Threat Model for Human-Operated Ransomware: Scope, Assumptions, and Design Implications

▼ Problem Statement

Commoditized ransomwares are *automated, spray-and-pray attacks,* generally making use of emails with malicious attachments, exploit kits, drive-by downloads etc. Commoditized ransomwares follow a scripted payload with usual targets at once being in thousand(s) with a hope of leverage or hefty payments, generally in crypto currencies. Once it begins execution, encryption is carried out within minutes/hours, examples include; WannaCry, Cerber etc.

Human-Operated Ransomware (HOR) on the other hand are real active attacks executed in real-time by ***Hands-on-keyboard*** attackers. These are multi-stage operations with deliberate, dynamic and contextual phases. **Initial access** is gained by Phishing, RDP brute force or other vulnerability exploits, followed by system reconnaissance of days or even weeks. Then follows series of attempts at credential theft, lateral movements and privilege escalation to gain access to valuable files, identify critical infrastructure or map assets etc. Then follows data exfiltration techniques if employed and or persistence mechanisms (establishing backdoors), then essential backups, shadow copies and security tools are either disabled, deleted, wiped. And finally the encryption of maximum files, networks and/or IT infrastructure begins as the final deployment. Dwell time is usual 7-60+ days between initial compromise and the final encryption. These are targeted attacks with specific intents and clearly determined goals, examples include; Conti, Ryuk, BlackMatter etc.

What makes Human-Operated Ransomware [HOR] are more lethal:

1. Maximum Business Disruption

- They encrypt production servers, backups, and domain controllers **simultaneously**
- Recovery becomes exponentially harder when backups are also encrypted or deleted

2. Double/Triple Extortion

- Data theft means even if we restore from backups, they threaten to leak sensitive data
- DDoS attacks, customer notification threats add pressure on the management.

3. Precision Targeting

- They identify and prioritize an organization’s most critical assets.
- Disable security tools before striking.
- Timed attacks for maximum damage (weekends, holidays).

4. Higher Ransom Demands

- Because impact is catastrophic, ransoms range from hundreds of thousands to millions
- They research an organization’s financials to calibrate demands to be made.

Commoditized ransomware defenses focus on **preventing execution** (antivirus, email filtering). But HOR attackers:

- Use legitimate admin tools (PowerShell, PsExec, WMI) that bypass traditional AV
- Move slowly to avoid behavioral detection
- Use living-off-the-land techniques
- Blend in with normal admin activity

▼ Proposed Idea

The long dwell time is the opportunity to make or break the day in this case, it is the opportunity to be seized, the critical insight of the proposed Anti-HOR (Human-Operated Ransomware) solution is that we have detection window of days or weeks before the actual attack is deployed, while the attackers are:

- Conducting network reconnaissance (port scanning, AD enumeration)

- Attempting credential theft (LSASS dumps, Kerberoasting)
- Moving laterally (unusual RDP sessions, SMB connections)
- Establishing persistence (new scheduled tasks, services)
- Testing access to backups and critical systems

The concentrated solution we need:

1. Behavioral Detection Over Signature-Based:
    - Monitor for attack patterns not individual malware
    - Credential access anomalies
    - Unusual lateral movement
    - Reconnaissance activity
    - Backup tampering attempts
2. **Threat Hunting Capabilities:**
    - Proactive searching for indicators of compromise
    - Historical log analysis to find dormant threats
    - Network traffic analysis for C2 beacons
3. **Rapid Containment:**
    - Network segmentation to isolate compromised systems
    - Automated credential rotation
    - Emergency response playbooks
4. **Deception Technology:**
    - Honeypot accounts that alert on any use
    - Canary files that trigger on access
    - Fake backup servers to detect reconnaissance

Anti-human-operated ransomware strategies focus on proactive detection of attacker TTPs (tactics, techniques, and procedures) before encryption, using endpoint detection & response (EDR), strong access controls (Zero Trust), network segmentation, and rapid response to disrupt the attackers' "hands-on-keyboard" infiltration, rather than just blocking malware, to stop them from escalating privileges, moving laterally, exfiltrating data, and deploying the final ransomware payload.

