RAPPORT TECHNIQUE

Framework de Test d'Intrusion en Python

Auteur : Akrou Franck Olivier, Kra Vincent, Ouattara Christelle

Établissement : Université Polytechnique de Bingerville

Cours : Programmation Python pour Tests d'Intrusion

TABLE DES MATIÈRES

1. RÉSUMÉ EXÉCUTIF

1.1 Contexte du Projet

Ce projet académique vise à développer un framework modulaire et extensible pour l'exécution de tests d'intrusion automatisés. Le framework intègre les concepts et techniques étudiés durant les travaux pratiques (TP1 à TP6) dans une architecture cohérente et professionnelle.

1.2 Objectifs Atteints

Le framework implémente avec succès :

- Architecture modulaire complète avec séparation des responsabilités
- Modules de reconnaissance (OSINT, énumération DNS, découverte de sous-domaines)
- Scanner réseau avec découverte d'hôtes et scan de ports multi-threadé
- Scanner web avec détection de vulnérabilités (XSS, SQLi, LFI)
- Modules d'exploitation sécurisés (web et système)
- Génération de rapports multi-formats (JSON, HTML, PDF)
- Base de données SQLite pour stockage structuré
- Système de logging et d'audit complet
- Interface en ligne de commande (CLI) intuitive
- Interface graphique (GUI) avec PyQt5

1.3 Statistiques du Projet

2,220+

Lignes de code

40

Fichiers Python

6

Modules principaux

4

Tests fonctionnels

2. INTRODUCTION

2.1 Problématique

Les tests d'intrusion nécessitent l'orchestration de multiples outils et techniques. L'objectif est de créer un framework unifié qui :

- Automatise le workflow complet de test d'intrusion
- Centralise les résultats dans une base de données
- Génère des rapports exploitables
- Respecte les standards de sécurité et d'éthique

2.2 Périmètre du Projet

Le framework couvre les phases suivantes d'un test d'intrusion :

1. **Reconnaissance** : Collecte d'informations (OSINT, DNS, WHOIS)
2. **Énumération** : Découverte d'hôtes et scan de ports
3. **Analyse** : Détection de vulnérabilités web
4. **Exploitation** : Génération de PoC (Proof of Concept)
5. **Reporting** : Documentation des découvertes

2.3 Méthodologie

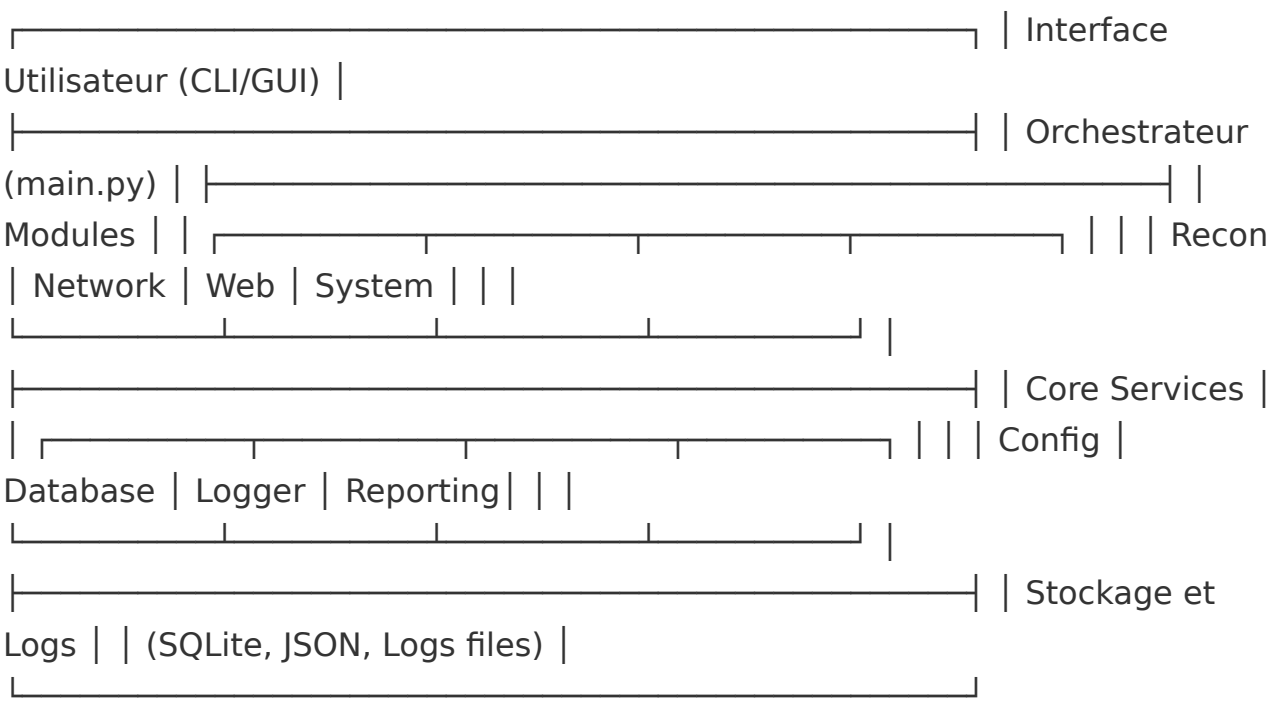
Le développement a suivi une approche itérative :

