Enhanced Security Monitoring & Evidence Collection System

Diamaris González Rodríguez
Computer Science
Advisor: Lisabel Rodríguez Espinosa, Ph.D.
Polytechnic University of Puerto Rico
Graduate Project EXPO, April 2025

Abstract *— This report details the deployment of an Enhanced Security Monitoring & Evidence Collection System to detect unauthorized login attempts and support forensic investigations. The system integrates key functionalities such as monitoring Windows event logs to identify suspicious activities, capturing memory dumps to preserve volatile data, and extracting file hashes to analyze potentially malicious files. The development process incorporates specialized tools like ProcDump for system process analysis, Volatility for memory forensics, and WinPmem for memory acquisition, ensuring thorough data collection and analysis. These tools contribute to a systematic and effective threat analysis workflow. By automating security monitoring and evidence collection, this system addresses the complexities of modern IT environments and enhances the organization's ability to respond to cyber threats. Its deployment significantly strengthens defenses, streamlines forensic investigations, and supports regulatory compliance, promoting a secure and efficient IT infrastructure.*

Key Terms *— Cybersecurity, Digital Forensics, Intrusion Detection, Security Logging*

INTRODUCTION

As individuals and businesses increasingly depend on digital infrastructures, the sophistication and nature of cyber threats are also growing. Insider threats, unauthorized access, APTs, and stolen credentials are significant threats that traditional security measures find hard to combat. Today, cybercriminals also employ stealthy attack techniques, i.e., fileless malware and in-memory exploits, to escape conventional security measures, so event-driven security monitoring and forensic analysis are essential to detect threats ahead of time [1].

Background

Among modern forensic techniques, memory forensics is essential for uncovering hidden cyberattacks that do not leave many traces on disk storage. Unlike traditional file-based malware, advanced cyber threats operate entirely in volatile memory, which makes them difficult to identify by conventional security tools. Through real-time memory analysis, security experts can identify malicious activity, unauthorized access attempts, and suspicious behavior that would otherwise go undetected [2].

Purpose of the Project

The Enhanced Security Monitoring & Evidence Collection System proactively detects threats and preserves evidence for forensic analysis. It monitors Windows Event Logs for suspicious login and uses memory dumps to capture volatile data (e.g., processes, modules, and network activity).

Using Volatility and ProcDump, the system analyzes memory for hidden threats like rootkits and code injections [3][4]. Cryptographic hashes ensured data integrity, making the evidence reliable for audits and legal proceedings.

Goal

The following are the goals, outlining the key priorities and actions to be achieved in this project:

- Detect unauthorized login attempts automatically.
- Collect and analyze memory using Volatility.
- Extract critical system files with integrity.
- Generate real-time forensic reports.

Features and Justification

The following highlights this system's capabilities in enhancing forensic investigations and securing critical data:

- **Unauthorized Login Detection**: Detects failed logins and anomalies via Windows Event Logs for real-time alerts against brute-force or insider threats [5].
- **Memory Dump Collection & Analysis**: Captures volatile data missed by disk forensics. Volatility reveals in-memory threats, hidden processes, and injected code.
- **File Extraction & Integrity**: Extracts vital files (e.g., registry, logs) and uses cryptographic hashes to maintain evidence integrity.
- **Real-Time Report Generation**: Produces JSON-formatted reports compatible with SIEM tools and aligned with ISO 27001 and NIST standards.

Justification for the Project

This approach enables early threat detection while ensuring the preservation of critical evidence. It maintains forensic-grade integrity, providing structured audit logs to support thorough investigations. Furthermore, it integrates industry-standard tools, such as Volatility and ProcDump, to enhance the depth and precision of forensic analysis.

This system bridges real-time monitoring and forensics, empowering security teams with the tools needed to respond to and investigate incidents efficiently.

SYSTEM ARCHITECTURE

Defines the foundational structure and view of the system.

Components

The following collectively ensures the streamlining of this system, which is designed to address key security challenges while supporting forensic analysis.

- Event Log Monitoring: Detects failed login attempts in real-time.
- Evidence Collection: Captures snapshots of system dumps and gathers hashes of files.
- Forensic Analysis: Uses Volatility to scan memory dumps for security threats.
- Security Report Generation: Generates formatted incident reports of detected events.

Technologies and Tools

The following for the backbone of this system, enabling its functionality and seamless integration.

