Projet final

Python pour les Tests d'Intrusion

Master 1 SAS

Dr. ATTA Amanvon Ferdinand

Année académique 2025-2026

Avertissement Critique

AVERTISSEMENT LÉGAL ET ÉTHIQUE CRITIQUE
RESTRICTIONS ABSOLUES :

- **Uniquement** sur machines virtuelles dédiées et isolées
- **Jamais** sur systèmes de production ou infrastructures réelles
- **Logging complet** : Toutes les actions sont tracées et auditables

TOUT USAGE NON AUTORISÉ CONSTITUE UNE INFRACTION PÉNALE GRAVE
Les techniques enseignées visent à comprendre les mécanismes d'attaque système pour mieux concevoir des défenses robustes. L'usage malveillant engage la responsabilité pénale et civile de l'utilisateur.

Table des matières

1 Projet Final - Framework Modulaire de Test d'Intrusion

1.1 Présentation du projet

1.1.1 Contexte et objectifs

Le projet final consiste à développer un framework modulaire de test d'intrusion qui intègre l'ensemble des compétences acquises durant les TP1 à TP6. Ce framework doit être capable de réaliser un audit de sécurité complet, de la reconnaissance initiale jusqu'à l'exploitation et la génération de rapports.

Information

Objectifs du projet

- Intégrer tous les modules développés dans les TP1-6
- Créer une architecture modulaire et extensible
- Développer une IHM et une interface en ligne professionnelle
- Automatiser le workflow complet de test d'intrusion
- Générer des rapports techniques et exécutifs
- Respecter les standards de sécurité et d'éthique

1.1.2 Durée et modalités

- **Deadline :** 31/10/2025
- **Modalité :** Projet en groupe
- **Soutenance :** Présentation de 20 minutes + 10 minutes de questions
- **Livrables :**
  - Code source complet du framework
  - Rapport technique (15-20 pages)
  - Documentation utilisateur
  - Démonstration vidéo (optionnelle, bonus)

1.2 Cahier des charges technique

1.2.1 Exigences fonctionnelles

Le framework doit implémenter les modules suivants :

Module	Fonctionnalités requises
Reconnaissance	OSINT, DNS enumeration, subdomain discovery, WHOIS
Scanning réseau	Port scanning, service detection, OS fingerprinting
Détection web	Crawling, vulnerability scanning (SQLi, XSS, LFI, etc.)
Exploitation web	Validation automatique, génération de PoC
Exploitation système	Buffer overflow, shellcode, reverse shells
Reporting	Génération JSON, HTML, PDF

TABLE 1 – Modules fonctionnels requis

1.2.2 Architecture technique

Le framework doit respecter l'architecture suivante :

```
# Architecture du framework
penetration_testing_framework/
|---- core/
|   |---- __init__.py
|   |---- config.py          # Configuration globale
|   |---- logger.py          # Système de logging
|   `---- database.py        # Gestion des résultats
|---- modules/
|   |---- reconnaissance/
|   |   |---- __init__.py
|   |   |---- osint.py       # TP2
|   |   `---- passive.py
|   |---- network/
|   |   |---- __init__.py
|   |   |---- scanner.py     # TP3
|   |   `---- enumeration.py
|   |---- web/
|   |   |---- __init__.py
|   |   |---- crawler.py     # TP4
|   |   |---- scanner.py
|   |   `---- exploiter.py   # TP5
|   `---- system/
|       |---- __init__.py
|       |---- exploiter.py   # TP6
|       `---- shells.py
|---- utils/
|   |---- __init__.py
|   |---- network_utils.py
|   `---- file_utils.py
|---- reporting/
|   |---- __init__.py
|   |---- report_generator.py
|   `---- templates/
|---- tests/
|   |---- __init__.py
|   `---- test_*.py
|---- main.py                # Point d'entrée
|---- requirements.txt
`---- README.md
```

