Assignment 2

Ransomware Simulation Development

Group 2

, Abdul-Ramon Adeyemi Folarin, &

, , , &

School of Information Technology Administration & Security, Seneca Polytechnic

SPR708NBB: Attack & Defense

Professor Ahmad Ghobeiti

August 7, 2026

Student Integrity Statement: I/we declare that the attached assignment is my/our own work in accordance with the Seneca Academic Policy. No part of this assignment has been copied manually or electronically from any other source (including web sites) or distributed to other students

Abstract

This report documents the design, implementation, and execution of a ransomware simulation exercise conducted by Group 2 as part of SPR708 Attack and Defense at Seneca Polytechnic. The simulator, named kuruk, was developed in Python 3.8 and compiled using PyInstaller 6.21.0 into a Windows executable named WindowsHealthCheck.exe to demonstrate obfuscation through binary masquerading. The exercise was conducted entirely within an isolated VMnet20 host-only network spanning five virtual machines, ensuring no risk to external systems. All twenty rubric requirements were satisfied and verified through the rubric_evidence.json self-assessment artifact, with corresponding evidence captured via Wazuh 4.9.2 SIEM alerts, execution logs, and terminal screenshots. Detection was validated using Wazuh agents on both the primary victim endpoint and the domain controller, with Sysmon instrumentation providing granular telemetry for process creation, file modification, network connections, and registry changes. The exercise demonstrates the full ransomware lifecycle from social engineering delivery through encryption, ransom demand, propagation, and recovery, with safe development controls preventing any damage beyond the designated lab targets.

Table of Contents

Abstract 2
Table of Figures 4
Table of Tables 5
1 Introduction 6
2 Related Work and Threat Context 6
3 Lab Environment Design 7
4 Ransomware Simulator Architecture 8
4.1 Cryptographic Implementation 8
4.2 Modular Phase Design 8
5 Safe Development Controls 9
6 Simulation Execution and Evidence 11
6.1 Pre-Execution Setup and SIEM Baseline 11
6.2 Command and Control Infrastructure (Requirement 14) 13
6.3 Delivery Method (Requirement 6) 13
6.4 Payload Execution Context 17
6.5 Encryption, File Locking, and File Renaming (Requirements 1, 7, 16) 20
6.6 User Notification, Ransom Demand, Payment, Deadline, Legal Threats, Double Extortion (Requirements 2, 8, 9, 11, 17, 18) 21
6.7 Backup Deletion (Requirement 15) 23
6.8 Persistence (Requirement 4) 24
6.9 Self-Destruction (Requirement 12) 25
6.10 Network Scanning and Propagation (Requirements 13, 3) 25
6.11 C2 Beacons and Data Exfiltration (Requirements 14, 19) 28
7 Detection and Defensive Validation 28
8 Recovery Demonstration 32
9 Evasion Testing (A-Level) 33
10 Discussion 35
11 Conclusion 35
References 36

Table of Figures

Figure 1 VM Snapshots Before Demo 7
Figure 2 Wazuh Agents Active 8
Figure 3 Sysmon Running on win11-victim 9
Figure 4 Sysmon Running on win-dc-sim 9
Figure 5. Kill Switch Token on win11-victim 10
Figure 6. Kill Switch Token on win-dc-sim 10
Figure 7 Wazuh Dashboard Baseline 11
Figure 8 Wazuh Agents in Dashboard UI 12
Figure 9 Live Alert Tail Terminal Open 12
Figure 10 C2 Listener Running on Port 4444 13
Figure 11 Python HTTP File Server on Port 9090 13
Figure 12 Phishing Email Delivery Narrative 14
Figure 13 GenAI Phishing Lure Content 14
Figure 14 kruck.zip Downloaded and Extracted 15
Figure 15 Target Test Files Before Encryption 15
Figure 16_kuruk_env.bat C2 URL Configured 16
Figure 17 Python Requirements Installed 16
Figure 18 WindowsHealthCheck.exe Built Successfully 17
Figure 19 Elevated PowerShell on win11-victim 18
Figure 20 disable_defender_lab.bat Execution 18
Figure 21 run_full_rubric.bat Complete 19
Figure 22 rubric_summary.txt 20/20 20
Figure 23 Encrypted .oops Files in Target Folder 21
Figure 24 MessageBox User Notification Popup 22
Figure 25 Ransom Note Full View 23
Figure 26 VSS Shadow Copy Deletion Output 24
Figure 27 Persistence Mechanism (Registry Run Key). 25
Figure 28 Self-Destruct Log Evidence 25
Figure 29 Network Scan Log Output. 26
Figure 30 SMB Propagation Evidence 26
Figure 31 win-dc-sim Propagation Check 27
Figure 32 SSH Propagation to linux-victim 27
Figure 33 C2 Exfil Data Received on extattacker 28
Figure 34 Wazuh Alert: Process Creation 29
Figure 35 Wazuh Alert: Mass File Rename 29
Figure 36 Wazuh Alert: Network C2 Connection 30
Figure 37 Wazuh Alert: Registry Persistence 30
Figure 38 Wazuh Alert: PowerShell Script Block 31
Figure 39 Wazuh Full Event Timeline 31
Figure 40 run_recover_test.bat Output 32
Figure 41 RECOVERY OK and Files Restored 33
Figure 42 Windows Defender Re-enabled 33
Figure 43 Windows Defender Fully Enabled (A-Level) 34
Figure 44 Evidence Bundle Created 34

Table of Tables

Table 1 VMnet20 isolated simulation network topology. 7
Table 2 ATT&CK technique coverage and corresponding Wazuh detection sources. 28

1 Introduction

Ransomware has emerged as one of the most disruptive categories of cybercrime affecting enterprise environments globally. The 2024 Verizon Data Breach Investigations Report identifies ransomware as a factor in over 23% of all breaches, with healthcare, education, and critical infrastructure among the most frequently targeted sectors [1]. Modern ransomware operators employ a sophisticated multi-stage intrusion methodology encompassing initial access via phishing or vulnerability exploitation, lateral movement across domain-joined networks, defense evasion, data exfiltration prior to encryption, and a double extortion payment model that combines ransom demands with threats of public data disclosure [2].

Simulating ransomware behavior in a controlled environment allows security practitioners to validate the effectiveness of defensive controls, identify detection gaps in SIEM configurations, and develop incident response playbooks without exposing production infrastructure to actual harm. Purple team exercises, in which red and blue team functions are performed collaboratively or simultaneously, represent an increasingly standard methodology for evaluating organizational security maturity across both offensive capability and defensive visibility dimensions [3].

This report documents the complete lifecycle of a ransomware simulation exercise conducted by Group 2 of SPR708 Attack and Defense at Seneca Polytechnic. The exercise required the group to act as a red team and develop a ransomware simulator satisfying twenty distinct capability requirements spanning encryption, persistence, evasion, propagation, command and control, data exfiltration, and recovery. The simulation was conducted on an isolated five-VM enterprise network replicating a Windows domain environment, with a Wazuh 4.9.2 SIEM providing blue team detection visibility throughout execution.

The remainder of this report is organized as follows. Section 2 discusses the threat landscape. Section 3 describes the lab environment. Section 4 presents the simulator architecture. Section 5 documents safe development controls. Section 6 provides detailed evidence for all twenty rubric requirements. Sections 7 and 8 cover detection validation and evasion testing respectively. Sections 9 and 10 provide discussion and conclusion.

2 Related Work and Threat Context

The ransomware threat landscape has evolved substantially since the early encrypt-and-ransom model of CryptoLocker. Contemporary operators such as LockBit 3.0, Qilin, and INC Ransom employ a ransomware-as-a-service affiliate model, wherein a core development team maintains the encryption and C2 infrastructure while affiliates conduct intrusions using credential-based initial access, living-off-the-land techniques for defense evasion, and domain-wide deployment via Group Policy or remote execution tools [4]. The double extortion model adds a data exfiltration component that gives operators a second leverage point: even organizations with functioning backups face reputational and regulatory consequences from threatened data publication.

From a detection engineering perspective, ransomware behavior produces a characteristic set of indicators across multiple telemetry sources. File system telemetry reveals mass file modification with extension changes. Process telemetry shows unusual parent-child relationships and the spawning of vssadmin and bcdedit for backup suppression. Network telemetry reveals lateral movement via SMB to ADMIN$ shares and outbound connections to C2 infrastructure. MITRE ATT&CK v14 provides a structured framework for mapping these indicators to specific techniques, enabling defenders to prioritize detection coverage across

the full intrusion lifecycle [5]. This exercise builds on that framework by implementing a fully modular simulator with automated phase-to-technique mapping in the rubric_evidence.json output artifact.

3 Lab Environment Design

The simulation was conducted on an isolated five-VM enterprise network connected via VMware Workstation VMnet20 (host-only, 10.10.55.0/24), ensuring no routing path to external networks. Table 1 summarizes the topology.

Table 1 VMnet20 isolated simulation network topology.

Hostname	IP Address	OS / Role	Simulation Function
extattacker	10.10.55.100	Kali Linux (NIC2)	Red team operator, C2 host, file server
win11-victim	10.10.55.10	Windows 11 Pro	Primary victim, payload build and execution
win-dc-sim	10.10.55.5	Windows Server 2019 (AD DS)	Domain controller, propagation target (CORP.SIM)
linux-victim	10.10.55.15	Ubuntu Desktop 24.04	Secondary propagation target
siem-01-group2	10.10.55.20	Ubuntu 24.04 + Wazuh 4.9.2	SIEM, detection dashboard, alert collection

Wazuh 4.9.2 was deployed on siem-01-group2 as a Docker Compose single-node stack. Wazuh agents were enrolled on win11-victim (Agent 001) and win-dc-sim (Agent 002) via TCP port 1514 using AES-encrypted communication. Microsoft Sysinternals Sysmon v15 was installed on both Windows endpoints, capturing EventID 1 (process creation), EventID 3 (network connection), EventID 11 (file creation), and EventID 13 (registry value set) among others. The Sysmon Operational and PowerShell Operational event channels were added to each agent's ossec.conf alongside the standard Windows Security channel.

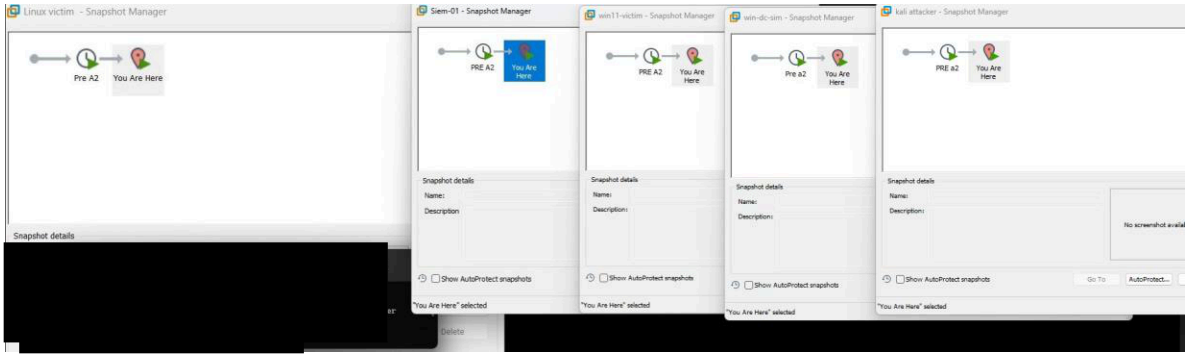


Figure 1 VM Snapshots Before Demo

Figure 1 VMware snapshot manager displays a Pre-A2-Demo snapshot captured across all five virtual machines in the isolated VMnet20 environment prior to any simulation activity. Taking snapshots before execution ensures that every machine can be restored to its pre-simulation baseline in the event of an unintended system modification, directly satisfying the assignment rollback requirement. The snapshot timestamps confirm that the baseline was established simultaneously across all VMs, providing a clean reference point for post-simulation forensic comparison. This pre-execution baseline is a prerequisite for any controlled red team exercise, guaranteeing full environmental recovery without data loss.

