A10:2021-Server-Side Request Forgery

Yasser Elsnbary

Agenda

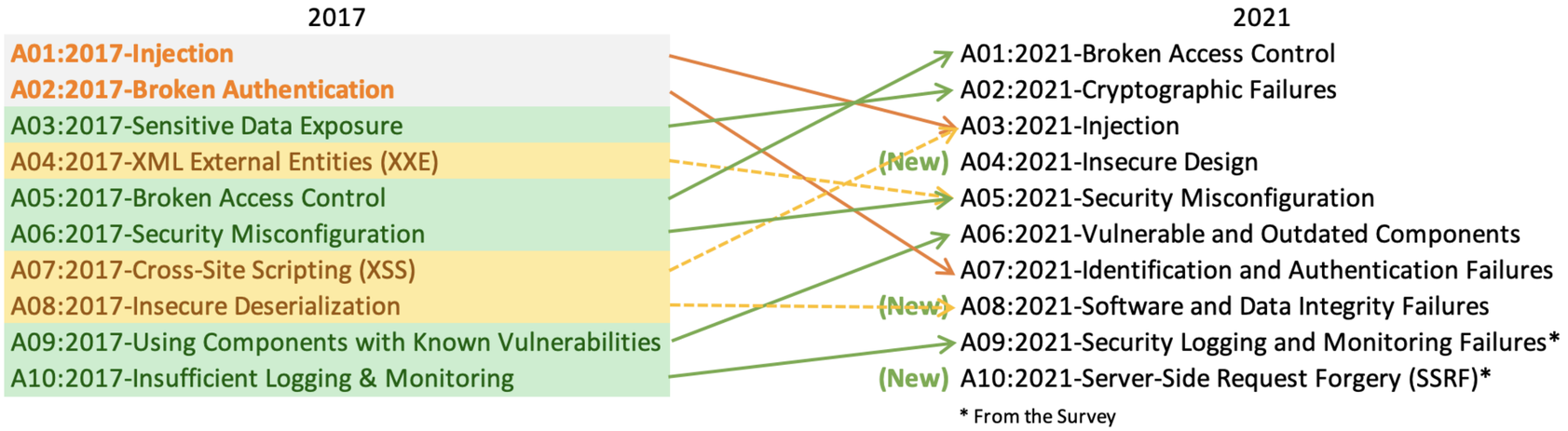
- What is SSRF?
- Types of SSRF
- Why SSRF Happen?
- Impact and Severity
- Exploitation
    - SSRF to LFI
    - SSRF to RCE
    - SSRFmap Tool
- SSRF Code Review
- Lab Demo
- Mitigation
- Resources

What is SSRF?

A10

This category was added to the OWASP Top 10 community survey (#1) in **2021**.