- Python 3.10+ (Core language)
- WinPmem – Performs memory dumps for forensic analysis.
- ProcDump – Takes memory dumps of critical system processes.
- Flask – Provides web-based dashboard for monitoring.
- smtplib – It sends security alert notifications via email [6].

IMPLEMENTATION

The process begins with preparing the necessary environment, ensuring the smooth functionality of the system.

Setting Up the Environment

The following steps describe the configuration of the environment and installation of the essential dependencies required for the system.

1. Install Python 3.10+ and Visual Studio Code.
2. Create a virtual environment and install the required dependencies:
    - python -m venv venv
    - venv\Scripts\activate
    - pip install pywin32
    - volatility3 flask smtplib

Log Monitoring (Detecting Failed Logins)

The script continuously monitors Windows Event ID 4625. When a failed login occurs, the system logs the details and triggers an alert [7].

Collecting Memory Dumps (WinPmem)

A memory dump is captured when a suspicious event is detected. The collected memory dump is stored securely for analysis.

Obtaining Hashes from the SAM File

The SAM file for Windows is obtained using *diskshadow.exe*. File hashes are computed to detect tampering and unauthorized modifications.

Analyzing Memory Dumps with Volatility

Volatility is a vital tool in digital forensics, enabling the analysis of memory dumps to uncover hidden threats that disk-based tools may miss. During incident response, it helps identify running processes, injected code, and critical network anomalies.

Key Analysis Steps

Key analysis steps are essential for effective forensic investigations, enabling precise threat detection and memory-based attack reconstruction.

1. **Running Processes:** Use *windows.pslist* to list processes active at the time of the dump. Look for suspicious processes like powershell.exe, often used in remote execution attacks [8].
   *Command**:*** vol -f memory.dmp windows.pslist
2. **Network Anomalies**: *windows.netscan* detects network connections during the dump. Unusual remote IPs or ports (e.g., powershell.exe connecting externally) may indicate C2 activity.
   *Command***:** vol -f memory.dmp windows.pslist
3. **Rootkit & Malware Detection:** Volatility uncovers malware and rootkits that alter system memory but evade file-based scanners. This is key for detecting advanced threats.
4. **Process Injection:** Volatility identifies injected code in legitimate processes (e.g., explorer.exe) by comparing memory patterns, flagging anomalies like unauthorized Dynamic Link Libraries.
5. **Event Correlation:** Memory artifacts help reconstruct attack timelines, showing how attackers gained access and what data was targeted.

Step-by-Step Volatility Investigation

Volatility analyzes memory dumps to uncover malicious activity by identifying running processes and network behavior.

1. **Process Analysis** (windows.pslist): The windows.pslist plugin retrieves details about processes that were active at the time of the memory dump. It provides key information, such as:
   - PID
   - Start Time
   - Process Name

   Command: vol -f memory.dmp windows.pslist
2. **Network Activity Analysis (windows.netscan):** Identifies network connections to spot suspicious activity like unauthorized data transfer or external IP communication. Key details:
   - Protocol
   - Local/Remote Address
   - PID & Process

   Command: vol -f memory.dmp windows.netscan

This analysis helps detect unauthorized or malicious network activity, such as connections to suspicious external IP addresses or unexpected ports.

Code Structure

The project consists of multiple Python scripts:

- **monitor_logs.py** – Monitors Windows Event Logs (see Figure 1). This monitor_logs.py script snippet utilizes the win32evtlog library to access and read windows security event logs, identifying failed login attempts (Event ID 4625) and generating real-time alerts with the timestamp of each failed attempt. The script (Figure 2) shows the implementation written in VS Code. The Script constantly monitors windows security event logs for unsuccessful login.
- **memory_dumper.py** – Handles memory dump collection (Figure 3). The memory_dumper.py script snippet which handles the memory dump collection.

```
import win32evtlog
hand = win32evtlog.OpenEventLog(None, "Security")
events = win32evtlog.ReadEventLog(hand, win32evtlog.EVENTLOG_BACKWARDS_READ, 0)
for the event in events:
   if event.EventID == 4625:
      print(f"[ALERT] Failed login at {event.TimeGenerated}")
```