▼ Scope

Target OS: Windows 10 and 11 + Windows Server

IN SCOPE:

1. **Credential Theft Detection**
    - LSASS process access monitoring
    - Mimikatz-like behavior detection
    - Suspicious process memory reads
    - SAM/SECURITY registry hive access
    - Credential dumping tool indicators
2. **Reconnaissance Activity**
    - Unusual network scanning from the host

- AD enumeration commands (net user, nltest, dsquery)
- SMB share enumeration
- Service enumeration
- Scheduled task enumeration
- Bloodhound-style queries

3. **Lateral Movement Preparation**
   - Remote service creation
   - WMI/PowerShell remoting usage
   - PsExec-like named pipe creation
   - Remote scheduled task creation
   - Pass-the-hash indicators
   - Unusual outbound RDP/SMB connections
4. **Privilege Escalation Attempts**
   - Token manipulation
   - UAC bypass techniques
   - Service exploitation
   - Unquoted service path exploitation
   - DLL hijacking attempts
5. **Persistence Mechanisms**
   - New scheduled tasks (especially hidden/suspicious ones)
   - New services creation
   - Registry run keys modification
   - WMI event subscriptions
   - Startup folder changes
6. **Pre-Encryption Indicators**
   - **CRITICAL**: VSS (Volume Shadow Copy) deletion attempts
   - Backup service tampering (vssadmin, wbadmin, bcdedit commands)
   - Mass file access patterns
   - Encryption tool staging (unusual .exe in temp folders)
   - Boot configuration changes
   - Security software tampering/disabling
7. **Behavioral Anomalies**
   - Unusual processes spawning from unexpected parents
   - PowerShell execution with obfuscation/encoding
   - LOLBin (Living-off-the-land binaries) abuse
   - Fileless malware indicators
   - Suspicious command-line arguments
8. **Host-Based Telemetry Collection**

- Windows Event Logs (Security, System, Sysmon if installed)
- Process creation/termination monitoring
- File system changes (specific directories)
- Registry monitoring (specific keys)
- Network connection tracking (per-host)

OUT OF SCOPE:

1. **Network-Level Detection**
   - Network traffic analysis (PCAP, NetFlow)
   - IDS/IPS signatures
   - East-west traffic monitoring
   - C2 beacon detection at network level
   - Cross-host correlation
2. **Initial Access Prevention**
   - Phishing detection
   - Email filtering
   - Endpoint antivirus/EDR replacement
   - Vulnerability scanning
   - Patch management
   - Firewall rules
3. **Data Exfiltration Detection**
   - Large data transfers to external IPs (network-level)
   - DLP (Data Loss Prevention) functionality
   - Cloud upload monitoring
4. **Active Remediation**
   - Automatic blocking/killing of processes
   - Network isolation/quarantine
   - Credential rotation
   - System restoration
5. **Encrypted/Kernel-Level Attacks**
   - Rootkit detection
   - Bootkits
   - Kernel-mode exploits
   - Hardware-level attacks
6. **Pre-Compromise Activities**
   - Vulnerability assessment
   - Configuration hardening
   - Attack surface reduction
7. **Non-Windows Platforms**

- Linux detection
- macOS detection
- Cross-platform support

8. **Advanced Evasion**
   - Anti-VM/sandbox detection
   - Detecting attackers who have completely disabled all logging

▼ Technical Metric Details and Goals

ATTACKER GOALS (Human-Operated Ransomware Attack Chain):