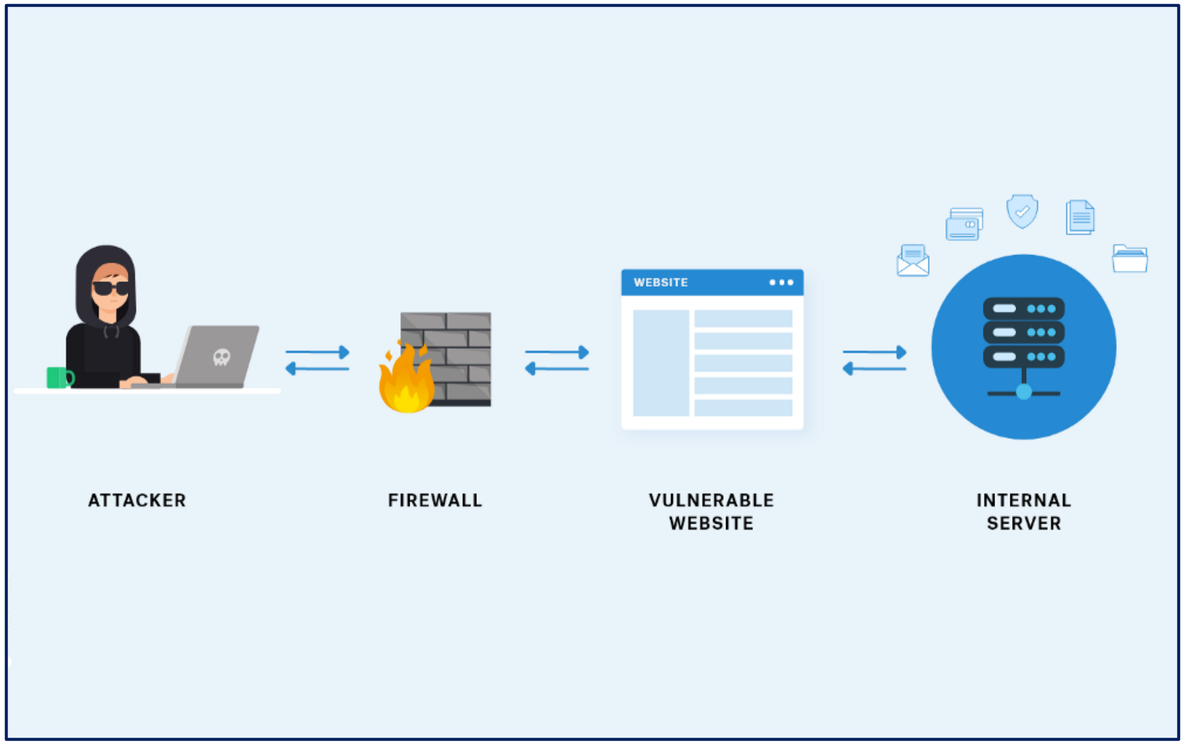
The data shows a relatively low **incidence** rate with above-average **testing coverage** and above-average **Exploit** and Impact **potential** ratings.



[CWE-918: Server-Side Request Forgery (SSRF)]

What is SSRF?

Server-side request forgery (**SSRF**) is a type of attack that allows an adversary to make arbitrary **outbound requests** from a server. In some cases, an attacker can use SSRF to **pivot** throughout corporate networks, **exploit** otherwise unreachable internal systems, or query **metadata** endpoints **to extract secrets**.



Types of SSRF

- Basic SSRF

Basic SSRF involves exploiting a web application's ability to make HTTP requests to arbitrary destinations. Attackers can forge requests to internal services or external systems, often leading to unauthorized data access.