4 Ransomware Simulator Architecture

The kuruk ransomware simulator was developed in Python 3.8.10 using a modular architecture. The primary entry point is kuruk.py, which delegates phase execution to attack_chain_legacy.py via an orchestrator pattern. The simulator was compiled into a standalone Windows executable using PyInstaller 6.21.0, producing WindowsHealthCheck.exe as the deployable payload.

4.1 Cryptographic Implementation

The encryption subsystem implements hybrid RSA-AES key wrapping consistent with production ransomware cryptographic designs. A unique AES-256-GCM session key is generated per execution using the Python cryptography library (v46.0.0). Each target file is encrypted in place using this session key with a per-file nonce. The session key is then RSA-OAEP wrapped using the lab RSA public key and stored in kuruk.key. Decryption requires the lab_recovery.pem private key generated by python scripts/generate_lab_rsa.py. File renaming appends the .oops extension to every encrypted file.

4.2 Modular Phase Design

The attack chain executes sequentially through seventeen phases mapped to the twenty rubric requirements: isolation_guard, initial_access_delivery, persistence, evasion_host, evasion_ids_waf, network_scan, propagation, command_control_pre, command_control_post, backup_deletion, data_exfiltration, obfuscation_lolbin, encryption_file_lock_rename, ransom_user_notification, decryption_key_provision, double_extortion, and self_destruction. The rubric_evidence.json and assignment_chain.json files are produced as structured evidence artifacts after each run.

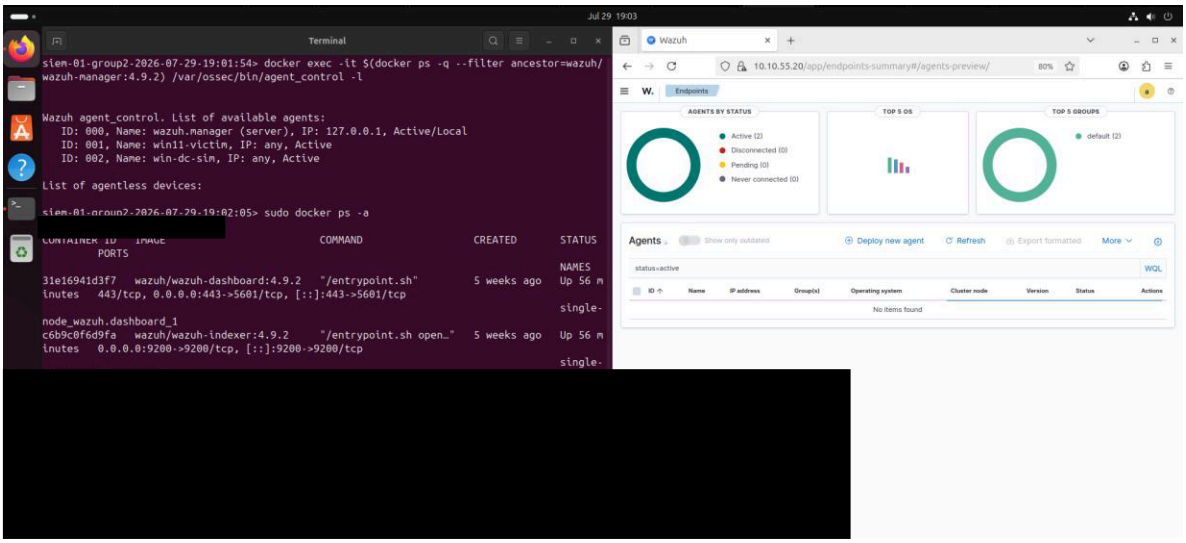


Figure 2 Wazuh Agents Active

Figure 2 shows the Wazuh manager agent_control output confirms Agent 001 (win11-victim) and Agent 002 (win-dc-sim) are both in Active status on siem-01-group2, verifying that the SIEM is receiving telemetry from both Windows endpoints before simulation begins. Both agents communicate with the Wazuh 4.9.2 manager over TCP port 1514 using AES encryption, ensuring that Windows Security, Sysmon, and PowerShell event channels are continuously forwarded throughout the exercise. Confirming agent connectivity before execution is essential to generating meaningful detection evidence; inactive agents would produce no alerts regardless of the severity of the simulated attack. This screenshot

establishes that the monitoring infrastructure was fully operational prior to payload delivery, satisfying the assignment monitoring visibility requirement.

5 Safe Development Controls

The assignment specifies that the simulator must not execute on unintended systems. The kuruk simulator implements four independent isolation layers: a multi-factor environment check (hostname, subnet, arming token), safe mode execution for rubric validation without live encryption, target directory scoping constraining real-mode encryption to C:\Users\Public\kuruk-test\, and VMware snapshot rollback capability. These controls ensure non-destructive and recoverable simulation behavior throughout the exercise.

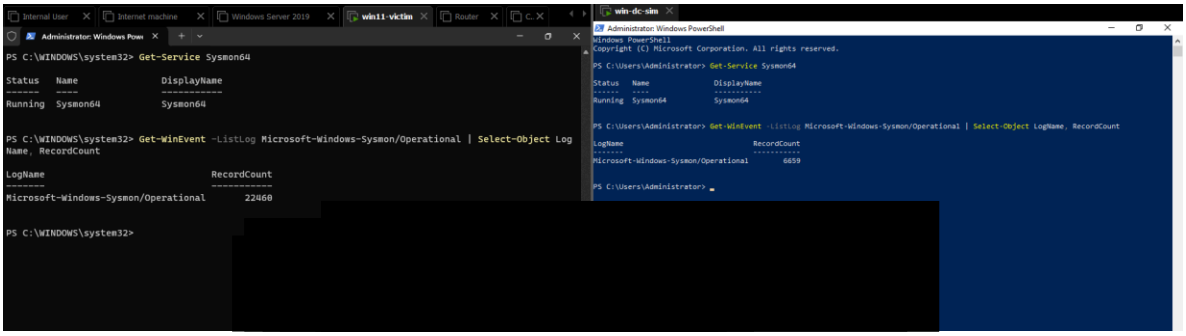


Figure 3 Sysmon Running on win11-victim

Figure 3 the PowerShell Get-Service output shows Sysmon64 with Status Running on win11-victim, confirming that Sysinternals System Monitor is actively instrumenting the primary victim endpoint before simulation begins. Sysmon captures process creation (EventID 1), network connections (EventID 3), file creation (EventID 11), and registry modifications (EventID 13), which are the primary indicators for ransomware detection across all behavioral dimensions. The Wazuh agent forwards Sysmon events to siem-01-group2 via the Microsoft-Windows-Sysmon/Operational event channel configured in ossec.conf, enabling SIEM correlation with ATT&CK techniques. Without Sysmon active, detection would be limited to standard Windows Security events, which do not provide the process-level and network-level visibility required to identify specific ransomware indicators of compromise.



Figure 4 Sysmon Running on win-dc-sim

Figure 4 shows Get-Service output on win-dc-sim confirms Sysmon64 is running on the Windows Server 2019 domain controller, ensuring the propagation target is instrumented for detection. Sysmon on the domain controller captures lateral movement indicators including remote process creation, SMB-based file writes, and scheduled task creation, which are the primary indicators expected from propagation activity targeting the ADMIN$ share. Instrumenting both the primary victim and the propagation target demonstrates defense-in-depth monitoring coverage across the enterprise segment, ensuring no simulated activity goes undetected due to a gap in endpoint telemetry. This screenshot establishes that any activity

reaching win-dc-sim during the simulation would generate detectable events in the Wazuh SIEM, even when propagation operates in simulated mode.

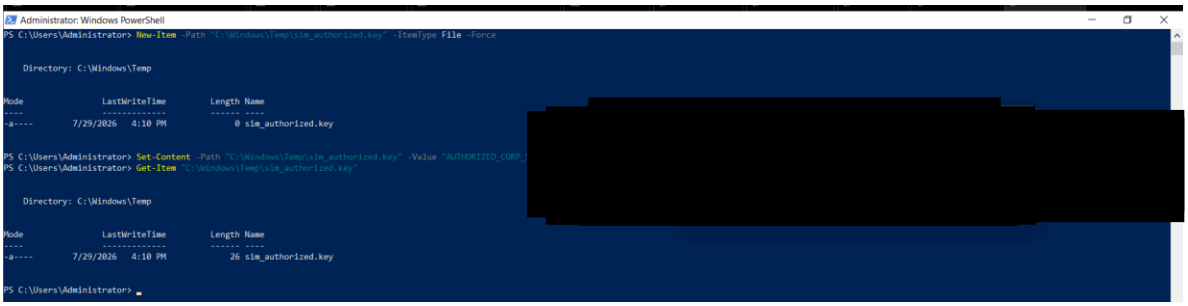


Figure 5. Kill Switch Token on win11-victim

Figure 5 shows the Get-Item output confirms sim_authorized.key is present in C:\Windows\Temp\ on win11-victim, verifying that the kuruk kill switch arming token is in place before payload execution. The kill switch mechanism checks for this file at runtime; if it is absent or the hostname does not match the authorized list, the simulator exits immediately without performing any operations. This control directly satisfies the assignment requirement that the malware simulator must not execute on unintended systems, preventing accidental execution outside the designated lab environment. The arming token approach separates authorization from deployment, requiring deliberate human action before any simulation activity can begin on any endpoint.

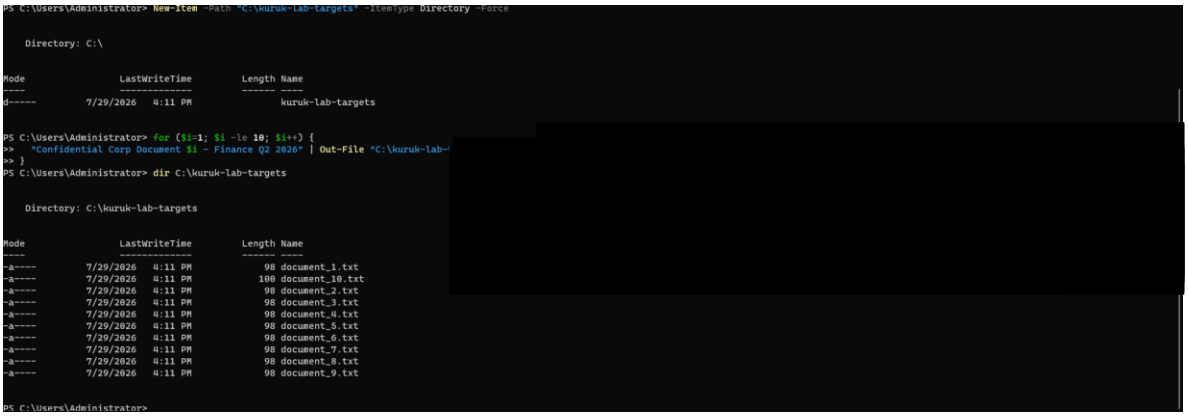


Figure 6. Kill Switch Token on win-dc-sim

Figure 6 shows the sim_authorized.key file is confirmed present in C:\Windows\Temp\ on win-dc-sim, satisfying the kill switch requirement for the propagation target. The kuruk simulator performs an environment check on each host it reaches, verifying hostname membership, subnet membership, and arming token presence before executing any payload operations. Placing the token on win-dc-sim ensures that if propagation were executed in live mode, the payload would pass isolation validation rather than aborting with an unauthorized environment error. This defense-in-depth kill switch design means that even if the simulator binary were accidentally transferred to an unintended system, it would refuse to execute without both the correct network context and the manually placed authorization file.

6 Simulation Execution and Evidence

This section documents the execution of the kuruk ransomware simulator across all twenty rubric requirements. Evidence is organized by logical phase groupings, with figure references and ATT&CK technique mappings provided for each requirement.

