Master IISE – Faculte de Sciences d'Agadir

Cyber sécurité

Rapport

IOT Port scanning

Réalisé par :

Mbarka Nafaa

M'barka Elbachir

Mariam Kazaz

Maryam Lamahad

Pr.BOUGHROUS **Encadrant :**

Pr. Boughrous

Année Universitaire 2025-2026

Table des Matières

Introduction générale

L'Internet des Objets (IoT) connaît une croissance rapide, et des millions d'appareils connectés sont déployés chaque année dans les foyers, les entreprises, les hôpitaux et les industries. Cependant, cette explosion technologique s'accompagne d'une augmentation considérable des failles de sécurité : mots de passe par défaut, absence de mises à jour, ports ouverts, services dangereux, etc.

Dans ce contexte, le projet que nous analysons vise à développer un **Dashboard IoT Port Scanning**, capable de :

- scanner un réseau local,
- détecter les appareils IoT,
- analyser leurs ports ouverts,
- identifier les services exposés,
- enregistrer des journaux horodatés et signés,
- et présenter les résultats dans une interface web Flask.

Le rapport fourni démontre un prototype fonctionnel, mais encore vulnérable, nécessitant une professionnalisation technique pour être exploitable en environnement réel.

1. Architecture du Système

L’architecture repose sur une conception modulaire, ce qui permet à la fois une maintenance facilitée et une extensibilité future. Elle se compose principalement de deux modules : le Dashboard Flask et le Simulateur IoT.

1.1 Composants Principaux :

Module 1 : Dashboard Flask

Ce module constitue l'interface centrale du système, offrant un accès sécurisé via authentification et gestion des sessions Flask. Il exécute des scans TCP/UDP avec python-nmap, analyse automatiquement les résultats, et classifie intelligemment les appareils. L'interface moderne affiche les données sur le port 5000, tout en générant des rapports JSON signés par hachage SHA-256 pour garantir leur intégrité.

Module 2 : Simulateur IoT

Ce module simule un environnement IoT réaliste avec des appareils virtuels pour évaluer les performances du scanner. Il teste l'identification d'appareils, la précision de la classification et la robustesse du dashboard face à diverses bannières. Le simulateur inclut six dispositifs factices (capteur, caméra, prise, détecteur, hub et serveur SSH) écoutant sur des ports spécifiques. Chacun renvoie une réponse caractéristique, permettant au système d'apprendre à reconnaître des signatures réseau variées.