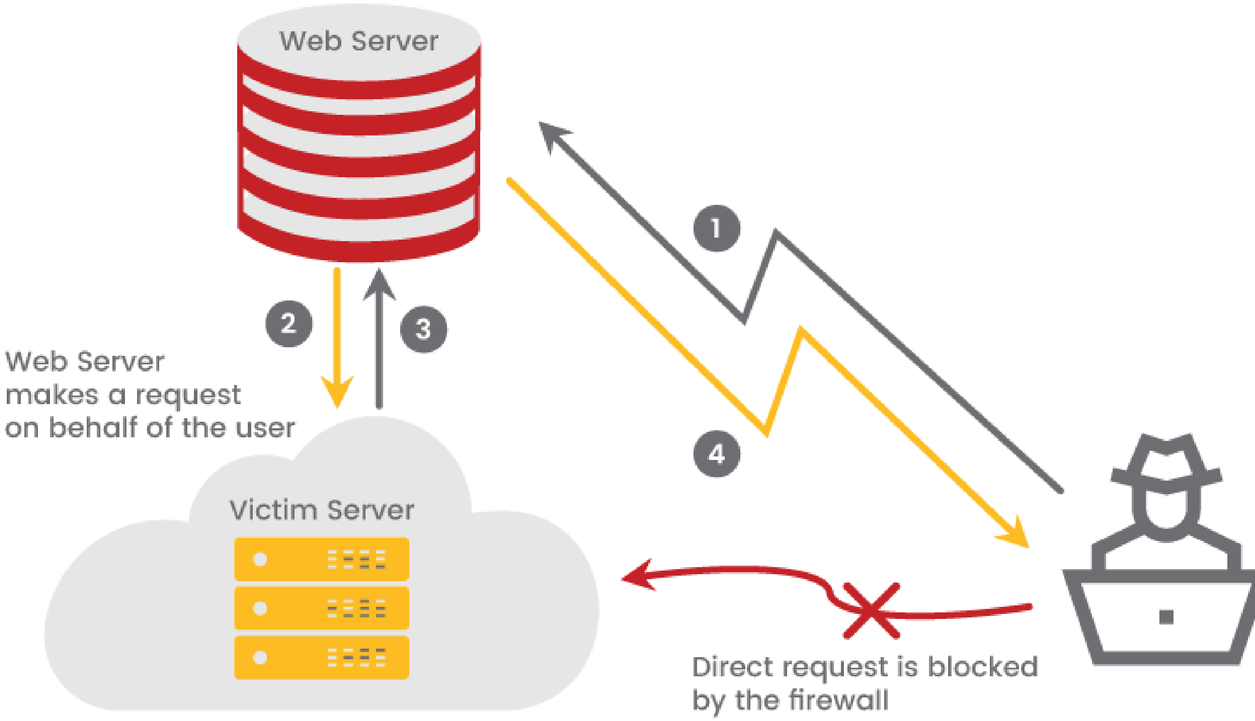
- Blind SSRF

Blind SSRF occurs when an attacker can trigger SSRF but doesn't receive the response directly. Instead, they rely on other techniques, like out-of-band (OOB) requests, to confirm the attack's success. This makes detection and mitigation more challenging.

Why SSRF Happen?

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.



Impact and Severity

SSRF is a very dangerous vulnerability that may cause serious security breaches. It is a very convenient way to **avoid firewalls** and **access internal resources** that would otherwise be inaccessible. SSRF is often used to escalate attacks further:

- Bypass Whitelisting
- Bypassing Authentication levels
- Enumerating Other Protocols
- Cross-Site Port Attack (XSPA)
- Denial of Service (DoS)
- Local File Inclusion (LFI)
- Remote Code Execution (RCE)

Exploitation

Where to look for SSRF?

- **Doc Parse**

  curl https://xyz.com/user/image?imgUrl=internal.target.local/images

- **Link Expansion**

  stockApi=http://localhost/admin

- **File Upload**

  Try to send URL as a filename to get blind SSRF, for example, filename=https://172.17.0.1/internal/file. You can also try to change type=”file” to type=”url” within a request.

Exploitation

Where to look for SSRF?

- **Doc Parse**

  curl https://xyz.com/user/image?imgUrl=internal.target.local/images

- **Link Expansion**

  stockApi=http://localhost/admin

- **File Upload**

  Try to send URL as a filename to get blind SSRF, for example, filename=https://172.17.0.1/internal/file. You can also try to change type=”file” to type=”url” within a request.

Exploitation

PHP Functions Vulnerable to SSRF:

- **file_get_contents():**
- **fopen()**
- **readfile()**
- **curl_exec()**
- **fsockopen()**
- **stream_context_create()**

```php
function make_request($url) {
    $response = file_get_contents($url);

    return $response;
}
```