1. **Establish Persistent Access**
   - Maintain multiple backdoors (scheduled tasks, services, registry keys)
   - Survive reboots and basic cleanup attempts
   - Goal: Remain undetected for days/weeks
2. **Escalate Privileges**
   - Obtain local administrator rights
   - Obtain domain administrator credentials (if domain-joined)
   - Goal: Unrestricted system access
3. **Harvest Credentials**
   - Dump LSASS memory for credentials
   - Extract cached credentials from registry (SAM, SECURITY hives)
   - Capture plain-text passwords if possible
   - Goal: Credentials for lateral movement
4. **Perform Reconnaissance**
   - Map network topology
   - Identify critical servers (file servers, domain controllers, backup servers)
   - Enumerate shares, databases, backup locations
   - Goal: Prioritize high-value targets
5. **Move Laterally** (out of scope for host-based, but they'll try)
   - Use stolen credentials to access other systems
   - Deploy tools across multiple hosts
   - Goal: Widespread network presence
6. **Sabotage Recovery Mechanisms**
   - **Delete Volume Shadow Copies** (`vssadmin delete shadows /all`)
   - **Disable Windows Recovery** (`bcdedit /set {default} recoveryenabled no`)
   - **Delete backups** (`wbadmin delete catalog -quiet`)
   - **Stop backup services** (VSS, Windows Backup, third-party agents)
   - Goal: Make restoration impossible
7. **Deploy Ransomware**
   - Stage encryption binary

- Execute simultaneously across critical systems
- Leave ransom notes
- Goal: Maximum business disruption

Attacker Success Criteria:

- Shadow copies deleted → victim can't restore files
- Backups encrypted/destroyed → no recovery path
- Critical systems encrypted → business operations halted
- Ransom paid (ultimate goal)

Attacker Constraints/Behaviors:

- **Wants to avoid detection** during reconnaissance phase (days/weeks)
- **Uses legitimate tools** (PowerShell, WMI, PsExec) to blend in
- **Operates during off-hours** (nights/weekends) when less monitoring
- **Tests before deploying** (small-scale encryption tests)
- **Moves deliberately, not rapidly** (unlike automated malware)

DEFENDER GOALS (Our Tool- *to be developed*)

Detect Human-Operated Ransomware during the **pre-encryption phase** and alert before catastrophic damage occurs.

Concrete Goals:

1. **Early Warning Detection**
   - Detect attacker reconnaissance activities
   - Detect credential harvesting attempts
   - Detect persistence establishment
   - **Goal**: Alert within minutes of suspicious behavior, ideally days before encryption
2. **Critical Asset Protection**
   - Detect any attempts to tamper with backup systems
   - Detect shadow copy deletion
   - Detect backup service manipulation
   - **Goal**: Zero missed detections for backup sabotage (top priority)
3. **High-Fidelity Alerting**
   - Generate actionable alerts with context
   - Minimize false positives (< 5% false positive rate acceptable)
   - Provide evidence for incident response and digital forensics (post incident)
   - **Goal**: Security team can respond confidently
4. **Continuous Monitoring**
   - Run 24/7 with minimal performance impact
   - Survive attacker attempts to disable it (within userland constraints)
   - Maintain alert capability even if partially compromised
   - **Goal**: Reliable operation under adversarial conditions
5. **Forensic Value**

- Log all detected behaviors with timestamps
- Capture process trees, command lines, user context
- Capture forensic disk image in case of incident
- Provide timeline of attacker activities
- **Goal**: Enable post-incident investigation

Defender Success Criteria:

- Detect shadow copy deletion **100% of the time**
- Detect LSASS access attempts **>90% of the time**
- Detect suspicious service/task creation **>85% of the time**
- Alert generated **before** encryption begins
- False positive rate **<5%** for critical alerts
- Performance impact **<5% CPU, <100MB RAM**
- Alerts include actionable context (what, when, who, how)

Defender Metrics (For Evaluation):

```
Detection Rate = (True Positives) / (True Positives + False Negatives)
```

```
False Positive Rate = (False Positives) / (False Positives + True Negatives)
```

```
Time to Detect = Time from suspicious activity → alert generation
```

```
Alert Quality = % of alerts with sufficient context for response
```

