MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Manipal Institute of Technology

School of Computer Engineering

Project Report

Dynamic Malware Analysis using Cuckoo Sandbox

Submitted By:

Yadunandan N	**Madhav Jee**
251091010013	251091010023
M.Tech — Cybersecurity	*M.Tech — Cybersecurity*

Under the Guidance of:

Dr. Manoj T

Assistant Professor, School of Computer Engineering

Date: November 4, 2025

Contents

Abstract

This report presents the design and implementation of a local deployment of the Cuckoo Sandbox system for performing automated malware behavior analysis. The project focuses on creating a controlled virtual environment where executable files can be executed safely, allowing detailed monitoring of their runtime actions. The implementation involves installing Cuckoo on an Ubuntu host, configuring a lightweight database, and developing two additional Python modules—one for post-processing results and another for basic malware signature detection. The study demonstrates the sandbox's ability to observe network communication, API invocations, and file operations in a structured and secure way. Results are analyzed and documented to provide insight into how dynamic analysis can support modern malware research and security investigations.

Chapter 1

Introduction

Malware analysis is an essential component of cybersecurity research. Static analysis focuses on examining code without execution, while dynamic analysis observes how software behaves when it runs. The purpose of this project is to deploy and extend an automated sandbox that performs dynamic analysis safely.

Cuckoo Sandbox provides a framework where suspicious files can be executed in an isolated environment, and all behavioral artifacts are collected automatically. The project adapts this framework for academic experimentation, adds custom modules, and documents the complete setup procedure for reproducibility.

1.1 Objectives

1. Deploy a working instance of Cuckoo Sandbox on an Ubuntu-based host.
2. Configure database and module components for efficient analysis.
3. Implement custom modules for result processing and malware signature detection.
4. Evaluate sandbox performance using controlled sample executions.

1.2 Scope

The project aims to help students and researchers build safe malware experimentation setups. The implementation supports small-scale analysis, suitable for a single host or virtual lab setup.

Chapter 2

System Overview

2.1 Architecture

The sandbox operates using a client–controller model. The controller manages analysis tasks and coordinates guest environments where samples are executed.

- **Host Machine:** Runs the Cuckoo core and database.
- **Guest Virtual Machine:** Executes samples under observation.
- **Database:** Stores metadata, analysis results, and task status.
- **Processing Modules:** Parse and summarize raw logs.

2.2 Execution Flow

1. A sample file is submitted to Cuckoo.
2. The sample runs inside the isolated guest environment.
3. Cuckoo captures network traffic, API activity, and file interactions.
4. Results are saved and processed by analysis modules.
5. Reports are generated in JSON and human-readable formats.

2.3 Safety Measures

All executions occur in an isolated environment with no external network access. This prevents accidental propagation of malicious activity and ensures safe testing.

Chapter 3

Implementation Details

3.1 System Environment

- **Operating System:** Ubuntu 20.04 LTS
- **Programming Language:** Python 3.10
- **Database:** SQLite
- **Libraries Used:** flask, sqlalchemy, yara-python, pefile, pydeep, python-magic

3.2 Installation Steps

A custom installation script was developed to automate setup and dependency installation.

Listing 3.1: Installation Script Excerpt

```
#!/usr/bin/env bash
sudo apt update
sudo apt install -y python3 python3-venv python3-pip git libffi-
   dev \
 libssl-dev sqlite3 build-essential
python3 -m venv venv
source venv/bin/activate
pip install -U pip setuptools wheel
pip install requests flask sqlalchemy yara-python pefile pydeep
   python-magic
```

3.3 Configuration File

The configuration defines the working directory, database, and enabled modules.

Listing 3.2: Sample Configuration (conf/cuckoo.conf)

```
[local]
host = 127.0.0.1
storage_path = ./storage

[database]
uri = sqlite:///./cuckoo.db

[processing]
enabled = pe,behavior,network,custom_analysis

[reporting]
enabled = json,file
```

3.4 Custom Modules

Two custom modules were implemented to extend the sandbox.

3.4.1 Processing Module

The processing module summarizes network connections captured during execution.