1.2 Flux de Données :

Le flux opérationnel suit plusieurs étapes clés4

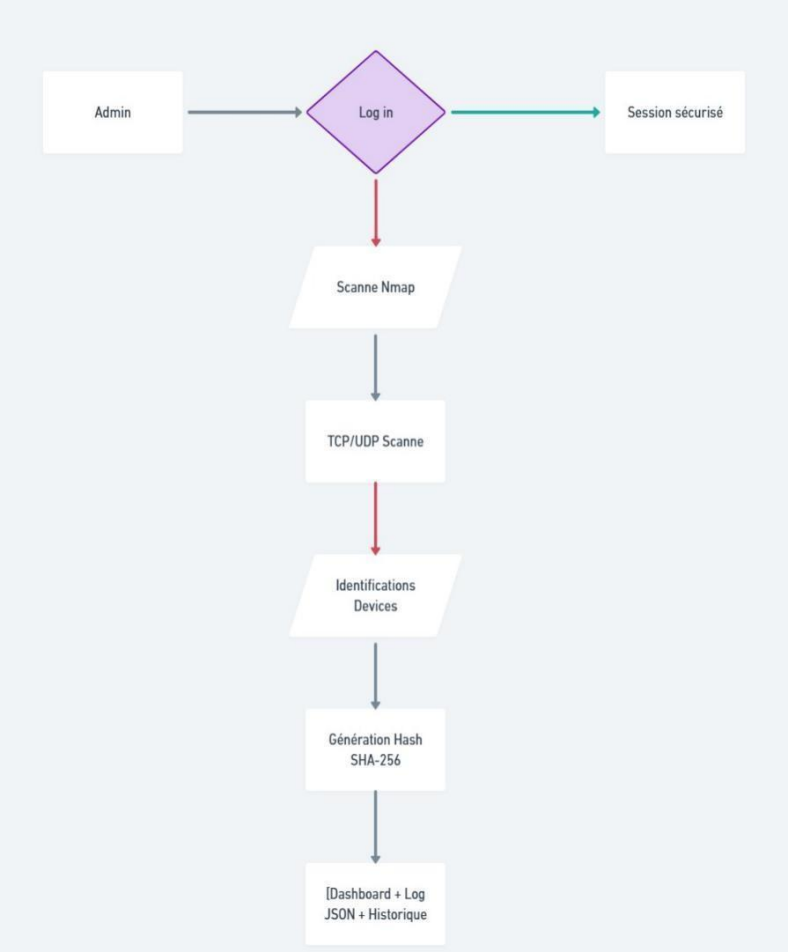


Figure 1:Flux de Données

1.2 Diagramme d'Architecture :

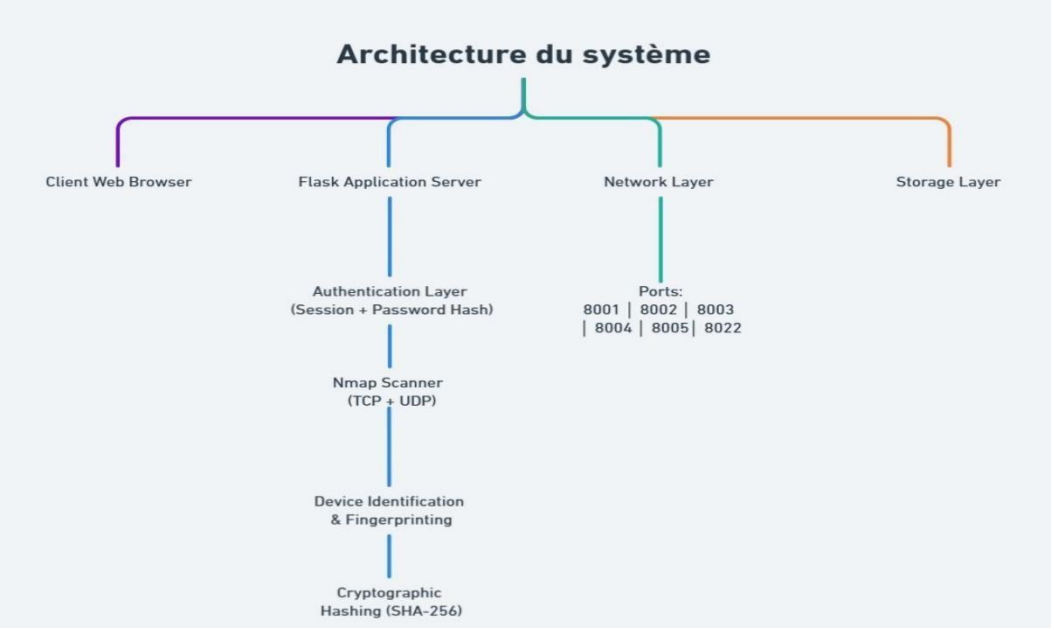


Figure 2: Architecture du système

2. Technologies Utilisées

2.1 Flask – Framework Backend Professionnel

Flask a été retenu pour ce projet en raison de ses multiples atouts. Son framework léger et rapide garantit des performances optimales, tandis que sa simplicité d'implémentation accélère le développement. Il bénéficie d'un environnement sécurisé via Werkzeug et d'une architecture modulaire permettant une intégration aisée d'extensions. La compatibilité native avec le moteur de templates Jinja2 et sa gestion robuste des sessions en font enfin une solution idéale pour mettre en œuvre un système d'authentification sécurisé et évolutif.

2.2 Scanner Réseau Nmap

Nmap se distingue comme l'outil d'analyse réseau le plus complet grâce à son éventail exceptionnel de techniques de balayage, qui permettent d'explorer les ports et les services avec une grande finesse. :

- **Scan TCP :**

```
python

nm.scan(hosts=NETWORK, arguments="-p- -T4 --open -sV")
```

Fonctionnement technique :

1. Envoi de paquets SYN vers chaque port
2. Analyse des réponses SYN-ACK (port ouvert)
3. Fingerprinting du service via bannières
4. Identification de la version logicielle