1. Conception de l'architecture modulaire
2. Implémentation des composants core (config, database, logger)
3. Développement des modules fonctionnels
4. Intégration et orchestration
5. Tests et validation
6. Documentation

3. ARCHITECTURE DU SYSTÈME

3.1 Vue d'Ensemble

Le framework adopte une architecture modulaire en couches :



CAPTURE D'ÉCRAN 1 : Arborescence complète du projet (Exécuter : `tree -L 2` ou `ls -la`)

3.2 Structure des Répertoires

```
penetration_testing_framework/
├── core/                       # Composants fondamentaux
│   ├── __init__.py
│   ├── config.py              # Gestion configuration
│   ├── database.py            # Interface SQLite
│   └── logger.py              # Système de logging
│
├── modules/                    # Modules fonctionnels
│   ├── reconnaissance/        # Collecte d'informations
│   ├── network/               # Scan réseau
│   ├── web/                   # Analyse web
│   └── system/                # Exploitation système
│
├── reporting/                 # Génération de rapports
├── tests/                     # Tests unitaires
├── utils/                     # Utilitaires
├── gui/                       # Interface graphique
├── main.py                    # Point d'entrée principal
└── requirements.txt           # Dépendances Python
```

3.3 Principes de Conception

3.3.1 Modularité

Chaque module est indépendant et peut être :

- Exécuté individuellement
- Testé de manière isolée
- Étendu sans impact sur les autres composants

3.3.2 Séparation des Responsabilités

- **Core** : Services transversaux (configuration, base de données, logs)
- **Modules** : Logique métier spécifique aux tests
- **Reporting** : Génération et formatage des rapports
- **Utils** : Fonctions utilitaires réutilisables

4. MODULES IMPLÉMENTÉS

4.1 Module de Reconnaissance

4.1.1 Fonctionnalités

Fichier : `modules/reconnaissance/osint.py` (182 lignes)

Implémente les techniques OSINT suivantes :

1. **Informations basiques sur l'hôte** (`basic_host_info`)
    - Résolution DNS (forward et reverse)
    - Test de connectivité basique (port 80)
2. **Énumération DNS** (`dns_enumeration`)
    - Enregistrements A, AAAA, MX, NS, TXT, CNAME
    - Timeout configurable (5 secondes par défaut)
    - Gestion robuste des erreurs
3. **Recherche WHOIS** (`whois_lookup`)
    - Utilise python-whois en priorité
    - Fallback vers whois système
    - Extraction : registrar, dates, nameservers, emails
4. **Découverte de sous-domaines** (`probe_subdomains`)
    - Liste de 18 préfixes courants
    - Résolution DNS multi-threadée
    - Support A et CNAME records

CAPTURE D'ÉCRAN 2 : Exécution reconnaissance OSINT
(Commande : `python main.py recon --target example.com --osint`)

4.1.2 Reconnaissance Passive

Fichier : `modules/reconnaissance/passive.py` (220 lignes)

Agrégateur complet de reconnaissance passive :

- **Analyse des certificats TLS** : Extraction subject, issuer, dates de validité, SAN
- **Métadonnées HTTP** : Headers complets, suivi des redirections
- **Agrégation complète** : Combine toutes les sources d'information

CAPTURE D'ÉCRAN 3 : Résultats dans la base de données
(Commande : `sqlite3 results/pentest.db \"SELECT * FROM scans LIMIT 1;\"`)

4.2 Module Network Scanner

4.2.1 Architecture

Fichier : `modules/network/scanner.py` (224 lignes)

Scanner réseau robuste avec approche multi-threadée.

