

Rendu TP02

Rendu du TP02 par Vincent LAGOGUE et Thomas PEUGNET.

Mise en place de l'environnement de travail

Nous créons notre environnement et obtenons l'architcture suivante.

```
● ● ●  ☺ 1 thomas@MacBook-Pro-de-Thomas:~/Desktop/Container_tmp_tp02
└─ thomas@MacBook-Pro-de-Thomas ~/Desktop/Container_tmp_tp02
    └── tree . -La 3

.
└── flaskapp
    ├── db
    │   └── init.sql
    ├── docker-compose.yml
    └── web
        ├── Dockerfile
        ├── app.py
        └── requirements.txt

5 directories, 4 files
└─ thomas@MacBook-Pro-de-Thomas ~/Desktop/Container_tmp_tp02
```

Notre `docker-compose.yml` a le contenu suivant.

Nous lançons notre application avec le démarrage de nos conteneurs.

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

Nous constatons que l'application tourne correctement et que la base de données fonctionne correctement.

```

● ● ●  ☺ 2 sudo docker compose logs -f
db-1 | CREATE DATABASE
db-1 |
db-1 |
db-1 | /usr/local/bin/docker-entrypoint.sh: running /docker-entrypoint-initdb.d/init.sql
db-1 |
db-1 |
db-1 | waiting for server to shut down....2025-01-29 13:21:13.439 UTC [41] LOG: received fast shutdown request
db-1 | 2025-01-29 13:21:13.445 UTC [41] LOG: aborting any active transactions
db-1 | 2025-01-29 13:21:13.447 UTC [41] LOG: background worker "logical replication launcher" (PID 48) exited with exit code 1
db-1 | 2025-01-29 13:21:13.447 UTC [43] LOG: shutting down
db-1 | 2025-01-29 13:21:13.486 UTC [41] LOG: database system is shut down
db-1 | done
db-1 | server stopped
db-1 |
db-1 | PostgreSQL init process complete; ready for start up.
db-1 |
db-1 | 2025-01-29 13:21:13.565 UTC [1] LOG: starting PostgreSQL 13.18 on x86_64-pc-linux-musl, compiled by gcc (Alpine 14.2.0) 14.2.0, 64-bit
db-1 | 2025-01-29 13:21:13.565 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db-1 | 2025-01-29 13:21:13.565 UTC [1] LOG: listening on IPv6 address "::", port 5432
db-1 | 2025-01-29 13:21:13.576 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.PGSQL.5432"
db-1 | 2025-01-29 13:21:13.588 UTC [58] LOG: database system was shut down at 2025-01-29 13:21:13 UTC
db-1 | 2025-01-29 13:21:13.596 UTC [1] LOG: database system is ready to accept connections

```

Nous modifions notre fichier `init.sql` pour avoir le contenu suivant. Ce contenu a été généré volontairement par Mistra AI.

```

-- Empty initialization file for now
-- Création de la base de données si elle n'existe pas
-- CREATE DATABASE mydb;
\c mydb;

-- Création de la table des utilisateurs avec une vulnérabilité classique (pas de préparation de requête)
DROP TABLE IF EXISTS users;
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(255) NOT NULL, -- Mauvaise pratique : stockage en clair
    email VARCHAR(100),
    role VARCHAR(20) DEFAULT 'user'
);

-- Insertion de quelques utilisateurs
INSERT INTO users (username, password, email, role) VALUES
('admin', 'admin123', 'admin@example.com', 'admin'),
('user1', 'password', 'user1@example.com', 'user'),
('user2', '123456', 'user2@example.com', 'user'),
('evil_hacker', 'p@ssw0rd', 'hacker@darkweb.com', 'user');

```

```
-- Table des logs (idéal pour tester l'injection dans les recherches)
DROP TABLE IF EXISTS logs;
CREATE TABLE logs (
    id SERIAL PRIMARY KEY,
    action TEXT,
    user_id INT REFERENCES users(id),
    timestamp TIMESTAMP DEFAULT NOW()
);

-- Exemples d'entrées de logs
INSERT INTO logs (action, user_id) VALUES
('User admin logged in', 1),
('User user1 changed password', 2),
('User user2 attempted login', 3),
('Evil hacker tried SQL injection', 4);
```