Figure 1

Monitor_logs.py Script Snippet

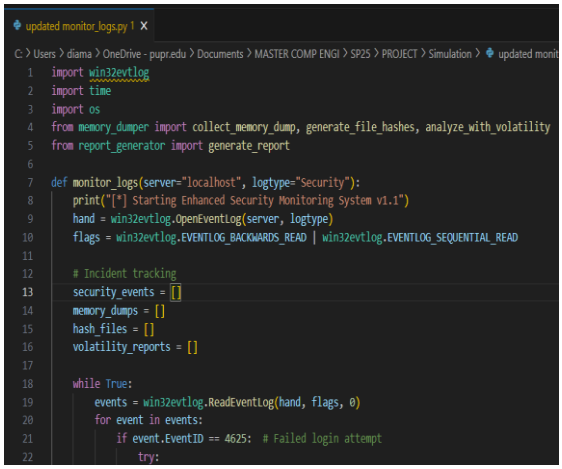

Figure 2

Monitors Windows Security Event Logs for Unsuccessful Login

```
import subprocess
subprocess.run(["procdump.exe", "-ma", "lsass.exe", "memory.dmp"])
```

Figure 3

Memory_dumper.py Script Snippet Which Handles the Memory Dump Collection

- **report_generator.py** – Generates structured security reports (see Figure 4). The report_generator.py script snippet that generates structure reports.

```
{   "timestamp": "2025-02-09 17:54:02",
    "failed_login_source": "C:\\Windows\\System32\\svchost.exe",
    "memory_dump": "evidence\\memory.dmp",
    "hashes": {
        "cmd.exe": "abc123...",
        "lsass.exe": "xyz456..."
    }3.5
}
```

Figure 4

Report_generator.py Script Snippet that Generates Structure Reports

Execution Steps

The following steps outline the execution process for testing and utilizing the system's forensic and security functionalities:

1. Run the monitoring script: python monitor_logs.py.
2. Trigger a failed login attempt (Test Scenario).
3. Observe real-time alerts & evidence collection.
4. Analyze a memory dump with Volatility.
5. Check the generated security report.

Data Storage Implementation

Efficient data handling is vital in digital forensics to maintain integrity and accessibility.

Storage Directory Hierarchy

Files are saved with timestamp-based names under strict access control (see Figure 5). The storage directory hierarchy where the files are saved with the corresponding timestamp.

```
/evidence
  memory_<timestamp>.dmp
  hashes_<timestamp>.txt
  volatility_<timestamp>.txt
/reports
  security_report_<timestamp>.json
```

Figure 5

Storage Directory Hierarchy

This system uses strict access controls to safeguard evidence and preserve its integrity.

Key Script Functions

Three scripts handle the collection, analysis, and reporting of forensic evidence, ensuring organized and secure storage of information.

- Memory Dump Collection (Memory_dumper.py): This script automates the collection of memory dumps, which is crucial for forensic investigations (Figure 6). Python script for automated memory dump collection (memory_dumper.py).

```python
def collect_memory_dump(output_dir="evidence"):
    """Create full memory dump using procdump"""
    try:
        # Verify Procdump exists
        if not os.path.isfile(PROCDUMP_PATH):
            raise FileNotFoundError(f"Procdump not found at {PROCDUMP_PATH}")

        # Verify Admin rights
        if not is_admin(): ···

        # Create output directory
        os.makedirs(output_dir, exist_ok=True)

        # Generate filename
        dump_file = os.path.join(output_dir, "memory.dmp")

        # Build command
        cmd = [ ···

        # Execute with a longer timeout
        result = subprocess.run(
            cmd,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True,
            check=True,
            timeout=600,  # Increased timeout
            shell=True
        )

        print(f"[SUCCESS] Memory dump created: {dump_file}")
        print(f"[PROCDUMP OUTPUT] {result.stdout}")
        return dump_file
```

Figure 6
Python Script for Automated Memory Dump Collection (memory_dumper.py)

Key functions

Procdump validation ensures the tool is available prior to execution, administrator privileges verify necessary access rights for the script, and time-based file naming with directory handling prevents overwriting by creating unique storage paths.

