

Riccardo Bonafede
Università di Padova
bonaff@live.it

File Disclosure and Server-Side Request Forgery



<https://cybersecnatlab.it>

License & Disclaimer

2

License Information

This presentation is licensed under
the Creative Commons BY-NC License



To view a copy of the license, visit:

<http://creativecommons.org/licenses/by-nc/3.0/legalcode>

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.

Outline

3

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

Outline

4

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

File Disclosure

5

- 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 class of vulnerabilities that lead to file disclosure**
 - For example, remote code execution is another type of impact that could also result in a file disclosure

File Disclosure

6

- Files inside a server are critical information:
 - In many applications, files uploaded by the users are the information that we want to protect
 - The disclosure of such files is a violation of the web app authentication itself

File Disclosure

7

- It is also possible to **leak configs files** from the webserver **which might contain credentials**
 - *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

8

- Finally, it is possible to **leak the source code** of the web application
 - For some business, the source code of the web application is its product/asset
 - An attacker with the source code is more effective
 - It is easier for the attacker to find other vulnerabilities, especially if the application was developed with a "security by obscurity" model, and to exploit them

File Disclosure

9

- How can a web app disclose sensible files?
 - Basically, **everything that works with files is dangerous**
 - 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

10

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

File Disclosure

11

- Some sinks
- Basically every programming language that manages files
 - Every flavor of shell
 - Flask server
 - ...
- Obviously, if the web app suffers from code execution

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

programming language

age

s if the web app

File Disclosure

12

- Some sinks
- Basically every programming language that manages files
 - Every flavor of C
 - Flask server
 - ...
- Obviously if the web app suffers from code execution

```
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
```

programming language

age

s if the web app

File Disclosure

13

- Other sinks less not 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);
```
 - **XML** parsing suffers from file disclosure: XML format has some special entities that permit to open and read files

File Disclosure

14

- Sometimes it is 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
 - Sometimes it is possible to trick a web server to return a .php file as an image...

Outline

15

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

Paths 101

16

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

Paths 101

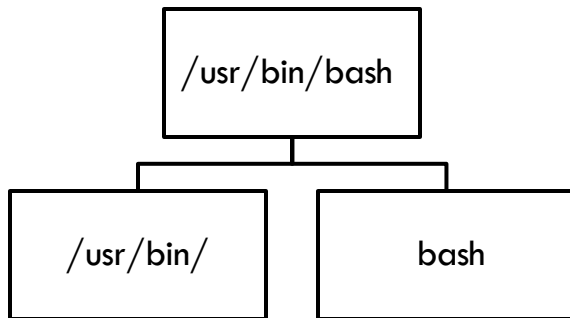
17

- 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

18

- 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

19

- Every directory has two special subdirectories:
 - The **current directory**, whose name is .
 - /foobar/. == /foobar/
 - And the **parent directory**, whose name is ..
 - /foobar../ == /
- The parent directory is especially useful for file disclosure because it permits to access every directory inside the file system

Paths 101

20

- 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

21

- 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

22

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

Path Traversal

23

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

Path Traversal

24

- Path traversal is a technique that allows an attacker to access files and directories outside the web root directory.
- It happens when the application does not properly sanitize user input before using it to construct file paths.
- If there is no security checks or security sanitization, 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

25

- Path traversal is a technique to access files and directories not intended to be accessed in a particular way
- disc
- It happens when an application does not properly sanitize user input before using it to access files or directories
- into
- If there is no security checks or security sanitization, 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

26

- Path traversal is a technique to access files and directories outside the file
- disclosure
- It happens when a program does not properly sanitize user input. **Sink** way
- into the file system
- If there is no security checks or security sanitization, 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 );
?>
```

Path Traversal

27

- There are a few cases that might happen:
 - A **plain** injection `open($input)`
 - A **prepended** injection `open($input + '/foobar')`
 - An **appended** injection `open('/foobar' + $input)`
 - **Appended** and **prepended** `open('/foo'+$input+'/bar')`

Full Plain Path Traversal

28

- `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`)

Full Plain Path Traversal

29

- 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

30

- `open('/somedir/' . $input)`
- It is the most common one
- It is basically 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

31

- 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

Appended Path Traversal

32

```
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

33

```
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>
  </context-param>
```

Prepended Path Traversal

34

- *open(\$input . 'someotherdata')*
- A little bit trickier than the previous one, normally in two forms:
 - An extension is enforced
 - *file_get_content(\$input . '.txt')*
 - Or a filename is enforced
 - *file_get_contents(\$input . '/somefile.txt')*

Prepended Path Traversal

35

- Allows the disclosure of files whose path finish with a hardcoded suffix
- There are some tricks

- 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
```

- Some scripting languages internally use the C function `open`
- Because of how C handles strings, `open` will ignore everything after a NULL character (`\x00`)
- This trick worked very well for older versions of PHP, but now is patched

