
**Riccardo BONAFEDE**
Università di Padova

Matteo Golinelli

# File Disclosure and Server-Side Request Forgery



*https://cybersecnatlab.it*

# License & Disclaimer



## License Information



## Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

# Goal



- Present common file disclosure vulnerabilities and their possible mitigations
- Learn how to perform a Server-Side Request Forgery attack

# Prerequisites



- Lecture:
  - *WS_1.1 - HTTP Protocol and Web Security Overview*

# Outline



- File Disclosure
  - Impact and Overview
  - Paths 101
  - Path traversal attacks
  - Fixes
- Server-Side Request Forgery

# Outline



- File Disclosure
  - Impact and Overview
  - Paths 101
  - Path traversal attacks
  - Fixes
- Server-Side Request Forgery

# File Disclosure



- A file disclosure is the **impact of certain vulnerabilities**
- As the name suggests, it consists of the ability to **disclose/leak important files from a server**
- Because it is an impact, there are **multiple classes of vulnerabilities** that **lead to file disclosure**
  - For example, remote code execution is another type of vulnerability that could results in a file disclosure

# File Disclosure



- Files inside a server are critical information:
  - In many applications, users-uploaded files are the sensitive information that the application is protecting
  - The disclosure of such files can be a violation of the site policy

# File Disclosure



- It is also possible to **steal configuration files** from the webserver **which might contain critical information items**
  - *Database configuration files* often contain the credentials to access the database
  - Files like the *tomcat-users.xml* contain the credentials to access the tomcat manager
  - Files like *flask configuration* or *web.config* in a .net application contain the secret used to sign the session

# File Disclosure



- Finally, it is possible to **steal the source code** of the web application
  - For some business, the source code of the web application is their **product/asset**
  - An attacker in possession the source code is more effective
    - It is easier for the attacker to find other vulnerabilities, especially if the application was developed according to a *security by obscurity* model

# File Disclosure



- How can a web app disclose internal files?
  - Basically, **everything that works with files can lead to a file disclosure vulnerability**
  - There are standard sinks, and some of them are a trivial
  - If a user-controlled input manages to go inside these sinks, the web app is at risk

# File Disclosure



- Some sinks are trivial...
  - Every function in every programming language that manages files
    - Every flavor of **`open`**/**`fopen`** in every language
    - Flask **`send_file`**
    - …
- As said before, it is also possible to leak files if the web app suffers from **code execution**

# File Disclosure



- Some sin
  - Every fu nguage that manage
    - Every f ge
    - Flask **s**
    - …
- As said b eak files if the web app **n**

```
fopen
tmpfile
bzopen
gzopen
SplFileObject->__construct
// write to filesystem (partially in combination with rea
chgrp
chmod
chown
copy
file_put_contents
lchgrp
lchown
link
mkdir
move_uploaded_file
rename
rmdir
```

# File Disclosure



- Some sinl
  - Every fu anguage that manages
    - Every fl age
    - Flask **s**
    - …
- As said be leak files if the web app **on**

```
readfile
readlink
realpath
stat
gzfile
readgzfile
getimagesize
imagecreatefromgif
imagecreatefromjpeg
imagecreatefrompng
imagecreatefromwbmp
imagecreatefromxbm
imagecreatefromxpm
ftp_put
ftp_nb_put
exif_read_data
read_exif_data
exif_thumbnail
exif_imagetype
```

# File Disclosure



- Other sinks are less trivial
  - **cURL** is used as a http client, but it can also be used to open files

```
$fd = curl_init('file:///etc/passwd');
echo curl_exec($a);
```

# File Disclosure



- It’s sometimes possible to leak important files just because they are publicly accessible
  - *.git* directory exposed
    - If you make your git directory open to the internet, everyone will be able to dump all files inside it
  - Web-server misrouting
    - It’s sometimes possible to trick a web server to return a .php file as an image...