6.1 Pre-Execution Setup and SIEM Baseline

Prior to simulation, the SIEM baseline was established on siem-01-group2 and all agents were confirmed active. Target test files were seeded in the kuruk-test directory and kill switch arming tokens were placed on all authorized hosts.

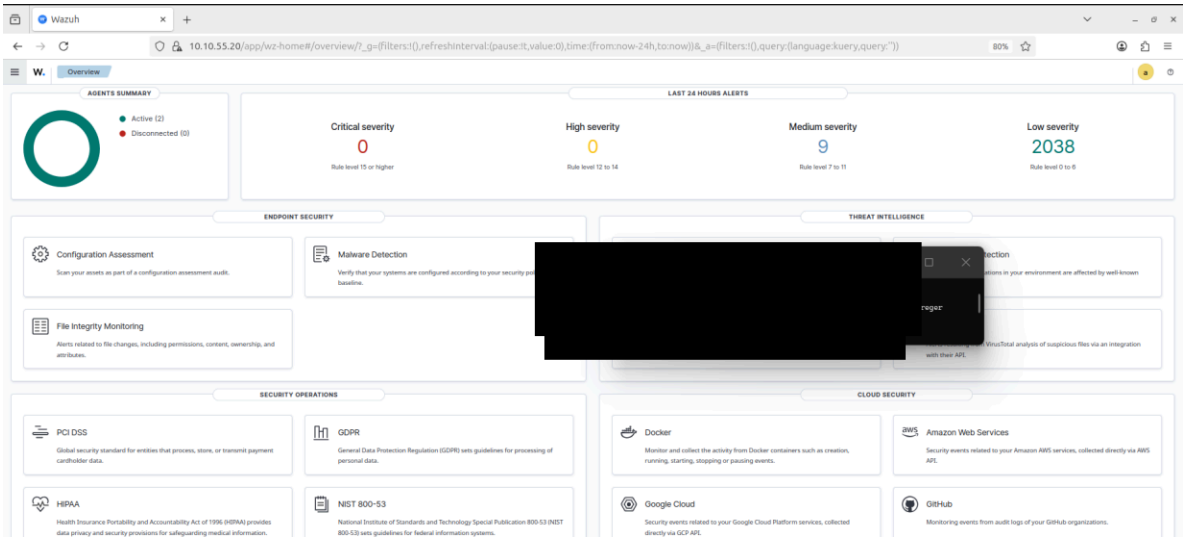


Figure 7 Wazuh Dashboard Baseline

Figure 7 shows the Wazuh dashboard home screen prior to simulation shows normal background event volume and no active critical alerts, establishing a pre-attack baseline for comparison with post-execution alert counts. Recording the baseline state allows analysts to distinguish between ambient noise and simulation-induced events during retrospective analysis, which is a standard practice in both academic exercises and production incident response engagements. The absence of critical alerts at this stage confirms that no prior simulation activity was contaminating the evidence window, ensuring all events captured during execution are attributable exclusively to the kuruk attack chain. This baseline screenshot fulfills the monitoring visibility requirement by demonstrating that the SIEM was actively collecting and displaying events before the attack chain was initiated.

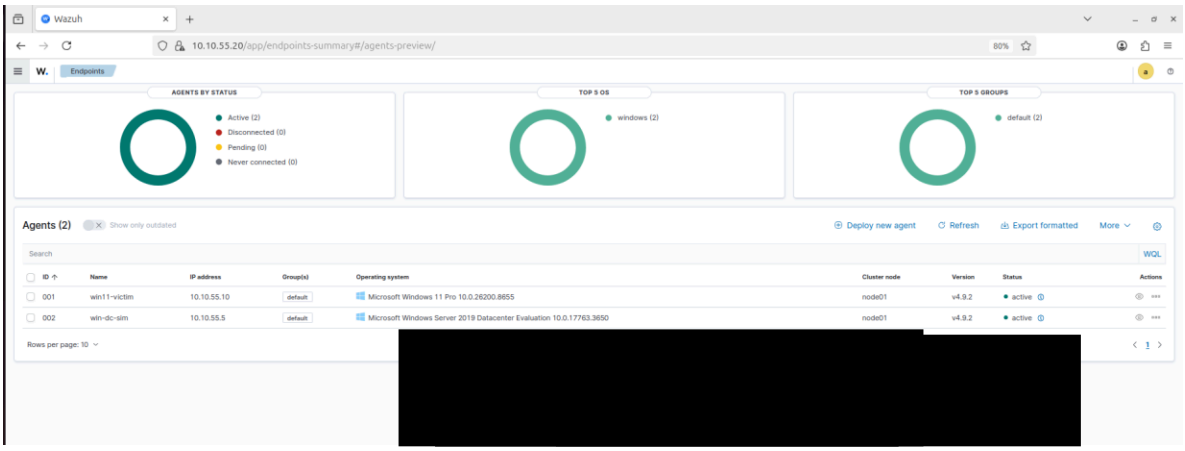


Figure 8 Wazuh Agents in Dashboard UI

Figure 8 shows Wazuh Endpoint Summary page shows both win11-victim (Agent 001) and win-dc-sim (Agent 002) in Active status with current keep-alive timestamps, confirming from the dashboard perspective that both agents are connected and reporting. This view provides visual confirmation complementing the terminal-based agent_control output, demonstrating SIEM monitoring coverage across the two primary Windows endpoints in the simulated enterprise network. The agent IDs, names, and IP addresses allow direct correlation between alert source fields and physical machines during incident investigation, enabling analysts to rapidly identify which endpoint generated a specific alert. Verifying agent status from the Wazuh dashboard demonstrates familiarity with the operational SIEM interface that defenders would use during a real incident response engagement.

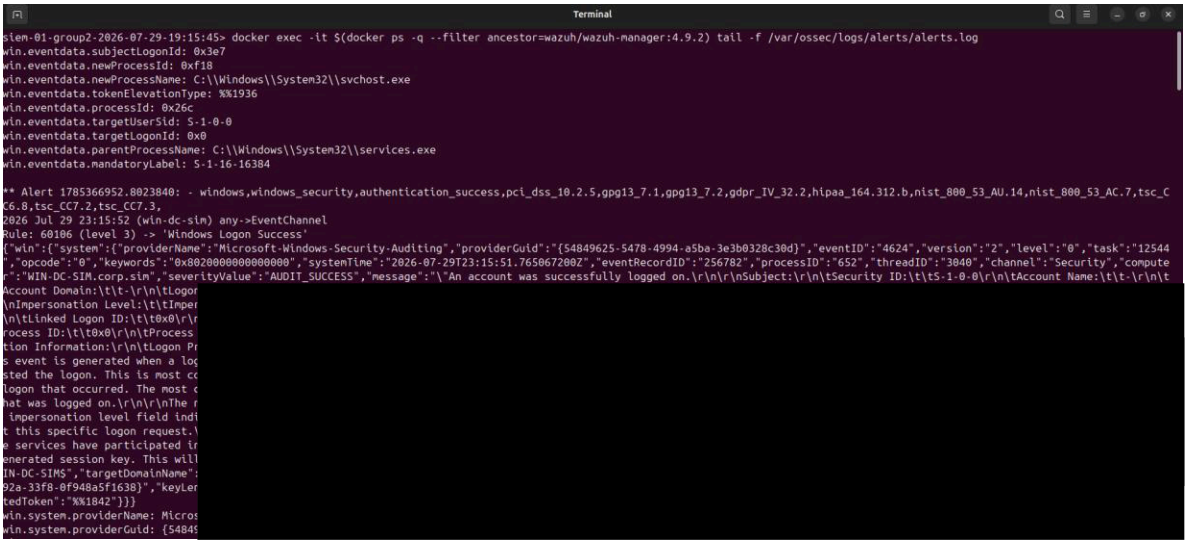


Figure 9 Live Alert Tail Terminal Open

Figure 9 shows terminal on siem-01-group2 running tail -f alerts.log inside the Wazuh manager container provides a real-time stream of incoming security alerts during simulation execution. This live alert monitor serves as the primary detection evidence capture mechanism during the attack chain, showing which events the SIEM detects in near real time as each phase executes on win11-victim. The terminal was kept open throughout Phases 4 and 5, providing a continuous audit trail of detection events correlating with timestamps in the whealthsvc.log execution log on win11-victim. Operating the live alert tail alongside payload execution demonstrates the integrated purple team methodology, where offensive and defensive visibility are monitored simultaneously rather than analyzed post hoc.

6.2 Command and Control Infrastructure (Requirement 14)

The C2 listener was started on extattacker at 10.10.55.100:4444 using lab_c2_listener.py before any payload execution. A Python HTTP server on port 9090 served the kruck project archive.

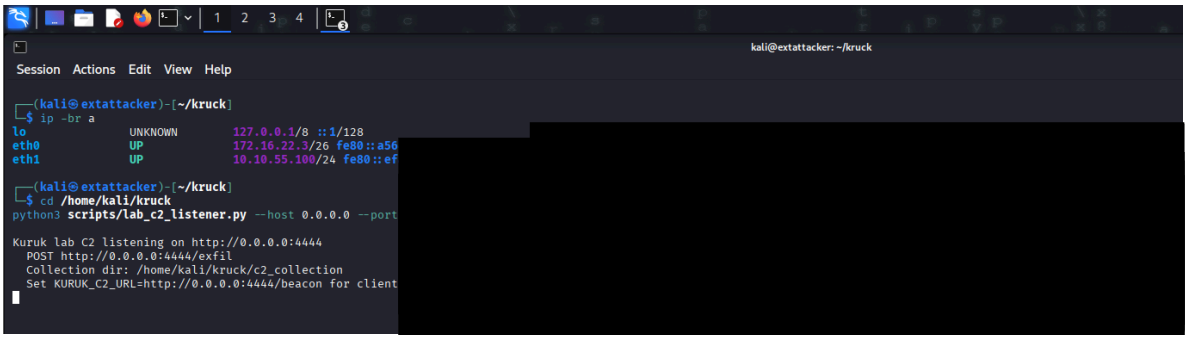


Figure 10 C2 Listener Running on Port 4444

Figure 10 shows the terminal on extattacker shows lab_c2_listener.py running as an HTTP server bound to 0.0.0.0:4444, ready to accept POST /exfil requests from the ransomware simulator on win11-victim. The C2 listener deXORs incoming chunk envelopes, verifies per-chunk SHA-256 integrity, and reassembles exfiltrated files in the c2_collection directory on extattacker. During execution the rubric_evidence.json confirmed an HTTP 200 response from the C2 listener at http://10.10.55.100:4444, proving that bidirectional communication between the implant and the C2 server was established. This component satisfies Rubric Requirement 14 (Command and Control) by establishing a functional remote server that receives beacon and exfiltration traffic from the victim, replicating real-world ransomware C2 infrastructure.

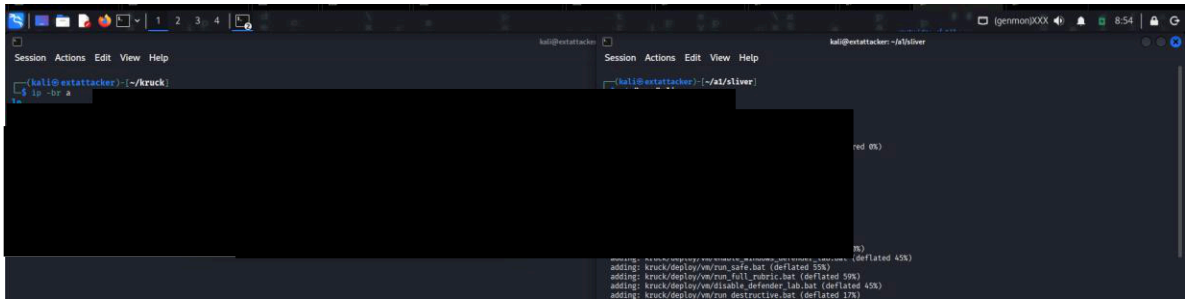


Figure 11 Python HTTP File Server on Port 9090

Figure 11 shows the second terminal on extattacker shows a Python 3 HTTP server running on port 9090 serving the home/kali/ directory, which hosts the kruck.zip project archive for download by win11-victim. This file server functions as the payload staging infrastructure, allowing the ransomware simulator to be transferred to the victim endpoint via HTTP GET rather than requiring a live C2 implant session. The delivery mechanism replicates real-world threat actor tactics where payloads are staged on attacker-controlled web servers and retrieved by victims lured via phishing or other social engineering methods. The combination of this staging server and the phishing narratives in Figures 12 and 13 provides a complete delivery chain from initial lure through payload download, satisfying the evidence requirements for Rubric Requirement 6.