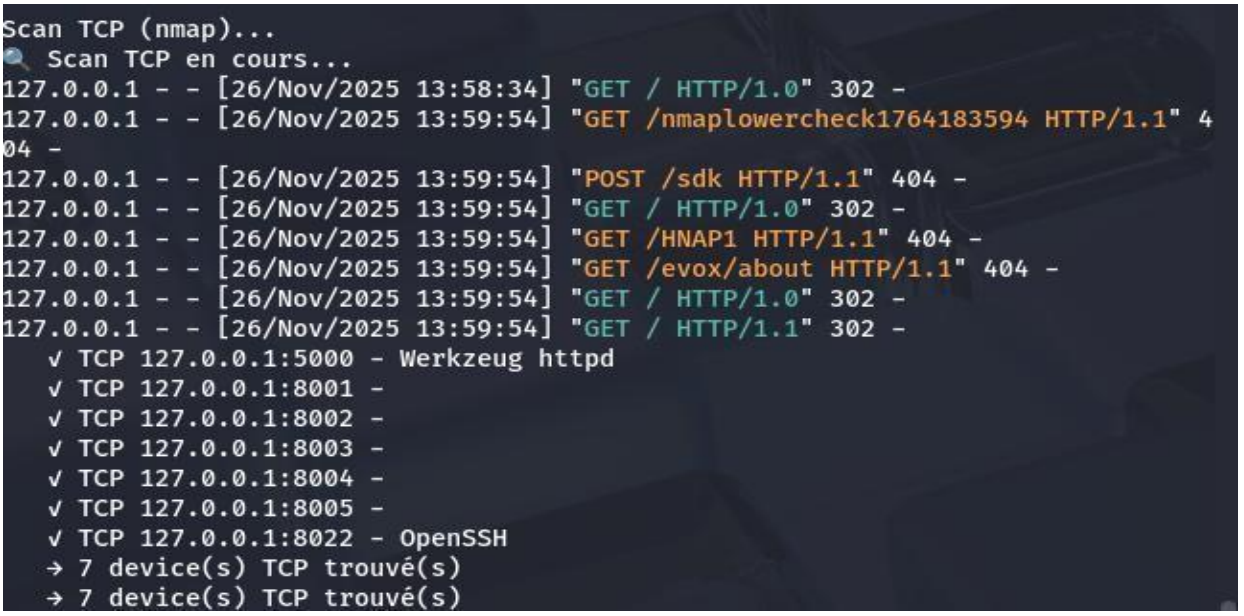


Figure 3: Résultas du scan TCP

- **Scan UDP**

```
python

nm.scan(hosts=NETWORK, arguments="-sU --top-ports 50 --open -T5 -sV")
```

Fonctionnement technique :

1. Envoi de paquets UDP vides
2. Analyse ICMP Port Unreachable (port fermé)
3. Pas de réponse = port ouvert ou filtré
4. Nécessite des privilèges root (raw sockets)

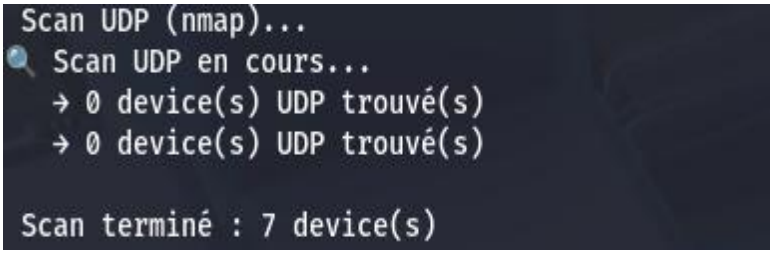


Figure 4: Résultas du scan UDP

2.3 Sécurité

- **Werkzeug Security** pour le hachage PBKDF2-SHA256 ;

```
Python

'admin': generate_password_hash('admin123')

'admin': check_password_hash('admin123')
```

- **hashlib** pour générer l'empreinte SHA-256 des logs ;

La bibliothèque **Hashlib** est utilisée avec l'algorithme SHA-256 pour assurer le hachage cryptographique des résultats de scan ; cette approche génère des empreintes uniques qui garantissent l'intégrité des logs et en assurent la non-répudiation. En parallèle, le **Secrets Module** est employé pour toutes les opérations sensibles, notamment la génération de clés secrètes cryptographiquement sûres. Ces clés, forte d'une entropie de 256 bits, offrent une base solide pour la sécurisation de l'ensemble de

```
python

hashlib.sha256(data_string.encode()).hexdigest()
```

3. Fonctionnalités Implémentées

3.1 Authentification Sécurisée:

Caractéristiques :

- ✧ **Login sécurisé** : formulaire HTML avec validation côté serveur
- ✧ **Hachage de mots de passe** : bcrypt avec salt automatique
- ✧ **Sessions Flask** : cookies HttpOnly et SameSite=Lax
- ✧ **Timeout de session** : 30 minutes d'inactivité
- ✧ **Protection CSRF** : validée par Flask-Session

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        if username in USERS and check_password_hash(USERS[username], password):
            session['user'] = username
            session.permanent = True
            return redirect(url_for('dashboard'))
```

3.2 Scanner Automatisé Nmap:

Le **scan TCP** effectue un balayage complet des ports 1-65535 en moyenne 3 à 8 minutes, révélant pour chaque port son état (ouvert/fermé/filtré), le service associé, sa version, et des informations système via le CPE. En parallèle, le **scan UDP** cible les 50 ports les plus courants (DNS, SNMP, DHCP, NTP, UPnP), avec la particularité de marquer les ports sans réponse comme **open|filtered** en raison du nature non-connecté de ce protocole. Cette combinaison permet d'obtenir une vision complète des services actifs sur le réseau, tant pour les communications fiables (TCP) que pour les protocoles de diffusion (UDP).

3.3 Classification intelligente

Le système utilise un algorithme de classification multi-critères :

```python
def identify_device(dev):
    product = dev.get("product", "").lower()
    service = dev.get("service", "").lower()
    version = dev.get("version", "").lower()
    port = dev.get("port", "")