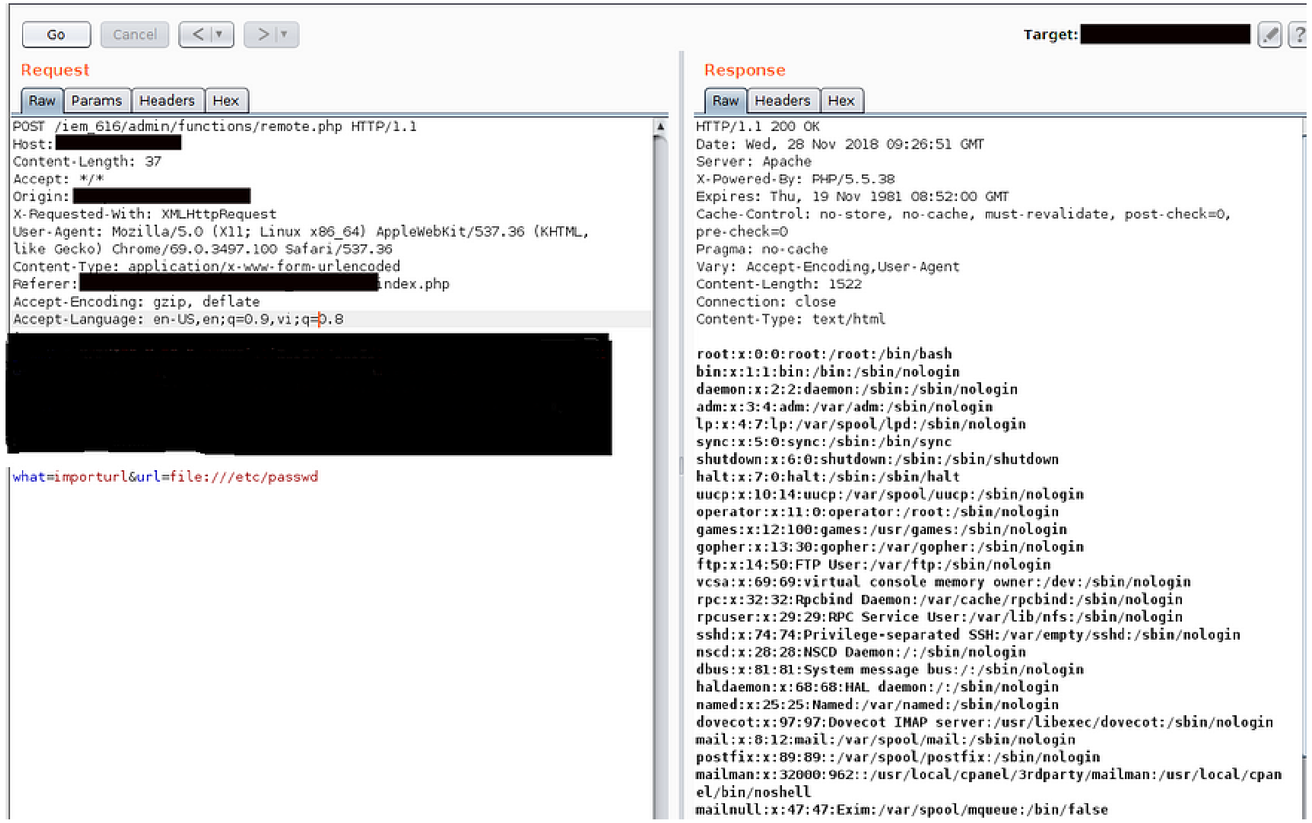
```php
function make_request($url) {
    $handle = fopen($url, "r");
    $response = stream_get_contents($handle);
    fclose($handle);

    return $response;
}
```

```php
function make_request($url) {
    readfile($url);
}
```

