Blind SSRF via feedparser accessing an internal URL – Plone 5.2.4

Affected Produce:

Produce & Version: Plone 5.2.4 (Test performed on the v5.2.4 another version might be affected too)

URL: https://plone.org/download

Severity: Low

What is SSRF Overview:

In a Server-Side Request Forgery (SSRF) attack, the attacker can abuse functionality on the server to read or update internal resources. The attacker can supply or modify a URL which the code running on the server will read or submit data, and by carefully selecting the URLs, the attacker may be able to read server configuration such as AWS metadata, connect to internal services like HTTP enabled databases, or perform post requests towards internal services which are not intended to be exposed.

Impact:

A successful SSRF attack can often result in unauthorized actions or access to data within the organization, either in the vulnerable application itself or on other back-end systems that the application can communicate with. In some situations, the SSRF vulnerability might allow an attacker to perform arbitrary command execution. An SSRF exploit that causes connections to external third-party systems might result in malicious onward attacks that appear to originate from the organization hosting the vulnerable application, leading to potential legal liabilities and reputational damage.

What is blind SSRF?

Blind SSRF vulnerabilities arise when an application can be induced to issue a back-end HTTP request to a supplied URL, but the response from the back-end request is not returned in the application's front-end response.

What is the impact of blind SSRF vulnerabilities?

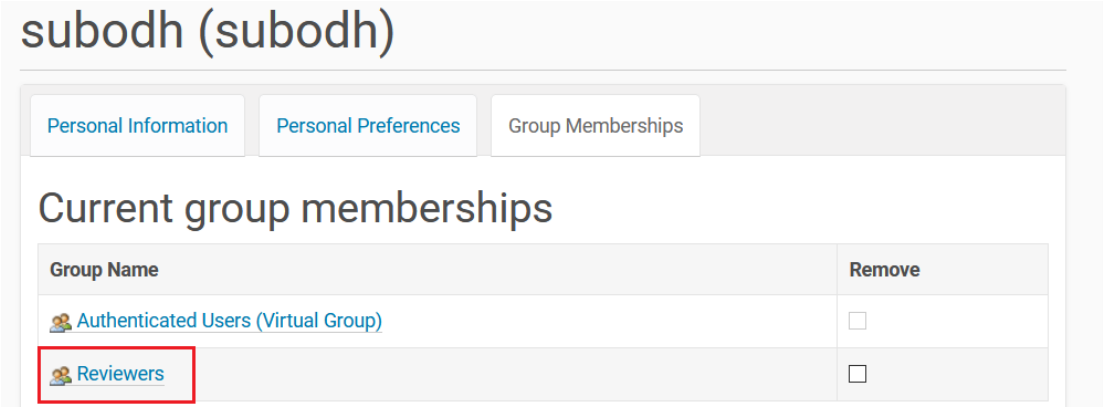
The impact of blind SSRF vulnerabilities is often lower than fully informed SSRF vulnerabilities because of their one-way nature. They cannot be trivially exploited to retrieve sensitive data from back-end systems, although in some situations they can be exploited to achieve full remote code execution.