Nous modifions également notre fichier `app.py` pour avoir le contenu suivant.

```
from flask import Flask, request, jsonify
import psycopg2
import logging

logging.basicConfig(filename='/app/logs/access.log', level=logging.INFO)

app = Flask(__name__)

# Connexion à PostgreSQL (⚠️ vulnérable car pas de paramètre sécurisé)
DB_CONFIG = {
    "dbname": "mydb",
    "user": "user",
    "password": "password",
    "host": "db",
    "port": 5432
}

def get_db_connection():
    return psycopg2.connect(**DB_CONFIG)

@app.route('/')
def home():
    return "🚀 API Flask vulnérable aux injections SQL"

# 🔥 1 Authentification vulnérable 🔥
@app.route('/login', methods=['POST'])
def login():
    data = request.json
    username = data.get('username', '')
    password = data.get('password', '')
```

```

conn = get_db_connection()
cursor = conn.cursor()

# ! Vulnérable aux injections SQL
query = f"SELECT id, username, role FROM users WHERE username = '{username}' AND password = '{password}'"
logging.info(f"[DEBUG] Query exécutée: {query}") # Pour voir ce qui est injecté
cursor.execute(query)
user = cursor.fetchone()

cursor.close()
conn.close()

if user:
    return jsonify({"message": "Connexion réussie", "user": user}), 200
return jsonify({"message": "Échec de l'authentification"}), 401

# 🔥 2 Recherche d'utilisateurs vulnérable 🔥
@app.route('/search', methods=['GET'])
def search_users():
    search = request.args.get('q', '')

    conn = get_db_connection()
    cursor = conn.cursor()

    # ! Vulnérable aux injections SQL
    query = f"SELECT id, username, email FROM users WHERE username LIKE '%{search}%'"
    logging.info(f"[DEBUG] Query exécutée: {query}")
    cursor.execute(query)
    results = cursor.fetchall()

    cursor.close()
    conn.close()

    return jsonify(results), 200

# 🔥 3 Affichage des logs vulnérable 🔥
@app.route('/logs', methods=['GET'])
def get_logs():
    filter_log = request.args.get('filter', '')

    conn = get_db_connection()
    cursor = conn.cursor()

    # ! Vulnérable aux injections SQL
    query = f"SELECT id, action, timestamp FROM logs WHERE action LIKE '%{filter_log}%'"
    logging.info(f"[DEBUG] Query exécutée: {query}")

```

```

cursor.execute(query)
logs = cursor.fetchall()

cursor.close()
conn.close()

return jsonify(logs), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