Exploitation

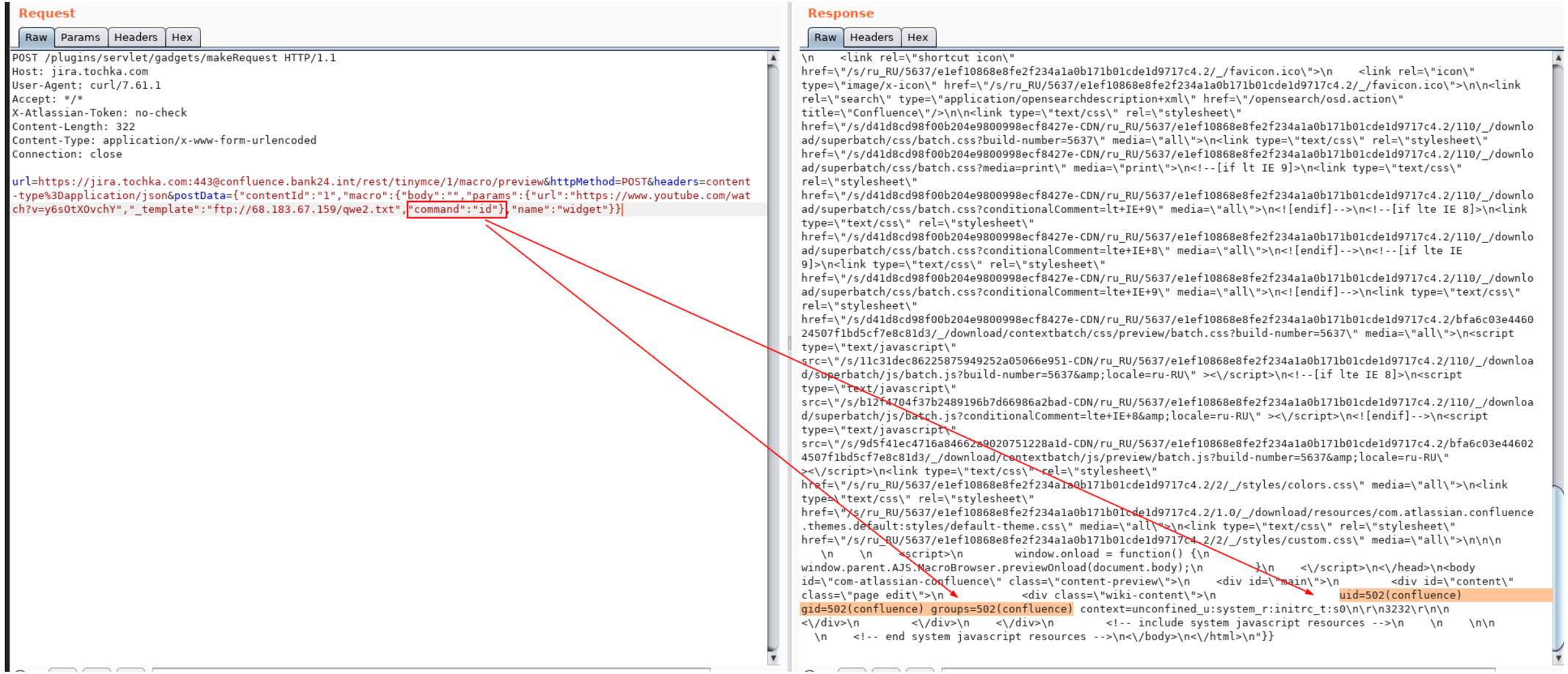
- SSRF to LFI

http://www.REDACTED.com/?url=file:///etc/passwd



Exploitation

- SSRF to RCE: https://hackerone.com/reports/713900



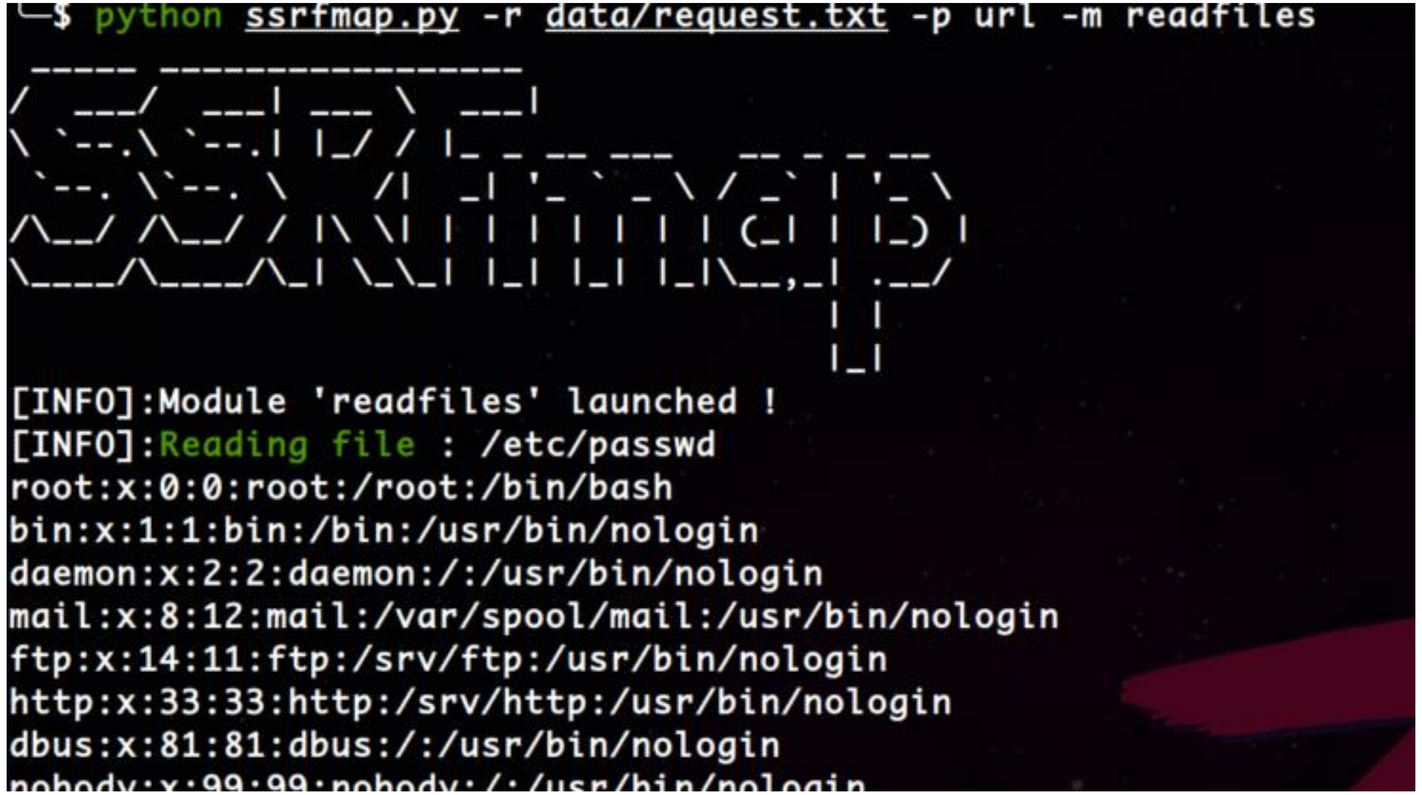
Exploitation

- SSRF to RCE:
    - AWS: http://169.254.169.254/latest/meta-data
    - GCP: http://metadata/computeMetadata/v1/project/project-id
    - Digital Ocean: http://169.254.169.254/metadata/v1/
    - Azure: http://169.254.169.254/metadata

Exploitation

- SSRFmap

SSRFMap is a tool used by security researchers and penetration testers to map internal networks and identify potential vulnerabilities using Server Side Request Forgery (SSRF) attacks. It is an open-source tool written in Python and is available on GitHub.



