

# Projet SCRIPTING SECURITE

Contributeurs : Nabiya CHERGUI - Olivier KOENIG

Référent pédagogique : Christophe GERMAIN

Présentation le jeudi 27 février 2025

# SOMMAIRE

Introduction

Organisation de l'équipe et du projet

Méthodologie et outils

Architecture et structure du projet

Présentation des scripts Bash & Python

Automatisation avec Cron

Problèmes rencontrés et solutions apportées

Améliorations possibles

Retour d'expérience & apprentissages

Remerciements

# INTRODUCTION

- ◆ Cybersécurité : un enjeu majeur

- ◆ Problématique

Comment sécuriser un système face aux menaces ?

Comment automatiser la détection des anomalies et attaques ?

- ◆ Objectif du projet

Développer

Automatiser

Proposer une solution simple et efficace

# Méthodologie et outils

**Planification** : Définition des tâches, choix des outils

**Développement** : Ecriture des scripts Bash & Python, documentation

**Tests et validation** : Tests unitaires, automatisation avec cron

**Finalisation** : Livraison des scripts, rédaction de la documentation.

**Git/GitHub** :  
Versionnement

**VS Code** : Éditeur  
de code

**VirtualBox** :  
gestionnaire de  
machines virtuels

**Kali-linux** :  
Machine à  
monitorer.

**Bash** :  
Automatisation

**Python3** :  
Scripting &  
Sécurité

**Nmap** : Scan des  
ports

**Fail2ban** :  
Protection contre  
les attaques

**Cryptography**  
: Chiffrement  
sécurisé

**Crontab** :  
Plannification des  
tâches

# Architecture et structure du projet

Notre objectif :

- Faciliter la navigation et l'évolution du projet
- Séparer chaque fonctionnalité
- Suivre les bonnes pratiques en cyber et scripting

```
✓ Projet_Scripting_S-curite
  > rapports
  ✓ scripts_logs
    🔗 analyse_logs.py
    ⚡ README_logs.md
    $ surveillance_logs.sh
  ✓ scripts_pwd
    ✓ backups
      ≡ passwords_backup_20250226_162932.e..
      > secrets
      $ backup_passwords.sh
      🔗 password_manager.py
      ⚡ README_pwd.md
    ✓ scripts_scan
      🔗 analyse_scan.py
      ⚡ README_scan.md
      $ scan_ports.sh
    > venv
    ⓘ README.md
    ≡ requirements.txt
```

# Présentation des scripts - Gestion sécurisée des mots de passe

Stockage et gestion  
sécurisée via chiffrement

Politique de robustesse des  
mots de passe garantie

Automatisation de la  
sauvegarde des mots de  
passe

## Scripts impliqués :

```
└─ scripts_pwd
   ├── backups
   ├── secrets
   ├── $ backup_passwords.sh
   ├── 🔗 password_manager.py
   └── ⬇ README_pwd.md
```

**Script Python (password\_manager.py) :** Gère les mots de passe en assurant leur **chiffrement**, leur **stockage sécurisé**, et leur **récupération contrôlée**.

**Script Bash (backup\_passwords.sh) :** Automatise la **sauvegarde** des mots de passe chiffrés pour éviter toute **perte de données**.

📖 **README\_pwd.md** → Documentation associée.



# Fonctionnement du script Python en détail

Script Python avec `password_manager.py` :


- Utilisation de la bibliothèque `cryptography` pour le chiffrement
- Génération de clé dans ***secrets/key.key***
- Stockage dans ***secrets/passwords.enc***
- Robustesse vérifiée
- Récupération sécurisée

```
(venv)-(nabs@kali)-[~/Projet_Scripting_Securite_V2/Projet_Scripting_S-curite]
$ python3 scripts_pwd/password_manager.py
Nouvelle clé générée.

🔑 Gestionnaire de mots de passe
1. Ajouter un mot de passe
2. Récupérer un mot de passe
- Choisissez une option : 1
🌐 Site : gmail
👤 Identifiant : moi@gmail.com

🔒 **Règles du mot de passe sécurisé** :
- Min. **8 caractères**
- Min. **1 majuscule** (A-Z)
- Min. **1 minuscule** (a-z)
- Min. **1 chiffre** (0-9)
- Min. **1 symbole spécial** (!@#$%^&*...)

🔑 Mot de passe : Akfdjkdhf8!
✅ Mot de passe sécurisé enregistré pour gmail.
```



```
└─ scripts_pwd
   └─ secrets
      ├── key.key
      ├── passwords.enc
      ├── backup_passwords.sh
      ├── password_manager.py
      └── README_pwd.md
```