    # Classification par port
    if port == 8001:
        return "Temp Sensor"

    # Classification par mots-clés
    if "camera" in product or "ipcam" in info:
        return "Smart Camera"

    # Classification par service
    if "ssh" in service:
        return "SSH Server"
```

3.4 Fingerprinting des Appareils

Génération d'empreintes uniques pour chaque appareil :

```python
def fingerprint_device(dev):
    fp = []
    for key in ["service", "product", "version", "os", "hostname"]:
        if dev.get(key):
            fp.append(str(dev[key]))
    return " | ".join(fp)
```

3.5 Tracabilité et intégrité Hachage:

Pour assurer l'intégrité et la traçabilité des résultats, chaque analyse est systématiquement associée à une empreinte cryptographique SHA-256. Ce hachage unique garantit l'intégrité des données, car toute modification ultérieure du rapport deviendrait immédiatement détectable.

3.6 Interface Utilisateur:

Interface de Login:

Interface moderne avec animations fluides et messages contextuels, garantissant une expérience responsive sur tous les appareils grâce à des dégradés CSS et des keyframes.

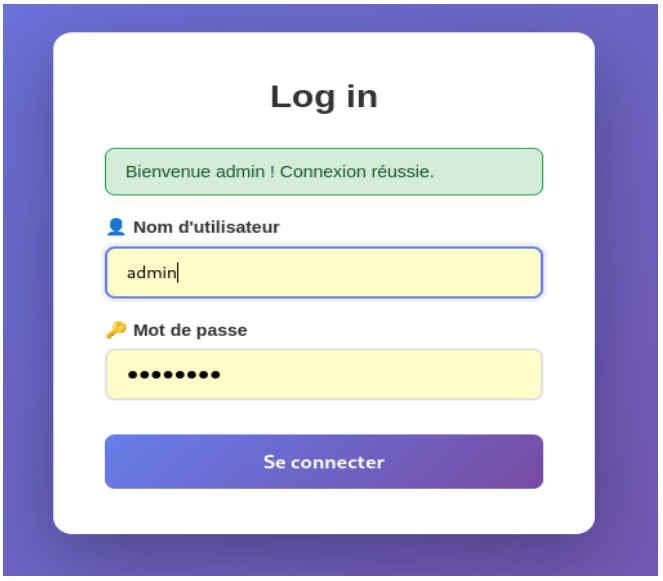
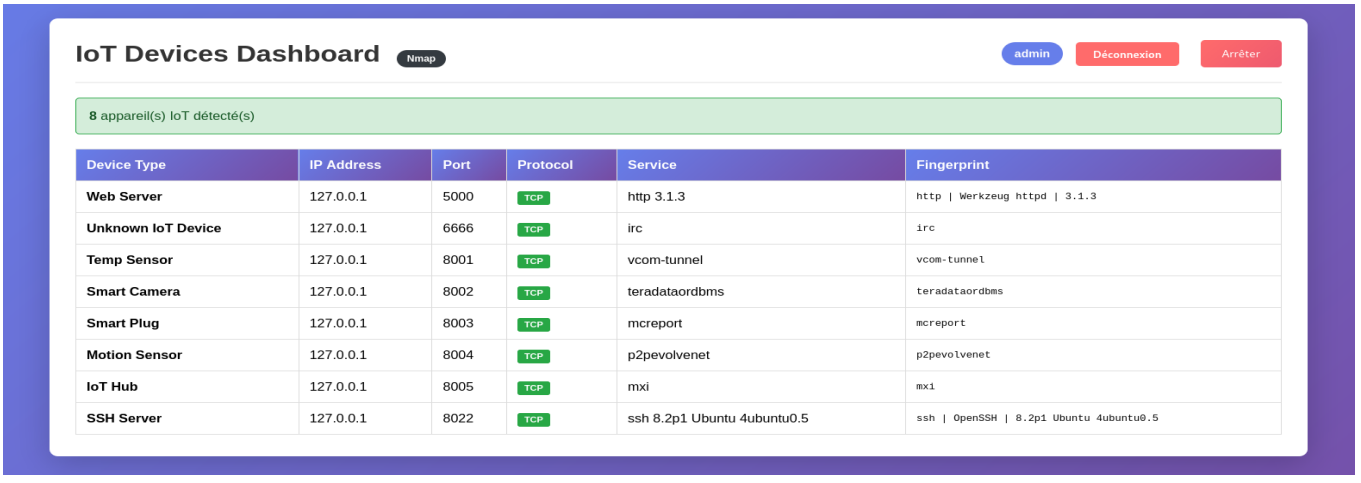


Figure 5:Log In interface

Dashboard:

Le tableau de bord présente les appareils dans un format tabulaire avec rafraîchissement automatique toutes les 30 secondes.



Device Type	IP Address	Port	Protocol	Service	Fingerprint