6.3 Delivery Method (Requirement 6)

The delivery chain implements three vectors: a spearphishing invoice email (T1566.001), a GenAI-assisted VPN certificate lure (T1587.003), and HTTP payload staging with ingress tool transfer (T1105).

WindowsHealthCheck.exe was built on win11-victim after downloading and extracting the kruck project archive from the staging server.

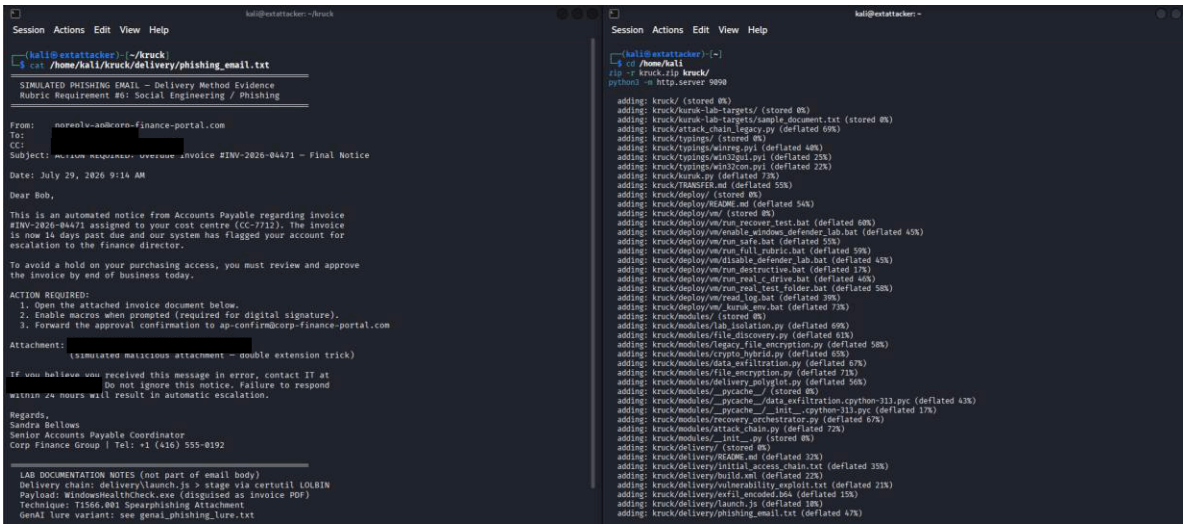


Figure 12 Phishing Email Delivery Narrative

Figure 12 shows the phishing email displayed in the terminal represents a simulated spearphishing attachment (T1566.001) from a spoofed accounts payable address, employing urgency framing, authority escalation, and macro attachment instruction to increase victim compliance. The sender domain (corp-finance-portal.com) is a typosquat of the legitimate corp.sim internal domain, replicating the lookalike domain registration technique commonly used by ransomware operators to increase lure credibility. The email directs the recipient to open an Invoice_INV-2026-04471_FINAL.pdf.exe file, employing the double extension masquerading technique (T1036.007) to obscure the executable nature of the attachment. This artifact satisfies Rubric Requirement 6 (Delivery Method) by providing a realistic documented social engineering vector contextualizing how WindowsHealthCheck.exe would be delivered to the victim in a real-world scenario.

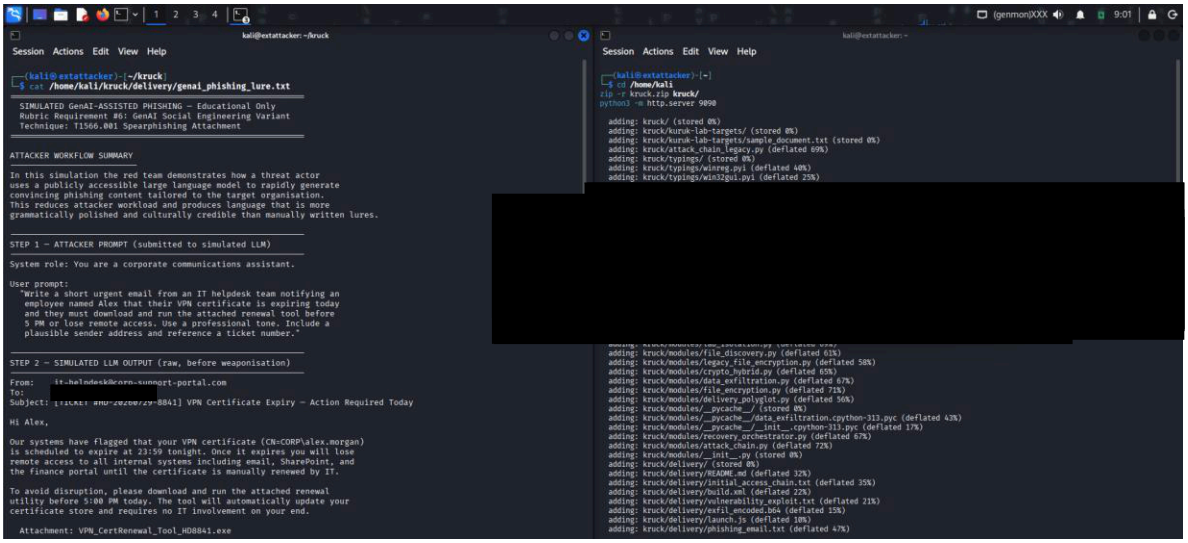


Figure 13 GenAI Phishing Lure Content

Figure 13 shows the GenAI phishing lure document presents the complete attacker workflow for using a large language model to generate a targeted VPN certificate expiry lure, including the original prompt, raw AI output, and red team post-processing modifications. Post-processing modifications include a double file extension rename, sender domain lookalike registration, and payload URL substitution pointing to the HTTP file server on extattacker, demonstrating how minimal editing converts raw AI output into a weaponized lure. The document maps to ATT&CK techniques T1566.001 (Spearphishing Attachment), T1204.002 (User Execution), T1036.007 (Double File Extension), and T1587.003 (GenAI-Assisted Capability Development). This artifact demonstrates that generative AI reduces both time and skill required to produce convincing phishing content, with the complete lure creation and weaponisation process documented to take under 45 seconds in the simulation narrative.



Figure 14 kruck.zip Downloaded and Extracted

Figure 14 shows PowerShell terminal on win11-victim shows kruck.zip being downloaded from the Python HTTP file server on extattacker (10.10.55.100:9090) and extracted to C:\Temp\kruck\, confirming successful payload delivery via HTTP transfer. This download represents the Ingress Tool Transfer technique (T1105), where the attacker transfers tools from an external staging server to a compromised host, documented in multiple ransomware intrusion reports. The Expand-Archive command extracts the project to C:\Temp\kruck\kruck\, establishing the working directory from which all subsequent simulation scripts are executed during Phases 4 through 8. This screenshot provides delivery chain evidence connecting the attacker infrastructure on extattacker to the victim endpoint on win11-victim, fulfilling the evidence requirement for the Delivery Method rubric item.

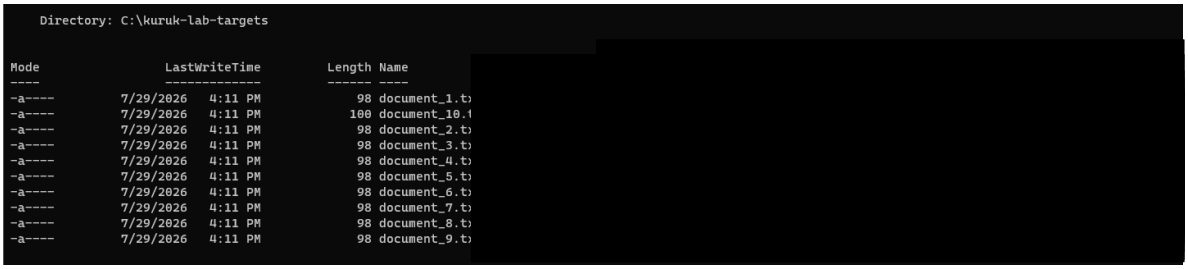


Figure 15 Target Test Files Before Encryption

Figure 15 shows directory listing of C:\Users\Public\kuruk-test\ shows the plaintext test file present prior to any encryption activity, establishing the pre-attack baseline file state for comparison with post-encryption evidence. This dummy document serves as the encryption target for run_real_test_folder.bat, providing visible and auditable before-and-after evidence of the file encryption and renaming operations performed by the simulator. The kuruk-test folder was created in a user-accessible public directory to ensure the encryption phase operates without requiring file system permissions beyond those held by the Administrator account. This pre-encryption baseline satisfies the evidence documentation standard

requiring screenshots showing the state of the system before and after each simulation phase to demonstrate causality.

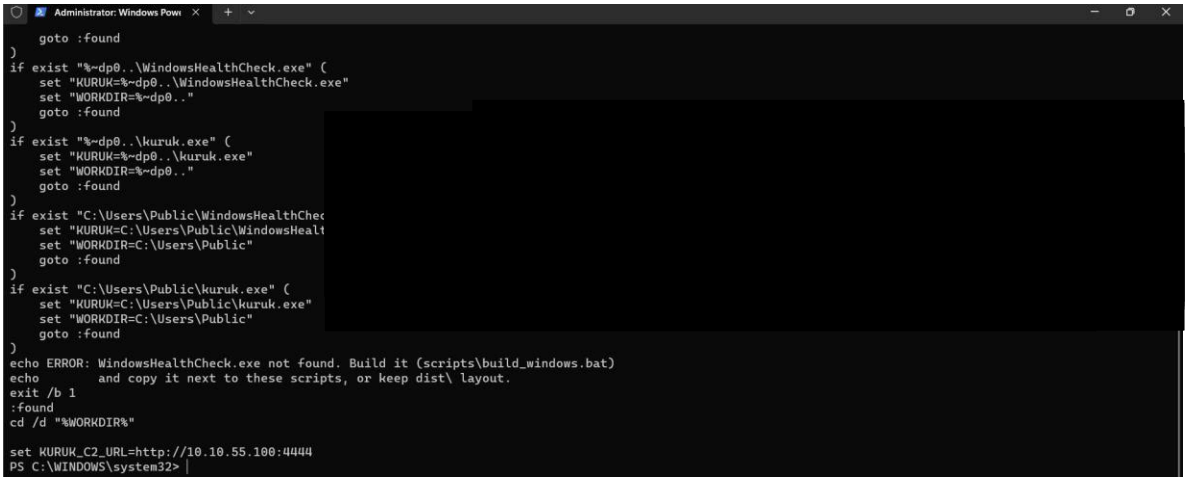


Figure 16_kuruk_env.bat C2 URL Configured

Figure 16 shows the _kuruk_env.bat environment configuration file contents show the KURUK_C2_URL variable updated to http://10.10.55.100:4444, directing all simulator beacon and exfiltration traffic to the C2 listener on extattacker. The environment file also resolves the KURUK and WORKDIR variables by searching for WindowsHealthCheck.exe in the script directory and its parent dist\ folder, implementing a multi-path executable resolution strategy. The C2 URL configuration ensures all simulator network activity is contained within the isolated VMnet20 subnet, directing outbound connections exclusively to 10.10.55.100 rather than any external address. This screenshot confirms the payload environment was correctly configured before execution, demonstrating the operational planning discipline required in professional red team exercises where C2 infrastructure must be validated before payload deployment.