```

Falco

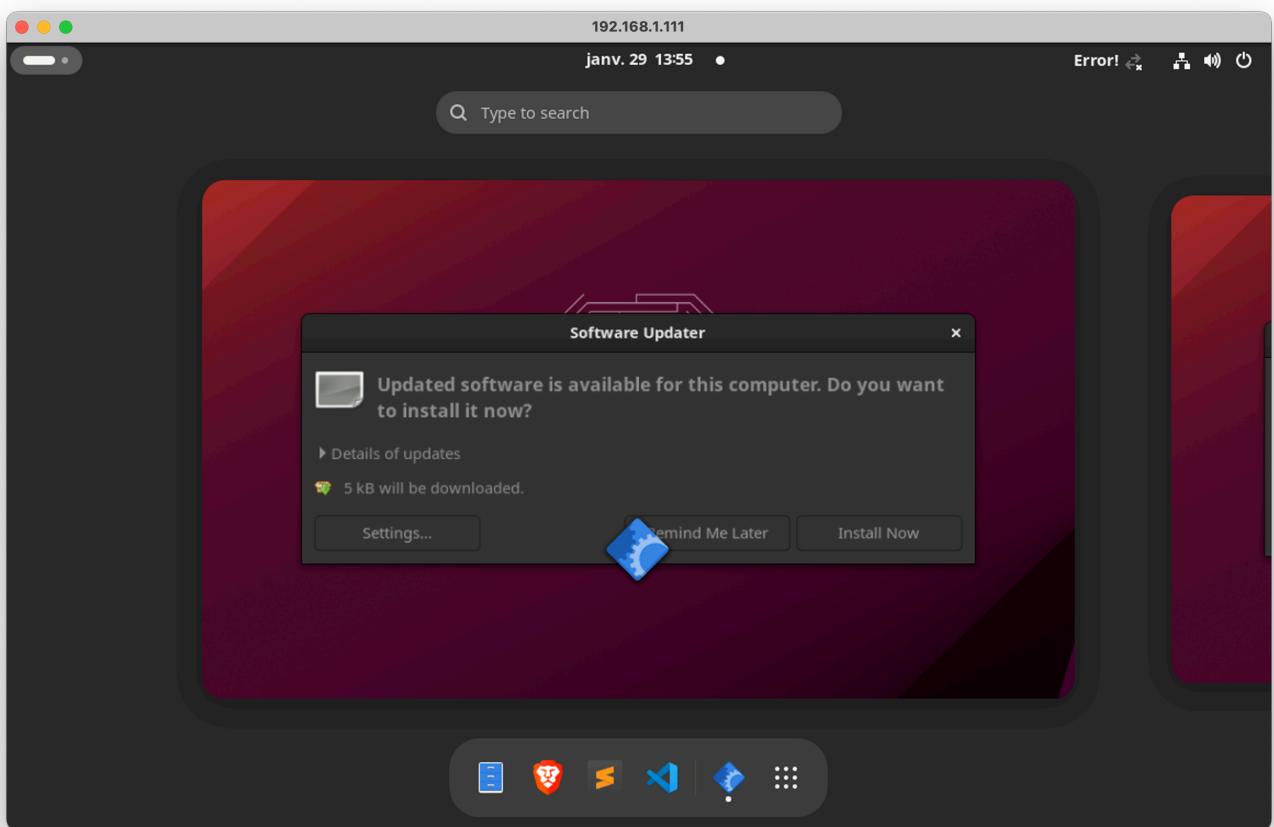
Nous lançons Falco à l'aide de la commande suivante.

```

docker run --rm -it \
--privileged \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /dev:/host/dev \
-v /proc:/host/proc:ro \
-v /boot:/host/boot:ro \
-v /lib/modules:/host/lib/modules:ro \
-e FALCO_UI_OUTPUT=/tmp/falco-events.json \
falcosecurity/falco:latest

```

Nous remarquons que, après plusieurs tentatives, `falco` ne peut pas fonctionner sur un système macOS. Nous nous connectons donc à une VM sur un hyperviseur Proxmox pour continuer la suite du TP.



Nous relançons la commande Docker.

```
user@xubuntu-desktop ~
└─$ sudo docker run --rm -it --privileged -v -v /dev:/host/dev -v -v -e FALCO_UI_OUTPUT=/tmp/falco-events.json -e FALCO_EVENTS_JSON=falco:latest
[falco] password for user:
2025-01-29T13:14:16+0000: Falco version: 0.40.0 (x86_64)
2025-01-29T13:14:16+0000: Falco initialized with configuration files:
2025-01-29T13:14:16+0000:     /etc/falco/falco.yaml | schema validation: ok
2025-01-29T13:14:16+0000: System info: Linux version 6.5.0-13-generic (buildd@lcy02-amd64-053) (x86_64-linux-gnu-gcc-13 (Ubuntu 13.2.0-4ubuntu3) 13.2.0, GNU ld (GNU Binutils for Ubuntu) 2.41) #13-Ubuntu SMP PREEMPT_DYNAMIC Fri Nov 3 12:16:05 UTC 2023
2025-01-29T13:14:16+0000: Loading rules from:
2025-01-29T13:14:16+0000:     /etc/falco/falco_rules.yaml | schema validation: ok
2025-01-29T13:14:16+0000:     /etc/falco/falco_rules.local.yaml | schema validation: none
2025-01-29T13:14:16+0000: The chosen syscall buffer dimension is: 8388608 bytes (8 MBs)
2025-01-29T13:14:16+0000: Starting health webserver with threadiness 6, listening on 0.0.0.0:8765
2025-01-29T13:14:16+0000: Loaded event sources: syscall
2025-01-29T13:14:16+0000: Enabled event sources: syscall
2025-01-29T13:14:16+0000: Opening 'syscall' source with modern BPF probe.
2025-01-29T13:14:16+0000: One ring buffer every '2' CPUs.
```