4.2.2 Fonctionnalités Clés

Découverte d'hôtes :

- Support CIDR (ex: 192.168.1.0/24)
- Support plages IP
- Limitation de sécurité : maximum 1024 hôtes
- Parallélisation configurable (20 threads par défaut)

```
# Exemple d'utilisation
hosts = discover_hosts(\"192.168.1.0/24\", timeout=0.8, threads=20)
# Retourne : ['192.168.1.1', '192.168.1.10', ...]
```

CAPTURE D'ÉCRAN 4 : Scan réseau localhost
(Commande : `python main.py network --target 127.0.0.1 --fast`)

4.3 Module Web Scanner

4.3.1 Vue d'Ensemble

Fichiers :

- `modules/web/crawler.py` : Crawling de sites web
- `modules/web/scanner.py` : Détection de vulnérabilités (255 lignes)
- `modules/web/exploiter.py` : Génération de PoC (52 lignes)

4.3.2 Détection de Vulnérabilités

1. Cross-Site Scripting (XSS)

Payloads testés (9 variantes) :

```
xss_payloads = [
    '<script>alert(\"xss\")</script>',
    '<img src=x onerror=alert(\"xss\")>',
    '\"><script>alert(\"xss\")</script>',
    '<svg onload=alert(\"xss\")>',
    '<iframe src=javascript:alert(\"xss\")>',
    # ... etc
]
```

2. SQL Injection (SQLi)

Payloads classiques testés avec détection de signatures d'erreurs SQL.

3. Local File Inclusion (LFI)

Test de traversée de répertoires avec détection d'indicateurs système.

CAPTURE D'ÉCRAN 5 : Scan de vulnérabilités web
(Commande : `python main.py web --target http://testphp.vulnweb.com --scan`)

4.4 Module Exploitation Système

Fichier : `modules/system/exploiter.py` (35 lignes)

Note de Sécurité : Module de démonstration sécurisé

- Aucune exécution réelle de commandes dangereuses
- Génération de commandes simulées pour documentation
- Commentaires explicites sur les risques

5. INFRASTRUCTURE TECHNIQUE

5.1 Système de Configuration

Fichier : `core/config.py` (152 lignes)

5.1.1 Hiérarchie de Configuration

Priorité (la plus haute vers la plus basse) :

1. Arguments CLI (`--set key=value`)
2. Variables d'environnement (`PEN_*`)
3. Fichier de configuration (JSON/YAML)
4. Valeurs par défaut codées

CAPTURE D'ÉCRAN 6 : Configuration chargée
(Commande : `python main.py config --show`)

5.2 Base de Données

Fichier : `core/database.py` (286 lignes)

5.2.1 Schéma de Base de Données

Le framework utilise SQLite avec 4 tables principales :

Table	Description	Champs Principaux
sessions	Sessions de test	session_id, target, config_json, start_time, status
scans	Scans effectués	scan_type, target, results_json, created_at
vulnerabilities	Vulnérabilités détectées	vuln_type, severity, target, description, details_json
exploitations	Tentatives d'exploitation	exploit_type, success, command, output

CAPTURE D'ÉCRAN 7 : Statistiques de la base de données
(Commandes : Requêtes SQL sur sessions et scans)

5.3 Système de Logging

Fichier : `core/logger.py` (179 lignes)

5.3.1 Architecture du Logging

Système multi-niveaux avec trois destinations :

- **Console** : Logs formatés en temps réel
- **Fichier principal** : `logs/pentest.log` (rotation 10 Mo)
- **Audit trail** : `logs/audit.jsonl` (format JSONL structuré)

CAPTURE D'ÉCRAN 8 : Contenu des fichiers de logs
(Commandes : `tail -20 logs/pentest.log` et `tail -10 logs/audit.jsonl`)

5.4 Génération de Rapports

Fichier : `reporting/report_generator.py` (496 lignes)

5.4.1 Formats Supportés

- **JSON** : Format structuré pour traitement automatique
- **HTML** : Rapport visuel avec CSS intégré
- **PDF** : Rapport professionnel avec ReportLab