1.2.3 Exigences non-fonctionnelles

- **Modularité** : Chaque module doit être indépendant et réutilisable
- **Extensibilité** : Possibilité d'ajouter de nouveaux modules facilement
- **Performance** : Support du multi-threading pour les scans
- **Robustesse** : Gestion d'erreurs complète avec retry mechanisms
- **Sécurité** : Logging de toutes les actions, validation des entrées
- **Documentation** : Code commenté, docstrings Python, README complet

1.3 Interface en ligne de commande

1.3.1 Commandes principales

Le framework doit fournir une interface CLI intuitive :

```
# Scan complet
python main.py scan --target example.com --full

# Reconnaissance uniquement
python main.py recon --target example.com --osint

# Scan réseau
python main.py network --target 192.168.1.0/24 --ports 1-1000

# Scan web
python main.py web --url http://example.com --crawl --scan

# Exploitation
python main.py exploit --target example.com --auto

# Génération de rapport
python main.py report --session-id abc123 --format pdf

# Configuration
python main.py config --set threads=20 --set timeout=30
```

1.3.2 Système de configuration

Exercice

Implémentation requise - Système de configuration
Développez un système de configuration flexible supportant :

- Fichiers de configuration JSON/YAML
- Variables d'environnement
- Arguments en ligne de commande
- Configuration par défaut

Exemple de fichier de configuration :

```
{
  "global": {
    "threads": 10,
    "timeout": 30,
    "user_agent": "PenTest-Framework/1.0",
    "output_dir": "./results"
  },
  "reconnaissance": {
    "osint_enabled": true,
    "subdomain_bruteforce": true,
    "dns_enumeration": true
  },
  "network": {
    "port_range": "1-65535",
    "scan_type": "syn",
    "os_detection": true
  },
  "web": {
    "crawl_depth": 3,
    "max_pages": 500,
    "scan_forms": true,
    "test_all_parameters": true
  },
  "exploitation": {
    "auto_exploit": false,
    "generate_poc": true,
    "safe_mode": true
  },
  "reporting": {
    "formats": ["json", "html", "pdf"],
    "include_screenshots": false,
    "executive_summary": true
  }
}
```

1.4 IHM

Pour l'IHM, le soin est laissé au groupe de présenter comme il le souhaite et en utilisant la bibliothèque de son choix(tkinter, pyqt, etc..)

1.5 Système de logging et traçabilité

1.5.1 Architecture du logging

Le framework doit implémenter un système de logging complet :

```python
#!/usr/bin/env python3
"""
Système de logging centralisé pour le framework
Support de plusieurs niveaux et destinations
"""
import logging
import json
from datetime import datetime
from pathlib import Path
from typing import Dict, Any

class PentestLogger:
    """Logger centralisé pour le framework"""

    def __init__(self, log_dir: str = "./logs"):
        self.log_dir = Path(log_dir)
        self.log_dir.mkdir(exist_ok=True)

        # Configuration du logger principal
        self.logger = logging.getLogger('PentestFramework')
        self.logger.setLevel(logging.DEBUG)

        # Handler pour fichier avec rotation
        from logging.handlers import RotatingFileHandler

        file_handler = RotatingFileHandler(
            self.log_dir / 'framework.log',
            maxBytes=10*1024*1024,  # 10MB
            backupCount=5
        )
        file_handler.setLevel(logging.DEBUG)

        # Handler pour console
        console_handler = logging.StreamHandler()
        console_handler.setLevel(logging.INFO)

        # Format détaillé
        formatter = logging.Formatter(
            '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
        )

        file_handler.setFormatter(formatter)
        console_handler.setFormatter(formatter)

        self.logger.addHandler(file_handler)
        self.logger.addHandler(console_handler)

        # Fichier d'audit séparé
        self.audit_file = self.log_dir / 'audit.jsonl'

    def log_action(self, action: str, target: str,
                   details: Dict[str, Any]):
```

```
        """Log une action avec détails"""

        audit_entry = {
            'timestamp': datetime.now().isoformat(),
            'action': action,
            'target': target,
            'details': details
        }

        # Log dans le fichier d'audit
        with open(self.audit_file, 'a') as f:
            f.write(json.dumps(audit_entry) + '\n')

        # Log dans le logger principal
        self.logger.info(f"{action} on {target}")

    def log_scan_start(self, scan_type: str, target: str):
        """Log le démarrage d'un scan"""
        self.log_action('SCAN_START', target, {'type': scan_type})

    def log_vulnerability(self, vuln_type: str, target: str,
                          severity: str):
        """Log une vulnérabilité détectée"""
        self.log_action('VULNERABILITY', target, {
            'type': vuln_type,
            'severity': severity
        })

    def log_exploitation(self, exploit_type: str, target: str,
                         success: bool):
        """Log une tentative d'exploitation"""
        self.log_action('EXPLOITATION', target, {
            'type': exploit_type,
            'success': success
        })
```