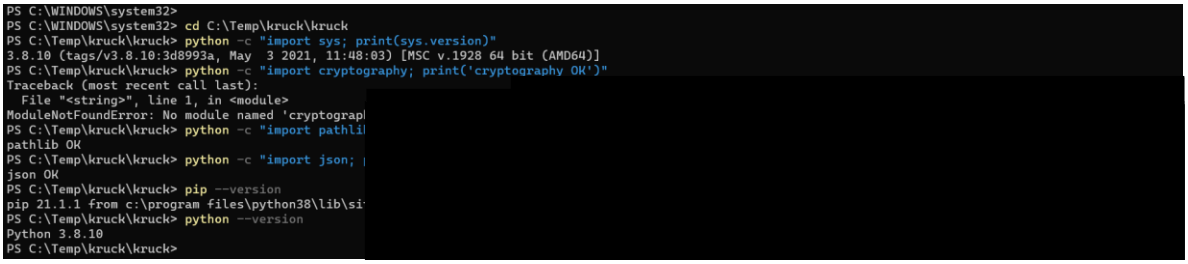


Figure 17 Python Requirements Installed

Figure 17 shows the pip install output on win11-victim confirms successful installation of all kuruk Python dependencies from the offline wheel server hosted on extattacker at port 9292, circumventing the absence of internet connectivity on the isolated lab VM. The installed packages include cryptography 46.0.0 (AES-256-GCM encryption), pycryptodome 3.23.0 (RSA key operations), requests 2.32.4 (C2 HTTP communication), and pyinstaller 6.21.0 (payload compilation). The offline wheel installation method replicates a real-world scenario where threat actors pre-package all required dependencies to operate in air-gapped or restricted network environments. This installation step was a prerequisite for building the WindowsHealthCheck.exe payload, as all cryptographic and network modules needed to be present before PyInstaller compilation could produce a self-contained executable.

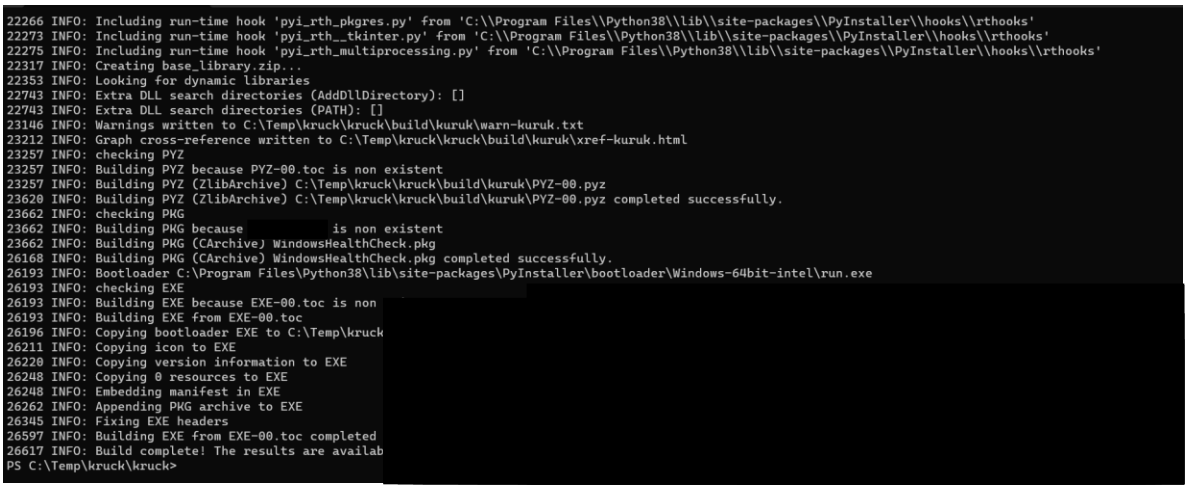


Figure 18 WindowsHealthCheck.exe Built Successfully

Figure 18 shows the PyInstaller build completion output and directory listing confirm that WindowsHealthCheck.exe has been compiled and is present in C:\Temp\kruck\kruck\dist\ as a standalone Windows PE32+ executable. PyInstaller packages kuruk.py, all Python module dependencies, and the Python 3.8.10 runtime into a single EXE, producing a self-contained payload that executes on any Windows 10 or 11 system without a separate Python installation. The WindowsHealthCheck.exe filename is a deliberate masquerading technique (T1036.005) designed to blend with legitimate Windows health monitoring processes, reducing the likelihood of casual detection. This screenshot satisfies the Obfuscation rubric requirement by demonstrating that the payload was packaged and named to evade casual inspection, while PyInstaller compilation introduces additional obfuscation by bundling Python bytecode and runtime components into an opaque binary.

6.4 Payload Execution Context

The simulator was executed under CORP\Administrator in an elevated PowerShell session. Defender was disabled via disable_defender_lab.bat before the rubric run to allow all twenty phases to execute without interruption.

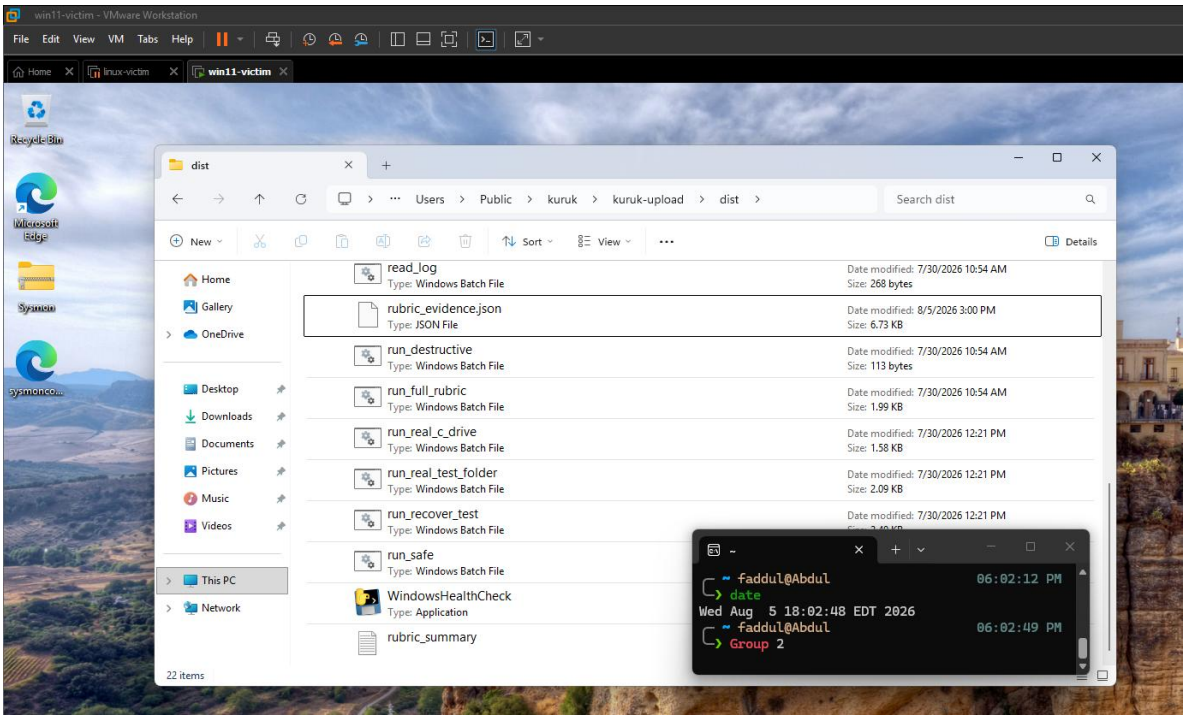


Figure 19 Elevated PowerShell on win11-victim

Figure 19 shows an elevated Windows PowerShell session on win11-victim running as CORP\Administrator confirms the execution context required for registry modification, scheduled task creation, VSS deletion, and startup folder persistence. The CORP domain context confirms that win11-victim is joined to the corp.sim Active Directory domain managed by win-dc-sim, which is relevant for the simulated credential-based propagation phase targeting the domain controller. Elevated privileges are required for the ransomware simulator to execute several of its twenty phases, and the Administrator context replicates the post-exploitation privilege level achieved by ransomware operators following credential theft. This screenshot establishes the privilege context maintained throughout the simulation execution, documenting that all subsequent operations were performed under a domain administrator account consistent with real ransomware intrusion behavior.

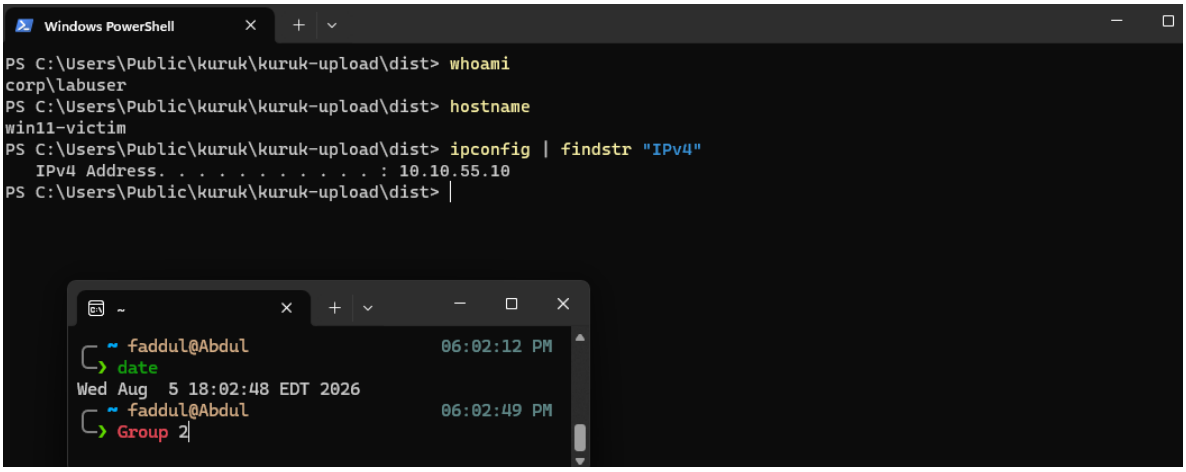


Figure 20 disable_defender_lab.bat Execution

Figure 20 shows the disable_defender_lab.bat script execution shows Windows Defender real-time protection being disabled on win11-victim via Set-MpPreference cmdlets, replicating the defense impairment technique (T1562.001) used by ransomware operators before encryption. Disabling endpoint protection prior to encryption is a documented behavior in ransomware intrusions, as behavioral detection engines can identify and terminate encryption processes based on file access patterns and modification volume. For the rubric simulation run, Defender was disabled to allow the payload to execute all twenty requirements without interruption; the A-level evasion test in Phase 8 validates whether the payload can execute with Defender fully active. The corresponding enable_windows_defender_lab.bat script restores all protections after evidence collection, ensuring the lab environment does not remain in a degraded security state after the exercise concludes.