1. **CONSTRAINTS:**
   a. **Userland Only (No Kernel Access)**
      - Cannot intercept system calls at kernel level
      - Cannot use kernel drivers or filter drivers
      - Cannot prevent actions, only detect after they occur
      - No minifilter for file system operations
      - No network filter driver for traffic inspection
   b. **What the tool CAN Do:**
      - Monitor processes via Windows APIs (`psutil`, `wmi`)
      - Read Windows Event Logs (`win32evtlog`)
      - Monitor file system changes via polling or directory watching (`watchdog`)
      - Query system state (services, tasks, registry)
      - Monitor network connections per-process (`psutil.net_connections()`)
   c. **Impact on Detection:**
      - **Timing**: Slight delay between action and detection (milliseconds to seconds)
      - **Evasion**: Sophisticated attackers could potentially detect and disable the agent
      - **Visibility**: Limited to what Windows APIs expose (can't detect kernel-mode rootkits)

2. **Python Language Constraints:**

   What the tool CANNOT do:

   - Not as performant as C/C++ native code
   - Slower startup time (Python interpreter initialization)
   - Larger memory footprint
   - Easier to reverse engineer (even if compiled to .exe)
   - Requires Python runtime (unless packaged with PyInstaller)

   What the tool CAN Do:

   - Rapid development and iteration
   - Rich library ecosystem (`psutil`, `watchdog`, `pywin32`)
   - Easy rule definition (YAML config files)
   - Cross-version compatibility (Win10/11/Server with same code)
   - Readable, maintainable code

   Mitigation Strategies:

   - Using `PyInstaller` to create standalone .exe (no Python install needed)
   - Optimizing hot paths (using generators, avoiding unnecessary copies)
   - Multi-threading for parallel monitoring tasks
   - Minimize dependencies (keeping it lightweight)

   Performance Targets:

   - CPU usage: <5% on average, <10% during detection event
   - Memory: <100MB RAM footprint
   - Startup time: <5 seconds
   - Alert latency: <2 seconds from event to alert

3. **Windows Userland API Constraints:**

   Process Monitoring:

   - ✓ Can enumerate processes (`psutil.process_iter()`)
   - ✓ Can read command-line arguments (if accessible)
   - × Cannot see processes started/stopped between polling intervals (unless using Event Logs)
   - × Protected processes may block access (PPL - Protected Process Light)

   File Monitoring:

   - ✓ Can watch directories for changes (`watchdog`)
   - × Real-time monitoring has slight delay (not instant)
   - × High I/O activity could cause missed events
   - × Cannot prevent file operations (detect only)

   Memory Access:

   - ✓ Can detect handles to LSASS process
   - × Cannot read memory of other processes (unless running as SYSTEM with SeDebugPrivilege)
   - × Cannot inject into other processes

Registry Monitoring:

- ✓ Can poll registry keys for changes
- × No real-time registry change notification in pure Python (without WMI)
- ✓ Can use WMI for registry event monitoring

Network Monitoring:

- ✓ Can see established connections per-process
- × Cannot see packet contents (no packet capture)
- × Cannot block connections
- × Ephemeral connections might be missed between polls

Event Logs:

- ✓ Can read Security, System, Application logs
- × Depends on audit policy being enabled (not your control)
- × Logs can be cleared by attacker (if they have admin rights)
- ✓ Can detect log clearing itself (Event ID 1102)

4. **Privilege Level Constraints**

   Ideal Scenario:

   - The tool runs as **Local System** or with **Administrator privileges**
   - This gives maximum visibility

   Constraints:

   - × Still userland (not kernel)
   - × Attacker with same privileges can terminate the tool process
   - × Attacker can modify the configuration/rules
   - × Attacker can clear the logs generated

   Mitigation:

   - Tool Run as Windows Service (harder to casually terminate)
   - Store critical alerts remotely (if possible) or in protected Event Log
   - Detect service tampering (monitor our own service status)
   - Code obfuscation (minor deterrent)

5. **Detection vs. Prevention Constraint**

   The Tool Can:

   - **Detect** suspicious behavior
   - **Alert** security team
   - **Log** forensic evidence

   The Tool CANNOT:

   - **Block** processes from executing
   - **Prevent** file encryption
   - **Kill** attacker processes (could, but risky - false positives)
   - **Isolate** the host from network (no firewall control)

Philosophy:
The tool is a **detection sensor**, not an EDR/prevention agent. It provides the "tripwire" that alerts defenders who then take action.