Forensic Report Generation

The *report_generator.py* script (See Figure 7) converts forensic analysis data into a JSON format, creating detailed security reports for auditing.

```python
def generate_report(events, memory_dumps, hash_files, volatility_reports, output_dir="reports"):
    """Generate comprehensive JSON report"""
    try:
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)

        report = {
            "metadata": {
                "generated_at": datetime.now().isoformat(),
                "system": os.environ.get("COMPUTERNAME", "UNKNOWN"),
                "analyst": "Security Team",
                "report_version": "1.1"
            },
            "security_events": events,
            "evidence": {
                "memory_dumps": [os.path.abspath(m) for m in memory_dumps],
                "hash_files": [os.path.abspath(h) for h in hash_files],
                "volatility_reports": [os.path.abspath(v) for v in volatility_reports]
            }
        }

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S%f")  # Adds microseconds for uniqueness

        #timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        report_file = os.path.join(output_dir, f"security_report_{timestamp}.json")

        with open(report_file, 'w') as f:
            json.dump(report, f, indent=2, ensure_ascii=False)

        print(f"[SUCCESS] Security report generated: {report_file}")
        return report_file
    except Exception as e:
        print(f"[ERROR] Report generation failed: {str(e)}")
        return None
```

Figure 7
JSON-Based Security Report Generator (report_generator.py)

Key Features

The system features a comprehensive report structure that organizes metadata, security events, and forensic evidence in JSON format for clarity and usability. It incorporates automated directory handling, ensuring the seamless creation of the "/reports" directory if it is missing. To maintain data integrity, unique timestamp-based naming is utilized, preventing report overwriting with microsecond precision. Additionally, robust error handling and logging mechanisms are in place to capture errors, facilitating effective debugging and validation.

System Benefits And Forensic Analysis

Memory dumps are crucial forensic tools, as they capture volatile system information that is lost during a reboot.

Incident Response Advantages

These advantages underscore the importance of memory dumps in preserving critical evidence for effective incident response.

- **Pre-Reboot Evidence Collection:** Memory dumps store crucial volatile data before a system reboot, such as running processes, open network connections, and plain-text credentials (see Figure 8).

```
if event.EventID == 4625:  # Triggers forensic collection
collect_memory_dump()
```

Figure 8
Memory Dump Collection

This approach aligns with the best practices of incident response**.**

- **Attack Reconstruction:** Analysts can use memory dumps to correlate logs with memory data, reconstructing an attacker's activities (see Figure 9).

```
# Extracting suspicious processes from memory
commands = [
    f"vol -f {memory_dump} windows.pslist",
    f"vol -f {memory_dump} windows.malfind"
]
```

Figure 9
Analyzing Memory Dumps

Legal & Compliance Value

Forensic investigations' legal and compliance significance is underscored by their ability to produce court-admissible evidence with verified integrity.

- **Court-Admissible Evidence:** Forensic investigations rely on cryptographic hashing (MD5, SHA-1, SHA-256), as seen in Figure 10, to ensure the integrity of memory dumps for legal admissibility, as outlined in NIST SP 800-86.

```
hashes = {
    'md5': hashlib.md5(),
    'sha1': hashlib.sha1(),
    'sha256': hashlib.sha256()
}
```

Figure 10
Initializing Cryptography Hashes for Verifying Data Integrity

The system adheres to NIST SP 800-86 guidelines [9] by implementing best practices for timely evidence preservation, data integrity verification, and structured reporting, ensuring compliance with regulatory standards for industries such as healthcare (HIPAA) and finance (ISO 27001).

Threat Detection

The system employs advanced techniques to detect sophisticated threats and mitigate vulnerabilities in memory-based attacks.

- Fileless Malware Detection: Memory analysis detects fileless malware that exists only in memory without disk traces (see Figure 11).

```
"vol -f memory.dmp windows.malfind"
```

Figure 11
Analyzes Memory Dump with Volatility to Detect Injected Malicious Code In Running Processes

This reveals injected code segments, making it easier to detect advanced malware strains such as Cobalt Strike and Meterpreter**.**

- Credential Theft Prevention (LSASS Dump Analysis): Tools like Mimikatz [10] target system memory to steal credentials from LSASS (see Figure 12).

```
cmd = [PROCDUMP_PATH, "-ma",
"lsass.exe", dump_file]
```

Figure 12
Captures an LSASS Memory dump with Procdump for Credential theft Analysis

Operational Efficiency Automated Triage & Threat Intelligence

The system automates memory dump collection and forensic analysis upon detecting security events, like failed logins, reducing investigation time and enabling real-time threat detection, as seen in Figure 13.

```
if event.EventID == 4625:  # Failed login
attempt
    collect_memory_dump()
    analyze_with_volatility()
```

Figure 13
Automation of Dump Collection and Analysis

This reduces investigation time and enables real-time threat detection, allowing security analysts to quickly respond to active threats [11].