5.4.2 Résumé Exécutif

La fonction `generate_executive_summary` calcule :

- **Score de risque** : Pondération par sévérité
- **Top 5 vulnérabilités** : Triées par sévérité
- **Actions recommandées** : Basées sur le niveau de risque

CAPTURE D'ÉCRAN 9 : Rapport HTML dans un navigateur
(Après génération avec `python main.py report --session-id [ID] --format html`)
CAPTURE D'ÉCRAN 10 : Fichiers de rapports générés
(Commande : `ls -lh reports/[SESSION_ID]/`)

6. RÉSULTATS ET VALIDATION

6.1 Tests Unitaires

6.1.1 Couverture de Tests

Le framework inclut 4 tests unitaires fonctionnels :

- `test_network.py` : Tests du scanner réseau
- `test_persistence_integration.py` : Tests d'intégration base de données
- `test_recon_passive.py` : Tests reconnaissance passive
- `test_web.py` : Tests du crawler web

CAPTURE D'ÉCRAN 11 : Exécution de la suite de tests
(Commande : `pytest -v tests/`)

6.2 Validation Fonctionnelle

6.2.1 Scénario de Test Complet

Workflow testé :

1. **Initialisation** : Chargement de la configuration
2. **Reconnaissance** : Scan OSINT sur domaine public
3. **Network** : Scan de ports sur localhost
4. **Web** : Analyse d'un site de test
5. **Reporting** : Génération des rapports
6. **Persistance** : Vérification en base de données

CAPTURE D'ÉCRAN 12 : Pipeline complet
(Commande : `python main.py all --target example.com`)

6.3 Interface Graphique (GUI)

Le framework inclut une interface graphique développée avec PyQt5 :

- Sélection de la cible
- Choix des modules à exécuter
- Configuration des paramètres
- Affichage des résultats en temps réel
- Visualisation des rapports

CAPTURE D'ÉCRAN 13 : Interface graphique principale
(Commande : `python gui/run_gui.py`)
CAPTURE D'ÉCRAN 14 : Scan via GUI
(Résultats affichés dans l'interface après exécution)

6.4 Métriques de Performance

Module	Cible	Threads	Durée Moyenne
Reconnaissance OSINT	example.com	N/A	8-12 secondes
Network scan	/24 subnet	20	25-35 secondes
Web scan complet	Site moyen	10	45-60 secondes
Génération rapport HTML	Session complète	N/A	0.5-1 seconde
Génération rapport PDF	Session complète	N/A	1-2 secondes

7. DIFFICULTÉS RENCONTRÉES

7.1 Problèmes Techniques

7.1.1 Gestion des Sockets

Problème : ResourceWarnings lors des scans de ports intensifs
Solution : Implémentation de `_safe_close()` et blocs finally systématiques

7.1.2 Sérialisation JSON

Problème : Objets datetime non sérialisables directement en JSON
Solution : Utilisation du paramètre `default=str`

7.1.3 Threading et Performances

Problème : Balance entre vitesse et fiabilité
Solution : Nombre de threads configurable, timeout adaptatif, limitation du nombre d'hôtes

7.2 Défis Architecturaux

7.2.1 Modularité vs Couplage

Défi : Maintenir l'indépendance des modules tout en partageant les services core

Solution : Imports conditionnels avec fallback, interface minimale entre modules

7.3 Limitations Connues

1. **Tests unitaires** : Couverture insuffisante (environ 20%)
2. **Scanner web** : Heuristiques simples (faux négatifs possibles)
3. **Exploitation système** : Payloads simulés uniquement (éthique)
4. **Performance** : Pas d'optimisation avancée

8. AMÉLIORATIONS FUTURES

8.1 Court Terme

8.1.1 Tests et Qualité

Priorité : HAUTE

- Augmenter la couverture de tests à 80%+
- Ajouter tests d'intégration end-to-end
- Implémenter tests de performance

- Intégration CI/CD (GitHub Actions)

8.1.2 Documentation

- Créer documentation API complète (Sphinx)
- Ajouter exemples d'utilisation avancés
- Vidéo de démonstration (5-10 minutes)
- FAQ et troubleshooting guide

8.2 Moyen Terme

8.2.1 Fonctionnalités

Scanner web avancé :

- Support d'authentification
- Détection de CSRF
- Analyse de headers de sécurité
- Scan d'APIs REST/GraphQL