6. **Time Constraint (Semester 2):**
   - No time for ML/AI models (data collection, training, tuning)
   - No time for complex UI/dashboard
   - No time for distributed architecture

   Rather the tool;

   - Focuses on **rule-based detection** (fast to implement and explain)
   - Focuses on **5-7 high-value detections**
   - Focuses on **proof of concept**, not production-hardened tool

---

CONCRETE EXAMPLE: Shadow Copy Deletion

Attacker Goal:

- Executes `vssadmin delete shadows /all /quiet` to destroy restore points.

Defender Goal:

- Detect this command execution within 2 seconds and generate high-severity alert.

Constraints in Action:

- **Userland Constraint:**
  - ✗ Cannot intercept the system call to `vssadmin.exe` at kernel level
  - ✓ Can monitor process creation via:
    - Polling `psutil.process_iter()` for new `vssadmin.exe` processes
    - Reading Event Log 4688 (Process Creation) if auditing enabled
    - Monitoring command-line arguments for "delete shadows"
- **Python Constraint:**
  - Detection happens in userland with ~100ms - 2s delay
  - Still acceptable (shadow copy deletion takes a few seconds)

Detection Method:

```
# Pseudocode
for process in psutil.process_iter(['name', 'cmdline']):
    if process.info['name'] == 'vssadmin.exe':
        cmdline = ' '.join(process.info['cmdline'])
        if 'delete' in cmdline and 'shadow' in cmdline:
            ALERT_CRITICAL("Shadow copy deletion detected!")
```

▼ SUMMARY TABLE

Aspect	Attacker	Defender (Tool)	Constraint Impact
Primary Goal	Encrypt systems for ransom	Detect pre-encryption behaviors	Must detect before encryption
Timeline	Days/weeks of reconnaissance	Minutes to detect suspicious activity	Userland = slight delay acceptable
Evasion	Use legitimate tools, blend in	Behavioral detection rules	Python = easier to reverse engineer

Aspect	Attacker	Defender (Tool)	Constraint Impact
Persistence	Multiple backdoors	Run as service, self-monitor	Userland = can be killed by admin attacker
Critical Action	Delete shadow copies	Detect shadow copy deletion 100%	Event Log + process monitoring achievable
Success Metric	Ransom paid	Alert before encryption	Time window = days (feasible)

▼ Hybrid Approach: Tiered Detection

Tier 1: Critical Single Behaviors (No Correlation Needed)
These generate immediate CRITICAL alerts:

```
CRITICAL_BEHAVIORS = [
    "shadow_copy_deletion",      # vssadmin, wmic, PowerShell
    "backup_deletion",           # wbadmin commands
    "boot_config_tampering",     # bcdedit
    "security_tool_disabling",   # Defender, firewall
]
```

Why: These are "last chance" detections - no time for correlation, act immediately.

Tier 2: Suspicious Behaviors (Correlation + Scoring)
These accumulate a score over a time window (e.g., 1 hour):

```
SUSPICIOUS_BEHAVIORS = {
    "lsass_access": 30,
    "encoded_powershell": 25,
    "lateral_movement": 20,
    "ad_enumeration": 15,
    "unusual_scheduled_task": 20,
    "backup_service_query": 15,
}

# Alert thresholds
SCORE_CRITICAL = 75   # Multiple strong indicators
SCORE_HIGH = 50       # Several suspicious activities
SCORE_MEDIUM = 30     # Worth investigating
```

Why: These behaviors are individually noisy but collectively indicative of HOR.

Tier 3: Informational (Log Only, No Alerts)
Context for investigations:

```
INFORMATIONAL = [
    "process_creation",
    "network_connection",
    "file_access",
    "registry_change",
]
```

Why: Too noisy to alert on, but useful for forensics after an alert.