Attaques

Injection SQL via HTTP

Nous lançons une première attaque, étant la requête `curl` suivante. Cette attaque fait une requête SQL au travers d'une requête POST faite à notre serveur web.

```

user@xubuntu-desktop ~
curl "http://localhost:5000/search?q=%27%20UNION%20SELECT%20id,%20username,%20password%20FROM%20users%20--"
[{"id": 3, "username": "user2", "password": "123456"}, {"id": 2, "username": "user1", "password": "user1@example.com"}, {"id": 3, "username": "user2", "password": "user2@example.com"}, {"id": 4, "username": "evil_hacker", "password": "p@ssw0rd"}, {"id": 1, "username": "admin", "password": "admin123"}, {"id": 1, "username": "admin", "password": "admin@example.com"}, {"id": 2, "username": "user1", "password": "password"}, {"id": 4, "username": "evil_hacker", "password": "hacker@darkweb.com"}]

```

[sudo] password for user:

```

2025-01-29T13:40:51+0000: Falco version: 0.40.0 (x86_64)
2025-01-29T13:40:51+0000: Falco initialized with configuration files:
2025-01-29T13:40:51+0000:   /etc/falco/falco.yaml | schema validation: ok
2025-01-29T13:40:51+0000: System info: Linux version 6.5.0-13-generic (build@lcy02-amd64-053) (x86_64-linux-gnu-gcc-13 (Ubuntu 13.2.0-4ubuntu3) 13.2.0, GNU ld (GNU Binutils for Ubuntu) 2.41) #13-Ubuntu SMP PREEMPT_DYNAMIC Fri Nov 3 12:16:05 UTC 2023
2025-01-29T13:40:51+0000: Loading rules from:
2025-01-29T13:40:51+0000:   /etc/falco/falco.rules.yaml | schema validation: ok
2025-01-29T13:40:51+0000:   /etc/falco/falco.rules.local.yaml | schema validation: none
2025-01-29T13:40:51+0000: The chosen syscall buffer dimension is: 8388608 bytes (8 MBs)
2025-01-29T13:40:51+0000: Starting health webserver with threadiness 6, listening on 0.0.0.0:8765
2025-01-29T13:40:51+0000: Loaded event sources: syscall
2025-01-29T13:40:51+0000: Enabled event sources: syscall
2025-01-29T13:40:51+0000: Opening 'syscall' source with modern BPF probe.
2025-01-29T13:40:51+0000: One ring buffer every '2' CPUs.

```

`sudo docker compose down && sudo docker compose up -d --build && sudo docker`

```

db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG: starting PostgreSQL 13.18 on x86_64-pc-linux-musl, compiled by gcc (Alpine 14.2.0) 14.2.0, 64-bit
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG:  listening on IPv6 address "::", port 5432
db-1  | 2025-01-29 13:39:59.603 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1  | 2025-01-29 13:39:59.623 UTC [27] LOG:  database system was shut down at 2025-01-29 13:35:25 UTC
db-1  | 2025-01-29 13:39:59.634 UTC [1] LOG:  database system is ready to accept connections
web-1  | * Serving Flask app 'app'
web-1  | * Debug mode: on
web-1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
web-1  | * Running on all addresses (0.0.0.0)
web-1  | * Running on http://127.0.0.1:5000
web-1  | * Running on http://172.18.0.3:5000
web-1  | Press CTRL+C to quit
web-1  | * Restarting with stat
web-1  | * Debugger is active!
web-1  | * Debugger PIN: 620-223-127
web-1  | 172.18.0.1 -- [29/Jan/2025 13:40:32] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 -- [29/Jan/2025 13:41:51] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 -- [29/Jan/2025 13:43:18] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 -- [29/Jan/2025 13:43:25] "GET /search?q='%20UNION%20SELECT%20id,%20username,%20password%20FROM%20users%20-- HTTP/1.1" 200 -

```

Nous constatons que l'attaque n'a pas été détectée par Falco (absence de changement dans les logs, fenêtre en haut à droite).