# Outline



- File Disclosure
  - Impact and Overview
  - Paths 101
  - Path traversal attacks
  - Fixes
- Server-Side Request Forgery

# Paths 101



- Let us focus on what happens if a user-controlled input finds a way to an open-like function
- We first need to understand few things about how paths work

# Paths 101



- An **absolute path** is a path that describes the location of a file regardless of the working directory

```
/etc/passwd
```

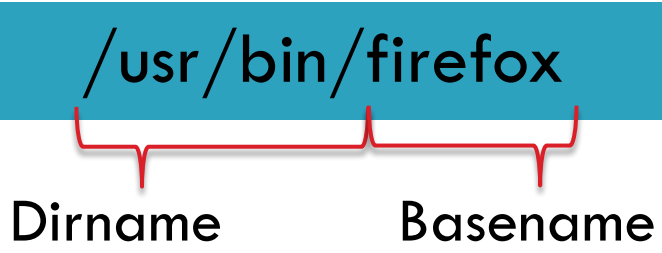
- A **relative path** is a path that describes the location of a file starting from the working directory

```
foo/bar
```

# Paths 101



- Paths are composed by a **dirname** and a **basename**
  - The **dirname** is the portion of the path up to the last /
  - The **basename** is the portion of the path after the last /



# Paths 101



- Every directory has two special subdirectories:
  - The **current directory**, whose name is .

/foobar/./ ➡ /foobar/

  - And the **parent directory**, whose name is ..

/foobar/../baz ➡ /baz

- The parent directory is useful for file disclosure because it permits to access deeper directories inside the file system

# Paths 101



- A path in its **shortest form** is called **normalized**
- For example:
  - */foo/bar* is normalized, there is no way to make it shorter
  - *//foo/bar* is not normalized, /foo/bar is shorter
  - */foo/./bar* is not normalized, /foo/bar is shorter
- What about /foo/test/../bar?

# Paths 101



- What about /foo/test/../bar?
- Its shortest form would be /foo/bar, but what happens if /foo/test/ does not exist?
  - If the path is normalized before opened, then everything is fine: we can access /foo/bar without any problem
  - If the path is not normalized, then the open would fail because /foo/test/ does not exist, and so ..

# Outline



- File Disclosure
  - Impact and Overview
  - Paths 101
  - Path traversal attacks
  - Fixes
- Server-Side Request Forgery

# Path Traversal



- **Path traversal** is a vulnerability that leads to a file disclosure
- It happens when a user-controlled input finds its way into an *open()* or an equivalent function
- If there are no security checks or security sanitizations, an attacker could inject paths that are not meant to be opened

# Path Traversal



- **Path** ile disc
- It h ds its way
- If th sanitizations, an attacker could inject paths that are not meant to be opened

```
<nowiki>
<?php
$template = 'blue.php';
if ( isset( $_COOKIE['TEMPLATE'] ) )
   $template = $_COOKIE['TEMPLATE'];
include ( "/home/users/web/templates/" . $template );
?>
</nowiki>
```

# Path Traversal



- **Path** ile disc
- It h ds its way
- If th sanitizations, an attacker could inject paths that are not meant to be opened

```
<nowiki>
<?php
$template = 'blue.php';
if ( isset( $_COOKIE['TEMPLATE'] ) )
    $template = $_COOKIE['TEMPLATE'];
include ( "/home/users/web/templates/" . $template );
?>
</nowiki>
```

Entry point

# Path Traversal



- **Path** ile disc
- It h ds its way
- If th

```
<nowiki>
<?php
$template = 'blue.php';
if ( isset( $_COOKIE['TEMPLATE'] ) )
    $template = $_COOKIE['TEMPLATE'];
include ( "/home/users/web/templates/" . $template );
?>
</nowiki>
```

Sink

sanitizations, an attacker could inject paths that are not meant to be opened

# Path Traversal



- Few cases might happen:
  - **Plain** injection — open($input)
  - **Prepended** injection — open($input + '/foobar')
  - **Appended** injection — open('/foobar' + $input)
  - **Appended** and **prepended** — open('/foo'+$input+'/bar)