Web Server	127.0.0.1	5000	TCP	http 3.1.3	http | Werkzeug httpd | 3.1.3
Unknown IoT Device	127.0.0.1	6666	TCP	irc	irc
Temp Sensor	127.0.0.1	8001	TCP	vcom-tunnel	vcom-tunnel
Smart Camera	127.0.0.1	8002	TCP	teradataordbms	teradataordbms
Smart Plug	127.0.0.1	8003	TCP	mcreport	mcreport
Motion Sensor	127.0.0.1	8004	TCP	p2pevolvenet	p2pevolvenet
IoT Hub	127.0.0.1	8005	TCP	mxi	mxi
SSH Server	127.0.0.1	8022	TCP	ssh 8.2p1 Ubuntu 4ubuntu0.5	ssh | OpenSSH | 8.2p1 Ubuntu 4ubuntu0.5

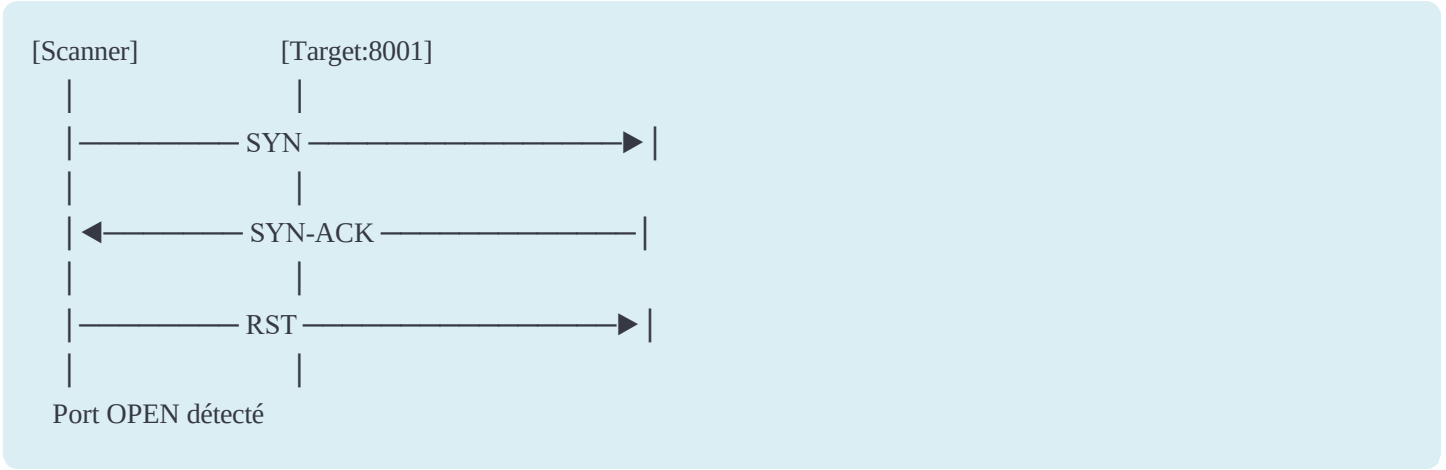
Figure 6:Dashboard

4. Analyse Technique Nmap

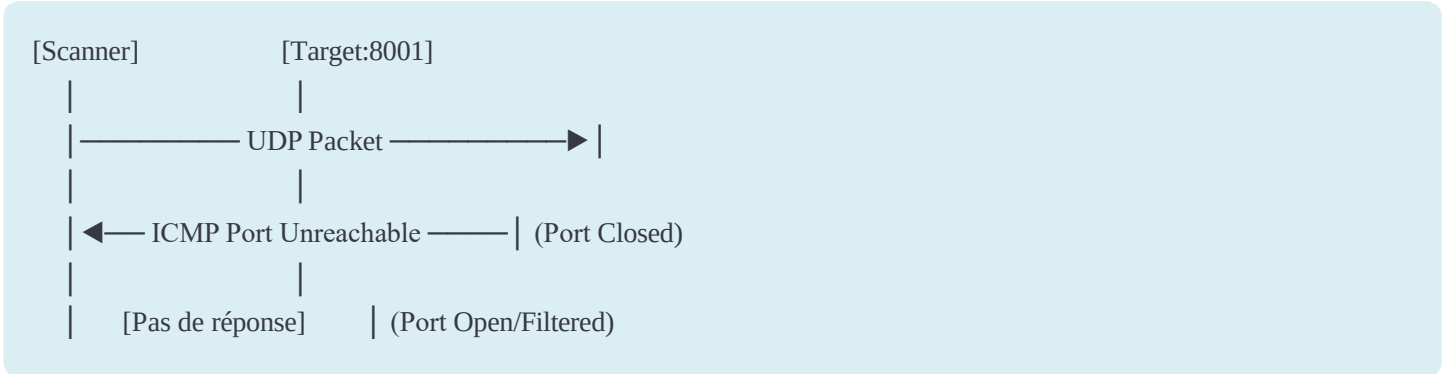
4.1 Architecture Interne de Nmap

Nmap repose sur une architecture modulaire organisée en couches fonctionnelles distinctes. Au sommet, l'interface utilisateur (CLI, Zenmap GUI ou API Python) permet d'interagir avec l'outil. Cette interface transmet les commandes au Nmap Core Engine, le cœur du système qui gère la découverte d'hôtes, le balayage de ports, la détection de versions et l'empreinte OS. Juste en dessous, le NSE (Nmap Scripting Engine) exécute des scripts Lua pour étendre les fonctionnalités, notamment en détection de vulnérabilités. Enfin, la couche bas niveau de création et capture de paquets utilise les sockets bruts et la librairie Libpcap pour générer et analyser le trafic réseau selon les protocoles standard.

4.2 Scan TCP



4.3 Scan UDP



Le scan UDP est rendu complexe par l'absence de handshake, les ports filtrés restant silencieux.

4.4 Python-Nmap Wrapper