1.6 Gestion des résultats et base de données

1.6.1 Stockage des résultats

Le framework doit stocker tous les résultats de manière structurée :

```
#!/usr/bin/env python3
"""
Système de gestion des résultats
Base de données SQLite pour stockage persistent
"""
import sqlite3
import json
from datetime import datetime
from typing import List, Dict, Any

class ResultDatabase:
    """Gestionnaire de base de données des résultats"""

    def __init__(self, db_path: str = "./results/pentest.db"):
        self.db_path = db_path
        self.conn = sqlite3.connect(db_path)
        self.create_tables()

    def create_tables(self):
        """Crée les tables nécessaires"""

```

```
22          cursor = self.conn.cursor()
23
24          # Table des sessions
25          cursor.execute('''
26              CREATE TABLE IF NOT EXISTS sessions (
27                  session_id TEXT PRIMARY KEY,
28                  target TEXT NOT NULL,
29                  start_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
30                  end_time TIMESTAMP,
31                  status TEXT DEFAULT 'running',
32                  config TEXT
33              )
34          ''')
35
36          # Table des scans
37          cursor.execute('''
38              CREATE TABLE IF NOT EXISTS scans (
39                  scan_id INTEGER PRIMARY KEY AUTOINCREMENT,
40                  session_id TEXT,
41                  scan_type TEXT NOT NULL,
42                  target TEXT NOT NULL,
43                  start_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
44                  end_time TIMESTAMP,
45                  results TEXT,
46                  FOREIGN KEY (session_id) REFERENCES sessions(session_id)
47              )
48          ''')
49
50          # Table des vulnérabilités
51          cursor.execute('''
52              CREATE TABLE IF NOT EXISTS vulnerabilities (
53                  vuln_id INTEGER PRIMARY KEY AUTOINCREMENT,
54                  session_id TEXT,
55                  scan_id INTEGER,
56                  vuln_type TEXT NOT NULL,
57                  severity TEXT NOT NULL,
58                  target TEXT NOT NULL,
59                  description TEXT,
60                  evidence TEXT,
61                  remediation TEXT,
62                  discovered_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
63                  FOREIGN KEY (session_id) REFERENCES sessions(session_id),
64                  FOREIGN KEY (scan_id) REFERENCES scans(scan_id)
65              )
66          ''')
67
68          # Table des exploitations
69          cursor.execute('''
70              CREATE TABLE IF NOT EXISTS exploitations (
71                  exploit_id INTEGER PRIMARY KEY AUTOINCREMENT,
72                  session_id TEXT,
73                  vuln_id INTEGER,
74                  exploit_type TEXT NOT NULL,
75                  target TEXT NOT NULL,
76                  success BOOLEAN NOT NULL,
77                  proof_of_concept TEXT,
78                  executed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
79                  FOREIGN KEY (session_id) REFERENCES sessions(session_id),
80                  FOREIGN KEY (vuln_id) REFERENCES vulnerabilities(vuln_id)
81              )
82          ''')
83
84          self.conn.commit()
```

```
85
86      def create_session(self, session_id: str, target: str,
87                         config: Dict[str, Any]) -> str:
88          """Crée une nouvelle session"""
89
90          cursor = self.conn.cursor()
91          cursor.execute('''
92              INSERT INTO sessions (session_id, target, config)
93              VALUES (?, ?, ?)
94          ''', (session_id, target, json.dumps(config)))
95
96          self.conn.commit()
97          return session_id
98
99      def add_vulnerability(self, session_id: str, vuln_data: Dict[str, Any]):
100         """Ajoute une vulnérabilité"""
101
102         cursor = self.conn.cursor()
103         cursor.execute('''
104             INSERT INTO vulnerabilities
105             (session_id, vuln_type, severity, target, description,
106              evidence, remediation)
107             VALUES (?, ?, ?, ?, ?, ?, ?)
108         ''', (
109             session_id,
110             vuln_data['type'],
111             vuln_data['severity'],
112             vuln_data['target'],
113             vuln_data.get('description', ''),
114             json.dumps(vuln_data.get('evidence', {})),
115             vuln_data.get('remediation', '')
116         ))
117
118         self.conn.commit()
119
120     def get_session_results(self, session_id: str) -> Dict[str, Any]:
121         """Récupère tous les résultats d'une session"""
122
123         cursor = self.conn.cursor()
124
125         # Informations de session
126         cursor.execute('''
127             SELECT * FROM sessions WHERE session_id = ?
128         ''', (session_id,))
129
130         session = cursor.fetchone()
131
132         # Vulnérabilités
133         cursor.execute('''
134             SELECT * FROM vulnerabilities WHERE session_id = ?
135         ''', (session_id,))
136
137         vulnerabilities = cursor.fetchall()
138
139         # Exploitations
140         cursor.execute('''
141             SELECT * FROM exploitations WHERE session_id = ?
142         ''', (session_id,))
143
144         exploitations = cursor.fetchall()
145
146         return {
147             'session': session,
```

```
            'vulnerabilities': vulnerabilities,
            'exploitations': exploitations
        }
```