# Full Plain Path Traversal



- *open($input)*
- Without security checks, it is possible to leak every file on the filesystem
- Other problems:
  - Protocols like HTTP / gopher / ssh could be used, making it a Server-Side Request Forgery
  - For some functions, it is possible to execute arbitrary code. (For example if the injection is inside Perl's open[1])

1: https://perldoc.perl.org/functions/open

# Full Plain Path Traversal



- The exploit for this kind of injection is trivial
  - Just put the path of the file to disclose
- A useful test file on Unix systems is **`/etc/passwd`**
- Why?
  - It always exists and is accessible by every user of the system
  - Is a good target to properly check if there is an actual injection inside an open-like function

# Appended Path Traversal



- *open('/somedir/' . $input)*
- It is the most common one
- It is a plain injection without the possibility to use other protocols
- If there is no protection, it is possible to leak every file in the filesystem

# Appended Path Traversal



- To exploit this, append some ../ in order to get to the root directory
- In this way, it is possible to access every file of the filesystem
- For example, try to inject:

```
../../../../../etc/passwd
```

# Appended Path Traversal



```
https://[redacted]/html/js/editor/editor.jsp?editorImpl=../../../WEB-INF/web.xml?
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: Microsoft-IIS/8.5
X-Powered-By: ASP.NET
Date: Thu, 30 Mar 2017 20:24:43 GMT
Connection: close
Content-Length: 54193


<?xml version="1.0"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
    <context-param>
        <param-name>contextClass</param-name>
        <param-value>com.liferay.portal.spring.context.PortalApplicationContext</param-value>
```

# Appended Path Traversal



```
https://[redacted]/html/js/editor/editor.jsp?editorImpl=../../../WEB-INF/web.xml?
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: Microsoft-IIS/8.5
X-Powered-By: ASP.NET
Date: Thu, 30 Mar 2017 20:24:43 GMT
Connection: close
Content-Length: 54193
```

```
<?xml version="1.0"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
    <context-param>
        <param-name>contextClass</param-name>
        <param-value>com.liferay.portal.spring.context.PortalApplicationContext</param-value>
```

# Prepended Path Traversal



- *open($input . 'someotherdata')*
- A little bit trickier than the previous one, normally in two forms:
  - With an appended extension
    - file_get_content($input . '.txt')
  - Or with an appended filename/directory
    - file_get_contents($input . '/somefile.txt')

# Prepended Path Traversal



- Allows the disclosure of files whose path finish with a hardcoded suffix
- There are some tricks
  - e.g., including a null byte

# Prepended Path Traversal



- Some languages support the file:// scheme.
- Particularly interesting because it is parsed as a URL
  - file://localhost**/path/to/file**?someotherdata == /path/to/file

```
ubuntu@ip-172-31-24-48:~$ curl file://localhost/etc/passwd\?someotherdata
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
```

# Prepended Path Traversal



- Some scripting languages internally use the C function *open*
- Because of how C handles strings, open will **ignore everything after a NULL character** (*\x00*)

```
http://foo.bar/?file=../../etc/passwd%00
```

- This trick worked very well for older versions of PHP, but now is patched

# Path Traversal



- A **denylist** is a common mitigation against these types of vulnerabilities
- A denylist is used to look for "dangerous" words inside a user-supplied input
- If a dangerous word is found, the system rejects the input or sanitizes it, thus removing the dangerous word

# Path Traversal



- Denylists **are insecure**, because they are error prone
  - You will never be able to insert all the edge cases!
- For example, does a denylist that contains the word '*proc*' prevent access to the '*/proc/*' directory?
  - No, */dev/fd/* is a link to */proc/self/fd*, so you can access every file of */proc/* with the directory */def/fd/../../*

