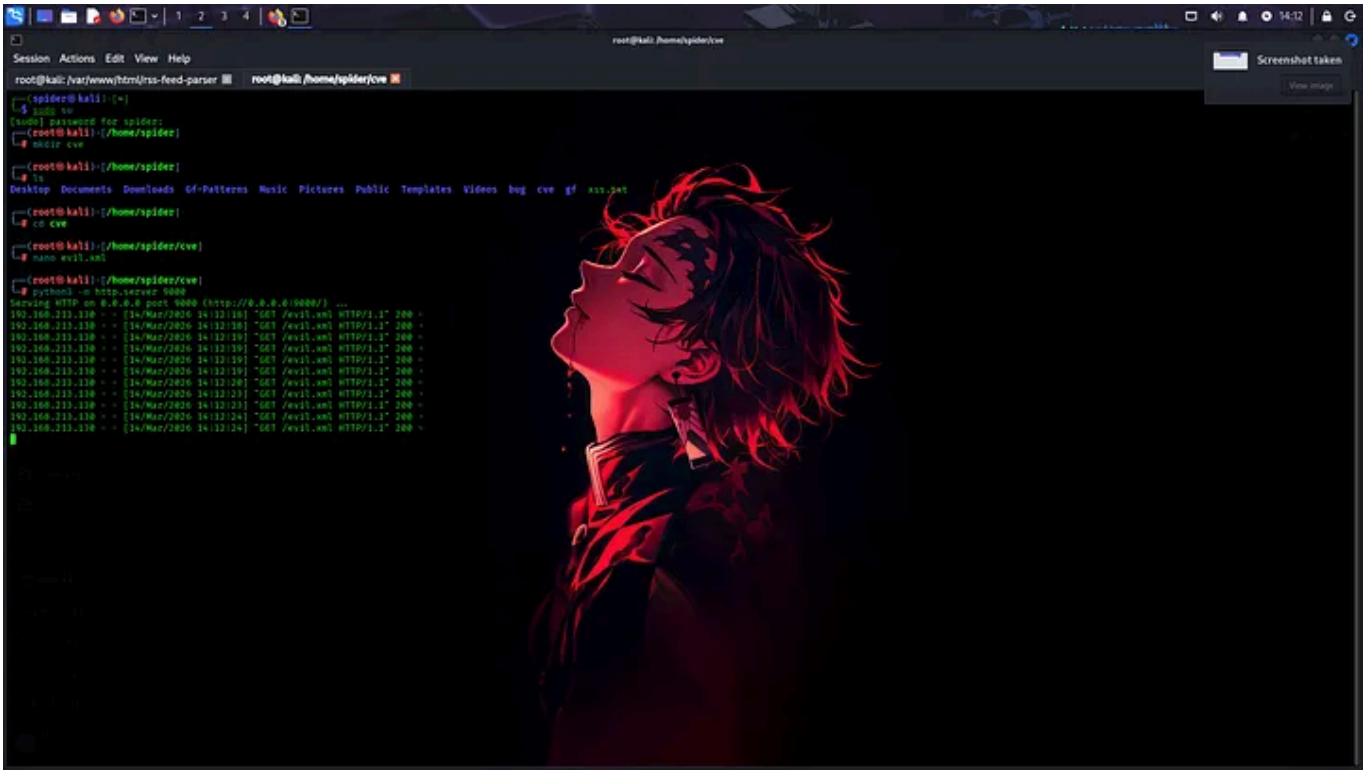


Figure: Server logs showing the target application fetching the malicious XML file, confirming the SSRF behavior.

[Insert Screenshot — Python HTTP Server Logs Showing Request]

This confirms that the application server made a request to the attacker-controlled resource.

Because the response is not returned to the user, this behavior is considered **Blind SSRF**.

. . .

Impact

If such a vulnerability exists in a production environment, it may allow attackers to:

- Interact with internal services
- Scan internal infrastructure
- Access internal APIs
- Attempt access to cloud metadata services

. . .

Mitigation

Developers can prevent SSRF by implementing the following security controls:

- Validate all user-supplied URLs
- Block internal IP ranges
- Restrict outbound network requests
- Use allowlists for trusted domains

Example validation:

```
if (!filter_var($url, FILTER_VALIDATE_URL)) {
    die("Invalid URL");
}
```

. . .

Conclusion

This case demonstrates how simple code patterns can introduce security risks when user input is not properly validated.

Regular code reviews and security testing are essential to identify and mitigate such vulnerabilities before deployment.

. . .

This research was conducted for educational and security testing purposes.

Cybersecurity | Bug Bounty | Ssrf | Web Security | Web Penetration Testing

Written by Hemant Raj Bhati

4 followers · 1 following

Cybersecurity Researcher | OSCP Certified | CVE Analysis & Exploitation | Web Application Security | Documenting real-world vulnerabilities and PoCs.

