FILELESS MALWARE DETECTION & IT' S MITIGATION

Detection and Mitigation of Fileless Malware in Operational Technology Environments

SUPERVISOR:
DR.AHMED SIKANDAR

TABLE OF CONTENTS

- Introduction
- Background of the Problem
- Project Motivation
- Problem Statement
- Objectives
- Literature Review
- Existing Methods/Technologies
- Comparative Analysis
- Proposed Solution
- Methodology & Architecture
- Implementation
- System Design
- Screenshots & Results
- References
- Q&A

INTRODUCTION

- OT systems manage critical infrastructure like power grids and manufacturing plants.
- Fileless malware poses unique challenges as it operates in memory.
- Detection in OT is difficult due to legacy hardware and limited system resources.

BACKGROUND OF THE PROBLEM

- Fileless malware uses legitimate tools like PowerShell, WMI.
- Hard to detect with traditional antivirus software.
- Attacks like Stuxnet, Industroyer, and Triton highlight the risk to OT.

PROJECT MOTIVATION

- Legacy OT systems lack modern security features.
- High-risk environments: continuous uptime and real-time responsiveness.
- Lack of existing lightweight solutions for fileless threats in OT.

PROBLEM STATEMENT

"Fileless malware evades traditional file-based detection systems, posing serious threats to OT systems which are not equipped with sufficient tools for memory-based analysis or anomaly detection."

OBJECTIVES

- **Develop a framework combining:**

  Process monitoring (psutil)

  Memory forensics (wmic.exe + base64 scripts + Invoke-Expression)

  Network inspection (Suricata)

  Alert system (smtplib, plyer)

  Web interface (Flask)

- **Evaluate performance in simulated OT environment**

LITERATURE REVIEW

- **Baldin (2019):** memory forensics & behavioral analytics
- **Kumar & Vardhan (2023):** lightweight detection for legacy OT
- **Langner (2013), Cherepanov (2017), Dragos (2017):** case studies on Stuxnet, Industroyer, and Triton

EXISTING METHODS/TECHNOLOGIES

Methodology	Tools	Limitations
Signature-Based	Antivirus	Cannot detect memory-based attacks
Behavioral	EDR	High resource consumption
Network-Based	Suricata	Requires custom rules
Memory Forensics	Volatility	Not optimized for real-time OT use

COMPARATIVE ANALYSIS

- **EDR:** High detection, not suitable for OT
- **AV:** Lightweight but ineffective for memory attacks
- **Proposed System:** Accurate, lightweight, real-time

PROPOSED SOLUTION

- Modular framework designed for OT
- Combines process, memory, and network-level detection
- Web-based admin dashboard for live control
- Alerts via desktop and email notifications

METHODOLOGY OVERVIEW

Memory Forensics (Python + Flask)

psutil scans active processes (PowerShell, WMI).

Flags encoded or obfuscated commands (IEX, -EncodedCommand).

Flask dashboard displays results (PID, Name, Justification).

Network Monitoring (Suricata IDS)

Custom rules detect:

PowerShell download cradles

LOLBin usage (e.g., rundll32, mshta)

Suspicious DNS (e.g., pastebin.com)

Alerts are logged and shown in dashboard.

Workflow

User clicks "Analyze Memory"

Backend scans processes

Suricata monitors traffic

Alerts & results shown in dashboard

Admin can block IPs or kill processes

ALGORITHM & FLOWCHART

Flow:

1. Collect process data
2. Scan for IOCs in memory
3. Apply Suricata rules
4. Alert admin on detection
5. Visualize on Flask dashboard

ARCHITECTURE DIAGRAM / WORKFLOW

- Device → Monitoring Module (CPU/memory)
- Memory Forensics → wmic.exe + base64 scripts + Invoke-Expression
- Network → Suricata IDS
- Alerts → smtplib/plyer
- UI → Flask Web Interface

IMPLEMENTATION

- **Windows 10 virtual OT setup**
- **Tools used:**
  - Python 3.9
  - psutil, Flask, jsonify
  - Suricata IDS
  - smtplib, plyer, pywin32
  - Bootstrap (front-end)

TOOLS & TECHNOLOGIES

- **psutil:** Live process monitoring
- **Flask:** Web dashboard for analysis
- **Suricata:** IDS with custom rules for
- Encoded PowerShell
- LOLBin (e.g., mshta, rundll32)
- WMI, registry abuse, suspicious DNS
- **smtplib:** Email notifications
- **plyer:** Desktop pop-up alerts

PROGRAMMING LANGUAGES, FRAMEWORKS

- Python 3.9
- Flask 2.0.1
- Bootstrap 5.1
- HTML, CSS, JavaScript

LIBRARIES/APIs

- psutil
- smtplib
- plyer
- pywin32
- Flask

SYSTEM DESIGN

- **Data Flow Diagram:**

  Input → Monitor/Analyzer → Flask → Admin Response

- **Use Case Diagram:**

  Admin can: View alerts, Kill process, Block IP, View analysis

SYSTEM COMPONENTS AND GUI

- Process Monitor
- Memory Analyzer
- Suricata Integration
- Alert System
- Dashboard (Home, Processes, Memory, Network)

COMPARATIVE RESULTS & GRAPHS

Module	Accuracy	CPU	RAM
Memory Forensics	90%	2-4%	~28MB
Suricata	95%	3-5%	~40MB
False Positives ~3–4.5% | Low system overhead			

REFERENCES

- Baldin, N. (2019). *Memory-based threats in OT.*
- Kumar, A., & Vardhan, R. (2023). *Security concerns in OT networks.*
- Langner, R. (2013). *Stuxnet Analysis.*
- Dragos (2017). *Triton ICS Analysis.*
- APA Format used. Additional in thesis bibliography.

THANK YOU

"Thank you for your attention. We look forward to your feedback."

QUESTION & ANSWER

"Any questions or suggestions?

We’d love to hear from you."