▼ Telemetry Feasibility Matrix

CRITICAL BEHAVIORS (Tier 1 - Pre-Encryption Indicators)

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexity
Shadow Copy Deletion	Entirely	`psutil` process monitoring for `vssadmin.exe`	Windows Event Log 8 (VSS events)	None - highly reliable	Yes (for Event Logs)	Low
Backup Catalog Deletion	Entirely	`psutil` process monitoring for `wbadmin.exe`	Command-line argument parsing	None	No (process enum)	Low
Boot Config Changes	Entirely	`psutil` process monitoring for `bcdedit.exe`	Event Log 4688 (process creation)	None	Yes (for Event Logs)	Low
VSS Service Stop/Disable	Entirely	`psutil.win_service_get('VSS')` polling	WMI service queries	Polling delay (1-5 sec)	Yes	Low
Windows Backup Service Stop	Entirely	`psutil.win_service_get('wbengine')`	WMI queries	Polling delay	Yes	Low
Event Log Clearing	Entirely	Windows Event Log 1102/104 monitoring	`win32evtlog` API	Requires audit policy enabled	Yes	Medium

CREDENTIAL THEFT BEHAVIORS (Tier 2)

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Comple
LSASS Process Access	Partial	`psutil` - enumerate processes with open handles to `lsass.exe`	Sysmon Event 10 (ProcessAccess)	Cannot see handle opens in real-time without polling	Yes (SeDebugPrivilege)	High
SAM/SECURITY Registry Access	Partial	Monitor processes accessing `C:\Windows\System32\config\SAM`	Event Log 4663 (object access)	Requires file auditing enabled	Yes	Mediun
Credential Manager Access	Partial	Monitor `vaultcmd.exe`, `cmdkey.exe` processes	Process command-line monitoring	Polling-based, may miss short-lived processes	No	Low
Process Memory Dumping	Not Feasible	N/A - kernel-level visibility needed	Sysmon Event 10 (if installed)	Cannot detect memory reads from userland	N/A	N/A
Mimikatz Execution	Partial	YARA scanning of process memory	String matching in command-lines	Can be obfuscated, renamed	Yes (memory access)	High

PROCESS & EXECUTION MONITORING (Tier 2)

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexit
Process Creation	Full	`psutil.process_iter()` polling	Event Log 4688 (if audit enabled)	Polling interval = 0.5-2 sec delay	No (basic enum) (cmdline)	Low
Process Termination	Full	`psutil.pid_exists()` tracking	WMI `Win32_ProcessStopTrace` event	Polling-based, miss rapid create/destroy	No	Medium
Process Command-	Partial	`psutil.Process(pid).cmdline()`	Event Log 4688 (command line	Protected processes may	Yes	Low

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexity
Line Args			logging)	block access		
Parent-Child Process Tree	Full	`psutil.Process(pid).ppid()`	Build tree via polling	Accurate if polled frequently	No	Medium
PowerShell Execution	Full	Monitor `powershell.exe` / `pwsh.exe` processes	Event Log 4103/4104 (script block logging)	Script block logging must be enabled	No (process) (logs)	Low
Encoded PowerShell	Full	Parse cmdline for `-enc`, `-e`, `-EncodedCommand`	N/A	Easy to detect in command-line	No	Low
WMI Process Creation	Partial	Monitor `wmic.exe` or `WmiPrvSE.exe`	Event Log 5857/5858 (WMI activity)	Indirect detection via parent process	Yes (Event Logs)	Medium

PERSISTENCE MECHANISMS (Tier 2)

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexity
Scheduled Task Creation	Full	`win32com.client` Task Scheduler API	Monitor `schtasks.exe` process	API access is comprehensive	Yes	Medium
Scheduled Task Modification	Full	Poll Task Scheduler, compare task list	Event Log 4698/4702	Polling delay	Yes	Medium
Service Installation	Full	`psutil.win_service_iter()` polling	Event Log 7045 (new service)	Polling delay (1-5 sec)	Yes	Low
Service Modification	Full	Monitor service config changes	Event Log 7040 (service state change)	Polling-based detection	Yes	Low
Registry Run Keys	Full	`winreg` module - poll registry keys	WMI registry event subscription	Polling delay, many keys to monitor	No	Medium
Startup Folder Changes	Full	`watchdog` monitoring `shell:startup` folder	File system polling	Real-time via watchdog	No	Low
WMI Event Subscription	Partial	Query `__EventFilter`, `__EventConsumer` WMI classes	PowerShell `Get-WmiObject` wrapper	Requires WMI knowledge	Yes	High