On peut récupérer le mdp grâce à la clé

```
(venv)-(nabs@kali)-[~/Projet_Scripting_Securite_V2/Projet_Scripting_S-curite]
$ ./scripts_pwd/password_manager.py

🔑 Gestionnaire de mots de passe
1. Ajouter un mot de passe
2. Récupérer un mot de passe
- Choisissez une option : 2
🌐 Site : gmail
👤 Entrez votre clé de chiffrement pour récupérer le mot de passe :
🔑 Identifiant : moi@gmail.com
🔒 Mot de passe : Akfdjkdhf8!
```

# Fonctionnement du script Bash en détail



Script Bash avec **backup\_passwords.sh**

- Vérification avec *passwords.enc*
- Copie et stockage dans *backups/*
- Récupération possible

```
(nabs@kali)-[~/Projet_Scripting_Securite_V2/Projet_Scripting_S-curite]
$ chmod +x scripts_pwd/backup_passwords.sh

(nabs@kali)-[~/Projet_Scripting_Securite_V2/Projet_Scripting_S-curite]
$ ./scripts_pwd/backup_passwords.sh
✓ Sauvegarde réalisée : scripts_pwd/backups/passwords_backup_20250226_162932.enc
```



```
▼ scripts_pwd
  ▼ backups
    ≡ passwords_backup_20250226_162...
```



```
Projet_Scripting_S-curite > scripts_pwd > backups > ≡ passwords_backup_20250226_162932.enc
1  gAAAAABnvyp12xi7XfEXm4NgMPeN8m04cSqNLfC2Rmzqj5GiNWgae4CJ8CghmloyBxrWoHq3Vk0tgQSs1
```



# Présentation des scripts - Scan des ports et services ouverts

🔍 Pourquoi utiliser Bash et Python ensemble ?

## ✅ Bash : Scanner les ports avec Nmap

- Le script Bash exécute nmap pour **identifier les ports ouverts et les services actifs**.
- Il **enregistre les résultats** dans un fichier pour une analyse ultérieure.
- Cela permet de **détecter les points d'entrée potentiels** sur un réseau.

## ✅ Python : Analyser les résultats et détecter les failles

- Python **lit le rapport Nmap** et extrait les services identifiés.
- Il **compare les versions des services** avec une base de vulnérabilités connues.
- Si un service vulnérable est détecté, il **génère une alerte** pour prévenir les risques.

➡ **Objectif : Automatiser la détection des failles avant qu'un pirate ne les exploite !**

```
PROJET_SCRIPTING_S-CURITE [SSH: 192.168.1.2]
└─ rapports
   ├── analyse_logs_report.txt
   ├── Bash_scan_result_192_168_1_2_20250227_082941.txt
   ├── Bash_scan_result_192_168_1_5_20250226_202926.txt
   ├── Python_scan_result_192_168_1_2_20250227_083118.txt
   ├── Python_scan_result_192_168_1_5_20250226_203016.txt
   └── Python_scan_result_192_168_1_5_20250226_204929.txt
```

## ./scripts\_scan/scan\_ports.sh 192.168.1.5



### Bash : Scanner les ports avec Nmap

- Le script Bash exécute nmap pour **identifier les ports ouverts et les services actifs**.
- Il **enregistre les résultats** dans un fichier pour une analyse ultérieure.
- Cela permet de **détecter les points d'entrée potentiels** sur un réseau.
- **générer un fichier : rapports/Bash\_scan\_result\_192\_168\_1\_5\_20250226\_202926.txt** contenant les résultats du scan.

```
(olivier@lan-olivier) - [~/Projet_Scripting_S-curite]
$ ./scripts_scan/scan_ports.sh 192.168.1.5
Scan des ports ouverts sur 192.168.1.5...
Résultats enregistrés dans rapports/Bash_scan_result_192_168_1_5_20250226_202926.txt
Starting Nmap 7.95 ( https://nmap.org ) at 2025-02-26 20:29 CET
Nmap scan report for 192.168.1.5
Host is up (0.00080s latency).
All 1000 scanned ports on 192.168.1.5 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 08:00:27:B7:1D:4B (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.62 seconds
```