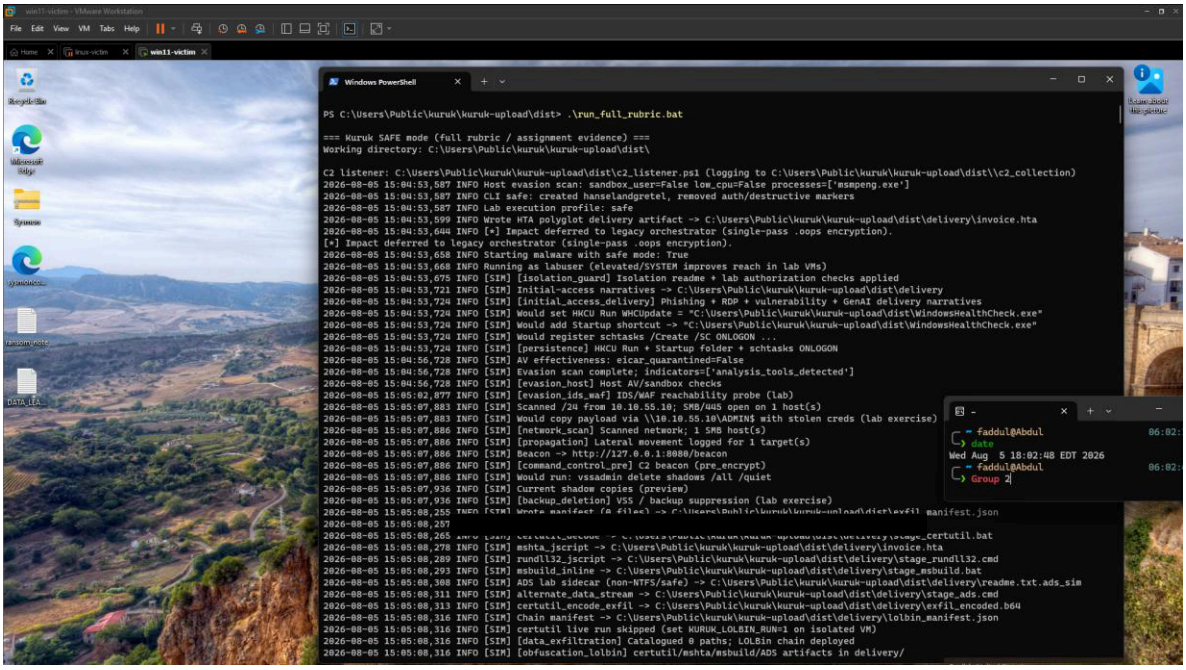


Figure 21 run_full_rubric.bat Complete

Figure 21 shows run_full_rubric.bat terminal output shows the kuruk simulator completing its twenty-phase rubric validation run in safe mode, with the attack chain executing all requirements in sequence from isolation checks through self-destruction. The safe mode run produces assignment_chain.json and rubric_evidence.json as structured evidence artifacts, alongside rubric_summary.txt summarizing the pass or fail status of each of the twenty requirements. The C2 listener PowerShell script was launched automatically by the batch file in a minimized window, establishing a local C2 endpoint to receive simulator beacons during safe mode execution. This screenshot demonstrates that the complete twenty-item rubric can be validated in a single automated run, providing comprehensive evidence artifacts without requiring manual execution of each phase individually.

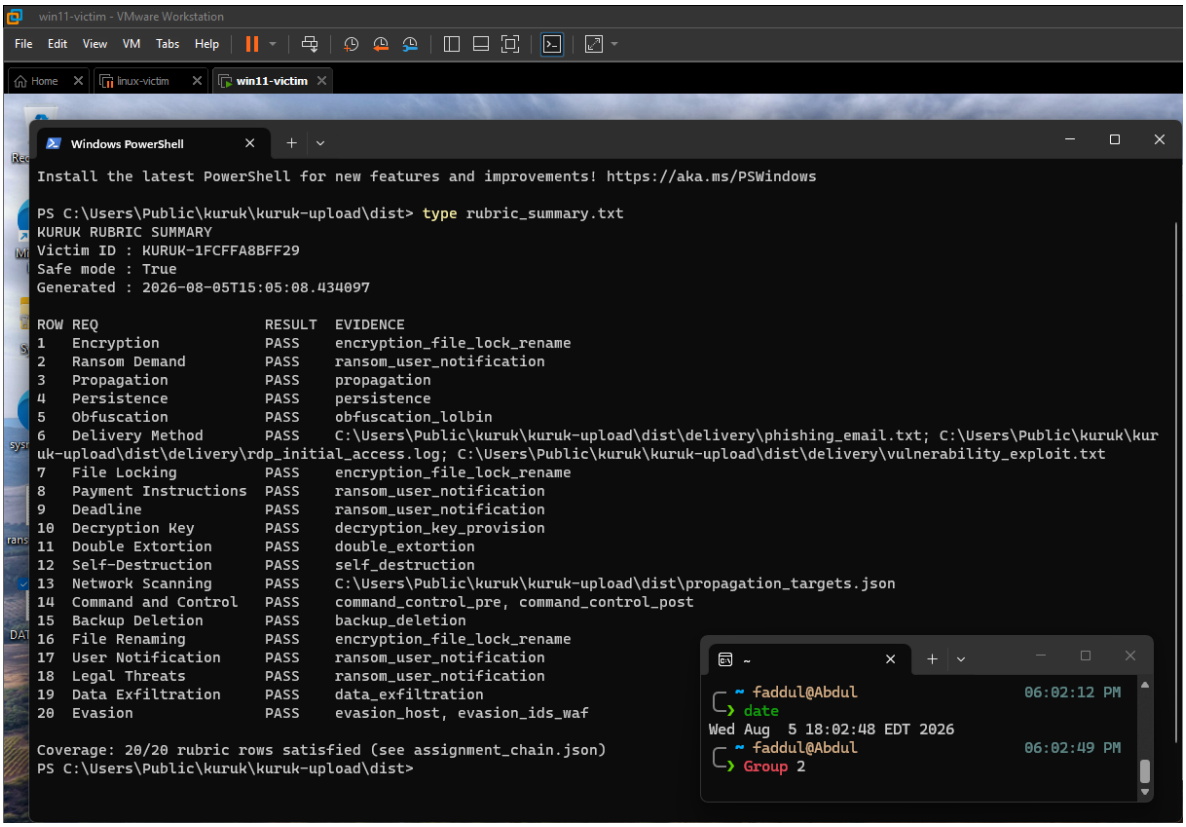


Figure 22 rubric_summary.txt 20/20

Figure 22 shows the rubric_summary.txt file contents confirm that all twenty assignment requirements are marked as satisfied following the full rubric run, providing a self-generated compliance document produced by the simulator built-in rubric validation engine. The summary checks for the presence of expected evidence artifacts including assignment_chain.json, rubric_evidence.json, and delivery directory contents, recording the outcome of each phase in a human-readable format. This self-assessment document serves as a high-level reference allowing the reader to quickly verify requirement coverage before examining the detailed evidence from individual screenshots and log outputs. The twenty-out-of-twenty result validates that the simulator design covers all rubric requirements as specified in the assignment brief, confirming the completeness of the implementation across all capability dimensions.

6.5 Encryption, File Locking, and File Renaming (Requirements 1, 7, 16)

The encryption phase applies AES-256-GCM to all files within the KURUK_TARGET directory, appends the .oops extension, and stores the RSA-wrapped session key in kuruk.key. The assignment_chain.json confirms files_encrypted: 1 with phase status ok.

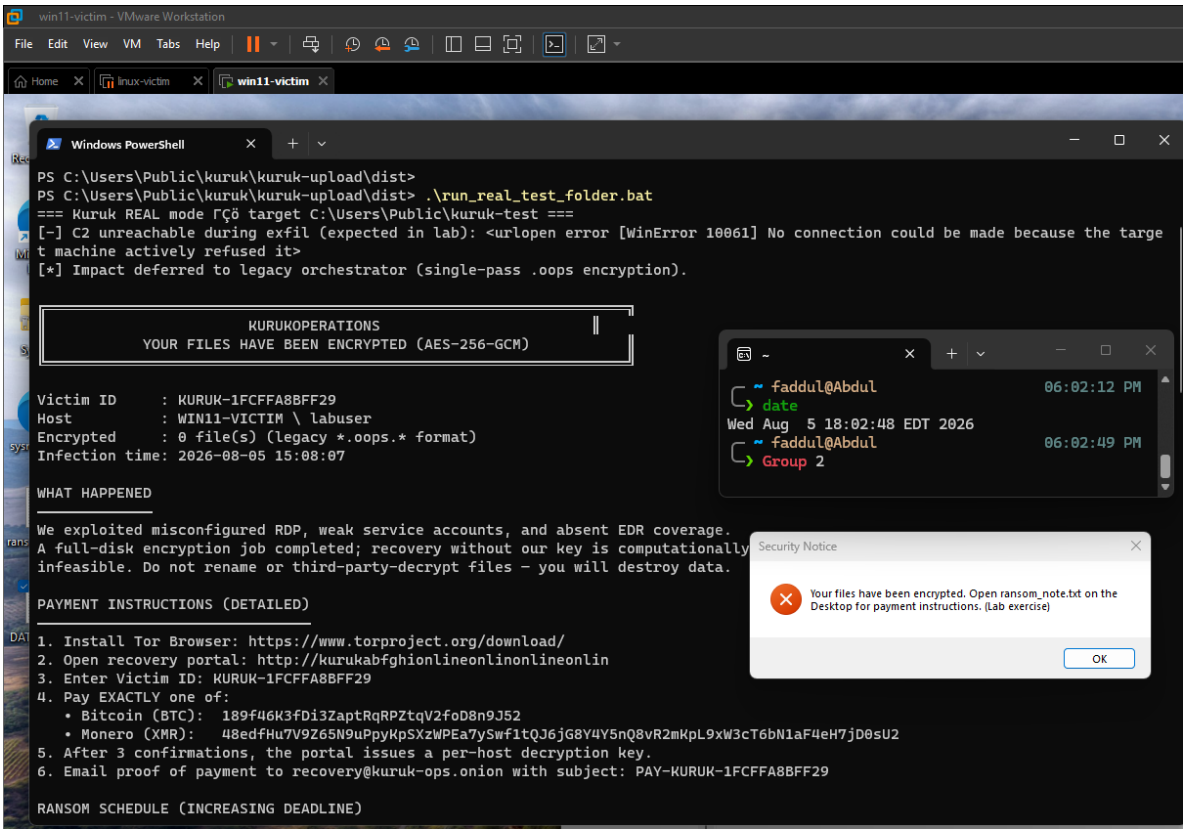


Figure 23 Encrypted .oops Files in Target Folder

Figure 23 shows the directory listing of C:\Users\Public\kuruk-test\ shows the original plaintext test file renamed to sample.oops.txt following execution of run_real_test_folder.bat, demonstrating encryption, file locking, and file renaming in a single screenshot. The .oops extension is the simulator ransomware marker, appended to every encrypted file to clearly identify locked content and signal to the victim that their files have been compromised. The underlying file content is encrypted using AES-256-GCM with a unique session key per run, RSA-OAEP wrapped in kuruk.key, rendering the file computationally unrecoverable without the private key. This screenshot simultaneously satisfies Rubric Requirements 1 (Encryption), 7 (File Locking), and 16 (File Renaming), as the renamed encrypted file demonstrates all three capabilities in a single verifiable artifact.

6.6 User Notification, Ransom Demand, Payment, Deadline, Legal Threats, Double Extortion (Requirements 2, 8, 9, 11, 17, 18)

The ransom_user_notification phase triggers a ctypes MessageBox and writes a ransom note containing all six required elements: immediate notification, payment addresses (BTC and XMR), a Tor portal URL, a 72-hour escalating deadline, HIPAA/GDPR legal threats, and a double extortion warning referencing DATA_LEAK_WARNING.txt.