LATERAL MOVEMENT INDICATORS (Tier 2)

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexity
SMB Outbound Connections	Full	`psutil.net_connections()` filter port 445	Event Log 5140/5145 (shares)	Polling-based, may miss ephemeral connections	No	Low
RDP Outbound Connections	Full	`psutil.net_connections()` filter port 3389	Event Log 4624 Type 10 (RDP logon)	Polling delay	No	Low
PsExec Usage	Partial	Detect `psexec.exe` or `PSEXESVC` service	Named pipe `\pipe\psexec*` creation	Can be renamed, obfuscated	Yes (named pipes)	Medium

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexity
WinRM/PowerShell Remoting	Partial	Monitor `wsmprovhost.exe` process	Event Log 4688, network port 5985/5986	Indirect detection	No (process) (logs)	Medium
Remote Service Creation	Partial	Monitor `services.exe` + network activity correlation	Event Log 7045 + source IP	Requires correlation, complex	Yes	High
Named Pipe Creation	Not Feasible	N/A - requires kernel driver	Sysmon Event 17/18 (if installed)	No userland API in Python	N/A	N/A

RECONNAISSANCE BEHAVIORS (Tier 3)

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexity
Network Scanning	Partial	Monitor unusual `connect()` patterns via `psutil`	Event Log 5156 (Windows Firewall)	Many false positives, hard to distinguish	No	High
AD Enumeration	Full	Monitor `net.exe`, `nltest.exe`, `dsquery.exe`	Event Log 4688 + cmdline	Process monitoring sufficient	No	Low
SMB Share Enumeration	Partial	Monitor `net.exe view`, `net.exe share`	Event Log 5140 (share access)	Indirect detection via commands	No	Medium
Bloodhound/SharpHound	Partial	Detect LDAP queries via process cmdlines	Network connection patterns to DC	Requires correlation	No	High
Port Scanning	Not Feasible	N/A - need packet-level visibility	Firewall logs (external to Python)	Cannot reliably detect from host only	N/A	N/A

FILE SYSTEM ACTIVITY (Tier 2/3)

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexity
Mass File Modifications	Full	`watchdog` directory monitoring + rate limiting	Poll file modification times	High I/O may cause watchdog delays	No	Medium
Suspicious File Extensions	Full	`watchdog` + pattern matching (`.encrypted`, `.locked`)	File system polling	Real-time via watchdog	No	Low
File Entropy Analysis	Partial	Read file bytes, calculate Shannon entropy	N/A	CPU-intensive, may impact performance	No	Medium

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexity
Canary File Access	Full	`watchdog` monitoring specific honeypot files	File access auditing (Event 4663)	Reliable with watchdog	No	Low
Rapid Directory Traversal	Partial	Monitor file handle creation rate	N/A	Hard to distinguish from legitimate batch operations	No	High

NETWORK ACTIVITY (Tier 3)

Behavior	Feasible?	Primary Method	Alternative Method	Limitations	Admin Required?	Code Complexity
Outbound Connections	Full	`psutil.net_connections()`	Event Log 5156 (Windows Firewall)	Per-process connections visible	No	Low
C2 Beacon Detection	Not Feasible	N/A - need packet inspection	External network monitoring	No payload visibility from userland	N/A	N/A
DNS Queries	Not Feasible	N/A - need packet capture	Windows DNS Client Event Log 3008	Cannot see query content reliably	N/A	N/A
Unusual External IPs	Partial	Parse connections, check against threat intel	N/A	Requires external IP reputation data	No	Medium

WINDOWS EVENT LOGS (Data Source)