Memory Dump Forensic Value

A three-step evidence preservation process ensures the integrity of memory dump data:

1. Data Acquisition**:** Memory dumps are captured when security events (e.g., failed logins) are triggered.
2. Integrity Verification**:** A triple-hashing mechanism (MD5, SHA-1, SHA-256) ensures data integrity and compliance with forensic standards (NIST SP 800-86).
3. Automated Analysis: The captured dumps undergo automated analysis to extract forensic intelligence (see Figure 13).

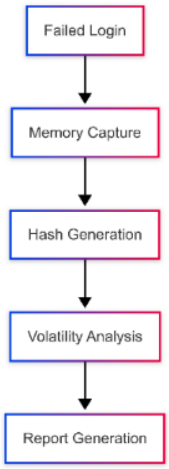

Figure 13
Automated Forensic Analysis from Memory Capture to Report Generation

Key Forensic Artifacts

Memory forensics helps reconstruct cyberattacks by identifying artifacts like credential theft, process injection, and network persistence (see Table 1).

Table 1
Key Forensic Artifacts aiding in Attack Reconstruction and Malware Detection

Artifact Type	Detection Method	Investigation Value
Credential Theft	windows.lsass +hashdump	Detects Mimikatz, Kerberoasting attacks.
Process Injection	windows.malfind	Identifies DLL hijacking and process hollowing.
Network Persistence	windows.netscan	Reveals attacker-controlled servers (C2).
Timestamp Evidence	windows.filescan + logs	Establishes forensic timeline.

Volatility Analysis Methodology

Automated Volatility 3 commands, as seen in Figure 14, extract forensic data from memory dumps to identify threats like credential dumping and malware execution:

```
# Automated analysis commands
commands = [
    f"vol -f {memory_dump} windows.pslist",
    f"vol -f {memory_dump} windows.netscan",
    f"vol -f {memory_dump} windows.malfind"
]
```

Figure 14
Volatility 3 Commands

In Figure 15, we can see the Sample Output Analysis:

PID	Process	Start Time	Command Line
672	lsass.exe	2025-03-01 08:05:32	C:\Windows\system32\lsass.exe
404	powershell.exe	2025-03-01 08:06:11	powershell -nop -w hidden -c IEX(...)

Figure 15
Sample Output Analysis

These anomalies serve as high-confidence indicators of compromise (IoCs) and aid in threat hunting [12].

Result And Analysis

The Enhanced Security Monitoring & Evidence Collection System was successfully tested to assess its potential to identify attempted unauthorized logins, collect forensic evidence, and produce structured reports.

- **Real-Time Detection of Malicious Login Attempts** (see Figure 16):
  - The system successfully detected and logged malicious login attempts in Windows Event Logs.
  - The malicious access attempts were indicated in real-time, giving timely notifications to security teams.
  - The real-time detection reduces response time, enabling faster mitigation of threats.

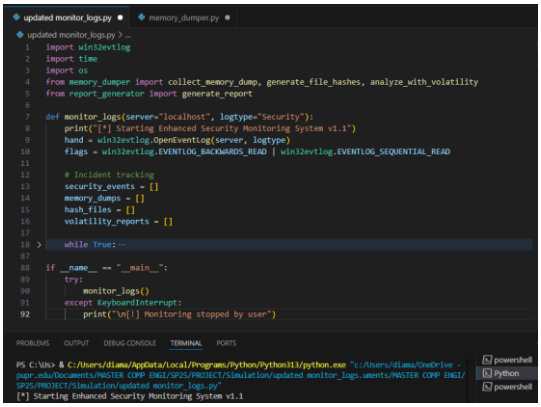

Figure 16
Running the updated_monitor_logs.py script in Visual Studio Code

- **Automated Collection and Analysis of Memory Dumps** (see Figure 17):

- After detecting a failed login, the system triggered forensic data collection, taking memory dumps for detailed analysis.
- This forensic methodology ensured that even advanced threats like fileless malware or credential theft would not go unnoticed.

- **Generation of Structured Security Reports:** Detailed reports listing information regarding security incidents were generated by the system, including, as seen in Figure 18:
  - Source and time of the unauthorized login attempt.
  - File hashes for ensuring data integrity and detecting changes.
  - These reports were stored in JSON format, which is simple to parse, analyze, and incorporate into broader security infrastructures.