# Path Traversal



- What if we denylist single dangerous characters like “.” or “/” ?
  - The problem here is congruence. Some languages, javascript in particular, don't handle malformed unicode characters.
  - e.g., the unicode character *u012e* ( Į ), when converted to ascii, is incorrectly  transformed to the byte *x2e* ( . )
  - You can see that if the denylist is using *unicode* but the open function is using the *ASCII* encoding then there is a problem

# Outline



- File Disclosure
  - Impact and Overview
  - Paths 101
  - Path traversal attacks
  - Fixes
- Server-Side Request Forgery

# Fixes



- **Normalize paths**
- In this way there are no "nasty points" inside paths, and it is possible to **enforce a dirname**
  - Pay attention that the function used for normalization parses paths the same way of the open function
    - In this way, you will be able to avoid problems caused by incongruences

# Fixes



- Another good mitigation is a **chroot**
- Chroots are "jails" enforced by the OS or by some programming languages
- If a path is set as a chroot, then every access outside this path would be denied by the OS/interpreter
- If an attacker manages to bypass all security checks, they will be stopped by the chroot

# Fixes



- In summary
  - **Denylists are insecure**, as they can be bypassed in different ways
  - **Allowlists work better**, but defeat the purpose of passing user input inside an open function
  - **Avoid incongruence**, check paths in the same way you open them

# Outline

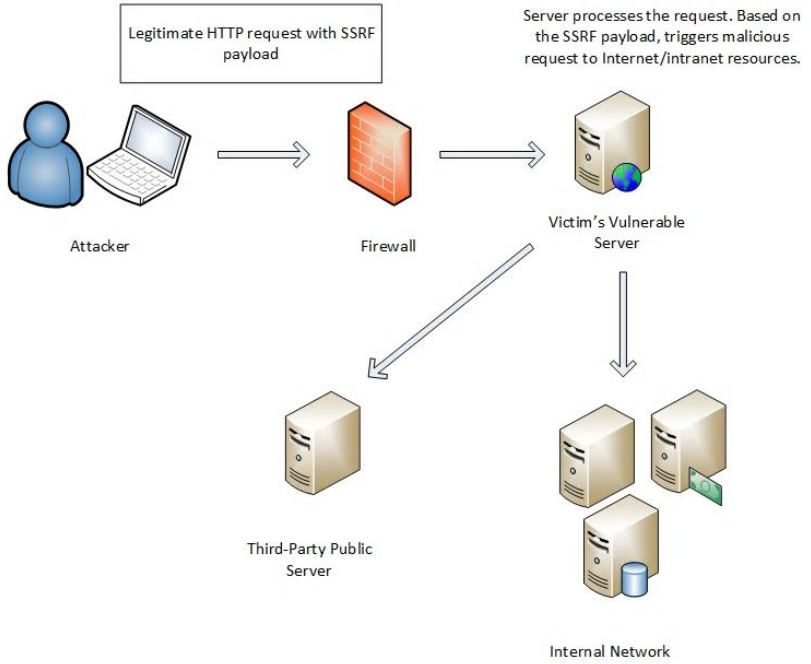


- File Disclosure
  - Impact and Overview
  - Paths 101
  - Path traversal attacks
  - Fixes
- Server-Side Request Forgery

# Server-Side Request Forgery



- A **Server-Side Request Forgery** is a vulnerability that allows an attacker to send a network request from the remote application



# Server-Side Request Forgery



- The impact varies a lot, depending on the control the attacker has on the forged request:
  - Control over the whole **TCP packet**
  - Control over some parts of an **HTTP request**
  - Control only over the **host/port** to which the request is made
  - …

# Server-Side Request Forgery



- SSRFs are dangerous because they allow bypassing the firewall
- If the internal network is not properly designed, it is possible to **access to sensible hosts**, like internal web applications and control panels

# Server-Side Request Forgery



- If the vulnerable web application is hosted on a **cloud instance**, things become more interesting
- Some instances have access to specials *URLs* that often contain **critical data**

