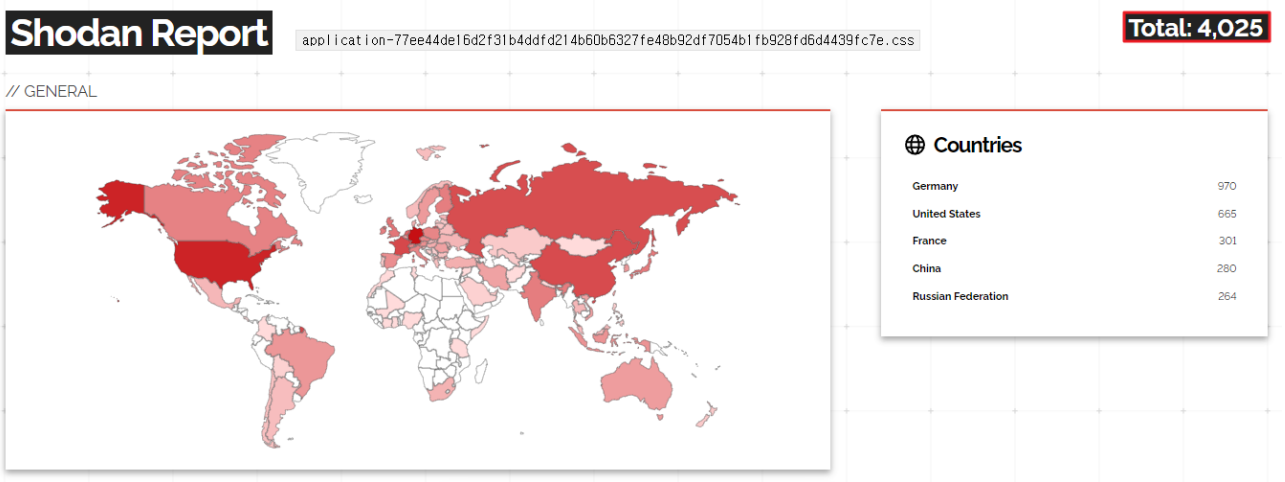
Research & Technique

GitLab arbitrary file reading vulnerability (CVE-2023-2825)

■ Overview of the vulnerability

In May 2023, an arbitrary file reading vulnerability was discovered in GitLab, a Git repository management solution used by individuals or organizations for software development and collaboration. Because the vulnerability can read or download any file on the server by utilizing the path exploration vulnerability, GitLab rated it as 10.0 points based on CVSS[1]. In particular, an unauthenticated attacker can manipulate the attached file download path of an open project and potentially gain access to detailed configuration information, source codes of the company, and sensitive user data, which are the key data files of the server.

Vulnerable GitLab, disclosed on the Internet, can be checked through OSINT search engines like Shodan. As a result of using Shodan to search for vulnerable servers on June 28, it was found that there are about 4,000 vulnerable GitLabs. Therefore, if you use a vulnerable version, you need to be extra careful.



*Source: Shodan Report

Figure 1. Results of the vulnerable server search

[1] CVSS (Common Vulnerability Scoring System) is a free and open industrial standard for evaluating the severity of the security vulnerability of a computer system.

■ Affected software version

The GitLab version vulnerable to CVE-2023-2825 is as follows:

S/W classification	Vulnerable version
GitLab CE(Community Edition)/EE(Enterprise Edition)	16.0.0

※ In order for the vulnerability to work, there is a condition that at least five groups must exist. The condition can be checked through detailed vulnerability analysis below.

■ Attack scenario

The attack scenario using the CVE-2023-2825 vulnerability is as follows:

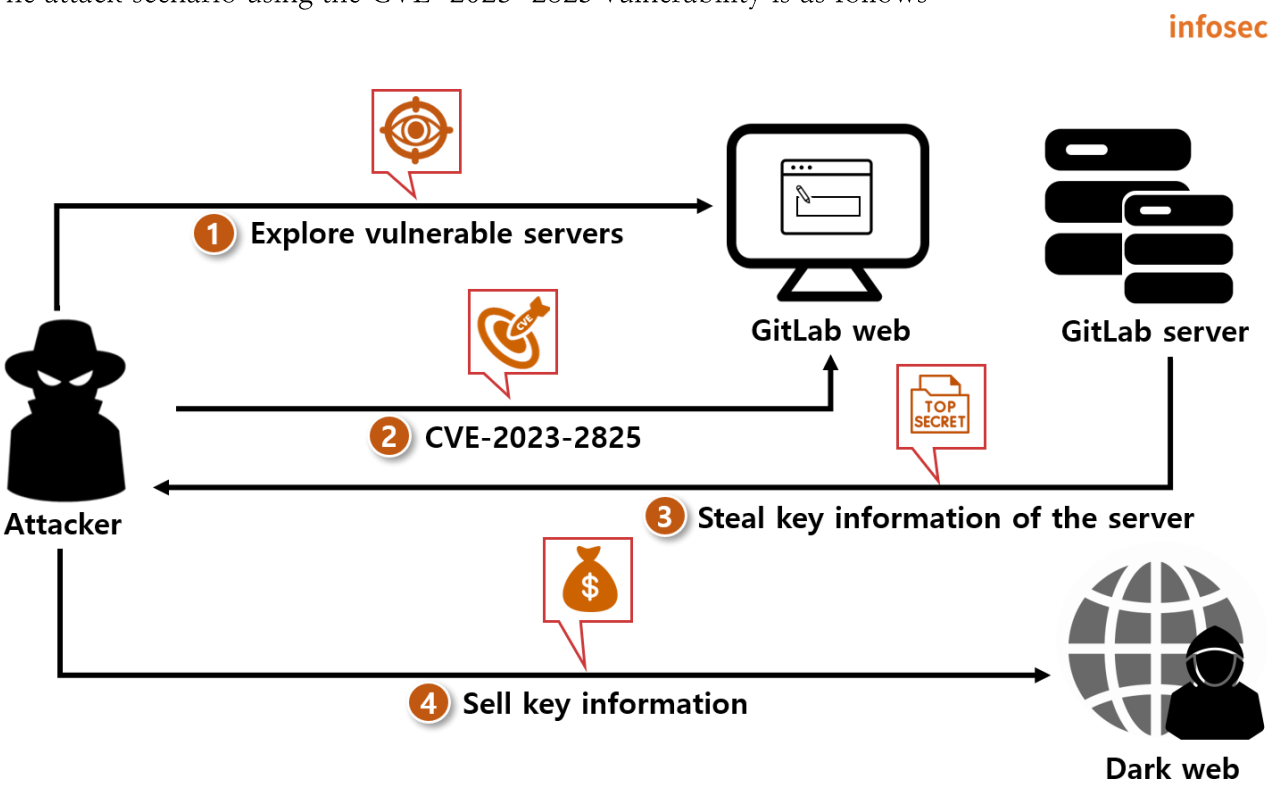


Figure 2. Attack scenario

① The attacker explores vulnerable GitLab web servers through the OSINT search engine.
② The attacker uses the CVE-2023-2825 vulnerability to access the victim's server.
③ Upon receiving the attacker's request, the server returns key information (development source codes, server environment configuration information, etc.) to the attacker.
④ The attacker sells the acquired key information to the dark web or other competitors.