SSRF Code Review - Vulnerable

Here's an example of a vulnerable PHP code that allows an attacker to make arbitrary requests to internal resources:

```php
<?php

$url = $_GET['url'];
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);
curl_close($ch);

echo $response;

?>
```

SSRF Code Review - Mitigated

To mitigate this vulnerability, you should always validate and sanitize user input, especially when it is used to construct URLs or make network requests. Here's an example of how you can modify the code to prevent SSRF attacks:

```php
<?php

$url = $_GET['url'];

// Validate the URL scheme and host
if (!preg_match('/^https?:\/\/[a-z0-9]+([-\.]{1}[a-z0-9]+)*\.[a-z]{2,5}(:[0-9]{1,5})?
(\/.*)?$/i', $url)) {
    die('Invalid URL');
}

// Sanitize the URL to remove any special characters
$url = filter_var($url, FILTER_SANITIZE_URL);

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);
curl_close($ch);

echo $response;

?>
```

Lab Demo

- Lab: SSRF with blacklist-based input filter:

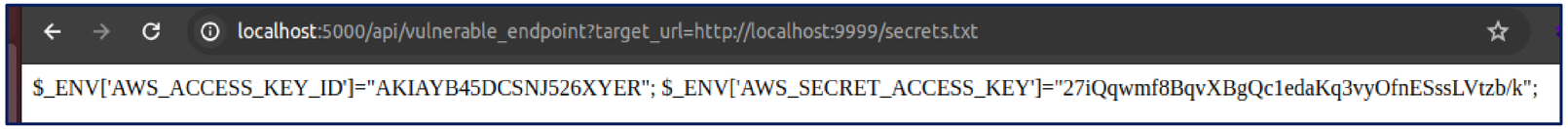
https://portswigger.net/web-security/ssrf/lab-ssrf-with-blacklist-filter

- Lab: SSRF with whitelist-based input filter:

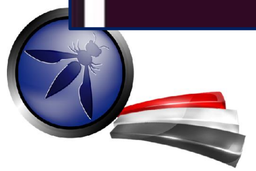
https://portswigger.net/web-security/ssrf/lab-ssrf-with-whitelist-filter

Lab Demo

```python
from flask import Flask, request
import requests

app = Flask(__name__)
@app.route('/api/vulnerable_endpoint', methods=['GET'])
def vulnerable_endpoint():
    # Get user input for the target URL
    target_url = request.args.get('target_url')
    # Make an HTTP request to the target URL
    response = requests.get(target_url)
    # Return the response to the user
    return response.text

if __name__ == '__main__':
    app.run(debug=True)
```




```
yasser@yasser:~/Data/OWASP_CHAPTER/SSRF/secrets$ python3 -m http.server 9999
Serving HTTP on 0.0.0.0 port 9999 (http://0.0.0.0:9999/) ...
127.0.0.1 - - [05/Feb/2024 17:29:43] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Feb/2024 17:29:57] "GET /secrets.txt HTTP/1.
127.0.0.1 - - [05/Feb/2024 17:30:17] "GET /secrets.txt HTTP/1.
127.0.0.1 - - [05/Feb/2024 17:30:58] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Feb/2024 17:31:02] "GET /secrets.txt HTTP/1.1" 200 -
```



Mitigation

- Sanitize and validate user input
- Disable unused URL schemas: file://, ftp://, and dict://
- Use Authentication on the internal services
- Disable HTTP redirections
- User white list for all of the IPs the web application needs to deal with
- Don’t use the black list for mitigation

Resources

- https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/
- https://portswigger.net/web-security/ssrf
- https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery
- https://www.cobalt.io/blog/a-pentesters-guide-to-server-side-request-forgery-ssrf
- https://blog.assetnote.io/2021/01/13/blind-ssrf-chains/
- https://learn.snyk.io/lesson/ssrf-server-side-request-forgery/#step-1732b042-9404-4d3c-f8f3-01498e6f84d3
- https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/
- https://www.briskinfosec.com/blogs/blogsdetail/Server-Side-Request-Forgery-SSRF
- https://brightsec.com/blog/ssrf-attack/#risks-of-ssrf-attacks
- https://medium.com/@adithyakrishnav001/server-side-request-forgery-ssrf-bf23802cfb12

Thanks for your time

Yasser Elsnbary