Event Log Type	Feasible?	Access Method	Key Event IDs	Limitations	Admin Required?	Code Complexity
Security Log	Full	`win32evtlog` or `evtx` library	4688 (process), 4624 (logon), 4663 (object access)	Requires audit policies enabled	Yes	Medium
System Log	Full	`win32evtlog`	7045 (service install), 7040 (service change)	Reliable if policies set	Yes	Medium
Application Log	Full	`win32evtlog`	Custom app events	Varies by application	No	Low
Sysmon Log	Full	`win32evtlog` (if Sysmon installed)	1 (process), 3 (network), 10 (process access)	Requires Sysmon installation	Yes (to install)	Medium
PowerShell Log	Full	`win32evtlog`	4103/4104 (script block logging)	Must enable script block logging	Yes	Medium
VSS Event Log	Full	`win32evtlog`	8 (shadow copy deletion)	Reliable for shadow operations	Yes	Low

OVERALL FEASIBILITY SUMMARY

Highly Feasible:

- Shadow copy deletion, backup sabotage, boot config changes
- Process creation/termination monitoring
- Service installation/modification
- Scheduled task creation/modification
- Registry run key monitoring
- File system mass modifications
- Outbound network connections (SMB/RDP)
- Windows Event Log parsing (Security, System, VSS)
- PowerShell execution detection (encoded commands)
- AD enumeration tool detection

Partially Feasible (Possible with Workarounds):

- LSASS process access (polling-based, may miss some)
- WMI process creation (indirect detection)
- PsExec usage (heuristics)
- Network scanning (high false positives)
- File entropy analysis (performance cost)
- Remote service creation (requires correlation)

Not Feasible (Out of Scope):

- Process memory dumping detection (kernel-level needed)
- Named pipe creation (no userland API)
- C2 beacon detection (packet inspection required)
- Kernel-mode exploits/rootkits
- Hardware-level attacks
- DNS query content inspection
- Port scanning from host only

PYTHON LIBRARIES REQUIRED:

Library	Purpose	Installation	License
`psutil`	Process, service, network monitoring	`pip install psutil`	BSD
`pywin32`	Windows API access (Event Logs, COM)	`pip install pywin32`	PSF
`watchdog`	File system monitoring	`pip install watchdog`	Apache 2.0
`winreg`	Registry access	Built-in (standard library)	PSF
`wmi`	WMI queries (optional)	`pip install WMI`	MIT
`evtx`	Event Log parsing (alternative)	`pip install python-evtx`	Apache 2.0

PERFORMANCE CONSIDERATIONS:

Monitoring Type	Polling Interval	CPU Impact	Memory Impact
Process enumeration	1-2 seconds	1-2%	20-30 MB
Service monitoring	5 seconds	<1%	5 MB

Monitoring Type	Polling Interval	CPU Impact	Memory Impact
File system (watchdog)	Real-time events	1-3%	10-20 MB
Event Log reading	5-10 seconds	<1%	10 MB
Network connections	2-5 seconds	<1%	5 MB
Total Estimated	N/A	**3-7%**	**50-100 MB**

Note: Performance to be tested on Windows 10 VM with 4 CPU cores, 8GB RAM

KEY LIMITATIONS TO ACKNOWLEDGE:

1. **Polling Delays**: Process/service changes detected with 1-5 second latency
2. **Ephemeral Processes**: Very short-lived processes may be missed between polls
3. **Protected Processes**: Some system processes block userland inspection
4. **Audit Policies**: Event Log detections require proper Windows audit configuration
5. **Kernel-Level Evasion**: Rootkits/bootkits cannot be detected from userland
6. **No Prevention**: Detection only - cannot block malicious actions
7. **Admin Privileges**: Most valuable detections require running as SYSTEM/Administrator

DEPLOYMENT REQUIREMENTS:

Minimum:

- Python 3.8+
- Windows 10/11 or Server 2016+
- Standard user privileges (limited detection capability)

Recommended:

- Python 3.10+
- Windows 10/11 or Server 2019+
- Administrator or SYSTEM privileges
- Audit policies enabled:
  - Process Creation (4688) with command-line logging
  - Object Access (4663) for sensitive files
  - Security Event Log clearing (1102)
- Optional: Sysmon installed for enhanced visibility