■ Test environment configuration information

Build a test environment and examine the operation process of CVE-2023-2825

Name	Information
Victim	Ubuntu 20.04.5 LTS (192.168.100.162) GitLab 16.0.0
Attacker	Kali Linux 6.1.0-kali5-amd64 (192.168.100.152)

■ Vulnerability test

Step 1. Environment configuration

1) Build a server of GitLab 16.0.0 version with vulnerabilities among GitLab CE images supported by the docker hub on the victim's PC.

Command	$ docker run -d -p 80:80 gitlab/gitlab-ce:16.0.0-ce.0 -d option: An option for executing the docker as the background in the detach mode -p option: An option for designating the local port and the port to run in the docker

```
root@ubuntu:/home/eqst# docker run -d -p 80:80 gitlab/gitlab-ce:16.0.0-ce.0
Unable to find image 'gitlab/gitlab-ce:16.0.0-ce.0' locally
16.0.0-ce.0: Pulling from gitlab/gitlab-ce
1bc677758ad7: Pull complete
633fcf47bc79: Pull complete
472c1ac0c258: Pull complete
5b665b492973: Pull complete
0bd8b5a23fe7: Pull complete
b385dd2cb2ca: Pull complete
38ac4d68d24c: Pull complete
e4588a97b783: Pull complete
Digest: sha256:ab90cdb096c4f81247088357b0e051f5b8a999284b2186cbd1b1ec1a41cca7e8
Status: Downloaded newer image for gitlab/gitlab-ce:16.0.0-ce.0
3e524103ef6858b7825c530db4ce0d2dd3c1eb5f1e36776ef413574655d61784
```

Figure 3. Build the environment through the docker

2) To reset the password for the GitLab root account, open the terminal of the container and execute the following command:

Command	Container access command : $ docker exec -it [container name or container ID] /bin/bash Password change command : # gitlab-rake "gitlab:password:reset[root]"

```
root@ubuntu:/home/eqst# docker ps
CONTAINER ID   IMAGE                          COMMAND             CREATED
  STATUS                    PORTS
NAMES
3e524103ef68   gitlab/gitlab-ce:16.0.0-ce.0   "/assets/wrapper"   4 minutes ago
  Up 4 minutes (healthy)   22/tcp, 443/tcp, 0.0.0.0:80->80/tcp, :::80->80/tcp
distracted_heyrovsky
root@ubuntu:/home/eqst# docker exec -it 3e524103ef68 /bin/bash
root@3e524103ef68:/# gitlab-rake "gitlab:password:reset[root]"
Enter password:
Confirm password:
Password successfully updated for user with username root.
```

Command to access a container

Command to change password

Figure 4. Reset the password for the GitLab root account password

3) For the vulnerability test, copy the git file where PoC is saved to the attacker's PC.

GitHub URL where PoC is saved is as follows:

- URL: https://github.com/Occamsec/CVE-2023-2825.git

command	$ git clone https://github.com/Occamsec/CVE-2023-2825.git

```
┌──(root㉿kali)-[/home/kali]
└─# git clone https://github.com/Occamsec/CVE-2023-2825.git
Cloning into 'CVE-2023-2825' ...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (33/33), done.
remote: Total 36 (delta 11), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (36/36), 13.60 KiB | 2.72 MiB/s, done.
Resolving deltas: 100% (11/11), done.
```

Figure 5. Copy PoC and check the path

4) Use the editor to enter information on the victim's server in the PoC file.

※ The reason for entering the root account is to create a project for the PoC test. In actual vulnerabilities, users without authentication information can attack an open project.

```
┌──(root㉿kali)-[/home/kali/CVE-2023-2825]
└─# ls
DISCLAIMER.txt  poc.py  README.md

    import requests
    import random
    import string
    from urllib.parse import urlparse
    from bs4 import BeautifulSoup


    ENDPOINT = "https://192.168.100.162"
    USERNAME = "root"
    PASSWORD = "Eqst12#$"
```

Enter victim server information

Figure 6. Enter victim's server information

5) PoC operates and makes it possible to check the "/etc/passwd" file of the victim's server.