Connexion en shell

Nous effetuons maintenant une connexion via un `docker exec` sur le conteneur de notre application web à l'aide de la commande suivante.

```
sudo docker exec -it flaskapp-web-1 /bin/bash
```

```

user@ubuntu-desktop ~%1 sudo docker exec -it flaskapp-web-1 /bin/bash
[sudo] password for user:
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
f17b76d7dcbe      falcosecurity/falco:latest   "/usr/bin/falco"   8 seconds ago    Up 7 seconds       sweet_davinc
i
21090e8b304a      flaskapp-web           "python app.py"    31 minutes ago   Up 31 minutes     flaskapp-web-1
cibe007ccf41      postgres:13-alpine      "docker-entrypoint.s..." 31 minutes ago   Up 31 minutes     flaskapp-db-1
user@ubuntu-desktop ~%1 sudo docker exec -it flaskapp-web-1 /bin/bash
root@21090e8b304a:/app# 

● ● ●  %2 sudo docker run --rm -it --privileged -v -v /dev:/host/dev -v -v -v
-sudo docker run --rm -it --privileged -v -v /dev:/host/dev -v -v -v -e
-fALCO_U1_OUTPUT=/tmp/falco-events.json \
-falcosecurity/falco:latest
2025-01-29T14:11:16+0000: Falco version: 0.40.0 (x86_64)
2025-01-29T14:11:16+0000: Falco initialized with configuration files:
2025-01-29T14:11:16+0000:   /etc/falco/falco.yaml | schema validation: ok
2025-01-29T14:11:16+0000:   System info: Linux version 6.5.0-13-generic (buildd@lcy02-amd64-053)
(x86_64-linux-gnu-gcc-13 (Ubuntu 13.2.0-4ubuntu3) 13.2.0, GNU ld (GNU Binutils for Ubuntu) 2.
41) #13-Ubuntu SMP PREEMPT_DYNAMIC Fri Nov 3 12:16:05 UTC 2023
2025-01-29T14:11:16+0000: Loading rules from:
2025-01-29T14:11:16+0000:   /etc/falco/falco.rules.yaml | schema validation: ok
2025-01-29T14:11:16+0000:   /etc/falco/falco.rules.local.yaml | schema validation: none
2025-01-29T14:11:16+0000: The chosen syscall buffer dimension is: 8388608 bytes (8 MBs)
2025-01-29T14:11:16+0000: Starting health webserver with threadiness 6, listening on 0.0.0.0:8
765
2025-01-29T14:11:16+0000: Loaded event sources: syscall
2025-01-29T14:11:16+0000: Enabled event sources: syscall
2025-01-29T14:11:16+0000: Opening 'syscall' source with modern BPF probe.
2025-01-29T14:11:16+0000: One ring buffer every '2' CPUs.
2025-01-29T14:11:16+0000: Notice A shell was spawned in a container with an attached
terminal (evt_type=execve user=root user_uid=0 user_loginuid=-1 process=bash proc_exepath=/u
s/r/bin/bash parent=containerd-shim command=bash terminal=34816 exe_flags=EXE_WRITABLE|EXE_LoWE
r_LAYER container_id=21090e8b304a container_name=<NA>

```

```

● ● ●  %2 sudo docker compose down && sudo docker compose up -d --build && sudo docker
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG: starting PostgreSQL 13.18 on x86_64-pc-linux-mu
nsl, compiled by gcc (Alpine 14.2.0) 14.2.0, 64-bit
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG:  listening on IPv6 address "::", port 5432
db-1  | 2025-01-29 13:39:59.603 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1  | 2025-01-29 13:39:59.623 UTC [27] LOG:  database system was shut down at 2025-01-29 13
:35:25 UTC
db-1  | 2025-01-29 13:39:59.634 UTC [1] LOG:  database system is ready to accept connections
web-1  | * Serving Flask app 'app'
web-1  | * Debug mode: on
web-1  | WARNING: This is a development server. Do not use it in a production deployment. Use a
production WSGI server instead.
web-1  | * Running on all addresses (0.0.0.0)
web-1  | * Running on http://127.0.0.1:5000
web-1  | * Running on http://172.18.0.3:5000
web-1  | Press CTRL+C to quit
web-1  | * Restarting with stat
web-1  | * Debugger is active!
web-1  | * Debugger PIN: 620-223-127
web-1  | 172.18.0.1 -- [29/Jan/2025 13:40:32] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 -- [29/Jan/2025 13:41:51] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 -- [29/Jan/2025 13:43:18] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 -- [29/Jan/2025 13:43:25] "GET /search?q=%20UNION%20SELECT%20id,%20username,%20password%20FROM%20users%20-- HTTP/1.1" 200 -

```

Nous constatons cette fois-ci que cette manipulation a bien été détectée par Falco, nous avons le résultat suivant.

```
2025-01-29T14:11:41.919363331+0000: Notice A shell was spawned in a container with an
attached terminal (evt_type=execve user=root user_uid=0 user_loginuid=-1 process=bash
proc_exepath=/usr/bin/bash parent=containerd-shim command=bash terminal=34816
exe_flags=EXE_WRITABLE|EXE_LOWER_LAYER container_id=21090e8b304a container_name=<NA>)
```

Un shell (`bash`) a été lancé à l'intérieur d'un conteneur.