1.7 Génération de rapports

1.7.1 Format du rapport technique

Le rapport technique doit suivre la structure imposée suivante :

Section	Pages	Contenu
Résumé exécutif	1	Vue d'ensemble, risques majeurs
Analyse des besoins	2-3	Objectifs, scope, contraintes
Conception	3-4	Architecture, choix techniques
Implémentation	4-5	Modules, algorithmes, code
Tests et validation	2-3	Scénarios de test, résultats
Analyse critique	2-3	Limites, améliorations futures
Annexes	Variable	Code source, captures, références

TABLE 2 – Structure du rapport technique

1.7.2 Template de rapport automatisé

```
#!/usr/bin/env python3
"""
Générateur de rapports automatisé
Support de plusieurs formats (JSON, HTML, PDF)
"""
from typing import Dict, Any, List
from datetime import datetime
import json
from pathlib import Path

class ReportGenerator:
    """Générateur de rapports multi-formats"""

    def __init__(self, session_id: str, results: Dict[str, Any]):
        self.session_id = session_id
        self.results = results
        self.output_dir = Path(f"./reports/{session_id}")
        self.output_dir.mkdir(parents=True, exist_ok=True)

    def generate_all(self):
        """Génère tous les formats de rapport"""

        self.generate_json()
        self.generate_html()
        self.generate_executive_summary()

        print(f"[+] Reports generated in {self.output_dir}")

    def generate_json(self):
        """Génère un rapport JSON détaillé"""

        report_data = {
            'metadata': {
                'session_id': self.session_id,
                'generated_at': datetime.now().isoformat(),
```

```
                'framework_version': '1.0.0'
            },
            'target': self.results.get('target', ''),
            'summary': self._generate_summary(),
            'reconnaissance': self.results.get('reconnaissance', {}),
            'network_scan': self.results.get('network', {}),
            'web_vulnerabilities': self.results.get('web_vulns', []),
            'system_vulnerabilities': self.results.get('system_vulns', []),
            'exploitations': self.results.get('exploitations', [])
        }

        output_file = self.output_dir / 'report.json'

        with open(output_file, 'w') as f:
            json.dump(report_data, f, indent=2, default=str)

        print(f"[+] JSON report: {output_file}")

    def generate_html(self):
        """Génère un rapport HTML interactif"""

        html_template = f'''
<!DOCTYPE html>
<html>
<head>
    <title>Penetration Testing Report - {self.session_id}</title>
    <style>
        body {{
            font-family: Arial, sans-serif;
            margin: 20px;
            background: #f5f5f5;
        }}
        .header {{
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
            padding: 30px;
            border-radius: 10px;
            text-align: center;
        }}
        .section {{
            background: white;
            padding: 20px;
            margin: 20px 0;
            border-radius: 5px;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        }}
        .critical {{ color: #dc3545; font-weight: bold; }}
        .high {{ color: #fd7e14; font-weight: bold; }}
        .medium {{ color: #ffc107; font-weight: bold; }}
        .low {{ color: #28a745; font-weight: bold; }}
        table {{
            width: 100%;
            border-collapse: collapse;
            margin: 15px 0;
        }}
        th, td {{
            border: 1px solid #ddd;
            padding: 12px;
            text-align: left;
        }}
        th {{
            background-color: #667eea;
            color: white;
```

```