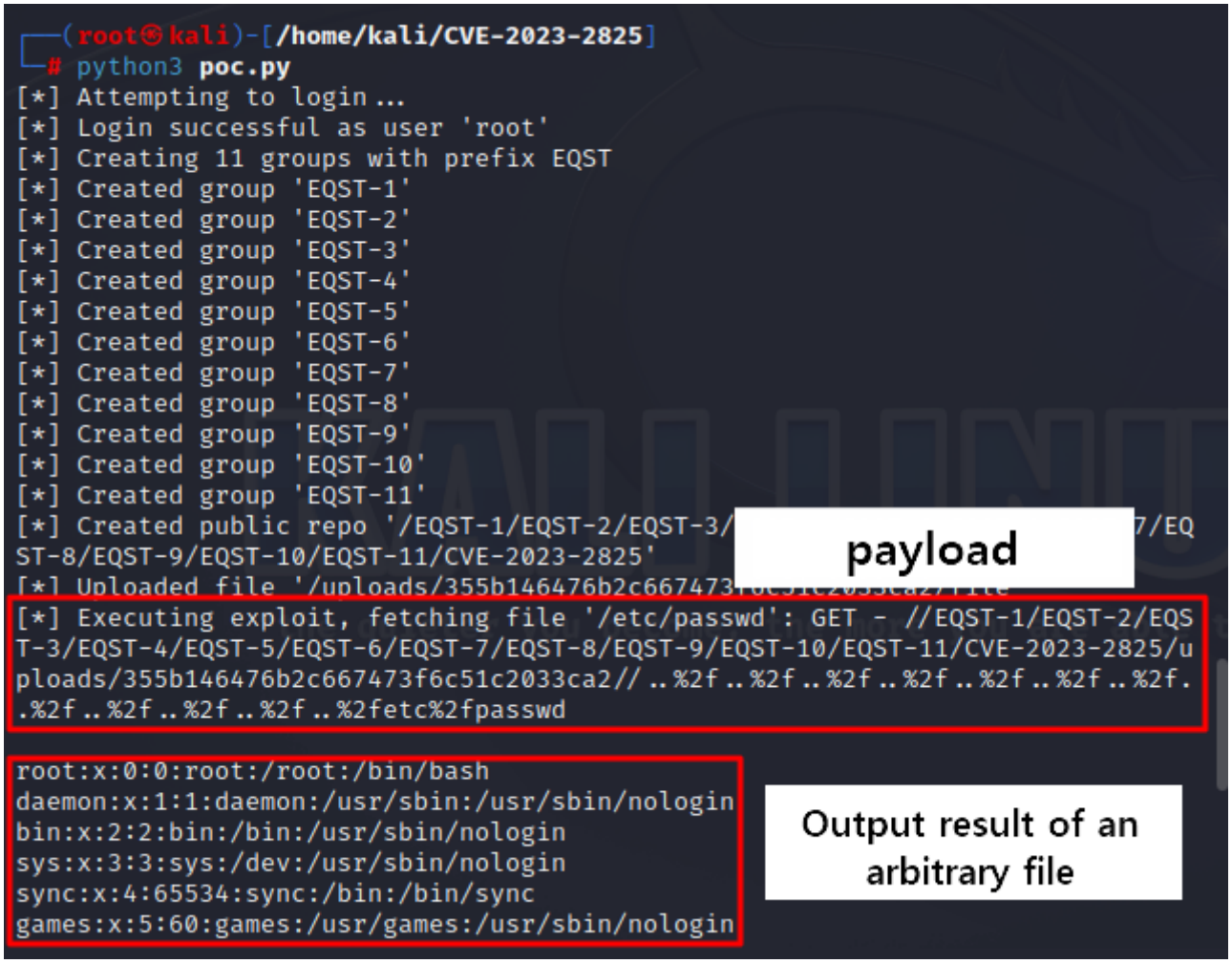


Figure 7. Exposure of a random file due to a vulnerability

■ Detailed analysis of the vulnerability

Step 1) Overview of the vulnerability

The CVE-2023-2825 vulnerability operates when an unauthenticated attacker accesses an attached file such as a GitLab project or Snippet[2] of an open vulnerable version. When the file does not exist in the requested URL path, the GitLab server processes it by decoding it after forwarding it to puma[3]. After that, the GitLab server retrieves the filename from the received URL. As the logic to check the filename is missing, however, a vulnerability occurs.

```
scope path: :uploads do
  # Note attachments and User/Group/Project/Topic avatars
  get "-/system/:model/:mounted_as/:id/:filename",
    to: "uploads#show",
    constraints: { model: %r{note|user|group|project|projects\/topic|achievements\/achievement},
    mounted_as: /avatar|attachment/, filename: %r{[^/]+} }
```

Figure 8. The source decodes through puma

When the attacker manipulates the packet and encodes and transmits the Path Traversal syntax to the attached filename, the server interprets the filename from the character string that has been decoded and processed. So a vulnerability occurs. Therefore, you can use the encoding character string "..%2f", "%2e%2e%2f" to access files in the parent directory.

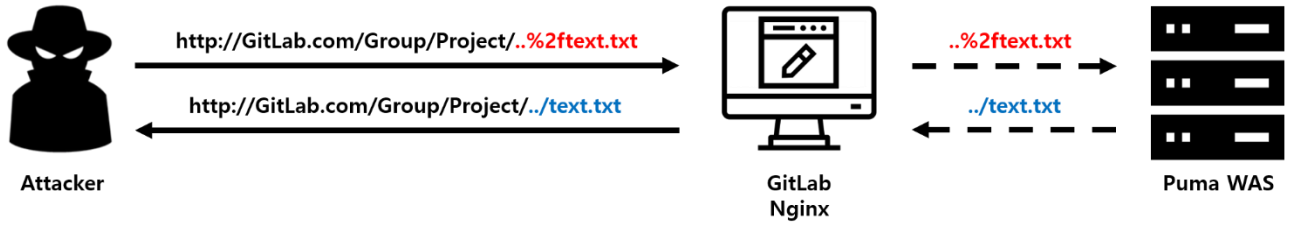


Figure 9. Illustration of decoding analysis

[2] A snippet is a page for storing frequently used codes or codes and texts to share with other users.

[3] As a type of WAS (Web Application Server), puma is a server for Ruby application programs. GitLab's Rails (a kind of Ruby's Web framework) is used to run application programs.

To understand the vulnerability, you need to know the storage structure related to GitLab's attached files. When creating a group project or snippet, the upload path and a specific directory path are combined and saved. The example below is the path that is saved on the server when a file is uploaded normally.

Figure 10. Storage path

However, when requesting the download of an attached file that does not exist in the download request URL path, GitLab transfers the file from the symbolic link[4] below through the puma server.

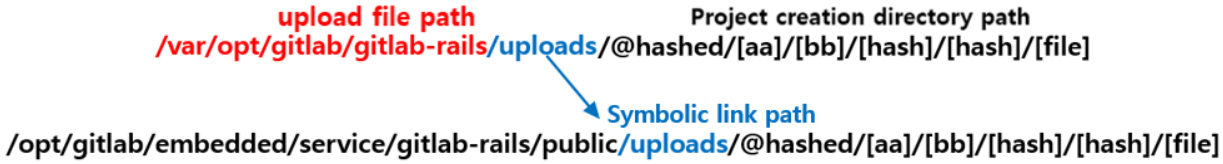


Figure 11. Symbolic link directory path

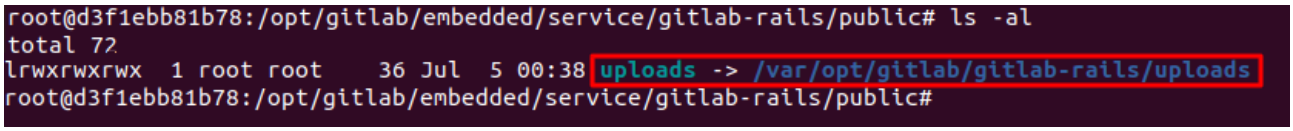


Figure 12. The symbolic link path of the server

The download request URL rule created when uploading an attached file in GitLab is as follows. Additional directories are created as many as the number of subgroups nested in the project.

GitLab address/[Group]/[Sub Group]/Project/uploads/[hash]/file

GitLab address/[Group]/[Sub Group1][Sub Group2]...[Sub Group N]/Project/uploads/[hash]/file

Figure 13. Download path request URL configuration

[4] A symbolic link is a file containing a reference to a specific file or directory, and functions like a shortcut in Windows OS.

If the download request URL has only one subgroup, as shown in the figure below, only the five paths of the WebRoot directory can be moved, and they can be moved only to the uploads directory of the symbolic link.