## ./scripts\_scan/analyse\_scan.py 192.168.1.5

✓ Python : Analyser les résultats et détecter les failles

- Python lit le **rapport Nmap** et extrait les services identifiés.
- Il **compare les versions des services** avec une base de vulnérabilités connues.
- Si un service vulnérable est détecté, il **génère une alerte** pour prévenir les risques.
- Il créera un fichier **rapports/Python\_scan\_result\_192\_168\_1\_5\_20250227\_095135.txt** contenant les résultats du scan.

```
(olivier@lan-olivier) - [~/Projet_Scripting_S-curite]
$ ./scripts_scan/analyse_scan.py 192.168.1.5
Scan des ports ouverts sur : 192.168.1.5...
⌚ Patientez, le scan peut prendre quelques instants...
✓ Scan terminé ! Résultats enregistrés dans : rapports/Python_scan_result_192_168_1_5_20250227_095135.txt
Starting Nmap 7.95 ( https://nmap.org ) at 2025-02-27 09:51 CET
Nmap scan report for 192.168.1.5
Host is up (0.0019s latency).
All 1000 scanned ports on 192.168.1.5 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 08:00:27:B7:1D:4B (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.63 seconds
```

✓ Aucun service vulnérable détecté.

# Présentation des scripts

-

## Surveillance et analyse des logs de sécurité


Détection des  
tentatives


Identification  
des attaques

### Scripts impliqués :

```
▼ scripts_logs  
  🔗 analyse_logs.py  
  ⬇ README_logs.md  
  $ surveillance_logs.sh
```

 **surveillance\_logs.sh (Bash)** → Surveillance en direct des tentatives échouées.

 **analyse\_logs.py (Python)** → Analyse détaillée et rapport des IP suspectes.

 **README\_logs.md** → Documentation associée.

# Fonctionnement du script Bash (surveillance\_logs.sh)

Depuis un autre terminal :

```
(nabs@kali)-[~/Projet_Scripting_Securite_V2/Projet_Scripting_S-urite]
$ chmod +x scripts_logs/surveillance_logs.sh

(nabs@kali)-[~/Projet_Scripting_Securite_V2/Projet_Scripting_S-urite]
$ ./scripts_logs/surveillance_logs.sh
● Surveillance en temps réel des échecs de connexion dans /var/log/auth.log
```



```
(nabs@kali)-[~/Projet_Scripting_Securite_V2]
$ ssh faux_utilisateur@localhost

faux_utilisateur@localhost's password:
Permission denied, please try again.
faux_utilisateur@localhost's password:
Permission denied, please try again.
faux_utilisateur@localhost's password:
faux_utilisateur@localhost: Permission denied (publickey,password).
```

```
(nabs@kali)-[~/Projet_Scripting_Securite_V2/Projet_Scripting_S-urite]
$ ./scripts_logs/surveillance_logs.sh
● Surveillance en temps réel des échecs de connexion dans /var/log/auth.log
▲ Tentative de connexion échouée : 2025-02-26T17:01:41.137770+01:00 kali sshd-session[927107]: Failed password for invalid user faux_utilisateur from ::1 port 58612 ssh2
▲ Tentative de connexion échouée : 2025-02-26T17:01:45.693950+01:00 kali sshd-session[927107]: Failed password for invalid user faux_utilisateur from ::1 port 58612 ssh2
▲ Tentative de connexion échouée : 2025-02-26T17:01:50.348235+01:00 kali sshd-session[927107]: Failed password for invalid user faux_utilisateur from ::1 port 58612 ssh2
□
```