# Server-Side Request Forgery



- For example, AWS instances can access the **metadata API**, at the URL http://169.254.169.254/
- This host contains sensitive information such as the **IAM security credentials** and general information about the vulnerable instance[1]

1: https://blog.appsecco.com/an-ssrf-privileged-aws-keys-and-the-capital-one-breach-4c3c2cded3af

# Server-Side Request Forgery



- If there is no output, the SSRF is called **blind SSRF**
- It is less dangerous than a normal SSRFs
- With a blind SSRF it is possible to
  - Map the internal network
  - **Trigger actions** on hosts behind the firewall[1]

1: A nice collection of payloads to use:
https://blog.assetnote.io/2021/01/13/blind-ssrf-chains/

# Server-Side Request Forgery



- It is possible to map the internal network by trying *URLs/ports*, and by looking at the response time
  - This can be done if the response time of the vulnerable endpoint **depends** on the response time of the SSRF request

# Server-Side Request Forgery



- To find an SSRF, you should:
  - Find suspicious endpoints: If you see a URL inside a parameter try putting a URL controlled by you (e.g., an ngrok)
  - If you have a pingback at your host, then probably you have an SSRF.
  - You can try to insert internal hostnames, like "localhost" or common internal IPs (192.168.1.1,10.0.0.1, and so on..)
  - Examine the response time!

# Server-Side Request Forgery



alyssa_herrera submitted a report to **U.S. Dept Of Defense**. Mar 15th (2 years ago)

**Summary:**

An end point on [redacted] allows an internal access to the network thus revealing sensitive data and allowing internal tunneling

**Description:**

OAuth Plugin allows you to provide a url that gives a snap shot of the web page. We can pass internal URLS and conduct SSRF.

## Impact

Critical

## Step-by-step Reproduction Instructions

https://[redacted]/plugins/servlet/oauth/users/icon-uri?consumerUri=http://169.254.169.254/latest/meta-data/hostname

We can see the follow data

ip-172-31-12-254.[redacted].compute.internal

https://[redacted]/plugins/servlet/oauth/users/icon-uri?consumerUri=http://169.254.169.254/latest/meta-data/public-ipv4

[redacted]

# Server-Side Request Forgery



alyssa_herrera submitted a report to U.S. Dept Of Defense. Mar 15th (2 years ago)

**Summary:**

An end point on [redacted] allows an internal access to the network thus revealing sensitive data and allowing internal tunneling

**Description:**

OAuth Plugin allows you to provide a url that gives a snap shot of the web page. We can pass internal URLS and conduct SSRF.

## Impact

Critical

## Step-by-step Reproduction Instructions

https://[redacted]/plugins/servlet/oauth/users/icon-uri?consumerUri=http://169.254.169.254/latest/meta-data/hostname

We can see the follow data

ip-172-31-12-254.[redacted].compute.internal

https://[redacted]/plugins/servlet/oauth/users/icon-uri?consumerUri=http://169.254.169.254/latest/meta-data/public-ipv4

[redacted]

# Server-Side Request Forgery



- Every piece of code that can issue a connection can lead to this vulnerability
- Common functions/libraries are:
  - PHP open-like functions
  - CURL
  - Python's urllib
  - ...

# Server-Side Request Forgery



Snippet of code from Graphite

```
def send_email(request):
    try:
        recipients = request.GET['to'].split(',')
        url = request.GET['url']
        proto, server, path, query, frag = urlsplit(url)
        if query: path += '?' + query
        conn = HTTPConnection(server)
        conn.request('GET',path)
        resp = conn.getresponse()
```

# Server-Side Request Forgery



- Generally speaking, SSRFs are really difficult to avoid
- The most effective way is to check the user-supplied host against an **allowlist**
- Another good mitigation is to make requests from a host that is **isolated from sensitive internal hosts**

Matteo Golinelli - 02/03/2023

# File Disclosure and Server-Side Request Forgery



UNIVERSITÀ DI TRENTO

**Riccardo BONAFEDE**

Università di Padova

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

*https://cybersecnatlab.it*