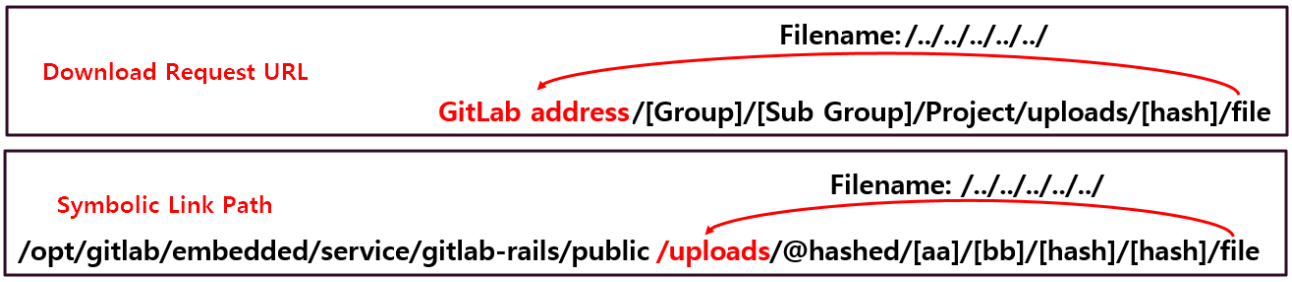


Figure 14. Go to the web root directory

However, since the download request URL creates as many directories as the number of nested subgroups, and the number of directories for symbolic links is fixed, it is possible to access directories higher than the Web Root directory.

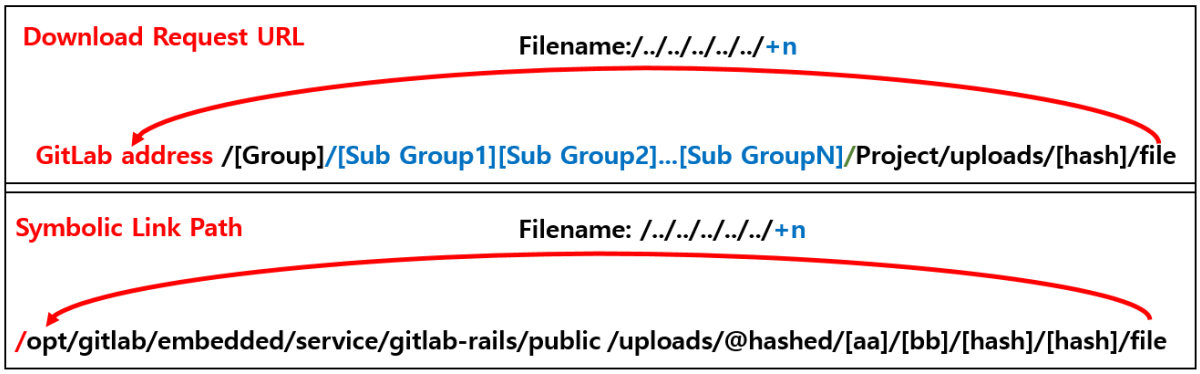


Figure 15. Go to the parent directory

The following figure is a diagrammatic representation of an example of a symbolic link referred to when the server receives an attached file download request.

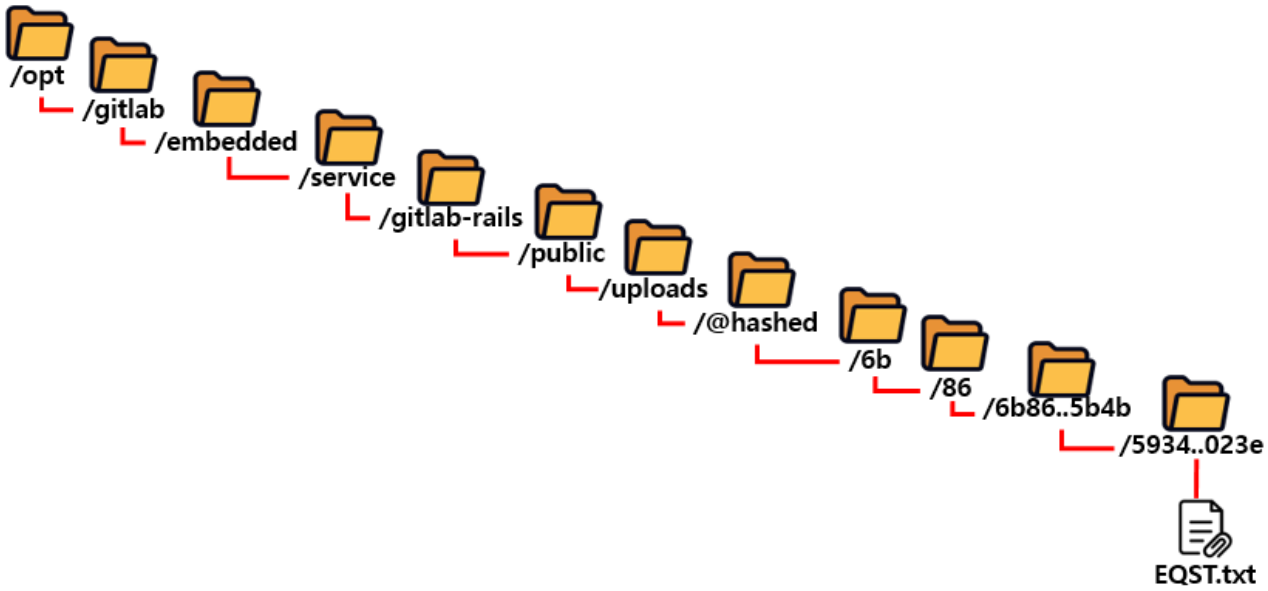


Figure 16. A diagrammatic representation of an example of a symbolic link

Step 2) Detailed analysis of operation

For detailed analysis of the vulnerability, create a group (EQSTLab) in the vulnerable version's GitLab and create a public project (Insight).

※ To analyze the vulnerability, a print.txt file that outputs the current path was created in each directory.