- L'utilisateur `root` (`user=root user_uid=0`) a exécuté cette commande, ce qui indique un accès avec des privilèges root.
- Le processus parent est `containerd-shim`, ce qui signifie que le shell a été lancé directement via Docker ou un runtime de conteneur.
- Le terminal est attaché (`terminal=34816`), ce qui veut dire que la session est interactive.
- Le fichier exécutable (`proc_exepath=/usr/bin/bash`) est situé dans `/usr/bin/`.

Injection SQL via un Shell

Nous tentons une nouvelle façon de faire une injection, avec une commande `docker exec`.

```
sudo docker exec -it flaskapp-db-1 psql -U user -d mydb -c "SELECT * FROM users WHERE
username = 'admin' OR '1'='1';"
```

```

user@ubuntu-desktop ~/flaskapp
└─$ sudo docker exec -it flaskapp-db-1 psql -U user -d mydb -c "SELECT * FROM users WHERE username = 'admin' OR '1='1';"
   id |  username  | password |      email       | role
-----+-----+-----+-----+-----+
   1 | admin     | admin123 | admin@example.com | admin
   2 | user1     | password  | user1@example.com | user
   3 | user2     | 123456   | user2@example.com | user
   4 | evil_hacker | p@ssw0rd | hacker@darkweb.com | user
(4 rows)

user@ubuntu-desktop ~/flaskapp
└─$ sudo docker run --rm -it --privileged -v /dev:/host/dev -v -v -e FALCO_UI_OUTPUT=/tmp/falco-events.json falcosecurity/falco:latest
--privileged \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /dev:/host/dev \
-v /proc:/host/proc:ro \
-v /boot:/host/boot:ro \
-v /lib/modules:/host/lib/modules:ro \
-e FALCO_UI_OUTPUT=/tmp/falco-events.json \
falcosecurity/falco:latest
2025-01-29T14:21:21+0000: Falco version: 0.40.0 (x86_64)
2025-01-29T14:21:21+0000: Falco initialized with configuration files:
2025-01-29T14:21:21+0000:   /etc/falco/falco.yaml | schema validation: ok
2025-01-29T14:21:21+0000: System info: Linux version 6.5.0-13-generic (build@lcy02-amd64-053) (x86_64-linux-gnu-gcc-13 (Ubuntu 13.2.0-4ubuntu3) 13.2.0, GNU ld (GNU Binutils for Ubuntu) 2.41) #13-Ubuntu SMP PREEMPT_DYNAMIC Fri Nov 3 12:16:05 UTC 2023
2025-01-29T14:21:21+0000: Loading rules from:
2025-01-29T14:21:21+0000:   /etc/falco/falco.rules.yaml | schema validation: ok
2025-01-29T14:21:21+0000:   /etc/falco/falco.rules.local.yaml | schema validation: none
2025-01-29T14:21:21+0000: The chosen syscall buffer dimension is: 8388608 bytes (8 MBs)
2025-01-29T14:21:21+0000: Starting health webserver with threadness 6, listening on 0.0.0.0:765
2025-01-29T14:21:21+0000: Loaded event sources: syscall
2025-01-29T14:21:21+0000: Enabled event sources: syscall
2025-01-29T14:21:21+0000: Opening 'syscall' source with modern BPF probe.
2025-01-29T14:21:21+0000: One ring buffer every '2' CPUs.