        }}
    </style>
</head>
<body>
    <div class="header">
        <h1>Penetration Testing Report</h1>
        <p>Session: {self.session_id}</p>
        <p>Generated: {datetime.now().strftime("%Y-%m-%d %H:%M:%S")}</p>
    </div>

    <div class="section">
        <h2>Executive Summary</h2>
        {self._generate_html_summary()}
    </div>

    <div class="section">
        <h2>Vulnerabilities Discovered</h2>
        {self._generate_html_vulnerabilities()}
    </div>

    <div class="section">
        <h2>Exploitation Results</h2>
        {self._generate_html_exploitations()}
    </div>

    <div class="section">
        <h2>Recommendations</h2>
        {self._generate_html_recommendations()}
    </div>
</body>
</html>
        '''

        output_file = self.output_dir / 'report.html'

        with open(output_file, 'w') as f:
            f.write(html_template)

        print(f"[+] HTML report: {output_file}")

    def generate_executive_summary(self):
        """Génère un résumé exécutif"""

        summary = f'''
PENETRATION TESTING - EXECUTIVE SUMMARY
Session: {self.session_id}
Date: {datetime.now().strftime("%Y-%m-%d")}

TARGET INFORMATION
Target: {self.results.get('target', 'N/A')}
Scope: {self.results.get('scope', 'N/A')}

RISK ASSESSMENT
Overall Risk Level: {self._calculate_risk_level()}
Critical Vulnerabilities: {self._count_by_severity('CRITICAL')}
High Vulnerabilities: {self._count_by_severity('HIGH')}
Medium Vulnerabilities: {self._count_by_severity('MEDIUM')}
Low Vulnerabilities: {self._count_by_severity('LOW')}

KEY FINDINGS
{self._generate_key_findings()}

IMMEDIATE ACTIONS REQUIRED
```

```
{self._generate_immediate_actions()}

COMPLIANCE IMPACT
{self._generate_compliance_assessment()}
        '''

        output_file = self.output_dir / 'executive_summary.txt'

        with open(output_file, 'w') as f:
            f.write(summary)

        print(f"[+] Executive summary: {output_file}")

    def _generate_summary(self) -> Dict[str, Any]:
        """Génère un résumé des résultats"""

        return {
            'total_vulnerabilities': len(self.results.get('web_vulns', [])) +
                                     len(self.results.get('system_vulns', [])),
            'critical_count': self._count_by_severity('CRITICAL'),
            'high_count': self._count_by_severity('HIGH'),
            'medium_count': self._count_by_severity('MEDIUM'),
            'low_count': self._count_by_severity('LOW'),
            'exploitations_attempted': len(self.results.get('exploitations', [])),
            'exploitations_successful': len([e for e in self.results.get('
    exploitations', [])
                                             if e.get('success')])
        }

    def _count_by_severity(self, severity: str) -> int:
        """Compte les vulnérabilités par sévérité"""

        web_vulns = self.results.get('web_vulns', [])
        system_vulns = self.results.get('system_vulns', [])

        count = 0

        for vuln in web_vulns + system_vulns:
            if vuln.get('severity') == severity:
                count += 1

        return count

    def _calculate_risk_level(self) -> str:
        """Calcule le niveau de risque global"""

        critical = self._count_by_severity('CRITICAL')
        high = self._count_by_severity('HIGH')

        if critical > 0:
            return "CRITICAL"
        elif high > 2:
            return "HIGH"
        elif high > 0:
            return "MEDIUM"
        else:
            return "LOW"

    def _generate_key_findings(self) -> str:
        """Génère les découvertes clés"""

        findings = []