Server IP: 172.20.10.7

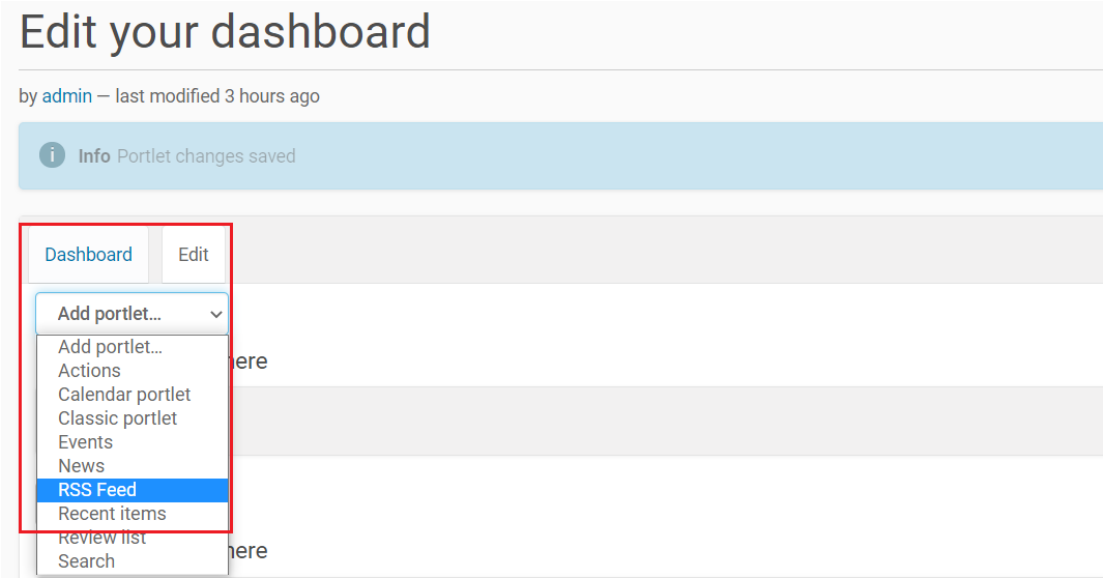
Attacker IP: 172.20.10.3

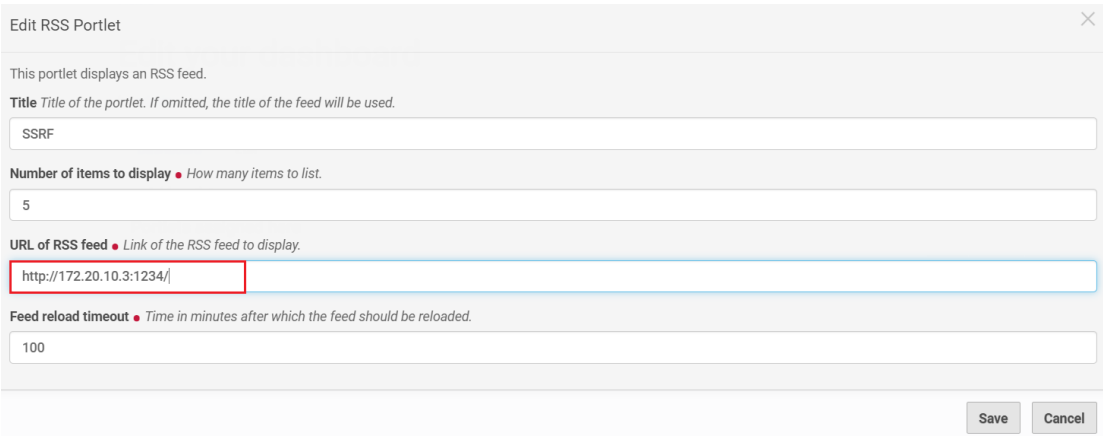
Steps to reproduce:

1. Setup an application and create a reviewer account. SSRF can be performed by admin but I tried to reproduce this with the least privileged user.



2. Enter attacker IP/URL in the URL of RSS Feed and click the Save button.





3. Start a netcat listener on the attacker machine on the same port as given in the URL using the command: *nc -nvlp 1234*
4. Reload the Dashboard page of any user.
5. Check back to the attacker machine, you will see a request coming from the application-hosted server.

Prevention:

Simple blacklists and regular expressions applied to user input are a bad approach to mitigating SSRF. In general, blacklists are a poor means of security control. Attackers will always find methods to bypass them. In this case, an attacker can use an HTTP redirect, a wildcard DNS service such as xip.io, or even alternate IP encoding.

Whitelists and DNS Resolution

The most robust way to avoid server-side request forgery (SSRF) is to whitelist the DNS name or IP address that your application needs to access. If a whitelist approach does not suit you and you must rely on a blacklist, it's important to validate user input properly. For example, do not allow requests to private (non-routable) IP addresses (detailed in RFC 1918).

However, in the case of a blacklist, the correct mitigation to adopt will vary from application to application. In other words, there is no universal fix to SSRF because it highly depends on application functionality and business requirements.

Response Handling

To prevent response data from leaking to the attacker, you must ensure that the received response is as expected. Under no circumstances should the raw response body from the request sent by the server be delivered to the client.

Disable Unused URL Schemas

If your application only uses HTTP or HTTPS to make requests, allow only these URL schemas. If you disable unused URL schemas, the attacker will be unable to use the web application to make requests using potentially dangerous schemas such as file:///, dict://, ftp://, and gopher://.

Authentication on Internal Services

By default, services such as Memcached, Redis, Elasticsearch, and MongoDB do not require authentication. An attacker can use server-side request forgery vulnerabilities to access some of these services without any authentication. Therefore, to ensure web application security, it's best to enable authentication wherever possible, even for services on the local network.

References:

https://portswigger.net/web-security/ssrf/blind

https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/19-Testing_for_Server-Side_Request_Forgery

https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/

https://blog.detectify.com/2019/01/10/what-is-server-side-request-forgery-ssrf/

https://portswigger.net/web-security/ssrf

Tools used:

Linux (web application server and attacker machine)

Web Browser

Netcat listener.

Researcher Details:

Name: Subodh Kumar

LinkedIn: https://www.linkedin.com/in/s-kustm/

GitHub: https://github.com/s-kustm

Reported on 6th Apr 2021 to the email security@plone.org - https://plone.org/security