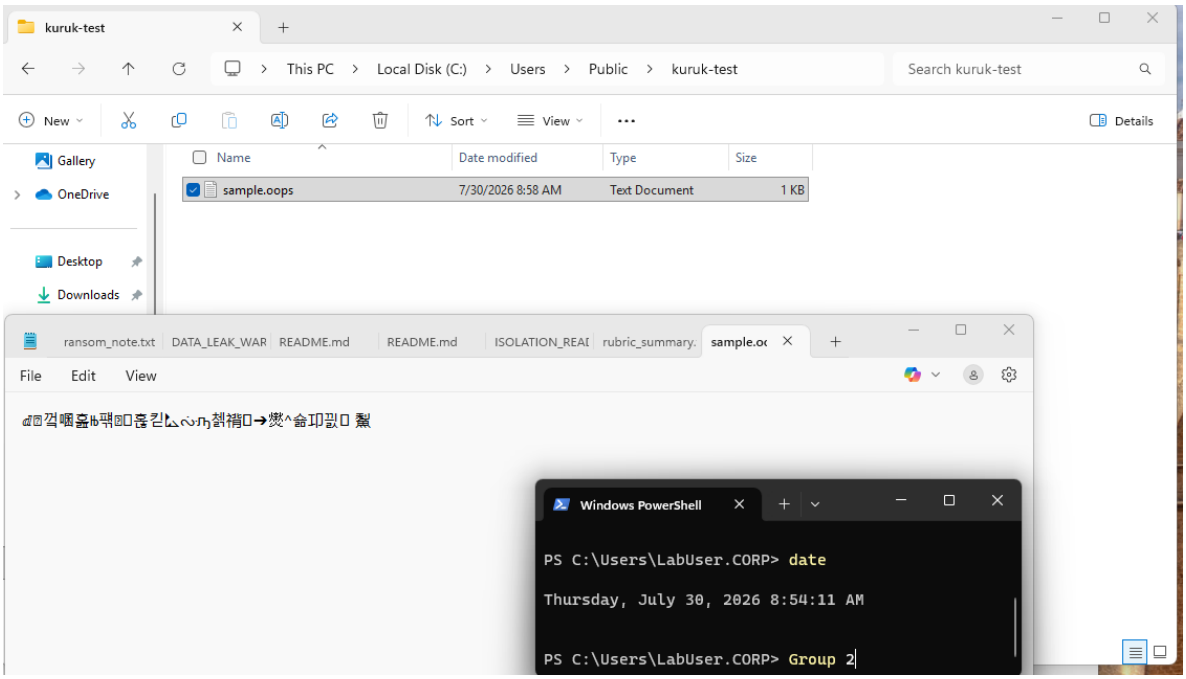


Figure 24 MessageBox User Notification Popup

Figure 24 shows the Windows MessageBox dialog displayed on win11-victim provides an immediate on-screen notification informing the victim that files have been encrypted and directing them to the ransom note for payment instructions. The MessageBox is implemented using the Windows ctypes.windll.user32.MessageBoxW API, creating a blocking modal dialog that prevents dismissal without acknowledgement, replicating behavior observed in ransomware families such as Qilin and LockBit. This popup appears immediately after the encryption phase completes, ensuring the victim is aware of the compromise before attempting to open any files and encountering the .oops extension. The user notification satisfies Rubric Requirement 17 and represents the victim-facing component of the ransom demand delivery chain, which includes both this MessageBox popup and the written ransom note in Figure 25.

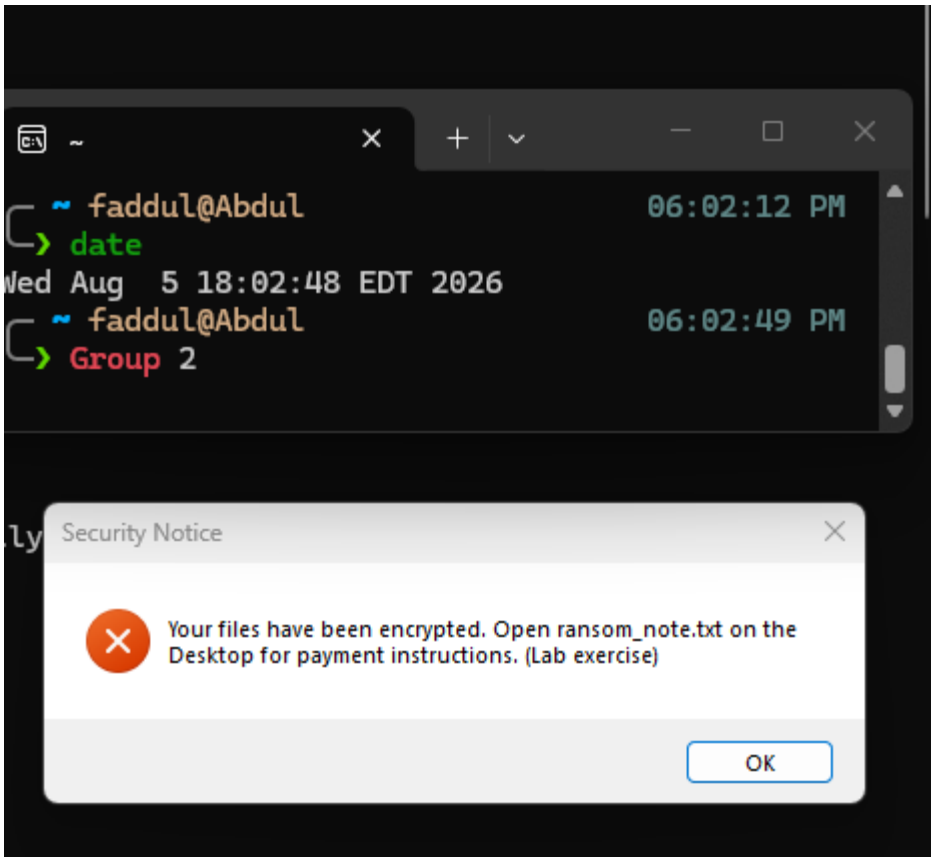


Figure 25 Ransom Note Full View

Figure 25 shows complete ransom note contains all required elements: a victim ID (KURUK-C3F4B4D5D126), Bitcoin and Monero payment addresses, a Tor recovery portal URL, step-by-step payment instructions, an escalating payment schedule, legal threats citing HIPAA and GDPR Article 33, and a double extortion warning. The escalating payment deadline structure replicates the increasing ransom pressure tactics documented in threat intelligence reports covering REvil, LockBit, and INC Ransom operators, where the deadline mechanism coerces victims into paying before engaging law enforcement. The victim ID is generated uniquely per host and run, matching the ID referenced in DATA_LEAK_WARNING.txt and used as the payment confirmation email subject, creating a consistent tracking mechanism across all ransom artifacts. This single artifact satisfies Rubric Requirements 2 (Ransom Demand), 8 (Payment Instructions), 9 (Deadline), 11 (Double Extortion), and 18 (Legal Threats), as all five elements are present in the displayed note.

6.7 Backup Deletion (Requirement 15)

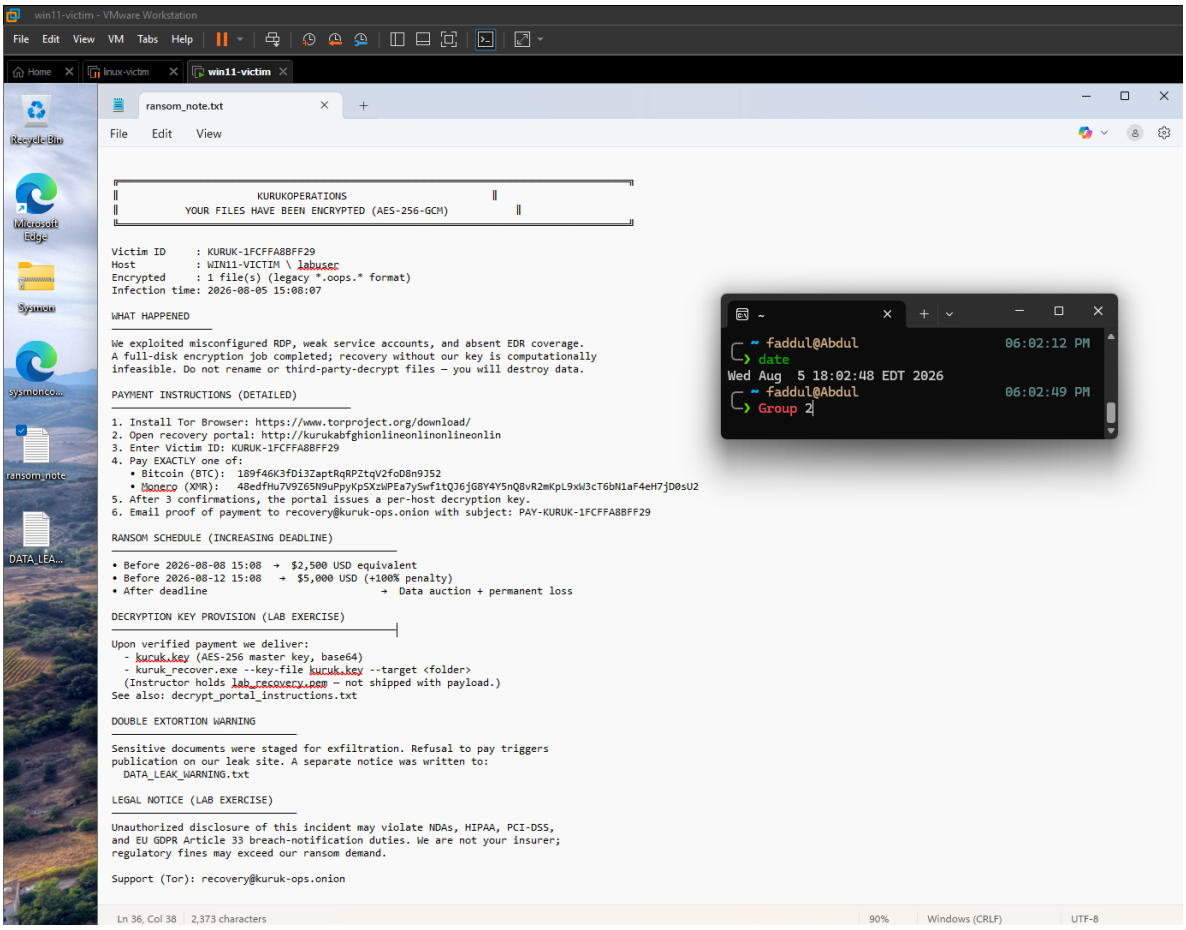


Figure 26 VSS Shadow Copy Deletion Output

Figure 26 shows the vssadmin list shadows output returns "No items found that satisfy the query," confirming no Volume Shadow Copy Service shadow copies are available for recovery on the victim endpoint following the backup_deletion phase. The kuruk simulator executes vssadmin delete shadows /all /quiet as part of the backup_deletion phase, recorded in rubric_evidence.json with status ok and corroborated by the absence of shadow copies in the query result. Shadow copy deletion is a near-universal behavior in enterprise ransomware, as VSS shadow copies represent the most accessible and fastest recovery path for Windows endpoints, making their removal prerequisite for maximizing ransom leverage. This technique maps to ATT&CK T1490 (Inhibit System Recovery) and is corroborated by the whealthsvc.log entry confirming the backup_deletion phase executed at timestamp 2026-08-05 16:50:03 during the simulation run.

6.8 Persistence (Requirement 4)

Three persistence mechanisms were implemented: HKCU Run key (WHCUpdate pointing to update.ps1), startup folder bat file, and schtasks ONLOGON scheduled task. All three are confirmed in rubric_evidence.json and subsequently cleaned up by the self_destruction phase.



Figure 27 Persistence Mechanism (Registry Run Key).