```

```
        vulns = self.results.get('web_vulns', []) + self.results.get('system_vulns', [])

        # Top 5 des vulnérabilités les plus critiques
        sorted_vulns = sorted(vulns,
                              key=lambda x: {'CRITICAL': 4, 'HIGH': 3,
                                             'MEDIUM': 2, 'LOW': 1}.get(x.get('severity', 'LOW'), 0),
                              reverse=True)

        for i, vuln in enumerate(sorted_vulns[:5], 1):
            findings.append(
                f"{i}. [{vuln.get('severity')}] {vuln.get('type')} on {vuln.get('target')}"
            )

        return '\n'.join(findings)

    def _generate_immediate_actions(self) -> str:
        """Génère les actions immédiates requises"""

        actions = []

        critical_vulns = [v for v in self.results.get('web_vulns', []) +
                          self.results.get('system_vulns', [])
                          if v.get('severity') == 'CRITICAL']

        for vuln in critical_vulns:
            actions.append(f"- Patch {vuln.get('type')} on {vuln.get('target')}")

        return '\n'.join(actions) if actions else "No critical vulnerabilities requiring immediate action"

    def _generate_compliance_assessment(self) -> str:
        """Évalue l'impact sur la conformité"""

        assessment = "Based on the vulnerabilities discovered:\n"

        critical = self._count_by_severity('CRITICAL')
        high = self._count_by_severity('HIGH')

        if critical > 0:
            assessment += "- Non-compliant with PCI-DSS, ISO 27001, GDPR\n"
            assessment += "- Immediate remediation required for compliance"
        elif high > 0:
            assessment += "- Partial compliance issues detected\n"
            assessment += "- Remediation recommended within 30 days"
        else:
            assessment += "- No major compliance issues identified\n"
            assessment += "- Regular monitoring recommended"

        return assessment

    def _generate_html_summary(self) -> str:
        """Génère le résumé HTML"""

        summary = self._generate_summary()

        return f'''
<p><strong>Total Vulnerabilities:</strong> {summary['total_vulnerabilities']}</p>
<p><strong>Risk Distribution:</strong></p>
<ul>
    <li class="critical">Critical: {summary['critical_count']}</li>
```

```
    <li class="high">High: {summary['high_count']}</li>
    <li class="medium">Medium: {summary['medium_count']}</li>
    <li class="low">Low: {summary['low_count']}</li>
</ul>
<p><strong>Exploitations:</strong> {summary['exploitations_successful']}/{summary['
    exploitations_attempted']} successful</p>
        '''

    def _generate_html_vulnerabilities(self) -> str:
        """Génère le tableau HTML des vulnérabilités"""

        vulns = self.results.get('web_vulns', []) + self.results.get('system_vulns',
     [])

        html = '<table><thead><tr><th>Severity</th><th>Type</th><th>Target</th><th>
    Description</th></tr></thead><tbody>'

        for vuln in vulns:
            severity_class = vuln.get('severity', 'LOW').lower()
            html += f'''
<tr>
    <td class="{severity_class}">{vuln.get('severity', 'UNKNOWN')}</td>
    <td>{vuln.get('type', 'Unknown')}</td>
    <td>{vuln.get('target', 'N/A')}</td>
    <td>{vuln.get('description', 'No description')[:100]}...</td>
</tr>
            '''

        html += '</tbody></table>'

        return html

    def _generate_html_exploitations(self) -> str:
        """Génère le HTML des exploitations"""

        exploits = self.results.get('exploitations', [])

        html = '<table><thead><tr><th>Type</th><th>Target</th><th>Status</th><th>PoC
    </th></tr></thead><tbody>'

        for exploit in exploits:
            status = " Success" if exploit.get('success') else " Failed"
            html += f'''
<tr>
    <td>{exploit.get('type', 'Unknown')}</td>
    <td>{exploit.get('target', 'N/A')}</td>
    <td>{status}</td>
    <td>{"Available" if exploit.get('proof_of_concept') else "N/A"}</td>
</tr>
            '''

        html += '</tbody></table>'

        return html

    def _generate_html_recommendations(self) -> str:
        """Génère les recommandations HTML"""

        recommendations = [
            "Implement input validation and output encoding",
            "Apply security patches to all systems",
            "Enable Web Application Firewall (WAF)",
            "Conduct regular security assessments",
```

```
            "Implement security awareness training",
            "Enable security logging and monitoring"
        ]

        html = '<ul>'

        for rec in recommendations:
            html += f'<li>{rec}</li>'

        html += '</ul>'

        return html
```