Utilisation de `tail -f /var/log/auth.log` pour suivre les nouvelles entrées en direct.

```
(nabs@kali)-[~/Projet_Scripting_Securite_V2/Projet_Scripting_S-urite]
$ tail -f /var/log/auth.log
2025-02-26T17:01:45.693950+01:00 kali sshd-session[927107]: Failed password for invalid user faux_utilisateur from ::1 port 58612 ssh2
2025-02-26T17:01:48.847850+01:00 kali sshd-session[927107]: pam_unix(sshd:auth): check pass; user unknown
2025-02-26T17:01:48.847995+01:00 kali sshd-session[927107]: pam_winbind(sshd:auth): getting password (0x00000388)
2025-02-26T17:01:48.848023+01:00 kali sshd-session[927107]: pam_winbind(sshd:auth): pam_get_item returned a password
2025-02-26T17:01:50.348235+01:00 kali sshd-session[927107]: Failed password for invalid user faux_utilisateur from ::1 port 58612 ssh2
2025-02-26T17:01:51.826157+01:00 kali sshd-session[927107]: Connection closed by invalid user faux_utilisateur ::1 port 58612 [preauth]
2025-02-26T17:01:51.827654+01:00 kali sshd-session[927107]: PAM 2 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=::1
2025-02-26T17:01:51.830234+01:00 kali sshd[718]: srclimit_penalise: ipv6: new ::1/128 deferred penalty of 5 seconds for penalty: failed authentication
2025-02-26T17:05:01.210593+01:00 kali CRON[928893]: pam_unix(cron:session): session opened for user root(uid=0) by root(uid=0)
2025-02-26T17:05:01.218189+01:00 kali CRON[928893]: pam_unix(cron:session): session closed for user root
□
```

# Fonctionnement du script Python (analyse\_logs.py)

Rendre exécutable le script si  
python3 non installé

```
(nabs@kali) - [~/Projet_Scripting_Securite_V2/Projet_Scripting_S-curite]
$ chmod +x scripts_logs/analyse_logs.py
```

Lancer la lecture et l'analyse des  
logs pour générer le rapport

```
(nabs@kali) - [~/Projet_Scripting_Securite_V2/Projet_Scripting_S-curite]
$ ./scripts_logs/analyse_logs.py
✓ Analyse terminée. Rapport généré : rapports/analyse_logs_report.txt
```

Génération du rapport dans le  
dossier rapports/

```
▼ Projet_Scripting_S-curite
  ▼ rapports
    ≡ analyse_logs_report.txt
```

Grace à Fail2Ban, adresses IP bannies au bout  
de 3 tentatives échouées pendant 10 minutes :

```
(nabs@kali) - [~/Projet_Scripting_Securite_V2/Projet_Scripting_S-curite]
$ sudo fail2ban-client status sshd
Status for the jail: sshd
- Filter
  - Currently failed: 0
  - Total failed: 0
  - Journal matches: _SYSTEMD_UNIT=ssh.service + _COMM=sshd
- Actions
  - Currently banned: 1
  - Total banned: 1
  - Banned IP list: 192.168.1.140
```

Rapport généré avec les IP suspectes et  
leur nombre de tentatives

```
Projet_Scripting_S-curite > rapports > ≡ analyse_logs_report.txt
1  📌 Rapport des tentatives de connexion échouées 📌
2  =====
3  🚫 :::1 - 25 tentatives échouées
4  🚫 192.168.1.191 - 3 tentatives échouées
```



# Automatisation avec Cron

Bien que par défaut, il soit souvent installé, voici les commandes pour l'installer sur les différentes distributions :

Installation Debian et Ubuntu :

➤ `sudo apt install cron`

Pour afficher le contenu du fichier **crontab** :

➤ `crontab -l`

Pour supprimer toutes les actions du fichier **crontab**

➤ `crontab -r`

Pour éditer les actions du fichier **crontab**

➤ `crontab -e`

A dark-themed cheat sheet for cron jobs. At the top, it lists several pre-defined cron expressions in boxes: '#every hour' (0 \* \* \* \* command), '#every 15 mins' (\* / 15 \* \* \* \* command), '#every 2 hours' (0 \* / 2 \* \* \* \* command), '#every Sunday midnight' (0 0 \* \* 0 command), '#every week' (@weekly command), '#every day' (@daily command), '#every year' (@yearly command), and '#every month' (@monthly command). Below these, the title 'CRON CHEATSHEET' is displayed in large, colorful letters. The main part of the cheat sheet shows the cron syntax: five asterisks representing the fields for Minute, Hour, Day, Month, and Weekday, followed by the 'command to be executed'. To the right of the syntax, a diagram shows the hierarchy of the fields: Weekday (0=Sun .. 6=Sat), Month (1..12), Day (1..31), Hour (0..23), and Minute (0..59). On the far right, there are two more boxes: '#every hour' (@hourly command) and '#every reboot' (@reboot command).