```

```

user@ubuntu-desktop ~/flaskapp
└─$ sudo docker compose down && sudo docker compose up -d --build && sudo docker
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG:  starting PostgreSQL 13.18 on x86_64-pc-linux-musl, compiled by gcc (Alpine 14.2.0) 14.2.0, 64-bit
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG:  listening on IPv6 address "::", port 5432
db-1  | 2025-01-29 13:39:59.603 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1  | 2025-01-29 13:39:59.623 UTC [27] LOG:  database system was shut down at 2025-01-29 13:35:25 UTC
db-1  | 2025-01-29 13:39:59.634 UTC [1] LOG:  database system is ready to accept connections
web-1  | * Serving Flask app 'app'
web-1  | * Debug mode: on
web-1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
web-1  | * Running on all addresses (0.0.0.0)
web-1  | * Running on http://127.0.0.1:5000
web-1  | * Running on http://172.18.0.3:5000
web-1  | Press CTRL+C to quit
web-1  | * Restarting with stat
web-1  | * Debugger is active!
web-1  | * Debugger PIN: 620-223-127
web-1  | 172.18.0.1 -- [29/Jan/2025 13:40:32] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 -- [29/Jan/2025 13:41:51] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 -- [29/Jan/2025 13:43:18] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 -- [29/Jan/2025 13:43:25] "GET /search?q=%20UNION%20SELECT%20id,%20username,%20password%20FROM%20users%20-- HTTP/1.1" 200 -

```

Nous constatons là encore que l'attaque n'est pas détectée, malgré une réussite évidente (liste de tous les users et leur mot de passe).

Conclusion

Il semblerait donc que Falco, en fonctionnant avec ses règles par défaut ne soit pas capable de détecter des injections SQL passant par une requête HTTP.

Nous avons longuement investigué sur la façon de gérer les règles pour améliorer ses capacités de détection, mais avons constaté que c'était le programme du TP03.

Nous ajoutons simplement une note ici, permettant de monter notre fichier de règles depuis notre host vers notre conteneur Falco.

```

sudo docker run --rm -it \
--privileged \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /dev:/host/dev \
-v /proc:/host/proc:ro \
-v /boot:/host/boot:ro \
-v /lib/modules:/host/lib/modules:ro \
-v /home/user/flaskapp/falco_rules.local.yaml:/etc/falco/falco_rules.local.yaml:ro \
-v /home/user/flaskapp/logs/access.log:/var/log/falco/flask_access.log:ro \
-e FALCO_UI_OUTPUT=/tmp/falco-events.json \
falcosecurity/falco:latest

```

À noter que la suite de nos tentatives étaient de rediriger l'output des logs de notre application Flask vers un fichier à lire par Falco lors de la création et du lancement du conteneur. Ensuite, nous aurions fait une règle analysant le contenu de notre fichier de log et vérifiant si une regex est validée par une des lignes du fichier de logs.

D'après la capture d'écran suivante, il ne serait pas forcément très difficile de détecter ce genre d'intrusion.

```
● ● ●  ☺ 22      sudo docker compose down && sudo docker compose up -d --build && sudo docker
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG:  starting PostgreSQL 13.18 on x86_64-pc-linux-mu
sl, compiled by gcc (Alpine 14.2.0) 14.2.0, 64-bit
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
db-1  | 2025-01-29 13:39:59.589 UTC [1] LOG:  listening on IPv6 address ":::", port 5432
db-1  | 2025-01-29 13:39:59.603 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1  | 2025-01-29 13:39:59.623 UTC [27] LOG:  database system was shut down at 2025-01-29 13
:35:25 UTC
db-1  | 2025-01-29 13:39:59.634 UTC [1] LOG:  database system is ready to accept connections
web-1  | * Serving Flask app 'app'
web-1  | * Debug mode: on
web-1  | WARNING: This is a development server. Do not use it in a production deployment. Use
a production WSGI server instead.
web-1  | * Running on all addresses (0.0.0.0)
web-1  | * Running on http://127.0.0.1:5000
web-1  | * Running on http://172.18.0.3:5000
web-1  | Press CTRL+C to quit
web-1  | * Restarting with stat
web-1  | * Debugger is active!
web-1  | * Debugger PIN: 620-223-127
web-1  | 172.18.0.1 - - [29/Jan/2025 13:40:32] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 - - [29/Jan/2025 13:41:51] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 - - [29/Jan/2025 13:43:18] "POST /login HTTP/1.1" 401 -
web-1  | 172.18.0.1 - - [29/Jan/2025 13:43:25] "GET /search?q=%20UNION%20SELECT%20id,%20usern
ame,%20password%20FROM%20users%20-- HTTP/1.1" 200 -
```