1.8 Tests unitaires et validation

1.8.1 Framework de tests

Le projet doit inclure une suite de tests complète :

```
#!/usr/bin/env python3
"""
Suite de tests unitaires pour le framework
"""
import unittest
from modules.reconnaissance import osint
from modules.network import scanner
from modules.web import vulnerability_scanner

class TestReconnaissanceModule(unittest.TestCase):
    """Tests pour le module de reconnaissance"""

    def test_dns_enumeration(self):
        """Test de l'énumération DNS"""
        pass

    def test_subdomain_discovery(self):
        """Test de la découverte de sous-domaines"""
        # Test avec un domaine connu
        pass

    def test_osint_data_collection(self):
        """Test de la collecte OSINT"""
        pass

class TestNetworkModule(unittest.TestCase):
    """Tests pour le module réseau"""

    def test_port_scanning(self):
        """Test du scan de ports"""
        pass

    def test_service_detection(self):
        """Test de la détection de services"""
        pass

    def test_os_fingerprinting(self):
        """Test du fingerprinting OS"""
        pass

class TestWebModule(unittest.TestCase):
    """Tests pour le module web"""

    def test_web_crawling(self):
```

```
        """Test du crawling web"""
        pass

    def test_sql_injection_detection(self):
        """Test de détection SQL injection"""
        pass

    def test_xss_detection(self):
        """Test de détection XSS"""
        pass

class TestIntegration(unittest.TestCase):
    """Tests d'intégration"""

    def test_full_pipeline(self):
        """Test du pipeline complet"""
        pass

    def test_report_generation(self):
        """Test de génération de rapports"""
        pass

if __name__ == '__main__':
    unittest.main()
```

1.9 Documentation technique

1.9.1 Structure de la documentation

Le projet doit inclure une documentation complète :

- **README.md** : Vue d'ensemble, installation, quick start
- **INSTALLATION.md** : Guide d'installation détaillé
- **USER_GUIDE.md** : Guide utilisateur avec exemples
- **API_REFERENCE.md** : Documentation de l'API
- **CONTRIBUTING.md** : Guide de contribution
- **CHANGELOG.md** : Historique des versions