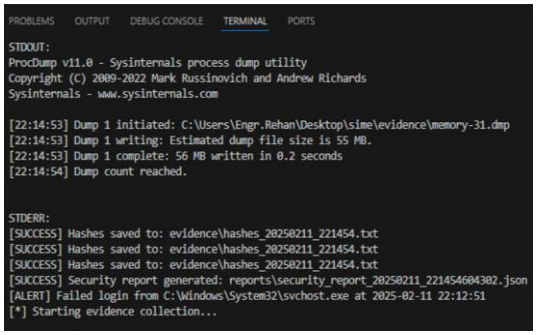

Figure 17
Running the updated_monitor_logs.py script, Showing Real-Time Security Monitoring and Evidence Collection

Figure 18
Example of a JSON-based Security Report with Metadata, Detected Security Events, and Forensic Evidence

In Figure 19, we see the Sample Output:

```
[ALERT] Failed login from C:\Windows\System32\svchost.exe at 2025-02-09 13:51:06
[*] Starting evidence collection...
[SUCCESS] Memory dump collected: evidence\memory.dmp
[SUCCESS] Hashes saved: evidence\hashes_20250209_175402.txt
[SUCCESS] Security report generated: reports\security_report_20250209_175403.json
```

Figure 19
Real-time Security Monitoring Execution and Report Generation

CHALLENGES FACED

A number of major challenges were encountered in the course of implementing the Enhanced Security Monitoring & Evidence Collection System, requiring systematic refinement to achieve optimal functionality and performance.

- ProcDump Execution Failures**.**
- Large Memory Dump Files**.**
- Event Log Parsing Issues**.**

CONCLUSION

The Enhanced Security Monitoring & Evidence Collection System is an advanced solution for detecting unauthorized access and facilitating forensic investigations. The system enhances data acquisition and analysis by integrating tools like ProcDump, Volatility, and WinPmem, streamlining security incident response. Its monitoring capabilities enable rapid threat detection and response, making it essential for IT administrators and forensic experts.

Applicable across various sectors, corporate, educational, government, and healthcare, the system protects sensitive data, provides actionable insights, and supports compliance with legal standards. Its role in forensic investigations aids effective evidence collection and incident response.

As cyber threats evolve, future enhancements like centralized dashboards, real-time alerts, proactive failed login responses, and expanded forensic capabilities will further enhance their value, reinforcing the system's ability to safeguard digital

infrastructures against future challenges. This proactive approach underscores its importance in modern cybersecurity.

REFERENCES

[1] A. T. Syed et al., "SPECTRE: A Hybrid System for an Adaptive and Optimized Cyber Threats Detection, Response and Investigation in Volatile Memory," *arXiv preprint*, arXiv:2501.03898, 2025.

[2] K. S. Segal et al., "UEFI Memory Forensics: A Framework for UEFI Threat Analysis," *arXiv preprint*, arXiv:2501.16962, 2025.

[3] Volatility Foundation. (2025, Feb. 24). *The Volatility Foundation* [Online]. Available: https://volatilityfoundation.org/

[4] Markruss. (2025, Feb. 24). *ProcDump - Sysinternals* [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/procdump

[5] Microsoft. (2025, Mar. 16). *Event Logging (Event Logging)* [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/eventlog/event-logging

[6] K. Roberts. (2025, Feb. 28). *Sending Automatic Email Notifications When An Active Directory Account Locks* [Online]. Available: https://www.sealingtech.com/2017/05/22/sending-automatic-email-notifications-when-an-active-directory-account-locks/

[7] EventSentry. (2025, Mar. 10). *Windows account lockouts: Real-time alerts and Dashboards* [Online]. Available: https://www.eventsentry.com/account-lockouts

[8] A. V. Novak and D. M. Wood, "PowerShell as a Weapon: A Comprehensive Guide to Detecting and Mitigating Malicious PowerShell Usage," *Cybersecurity Intelligence*, vol. 29, no. 4, pp. 76–89, 2024.

[9] National Institute of Standards and Technology (NIST), Guide to Integrating Forensic Techniques into Incident Response, NIST SP 800-86, Gaithersburg, MD, 2023.

[10] Microsoft. (2024). *Mimikatz detection and prevention* [Online]. Available: https://learn.microsoft.com/

[11] FireEye, *Advanced Memory Analysis in Threat Hunting*, FireEye Threat Report, 2024.

[12] R. Smith and M. J. Thompson, "Indicators of Compromise: A Real-Time Analysis of Memory Artifacts for Threat Detection," *International Journal of Cyber Threat Intelligence*, vol. 38, no. 5, pp. 124–138, 2025.