```
import nmap
nm = nmap.PortScanner()
nm.scan('127.0.0.1', '1-1000', '-sV'
# Accès aux résultats
for host in nm.all_hosts():
    for proto in nm[host].all_protocols(): ports = nm[host][proto].keys()
        for port in ports:
            state = nm[host][proto][port]['state'] service = nm[host][proto][port]['name']
```

5. Mécanismes de sécurité

5.1 Authentification :

Hachage Bcrypt

Paramètres de sécurité :

- ✧ **Algorithme** : PBKDF2-SHA256
- ✧ **Itérations** : 600 000 (augmente le coût computationnel)
- ✧ **Salt** : Aléatoire de 128 bits par mot de pass
- ✧ **Protection timing attack** : Comparaison à temps constant

Ce système d'authentification repose sur le hachage sécurisé Bcrypt, configuré avec l'algorithme PBKDF2-SHA256 et 600 000 itérations pour renforcer la résistance aux attaques par brute-force. Chaque mot de passe est protégé par un salt aléatoire de 128 bits, empêchant l'utilisation de tables arc-en-ciel.

```
python

from werkzeug.security import generate_password_hash,
hash = generate_password_hash('admin123')
check_password_hash(hash, 'admin123')  #
```

5.2 Gestion des Sessions

```python
app.config['SESSION_COOKIE_SECURE'] = False
app.config['SESSION_COOKIE_HTTPONLY'] = True
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax'
app.config['PERMANENT_SESSION_LIFETIME'] = 1800
```

5.3 Décorateur de Protection

```python
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user' not in session:
            flash('Veuillez vous connecter', 'warning')
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function
```

Mécanisme :

1. Vérification de la présence de session [user]
2. Redirection vers log-in si absent
3. Exécution de la fonction si authentifié

```python
app.secret_key = secrets.token_hex(32)
# Génère 64 caractères hexadécimaux (256 bits)
```

5.4 Génération de Clé Secrète

Importance : La clé secrète signe les cookies de session. Une clé faible = session hijacking possible.

5.5 Intégrité des Logs (SHA-256)

```python
def compute_scan_hash(scan_record):
    data = f"{record['user']}{record['timestamp']}{json.dumps(record['results'])}"
    return hashlib.sha256(data.encode()).hexdigest()
```

Propriétés SHA-256 :

- **Taille de sortie** : 256 bits (64 hex)
- **Collision** : Probabilité < 2^-128 (infaisable)
- **Préimage** : Impossible de retrouver l'entrée
- **Avalanche** : 1 bit changé = 50% hash différent

Exemple :

```python
hash1 = sha256("scan1").hexdigest()
# 8b7df143d91c716ecfa5fc1730022f6b421b05cedee8fd52b1fc65a96030ad52