Path Traversal

38

- A **blacklist** is a common mitigation against these types of vulnerabilities
- A blacklist checks for "dangerous" words inside a user-supplied input
- If a dangerous word is found, the blacklist refuses to process the input or it sanitizes it, thus removing the dangerous word

Path Traversal

39

- Blacklists are insecure, because they are error prone
 - You will never be able to insert all the edge cases!
- For example, does a blacklist that looks for 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 */dev/fd/../../*

Path Traversal

40

- What if we blacklist single dangerous characters like . Or / ?
 - The problem here is congruency. Some languages, javascript in particular, don't handle well malformed unicode characters.
 - For example, the unicode character `\u012e (Ĳ)`, when converted to ascii, is incorrectly transformed to the byte `\x2e (.)`
 - You can see that if the blacklist is using unicode but the open function is using ascii there is a problem

Outline

41

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

Fixes

42

- **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
 - No nasty unicode characters

Fixes

43

- Another good mitigation is **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, he will be stopped by the chroot

Fixes

44

- In summary
 - Blacklists are useless, as they can be bypassed in different weird ways
 - Whitelists work better, but defeat the purpose of passing user input inside an open function
 - Do not create incongruency, check paths the same way you open them

Outline

45

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

Server-Side Request Forgery

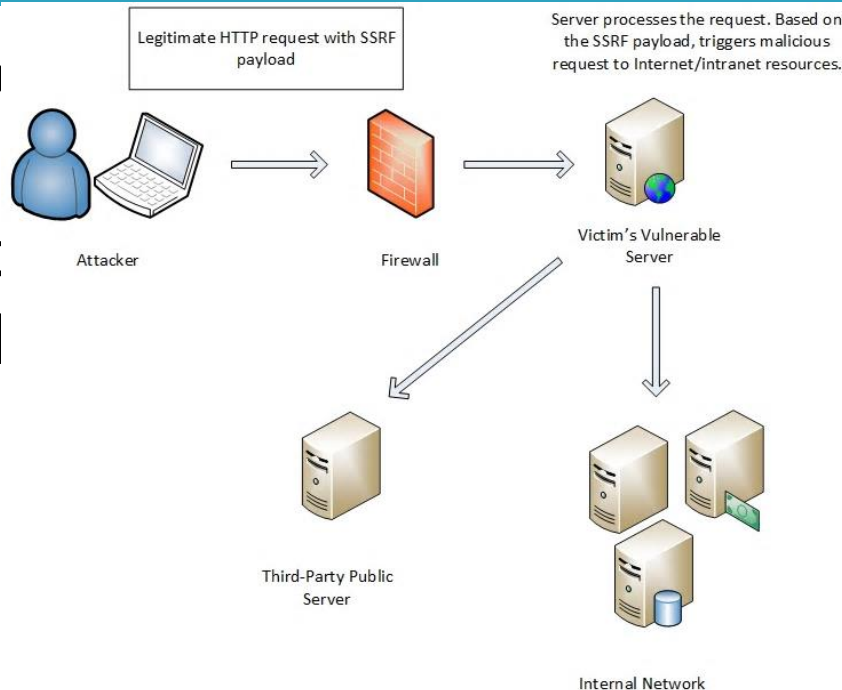
46

- A Server-Side Request Forgery (SSRF from now on) is a vulnerability in which an attacker abuses a functionality of an application to send requests from the server backend

Server-Side Request Forgery

47

- A Server-Side Request Forgery (SSRF) attack is a vulnerability that allows an attacker to make the server make requests to internal resources.



From now on) is
uses a
requests from

Server-Side Request Forgery

48

- The impact varies a lot, and it depends on the control present in the forged requests
 - Total TCP controlled request
 - Total HTTP controlled request
 - Control on some part of HTTP
 - Control only on the host/port

Server-Side Request Forgery

49

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

Server-Side Request Forgery

50

- If the vulnerable web application is hosted on a cloud instance, things become more interesting
- Some instances have access to special URLs that often contain critical data such as API key, used to manage the instances themselves

Server-Side Request Forgery

51

- For example, AWS instances can access the metadata API, at the url <http://169.254.169.254/>
- This host contains sensible information such as the IAM security credentials and general information about the instance

Server-Side Request Forgery

52

- 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

Server-Side Request Forgery

53

- It is possible to map the internal network by trying url/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

54

- To find an SSRf, you should:
 - Find suspicious endpoints: If you see a url inside a parameter try to put a URL controlled by you. You can use a tool like ngrok
 - If you have a pingback at your host, then probably you have an SSRF. Things you should try now is to insert internal hosts, like "localhost" or common internal IPs (192.168.1.1,10.0.0.1, and so on..)
 - Look at the response time!

Server-Side Request Forgery

55



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

[REDACTED]

Server-Side Request Forgery

56



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`

Server-Side Request Forgery

57

- 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

58

```
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

59

```
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()
```

Snipped of code from
Graphite

Server-Side Request Forgery

60

- Generally speaking, SSRFs are really difficult to fix
- The most effective way is to check the user-supplied host against a whitelist
- Another good mitigation is to make requests from a host isolated from the internal network