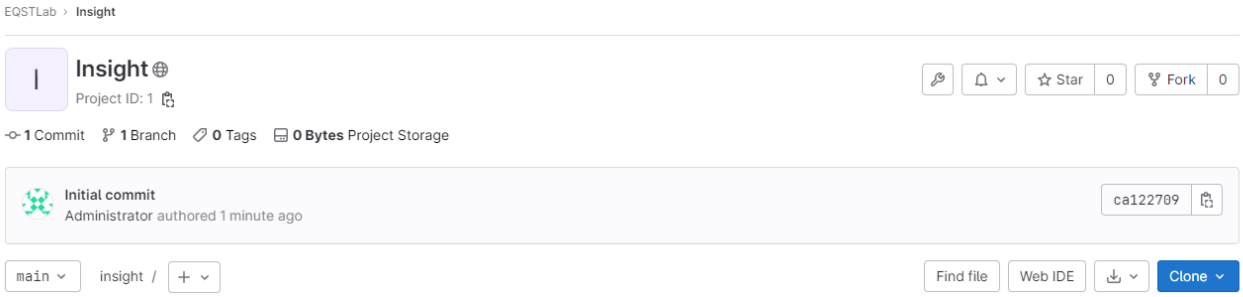


Figure 17. Creation screen

After creating the project, in order to exploit the attached file, create an issue, a space where you can write contents related to the project, and upload the attached file (EQST.txt).

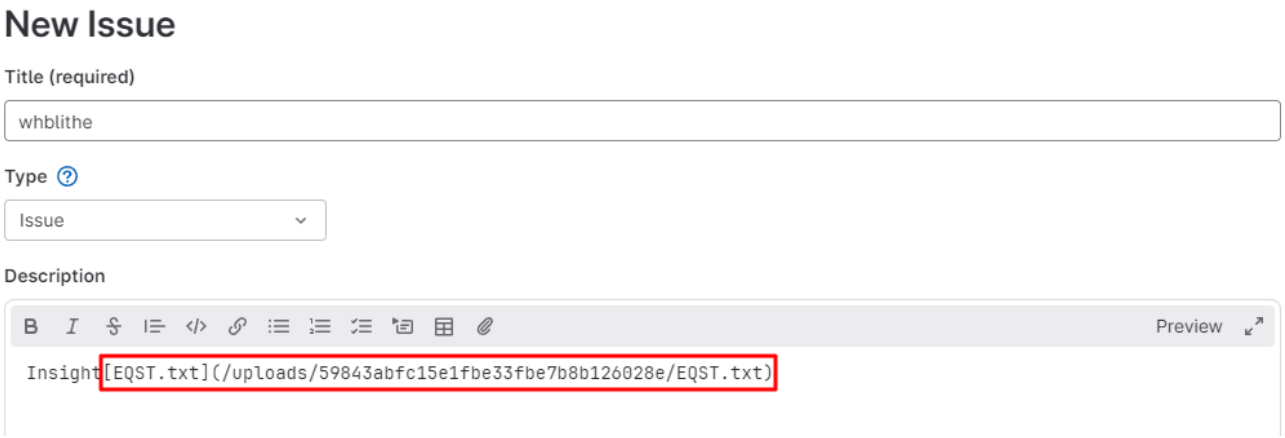


Figure 18. Upload the attached file

Download the attached file (EQST.txt) through the path below.

- http://192.168.100.162/eqstlab/insight/uploads/59843abfc15e1fbe33fbe7b8b126028e/EQST.txt

Figure 19. Download the file

To exploit the CVE-2023-2825 vulnerability, if you use a proxy tool to modify the file name, perform URL encoding for the "../"character string and deliver the URL-encode the "..%2fprint.txt" or "%2e%2e%2fprint.txt" payload to the victimized server, you can reach the upper path.

The response value of print.txt, which displays the current path of the parent path using the proxy tool, is as follows:

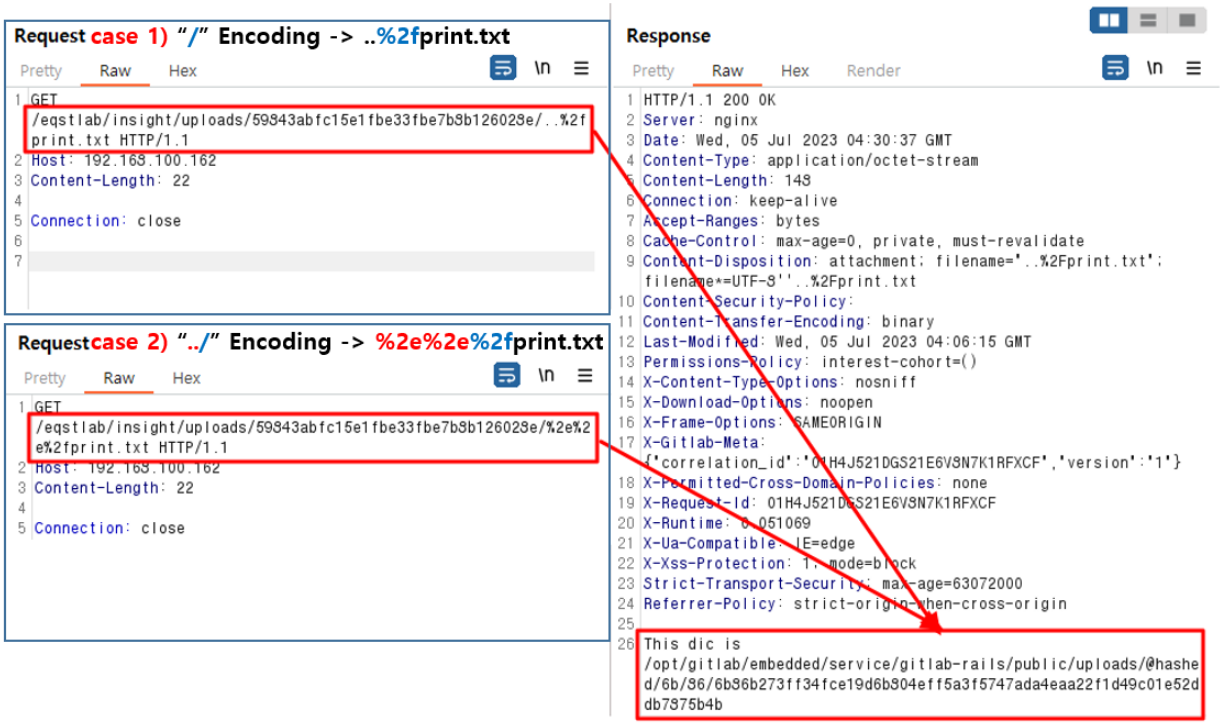


Figure 20. Go to the parent path and display file information

As it repeats moving to the upper path one step at a time, and when moving to the top five directories, a 400 Bad Request error is returned. Since this is a project configuration that does not create any subgroup, the download request URL includes only four directories:

/eqstlab/insight/uploads/59843abfc15e1fbe33fbe7b8b126028e/. Therefore, it is impossible to move up five directories higher than the WebRoot directory.