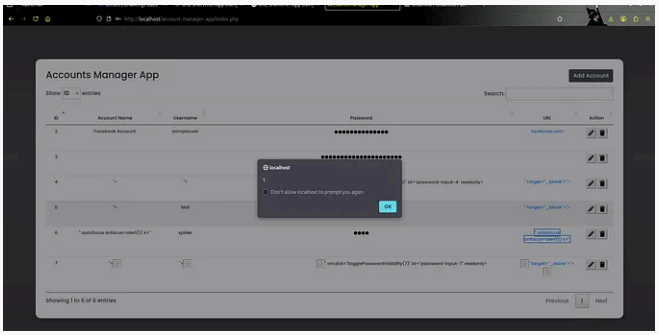
Follow

No responses yet

Write a response

What are your thoughts?

More from Hemant Raj Bhati

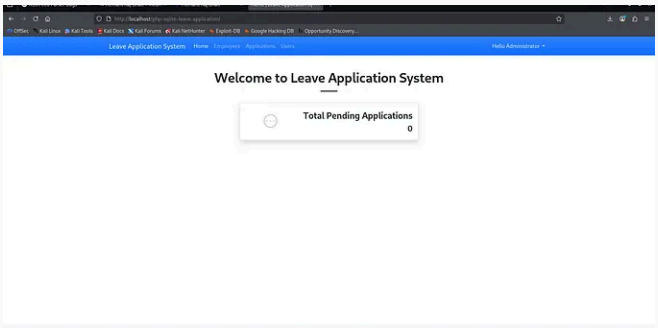


Hemant Raj Bhati

Stored Cross-Site Scripting (XSS) in Accounts Manager App Using...

Introduction

Feb 20

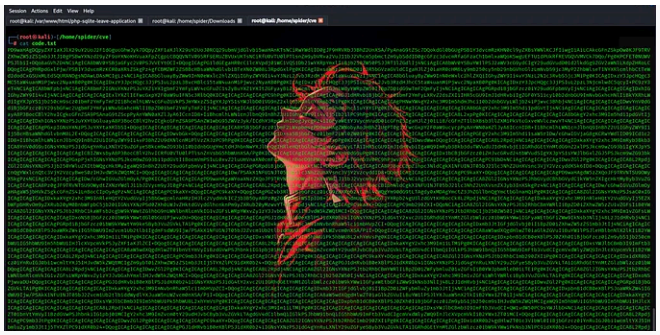


Hemant Raj Bhati

Stored Cross-Site Scripting (XSS) in PHP Leave Application System

Overview

Mar 15 2

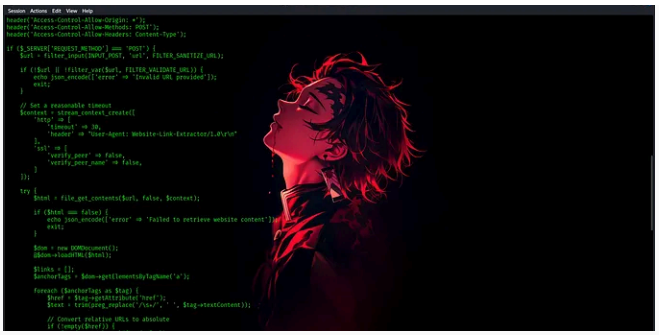


Hemant Raj Bhati

Local File Inclusion (LFI) in Leave Application System (PHP &...

Introduction

Mar 15



Hemant Raj Bhati

SSRF Vulnerability in SourceCodester Website Link...

Overview

Feb 15

See all from Hemant Raj Bhati

Recommended from Medium

In OSINT Team by Vivek PS

Kayra Öksüz

How a $20,000 Bug Was Hidden Inside YouTube's API — A Bug...

Note: This is a review and narrative of a bug bounty write-up originally published by...

4d ago 110

Remember Me. Forget MFA

Some of the most interesting vulnerabilities are not where you expect them.

Mar 19 112 1

In InfoSec Write-ups by ***Lostsec***

A Practical Workflow for Fuzzing and Scanning in Bug Bounty

How to Systematically expand your attack surface, eliminate noise and find the bugs...

4d ago 327 2

Aland DIshad (HexaPhp)

File Upload Bypass to Remote Code Execution (RCE) HEXAPHP

Introduction

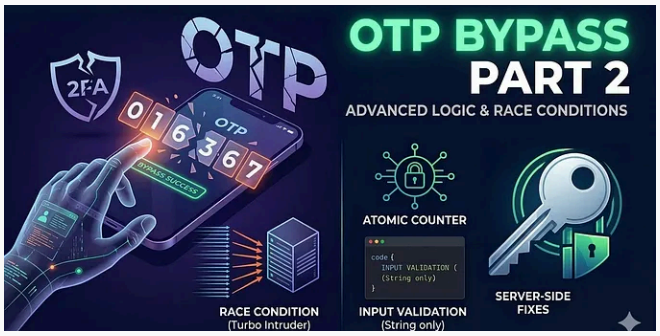
4d ago 12

Muhammad Wageh

My $150 Bug Bounty: A Low-Severity Access Control Bug

Hey Everyone!

Mar 14 322 2



Arrhenius Paelongan

OTP Bypass Part 2: Advanced Logic Flaws and Race Conditions

Introduction

Mar 17 5