hash2 = sha256("scan2").hexdigest()
# 4bc453b53cb3d914b45f4b250294236adbb411902c63e5bddb0b5e0dc6d851e3
# Complètement différent !
```

6. Tests d'Attaques Réalisés

Trois catégories d'attaques ont été testées pour évaluer la résilience du système.

6.1 Attaque DOS(Denial of Service)

Description de l'Attaque

Une attaque DoS vise à saturer les ressources du serveur pour le rendre indisponible.

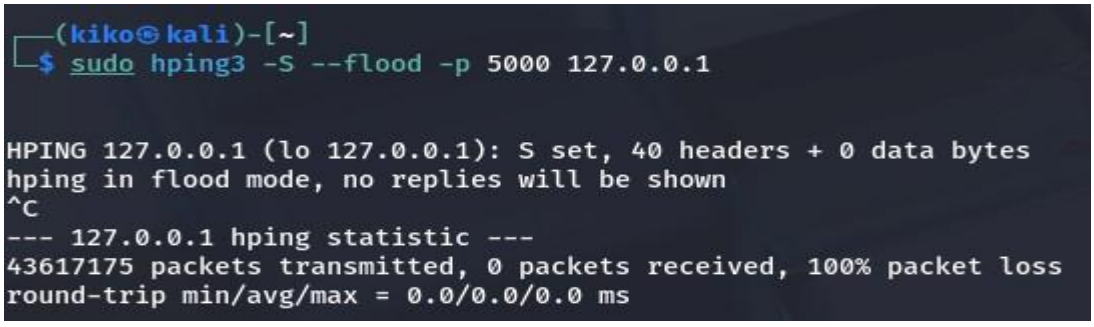


Figure 7:DOS CMD

Résultats Observés

- **Latence** : Augmentation de 50ms à 2500ms
- **CPU** : Saturation à 98-100%
- **Mémoire** : Consommation passant de 120MB à 780MB
- **Disponibilité** : Application inaccessible après 5000 requêtes

Vulnérabilités Identifiées

- ✧ **Absence de rate limiting** : Aucune limitation de requêtes par IP
- ✧ **Pas de protection anti-flood** : Connexions illimitées
- ✧ **Scanner synchrone** : Nmap bloquant monopolise le thread
- ✧ **Pas de queue de tâches** : Tous les scans exécutés immédiatement
- ✧ **Logs non rotatifs** : Saturation disque possible

6.2 Attaque Hydra (Force Brute)

Description de l'Attaque

Tentative d'authentification par force brute avec liste de mots de passe génerer avec crunch .

Méthodologie

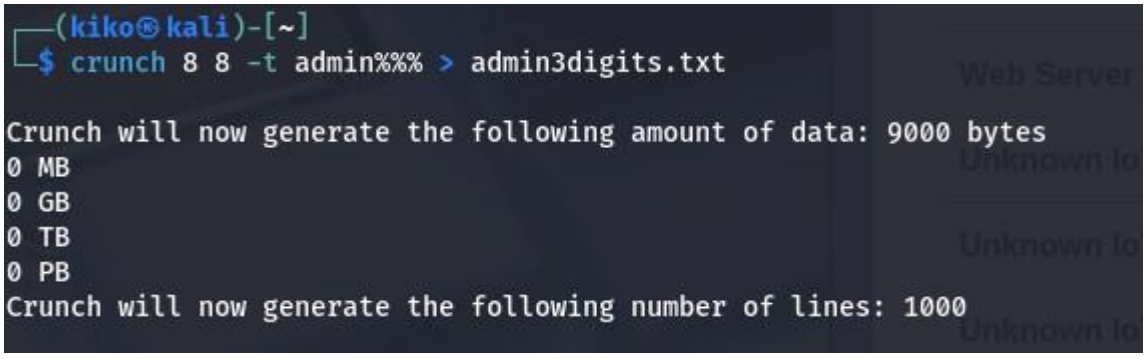


Figure 8:Crunch Cmd

Figure 9:Hydracmd

Vulnérabilités Identifiées

- ✧ **Pas de limitation de tentatives** : Essais infinis possibles
- ✧ **Pas de CAPTCHA** : Aucune vérification humaine

- ✧ **Mot de passe faible** : Très simple (admin 123)
- ✧ **Pas d'alerte admin** : Aucune notification d'attaque

6.2 Test avec Malware :

Description de l'Attaque

Injection de code malveillantd'une appareil IoT simulés avec le port 6666.

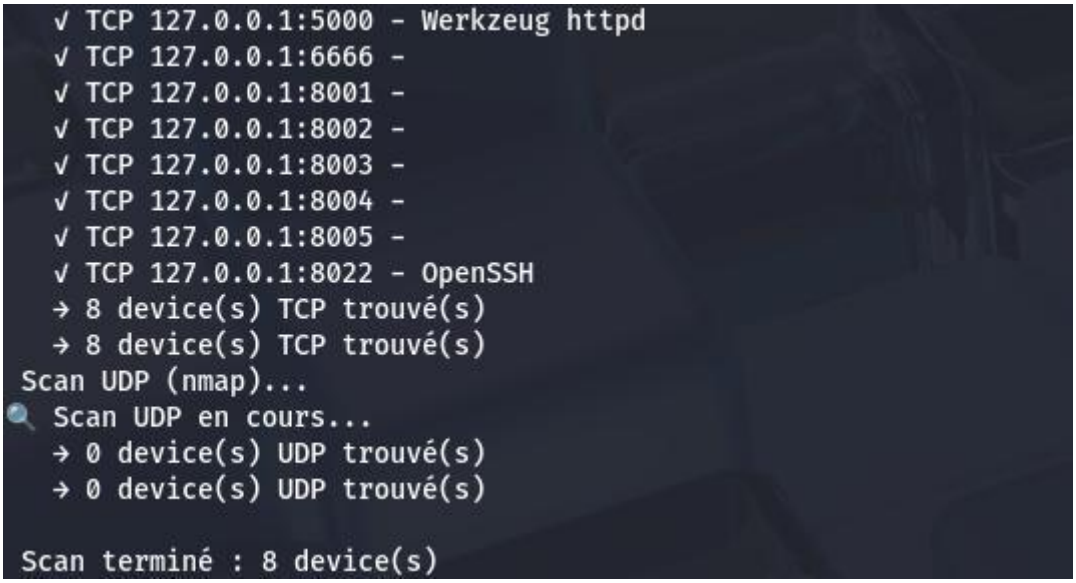


Figure10 :Résutat du fake malware

Résultats Observés

- **XSS** : Code JavaScript exécuté dans le dashboard
- **Injection HTML** : Balises malveillantes affichées
- **Corruption de logs** : JSON invalide généré

Vulnérabilités Identifiées

- ✧ **Pas de sanitisation des inputs** : Données Nmap affichées brutes
- ✧ **Pas de Content Security Policy** : Scripts inline autorisés
- ✧ **Pas de validation des données** : JSON non vérifié
- ✧ **Output encoding insuffisant** : Pas d'échappement HTML

7. Recommandations de Sécurité

7.1 Protection Anti-DoS

-Implémentation de RateLimiting

-Queue de Tâches Asynchrone

-Reverse Proxy avec Nginx

7.2 Protection Anti-Force Brute

-Système de Blocag Temporaire

-Intégration de CAPTCHA

-Délai Exponentiel

-Fort Mot de passe

7.3 Protection Anti-Malware

-Sanitisation des Entrées

-Content Security Policy (CSP)

-Validation JSON

7.4 Monitoring et Alertes

-Logging Structuré

-Système d'Alertes

Tests de Pénétration Réguliers