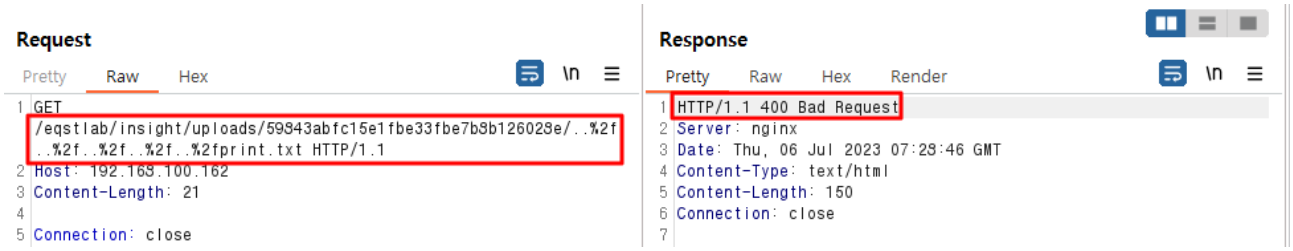


Figure 21. Return an error

The figure below shows the process of returning a 400 Bad Request error when moving five paths, a directory higher than the WebRoot directory.

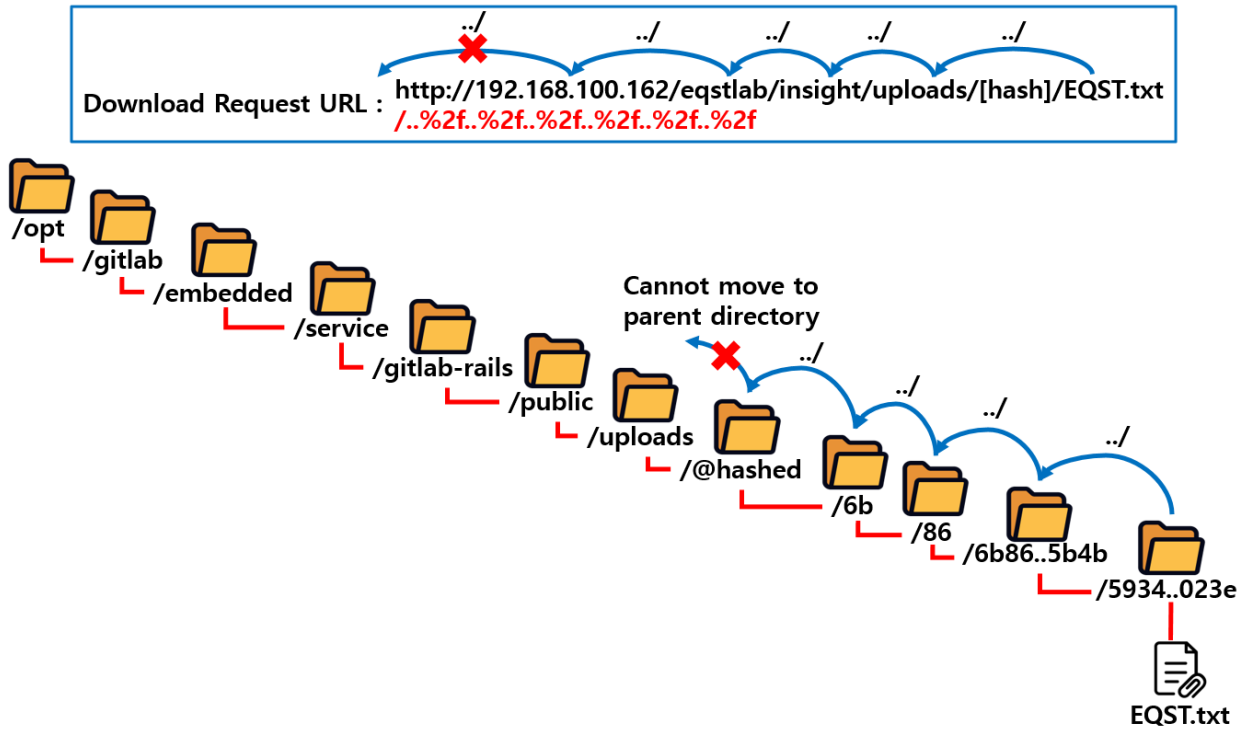


Figure 22. A diagrammatic representation of “Unable to move”

To access a directory higher than the WebRoot directory by exploiting the CVE-2023-2825 vulnerability, you must add nested subgroups to increase the number of directories included in the download request URL.

Therefore, in order to access "/opt/gitlab/embedded/service/gitlab-rails/config/secrets.yml," which is one of the main information items located in a directory higher than WebRoot, you must move up a total of seven higher directories by adding three to the existing four directories. Therefore, you will exploit the CVE-2023-2825 vulnerability by adding three nested subgroups.

```
root@d3f1ebb81b78:/opt/gitlab/embedded/service/gitlab-rails# ls -al config/ | grep secrets.yml
lrwxrwxrwx 1 root root     44 Jul  5 00:38 secrets.yml -> /var/opt/gitlab/gitlab-rails/etc/secrets.yml
-rw-r--r-- 1 root root    404 May 18 18:02 secrets.yml.example
```

Figure 23. "/config/secrets.yml" path within the server

In the figure below, three nested subgroups were created for access to seven higher directories.

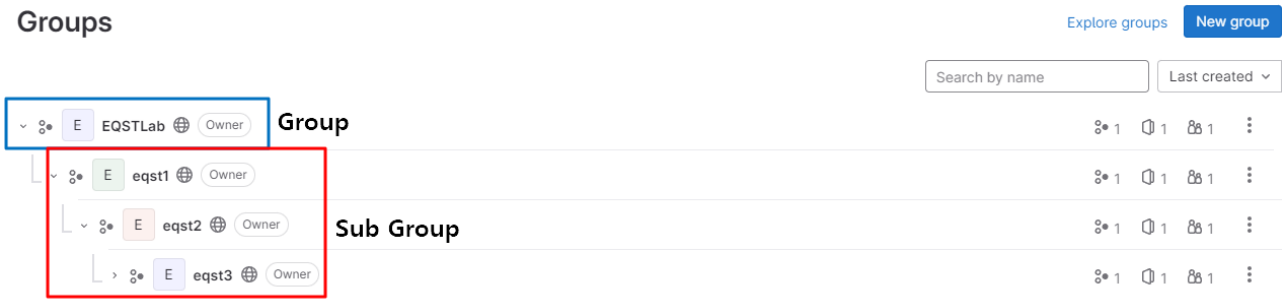


Figure 24. Create subgroups

The figure below illustrates the process of accessing the "/config/secrets.yml" file in gitlab-rails, a directory higher than the WebRoot directory, by adding three nested subgroups.