1.9.2 Exemple de README.md

````
# Penetration Testing Framework

Framework modulaire de test d'intrusion développé en Python.

## Fonctionnalités

- **Reconnaissance** : OSINT, DNS, sous-domaines
- **Scan réseau** : Ports, services, OS
- **Scan web** : Crawling, détection de vulnérabilités
- **Exploitation** : Automatisation web et système
- **Reporting** : JSON, HTML, PDF

## Installation
```bash
# Cloner le dépôt
git clone https://github.com/username/pentest-framework.git
````
````

```
cd pentest-framework

# Installer les dépendances
pip install -r requirements.txt

# Configuration
cp config.example.json config.json

# Scan complet
python main.py scan --target example.com --full

# Reconnaissance uniquement
python main.py recon --target example.com

# Génération de rapport
python main.py report --session-id <session_id>

Architecture

framework/
|---- core/           # Modules core
|---- modules/        # Modules fonctionnels
|---- utils/          # Utilitaires
`---- reporting/     # Génération de rapports

Documentation
Voir le dossier docs/ pour la documentation complète.

License
MIT License - Voir LICENSE pour détails

Avertissement
USAGE ÉDUCATIF UNIQUEMENT
Ce framework est destiné uniquement à des fins éducatives
et de tests autorisés. Tout usage malveillant est interdit.
```

2 Critères d'évaluation

2.1 Grille d'évaluation détaillée

2.2 Critères de bonus

Des points bonus peuvent être attribués pour :

- **Démonstration vidéo** (+2 pts) : Vidéo de démonstration professionnelle
- **Interface graphique** (+3 pts) : GUI en plus du CLI
- **Tests avancés** (+2 pts) : Couverture de tests > 80
- **CI/CD** (+1 pt) : Pipeline d'intégration continue
- **Containerisation** (+1 pt) : Docker/Docker-compose
- **Documentation avancée** (+1 pt) : Wiki ou documentation web

Critère	Points	Détails
Qualité technique du code (40%)		
Architecture	8 pts	Modularité, séparation des responsabilités
Qualité du code	8 pts	Lisibilité, conventions PEP8, docstrings
Gestion d'erreurs	8 pts	Try-catch, logging, récupération
Performance	8 pts	Multi-threading, optimisations
Sécurité	8 pts	Validation entrées, sanitization, logging
Fonctionnalités implémentées (25%)		
Reconnaissance	5 pts	OSINT, DNS, sous-domaines complets
Scan réseau	5 pts	Ports, services, OS fonctionnels
Scan web	5 pts	Crawling et détection vulnérabilités
Exploitation	5 pts	Web et système implémentés
Reporting	5 pts	Multi-formats, complets et clairs
Documentation et rapport (20%)		
Rapport technique	10 pts	Structure respectée, complet (15-20 pages)
Documentation code	5 pts	README, guides, API reference
Commentaires	5 pts	Code bien commenté, docstrings
Innovation et originalité (15%)		
Fonctionnalités avancées	8 pts	Features au-delà des exigences
Interface utilisateur	4 pts	CLI&IHM intuitive, ergonomique
Créativité	3 pts	Solutions originales, optimisations
TOTAL		
Total général	**100 pts**	**Ramené sur 20**

TABLE 3 – Grille d'évaluation détaillée du projet

2.3 Pénalités

Des pénalités seront appliquées pour :

— **Retard de livraison** : -2 points par jour de retard
— **Code non fonctionnel** : -10 points si le framework ne s'exécute pas
— **Plagiat détecté** : 0/20 + sanctions disciplinaires
— **Documentation manquante** : -5 points
— **Tests absents** : -3 points

3 Modalités de soutenance

3.1 Déroulement de la soutenance

3.1.1 Format de présentation

— **Durée totale :** 30 minutes
  — Présentation : 20 minutes
  — Questions : 10 minutes
— **Support :** Présentation PowerPoint/PDF obligatoire
— **Démonstration :** Live demo du framework (5-7 minutes)