Field	Range
Weekday	(0=Sun .. 6=Sat)
Month	(1..12)
Day	(1..31)
Hour	(0..23)
Minute	(0..59)

# Analyse du fichier crontab

## BASH - Tâches exécutées à 02h00 du matin :

### ✓ Scan des ports

```
0 2 * * * bash ~/Projet_Scripting_Seurite/scripts_scan/scan_ports.sh 192.168.1.1
```

Analyse les ports de l'adresse IP 192.168.1.1.

### ✓ Surveillance des logs

```
0 2 * * * bash ~/Projet_Scripting_Seurite/scripts_logs/surveillance_logs.sh
```

Exécute un script qui surveille les logs (journal des événements).

### ✓ Sauvegarde des mots de passe

```
0 2 * * * bash ~/Projet_Scripting_Seurite/scripts_pwd/backup-passwords.sh
```

Effectue une sauvegarde des mots de passe.

## PYTHON - Tâches exécutées à 08h30 du matin :

### ✓ Analyse des scans

```
30 8 * * * python3 ~/Projet_Scripting_Seurite/scripts/analyse_scan.py
```

Exécute un script Python qui analyse un scan (probablement généré par le scan des ports à 02h00).

### ✓ Analyse des logs

```
30 8 * * * python3 ~/Projet_Scripting_Seurite/scripts_logs/analyse_logs.py
```

Analyse les logs du système.

### ✓ Gestionnaire de mots de passe

```
30 8 * * * python3 ~/Projet_Scripting_Seurite/scripts_pwd/password_manager.py
```

Lance un gestionnaire de mots de passe.



```
GNU nano 8.3 /tmp/crontab.d5
0 2 * * * bash ~/Projet_Scripting_Seurite/scripts_scan/scan_ports.sh 192.168.1.1
0 2 * * * bash ~/Projet_Scripting_Seurite/scripts_logs/surveillance_logs.sh
0 2 * * * bash ~/Projet_Scripting_Seurite/scripts_pwd/backup-passwords.sh
30 8 * * * python3 ~/Projet_Scripting_Seurite/scripts/analyse_scan.py
30 8 * * * python3 ~/Projet_Scripting_Seurite/scripts_logs/analyse_logs.py
30 8 * * * python3 ~/Projet_Scripting_Seurite/scripts_pwd/password_manager.py
```

# Problèmes rencontrés et solutions apportées

- ◆ Fichier `auth.log` introuvable
  - ➔ Pas de logs SSH disponibles
  - ✓ Activation de la journalisation
- ◆ Droits administrateur insuffisants
  - ➔ Commandes bloquées sans `sudo`
  - ✓ Ajout utilisateur au groupe `sudo`
- ◆ Problème de push GitHub
  - ➔ Conflit empêchant le `git push`
  - ✓ `git pull --rebase` pour résoudre





◆ **Renforcer la sécurité**

- **Intégration d'un pare-feu** (UFW, iptables)
- Amélioration de la **politique de mots de passe**



◆ **Automatisation avancée**

- **Ajout de notifications** (email, Slack) pour alertes
- Génération de **rapports détaillés** en PDF



◆ **Tests & évolutions**

- **Simulation d'attaques** avec Hydra & Nikto
- **Compatibilité** avec d'autres systèmes Linux

Améliorations  
possibles

# Retour d'expérience & apprentissages

- ◆ Approfondissement des outils de cybersécurité

- ◆ Montée en compétences sur l'automatisation

- ◆ GitHub & travail en équipe

- ◆ Résolution de problèmes & autonomie

# Remerciements 🙏

Nous tenons à remercier :



- ◆ **Notre formateur Christophe** pour son accompagnement et ses précieux conseils.
- ◆ **Nos camarades de classe** pour leurs échanges et leur entraide.
- ◆ **Notre binôme** pour la collaboration efficace et la répartition des tâches.
- ◆ **Les ressources en ligne et la documentation officielle** qui nous ont aidés à surmonter les obstacles.

💡 *Merci pour votre attention ! Nous sommes disponibles pour vos questions*