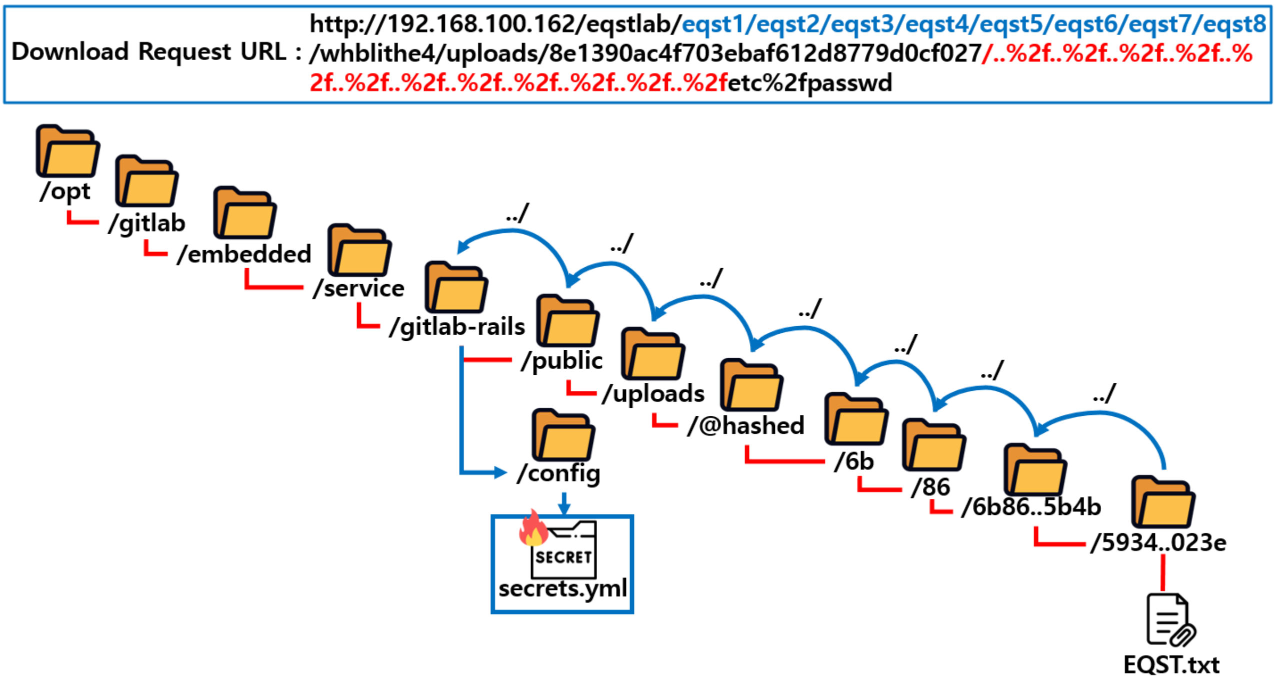


Figure 25. A diagrammatic representation of "/config/secrets.yml"

The payload that outputs "/config/secrets.yml" information after creating nested subgroups through the previous process is as follows:

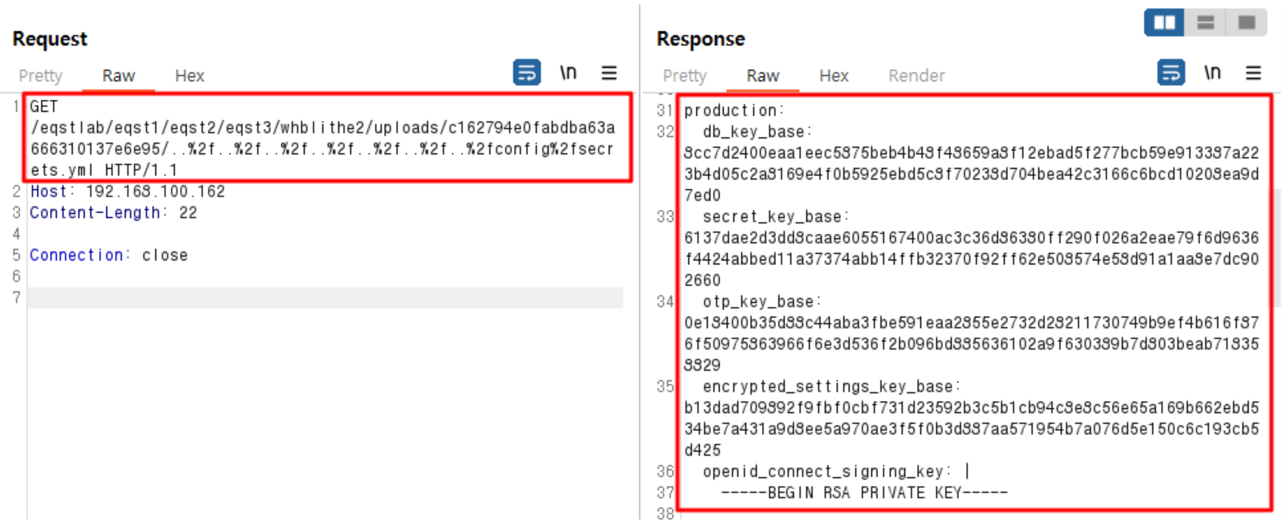


Figure 26. Access the "/config/secrets.yml" file of the parent directory

Therefore, in order to access the top-level root directory, you have to move up a total of 12 upper directories. So you can access "/etc/passwd" by manipulating the attached file path of one group and eight subgroup projects.

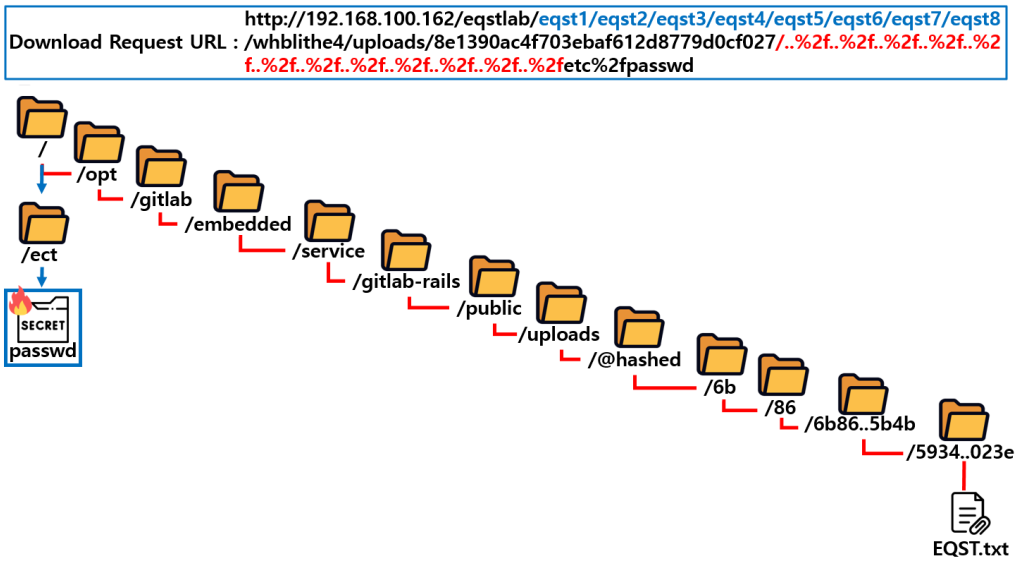


Figure 27. Access the root directory

Through the previous process, you can move up 12 higher directories, move to the top root directory, and output "/etc/passwd" or download it.