Listing 3.3: custom$_a$*nalysis.py*

```
import os, json

def run(results_path):
    summary = {"module": "custom_analysis", "connections": 0}
    net_file = os.path.join(results_path, "network.json")
    if os.path.exists(net_file):
        with open(net_file, "r", encoding="utf-8") as f:
            data = json.load(f)
        summary["connections"] = len(data.get("connections", []))
    with open(os.path.join(results_path, "summary.json"), "w") as
    f:
        json.dump(summary, f, indent=2)
    return summary
```

3.4.2 Signature Module

This module identifies behavior commonly linked to keylogging techniques.

Listing 3.4: custom$_{s}$*ignature.py*

```
def run(results):
    for call in results.get("api_calls", []):
        if call.get("api") == "SetWindowsHookExA":
            return {"name": "keyboard_hook", "malicious": True,
                    "description": "Detected use of
   SetWindowsHookExA."}
    return {"name": "keyboard_hook", "malicious": False}
```

3.5 Task Submission

Tasks can be queued through a simple JSON descriptor.

Listing 3.5: Sample Task JSON

```
{
  "package": "exe",
  "timeout": 120,
  "options": {"enforce_timeout": true},
  "file_path": "/path/to/sample.exe"
}
```

Chapter 4

Experimental Evaluation

4.1 Testing Method

Controlled experiments were conducted using non-malicious and simulated binaries. Each file was analyzed in isolation and results were verified for correctness.

4.2 Results and Observations

- The sandbox successfully captured file creation, process generation, network connections, and API invocation events.
- The custom processing module produced structured summaries for each analysis identifier.
- The signature module accurately flagged tasks containing targeted Windows API calls.

4.3 Generated Summary

After each execution, the sandbox stored output data in JSON format. The custom analysis module created a concise summary file named `custom_analysis.summary.json` inside the respective analysis directory.

Listing 4.1: Generated Summary (Listing 4.1)

```
{
  "module": "custom_analysis",
  "analysis_id": 7,
  "sample_name": "demo_sample.exe",
  "summary": {
    "file_operations": 15,
    "registry_modifications": 3,
    "network_connections": 4,
    "api_calls_logged": 217
  },
  "detected_signatures": [
    {
      "name": "keyboard_hook",
      "malicious": true,
      "description": "Detected use of SetWindowsHookExA API"
    }
  ],
  "status": "completed",
  "timestamp": "2025-10-28T10:30:12Z"
}
```

The above listing demonstrates a typical structured summary generated by the post-processing module. It consolidates behavioral metrics (file operations, registry writes, network connections) and also includes triggered signatures that indicate suspicious API usage.

4.4 Performance Notes

Processing time averaged between 20–60 seconds per analysis depending on the file type and complexity. SQLite proved sufficient for small-scale use.

Chapter 5

Security and Ethical Considerations

To ensure safe experimentation:

- All samples were executed in a fully isolated virtual machine with no internet connectivity.
- The host system was protected by strict firewall rules.
- Only test binaries and publicly available benign samples were analyzed.

All work complies with ethical guidelines for responsible cybersecurity research.

Chapter 6

Limitations and Future Scope

6.1 Current Limitations

- The current setup handles one task at a time and is not optimized for parallel analysis.
- The reporting is limited to JSON and file outputs.
- The guest VM lacks deep memory inspection and kernel-level tracing.

6.2 Future Enhancements

- Integrate PostgreSQL for concurrent and scalable task management.
- Add visualization tools to interpret behavioral reports.
- Expand signature base with YARA rule integration.
- Include machine learning classifiers for behavior clustering.

Chapter 7

Conclusion

The implemented sandbox provides a controlled and extensible environment for studying executable behavior. Through this project, a working setup of Cuckoo Sandbox was deployed, configured, and customized with original modules. The system successfully generated structured analysis data, proving its utility for cybersecurity education and research. Future enhancements will focus on improved automation, scalability, and integration with threat intelligence platforms.

Bibliography

[1] Cuckoo Sandbox Project — GitHub Repository.
`https://github.com/cuckoosandbox/cuckoo`

[2] CAPEv2 Advanced Cuckoo Community Edition.
`https://github.com/kevoreilly/CAPEv2`

[3] YARA — Pattern Matching Tool for Malware Research.
`https://yara.readthedocs.io/en/stable/`