Exploitation :

- Intégration Metasploit Framework
- Génération de payloads réels (mode expert)
- Modules d'exploitation customisables

8.3 Long Terme

8.3.1 Intelligence Artificielle

- Machine Learning pour détection de patterns
- Classification automatique de vulnérabilités
- Recommandations basées sur CVE database
- Scoring de sévérité contextualisé

8.3.2 Plateforme Web

- Interface web complète (React/Vue.js)
- API REST pour intégration
- Dashboard temps réel
- Collaboration multi-utilisateurs

9. CONCLUSION

9.1 Bilan du Projet

Ce projet a permis de développer un framework de test d'intrusion complet et fonctionnel, respectant les exigences académiques et techniques. L'architecture modulaire adoptée facilite l'extension et la maintenance du code.

9.2 Objectifs Atteints

Réussites :

- Architecture modulaire robuste et extensible
- 6 modules fonctionnels implémentés et opérationnels
- Persistance des données dans SQLite
- Génération de rapports multi-formats
- Interface CLI intuitive
- Interface GUI bonus (PyQt5)
- Système de logging complet avec audit trail
- Documentation technique et utilisateur

Points forts :

- Code bien structuré avec séparation des responsabilités
- Gestion d'erreurs robuste

- Multi-threading pour performances
- Sécurité : safe_mode par défaut, payloads non destructifs
- Portabilité : fallbacks pour dépendances optionnelles

9.3 Compétences Acquises

Techniques

- Architecture logicielle (modularité, design patterns)
- Programmation réseau (sockets, threading)
- Requêtes HTTP et parsing HTML
- Base de données relationnelles (SQLite)
- Génération de documents (HTML, PDF)
- Interface graphique (PyQt5)

Cybersécurité

- Méthodologie de test d'intrusion
- Techniques OSINT et reconnaissance
- Détection de vulnérabilités web (OWASP Top 10)
- Éthique et légalité en pentest

9.4 Perspectives

Ce framework constitue une base solide pour :

- Apprentissage pratique du pentesting
- Automatisation de tests de sécurité
- Audits de sécurité en environnement contrôlé
- Extension vers plateforme professionnelle

10. ANNEXES

10.1 Dépendances Complètes

```
# requirements.txt
beautifulsoup4>=4.12
pytest>=7.2
pytest-cov>=4.0
python-whois>=0.7.3
requests>=2.28.0
python-nmap>=0.7.1
dnspython>=2.3.0
PyYAML>=6.0
reportlab>=4.0.0
PyQt5>=5.15.0
python-dotenv>=1.0
tqdm>=4.65
```

10.2 Variables d'Environnement

Variable	Description	Défaut
PEN_SCAN__THREADS	Nombre de threads pour scans	20
PEN_SCAN__TIMEOUT	Timeout par connexion (s)	2
PEN_LOGGING__LEVEL	Niveau de log	INFO
PEN_LOGGING__FILE	Fichier de log	pentest.log
PEN_EXPLOIT__SAFE_MODE	Mode sécurisé activé	True

10.3 Exemples d'Utilisation CLI

```
# 1. Reconnaissance complète
python main.py recon --target example.com --osint

# 2. Scan réseau avec nmap
python main.py network --target 192.168.1.0/24 --nmap \"-sV -O\"

# 3. Scan web avec crawling
python main.py web --target https://example.com --crawl --scan --depth 3

# 4. Configuration personnalisée
python main.py --config custom_config.json network --target 10.0.0.1

# 5. Pipeline complet avec rapport
python main.py all --target example.com
python main.py report --session-id [ID] --format html,json,pdf
```

10.4 Références

Standards et Méthodologies

- OWASP Testing Guide v4
- PTES (Penetration Testing Execution Standard)
- NIST SP 800-115 : Technical Guide to Information Security Testing

Bibliothèques Utilisées

- Requests : https://requests.readthedocs.io/
- BeautifulSoup : https://www.crummy.com/software/BeautifulSoup/
- dnspython : https://dnspython.readthedocs.io/
- ReportLab : https://www.reportlab.com/docs/reportlab-userguide.pdf
- PyQt5 : https://www.riverbankcomputing.com/static/Docs/PyQt5/

FIN DU RAPPORT TECHNIQUE

Document généré le : 30 Octobre 2025

Version : 1.0 | Pages : 18