Figure 28. Get the "/etc/passwd" file information of the root folder

If you enter it in the URL address, you can download the "/etc/passwd" file.

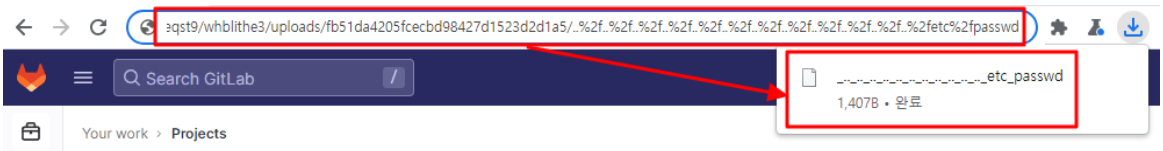


Figure 29. Download the "/etc/passwd" file

■ Countermeasure

If you are operating a vulnerable version of GitLab server, an attacker can exploit the vulnerability by creating an issue in the project or registering an attached file in a public snippet. In order to cope with this, it is safe to update it to GitLab 16.0.1 or higher with logic that checks whether the character string decoded based on a regular expression is a 'path traversal pattern.'

```
def check_path_traversal!(path)
  return unless path

  path = path.to_s if path.is_a?(Gitlab::HashedPath)
  raise PathTraversalAttackError, 'Invalid path' unless path.is_a?(String)

  path = decode_path(path)
  path_regex = %r{(\A(\.{1,2})\z|\A\.\.[/\\]|[/\\]\.\.\z|[/\\]\.\.[/\\]|\n)}

  if path.match?(path_regex)
    logger.warn(message: "Potential path traversal attempt detected", path: "#{path}")
    raise PathTraversalAttackError, 'Invalid path'
  end

  path
end
```

Figure 30. Path traversal detection via regular expression

In version 16.0.1 or later, it can be confirmed that bad_reqeust is returned when check_path_traversal logic is added in the module involved in upload, and path traversal is detected.

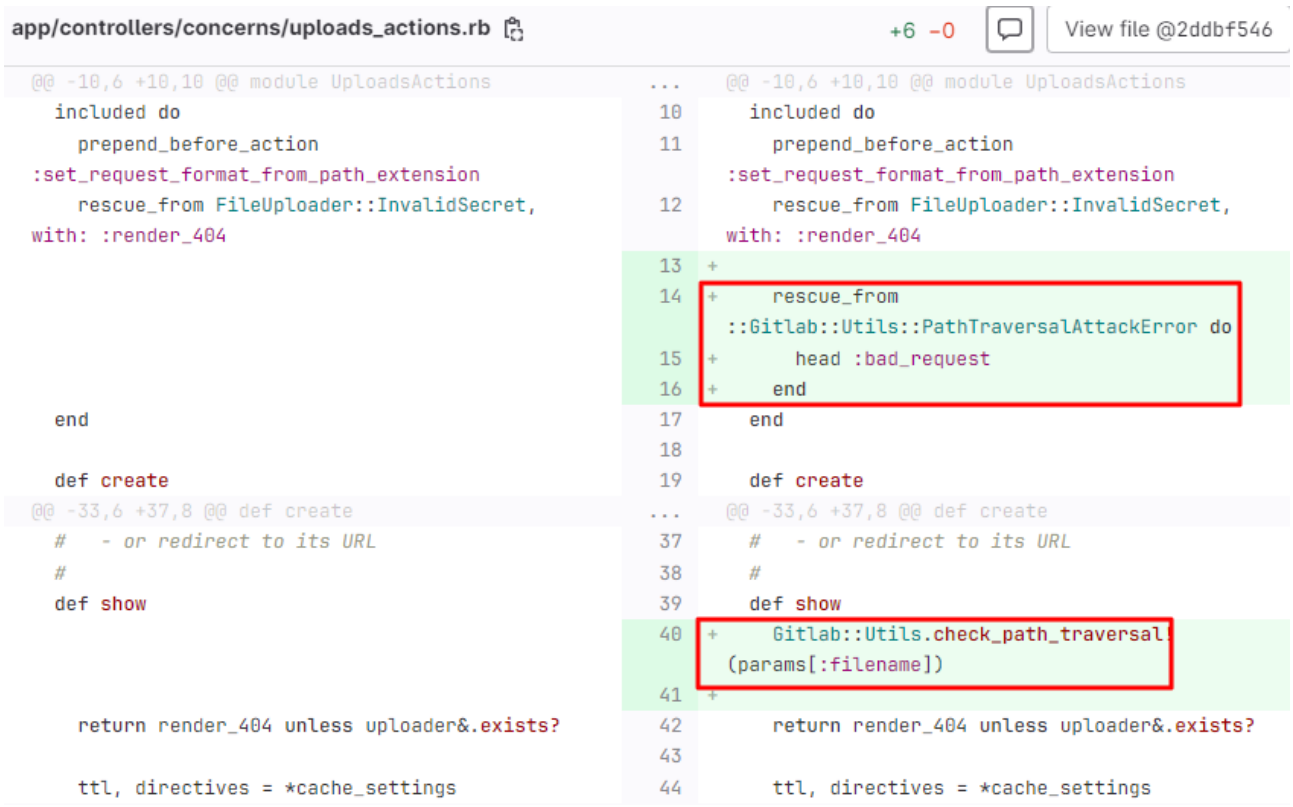


Figure 31. Add inspection logic

If it is difficult to update the version, it is necessary to disable the open project or inspect the packet using the web firewall. However, since this is not a perfect countermeasure for vulnerabilities, it is recommended to update to version 16.0.01 or higher with the inspection logic added.

■ Reference sites

- URL: https://labs.watchtowr.com/gitlab-arbitrary-file-read-gitlab-cve-2023-2825-analysis/
- URL: https://github.com/Occamsec/CVE-2023-2825.git
- URL: https://about.gitlab.com/releases/2023/05/23/critical-security-release-gitlab-16-0-1-released/