Directory Traversal

Kameswari Chebrolu

Background

- Location of a file described by a filepath
  - E.g. /var/www/images/chebrolu.jpg (in unix)
    - var, www, images are directories starting from root directory “/”
      - This is an absolute path since it starts from root
      - “/” is a path separator also
    - chebrolu.jpg is a file within the images directory

- If one is already within image directory, the file can also be specified as ./chebrolu.jpg
    - This is a relative path
    - Period (.) character denotes the current directory
    - “current” directory depends on the context where the user is currently
- “..” is used to indicate parent directory
    - If one is inside the images folder and do “cd ../../../”, one will end up in root directory

File Access

- Direct File Access:
  - Example: http://example.com/files/shell.php
    - Web server checks the file's location (likely /var/www/html)
    - Apache normally configured to allow access to such files
    - Web server will return the content or output of the file based on whether appropriate mime type is associated with the extension
  - Example: http://example.com/files/../../../shell.php
    - Web server checks the file's location (likely /)
    - Apache normally configured to disallow access to files in root
    - Web server will return an unauthorized error

- Accessing Files via an Executable Script
  - Example: http://example.com/download.php?file=../../../etc/passwd
  - Script checks for the requested file (/etc/passwd)
  - If script has permissions to access root folder, server will return file
    - Script has whatever user permissions given to web server
    - Many people often install web servers as root → trick script to access unauthorized files!

- Why all this relevant?
  - If server-side code allows evaluation of relative path, attacker can navigate the filesystem for sensitive files breaking access control!

Directory Traversal

- Also called Path Traversal
- Allows attackers to access restricted directories and execute commands outside of the web server’s root directory
  - Depends on how the website access is set up
  - Attacker can impersonate a user associated with “the website” i.e. webadmin or www
    - Can access anything the webadmin has access to

- Attack vector: website has some executable that accepts URLs containing parameters describing paths to files
- Attack involves replacing the URL parameter with a relative file path that uses the ../ syntax to “climb out” of webroot

Scenario

https://vulnerable-website.com/loadImage?filename=cat.png

- User supplies the filename
- loadImage URL takes the filename parameter and returns the contents of the file
  - May append a base path (e.g. /var/www/images/ and use an API to fetch the file (/var/www/images/cat.png)

Attack: Reveal Sensitive Information

```
GET
https://vulnerable-website.com/loadImage?filename=../../../etc/passwd HTTP/1.1
Host: vulnerable-website.com
```

- Why ../../../?
  - ../ sequences moves up the folder /var/www/images/ landing in root (/) folder
  - File read is /etc/passwd
  - Can work in windows also (e.g. ..\..\..\boot.ini)

Attack: Execute Commands

GET https://vulnerable-website.com/script.php?file=../../../../../../Windows/System32/cmd.exe?/c+dir+c:\ HTTP/1.1
Host: vulnerable-website.com

- Web server (not app code) itself is vulnerable (but this is patched now)
  - Navigate and execute command line with specified command
    - ?/c is execute command and terminate
    - Command is dir c:\
- In unix similar to above: https://vulnerable-website.com/loadImage?filename=../../../bin/cat%20/etc/passwd

Circumventing common defenses

- Defense:
  - Strip or block directory traversal sequences (../../) from the user-supplied filename
  - Check file ends in expected file extension (e.g. jpg or png)

- Circumvention
  - Use ....//
    - ../ is stripped, resulting in ../ which works!
  - URL encoding or double URL encoding
    - ../ same as %2e%2e%2f or %252e%252e%252f
  - Use a null byte to terminate the file path
  - E.g. filename=../../../etc/passwd%00.png
    - %00 is null charcter

Best Practices

- Host static files in CDN or cloud storage if possible
  - They employ best practices like below
- If writing own code to serve files from disk, use opaque IDs
  - Assign each file an opaque ID that corresponds to a file path
  - Have URLs reference each file by the ID
  - Needs to maintain state: file ID to a path (in database)
  - Covered more under file upload vulnerabilities

- If using legacy code and cannot refactor, ban path separator characters, including encoded separator characters
  - Validate file names against a regular expression (regex) to filter out anything that looks like path syntax (blacklist)
  - Compare against whitelist of permitted values
    - Input contains only alphanumeric characters
    - Canonicalize the path and verify it starts with the expected base directory
      - Canonicalization: shortest absolute path (unique)
      - abc/../abc/file.txt → abc/file.txt
  - Use well tested third party libraries for this
  - But overall, this is not very secure, attackers often find some way to circumvent

Real Life Examples

Checkout:

https://www.cvedetails.com/vulnerability-list/opdirt-1/directory-traversal.html

Summary

- Path traversals allows attackers to
  - Access restricted directories
  - Also execute commands outside webroot
- Input sanitization is not as secure, should follow best practices as mentioned!

References

- https://portswigger.net/web-security/file-path-traversal
- https://gist.github.com/SleepyLctl/823c4d29f834a71ba995238e80eb15f9 (Windows cheatsheet)
- https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Directory%20Traversal/README.md (Payloads)