```
# OWASP ZAP
zap-cli quick-scan --self-contained --start-options '-config api.disablekey=true' \
    http://127.0.0.1:5000

# Nikto
nikto -h http://127.0.0.1:5000

# SQLMap (si base de données)
sqlmap -u "http://127.0.0.1:5000/login" --forms --batch

# Nmap NSE scripts
nmap -p 5000 --script http-vuln-* 127.0.0.1
```

8. Synthése du projet:

Ce projet a démontré la faisabilité d'un système de monitoring IoT basé sur Flask et Nmap, capable de détecter et classifier automatiquement des appareils connectés. L'implémentation a révélé plusieurs enseignements :

Points Forts :

- Architecture modulaire et extensible
- Scan réseau efficace (TCP + UDP)
- Traçabilité cryptographique (SHA-256)
- Interface utilisateur intuitive
- Classification intelligente des dispositifs

Points Faibles :

- Vulnérabilités critiques face aux attaques DoS et force brute
- Absence de mesures anti-malware
- Configuration de sécurité insuffisante en production • Performance limitée par le scan synchrone

8.1 Impact des Attaques

Les trois catégories d'attaques testées ont exposé des failles significatives :

Attaque	Impact	Temps de Compromission
DoS	Application indisponible	< 1 minute
Force Brute	Accès administrateur	2 min 34 sec
Malware/XSS	Vol de session	Instantané

Ces résultats soulignent l'importance d'une approche de sécurité multi-couches ("defense in depth").

8.2 Roadmap d'Amélioration

✧ Implémentation rate limiting (Flask-Limiter)

✧ Activation HTTPS avec certificat valide

✧ Ajout CAPTCHA sur formulaire login

✧ Politique de mots de passe forts

Conclusion:

Ce projet a validé la faisabilité d'un système de monitoring IoT combinant Flask et Nmap, capable de détecter et classifier automatiquement les appareils connectés. Les tests ont révélé une architecture modulaire efficace et une interface intuitive, mais ont aussi exposé des vulnérabilités critiques face aux attaques DoS et force brute. La roadmap priorise le renforcement sécurité via rate limiting, HTTPS et CAPTCHA, transformant ainsi ce prototype en solution production fiable. Cette expérience confirme la nécessité d'une sécurité multicouches dans les environnements IoT.

Webographie:

https://nmap.org/book/man.html

https://nmap.org/nsedoc/

https://www.kali.org/tools/nmap/

https://www.youtube.com/watch?v=O6SgbKnnHYk

https://www.youtube.com/watch?v=4t4kBkMsDbQ

https://ravitejbandlekar.medium.com/nmap-101-a-beginners-guide-to-network-mapping-and-security-a7ec875e52b

https://www.reddit.com/r/tryhackme/comments/1alan6y/fast_way_to_scan_all_ports_using_nmap/

https://github.com/topics/kali-tools?o=asc&s=forks

https://prezi.com/p/hh621xfgyed3/exploring-nmap-tool-in-kali-linux/