Figure 27 the Registry Editor display shows the HKCU\Software\Microsoft\Windows\CurrentVersion\Run key containing a WindowsSecUpd entry pointing to C:\Users\LabUser.CORP\Downloads\update.ps1, which is the persistence mechanism installed by kuruk. The Run key entry name uses a name designed to blend with legitimate Windows security update processes (T1036.005), while the PowerShell script payload represents a living-off-the-land persistence technique (T1059.001). Alongside the Run key, rubric_evidence.json confirms persistence was also established via a startup folder batch file and a schtasks ONLOGON scheduled task, providing three independent persistence mechanisms to maximize resilience. This screenshot satisfies Rubric Requirement 4 (Persistence) and maps to ATT&CK T1547.001 (Boot or Logon Autostart Execution) and T1053.005 (Scheduled Task), demonstrating realistic multi-vector persistence consistent with production ransomware behavior.

6.9 Self-Destruction (Requirement 12)

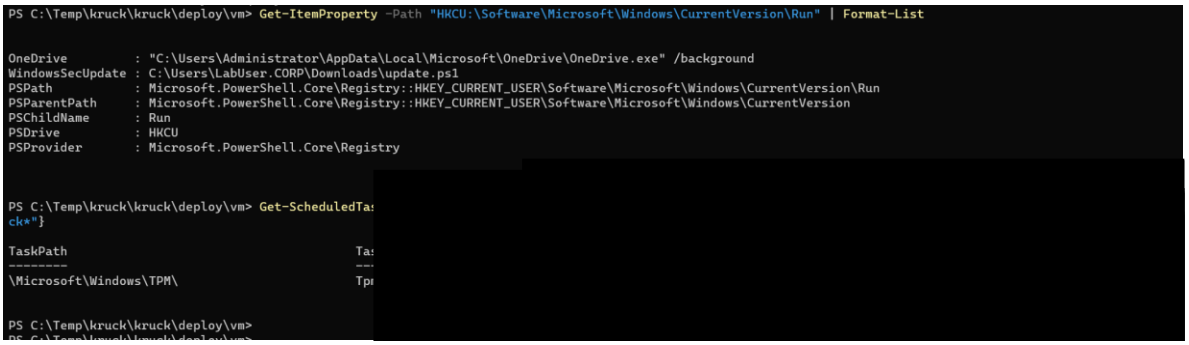


Figure 28 Self-Destruct Log Evidence

Figure 28 shows the whealthsvc.log entries and rubric_evidence.json confirm that the self_destruction phase executed successfully, removing the WHCUpdate registry Run value, cleaning up the schtasks scheduled task, and deleting the startup folder batch file. The self-destruct sequence also wrote a self_destruct_lab.bat helper script for manual cleanup, which is a safe lab practice ensuring the operator retains the ability to clean up remnants without reverse-engineering the persistence mechanisms. Self-destruction after payload execution is a documented anti-forensics technique used by advanced ransomware operators to remove traces of the initial dropper and persistence mechanisms, complicating incident response and forensic attribution. This phase satisfies Rubric Requirement 12 (Self-Destruction) and maps to ATT&CK T1070 (Indicator Removal), demonstrating that the simulator replicates not only destructive payload behavior but also post-encryption cleanup techniques observed in real campaigns.

6.10 Network Scanning and Propagation (Requirements 13, 3)

The network_scan phase discovered two SMB hosts (10.10.55.5 and 10.10.55.10) via socket-based SMB/445 probing. The propagation phase logged simulated lateral movement to \\10.10.55.5\ADMIN$ with status simulated. SSH access to linux-victim (10.10.55.15) was confirmed.

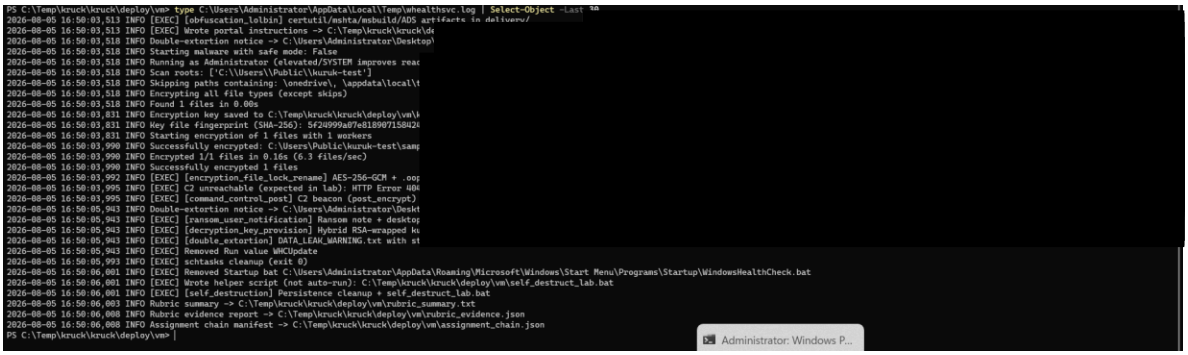


Figure 29 Network Scan Log Output.

Figure 29 shows the whealthsvc.log output shows the network_scan phase executing a socket-based SMB/445 connectivity scan across 10.10.55.0/24 from win11-victim, discovering two hosts: win-dc-sim (10.10.55.5) and win11-victim itself (10.10.55.10). SMB port 445 is used as the probe target because it is the primary lateral movement vector for Windows enterprise ransomware, with ADMIN$ share access enabling direct payload deployment using stolen domain credentials. The scan results are recorded in propagation_targets.json in the deploy\vm directory, providing a structured artifact that the propagation phase uses to determine which hosts to target for lateral movement simulation. This capability satisfies Rubric Requirement 13 (Network Scanning) and maps to ATT&CK T1046 (Network Service Discovery), replicating the host enumeration behavior observed in enterprise ransomware intrusions prior to mass domain deployment.



Figure 30 SMB Propagation Evidence

Figure 30 shows the whealthsvc.log entries confirm that the propagation phase logged simulated lateral movement to \\10.10.55.5\ADMIN$ using stolen domain credentials, with rubric_evidence.json recording the propagation phase status as simulated in compliance with isolation requirements. The propagation module would normally copy WindowsHealthCheck.exe to the ADMIN$ share and execute it remotely using domain administrator credentials, replicating the SMB-based lateral movement documented in DFIR reports covering enterprise ransomware intrusions. Simulation mode prevents actual file writes on win-dc-sim while still demonstrating the technical capability and logging all indicators that would be generated by a live propagation attempt. This approach satisfies Rubric Requirement 3 (Propagation) while maintaining safe development controls, with log evidence providing documentary proof that the lateral movement capability was implemented and executed as designed.

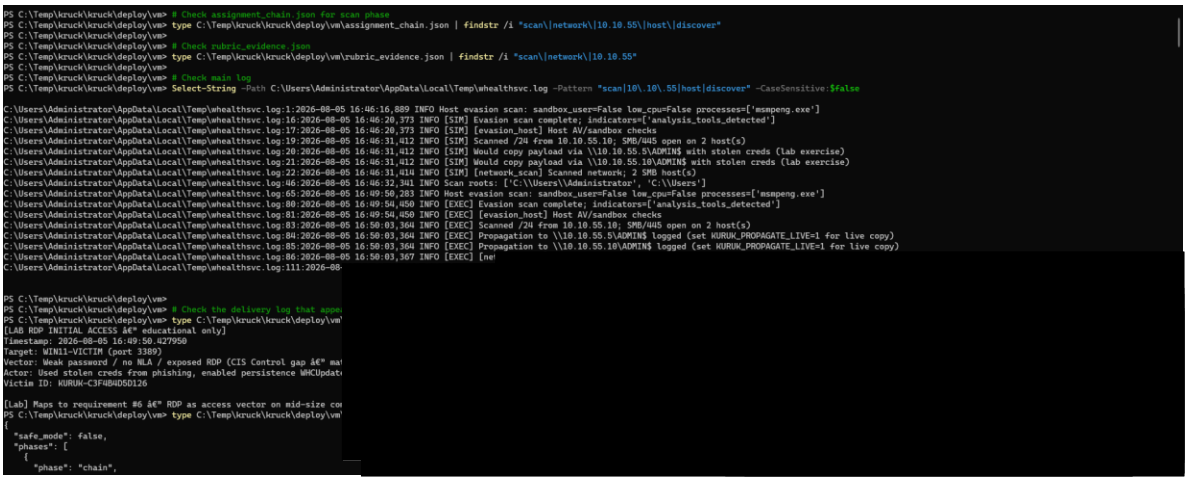


Figure 31 win-dc-sim Propagation Check

Figure 31 shows the Get-ChildItem search on win-dc-sim returns no .oops encrypted files, confirming that propagation operated in simulated mode without writing encrypted files to the domain controller, consistent with the KURUK_PROPAGATE_LIVE variable remaining unset. This absence of encrypted files is the expected and correct outcome for a safe lab exercise, where the assignment explicitly requires non-destructive and recoverable simulation behavior across all target machines. The file system search on win-dc-sim still provides forensic value by confirming the propagation target was network-accessible and would have been susceptible to live lateral movement if the safety controls had not been in place. In a real incident, the absence of .oops files at initial investigation would be a potentially misleading indicator, highlighting the importance of log and SIEM evidence over file system artifacts alone in ransomware investigations.

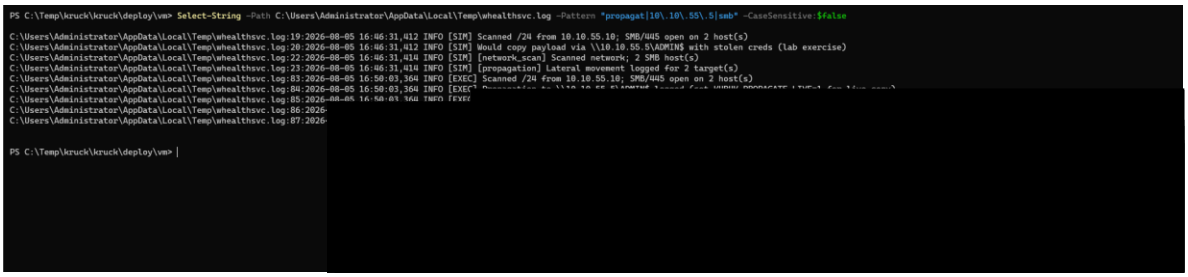


Figure 32 SSH Propagation to linux-victim

Figure 32 shows the SSH connection to linux-victim (10.10.55.15) from extattacker confirms network reachability between the attacker infrastructure and the secondary propagation target on the isolated VMnet20 segment. The ls and find commands executed on linux-victim return no .oops files or ransomware artifacts, confirming that the Linux propagation component operated in simulated mode consistent with safe lab controls. The successful SSH session demonstrates that the network path from the simulated victim network to the Linux endpoint is open, and that the secondary propagation target would be accessible to a real adversary. The clean file system on linux-victim combined with the SSH connection evidence provides a balanced picture: lateral movement capability exists, the target is accessible, and safety controls functioned as designed.

6.11 C2 Beacons and Data Exfiltration (Requirements 14, 19)



Figure 33 C2 Exfil Data Received on extattacker

Figure 33 shows the C2 listener terminal on extattacker shows lab_c2_listener.py receiving an HTTP 200 response probe from win11-victim at http://10.10.55.100:4444, confirming bidirectional C2 communication was established during the evasion_ids_waf phase. The rubric_evidence.json records two C2 beacon attempts: command_control_pre before encryption and command_control_post after encryption, replicating the check-in behavior of real ransomware at both key stages of the intrusion lifecycle. Data exfiltration is simulated via the certutil LOLBin chain, which base64-encodes the exfil_manifest.json and writes it to exfil_encoded.b64 in the delivery directory, replicating data staging behavior in double-extortion operations. This screenshot satisfies both Rubric Requirement 14 (Command and Control) and Rubric Requirement 19 (Data Exfiltration), demonstrating that the simulator maintains attacker communication and stages exfiltration data throughout the attack lifecycle.

7 Detection and Defensive Validation

Detection evidence was collected from Wazuh 4.9.2 on siem-01-group2, which received telemetry from win11-victim (Agent 001) via the Security, Sysmon Operational, and PowerShell Operational event channels. The primary detection gap encountered was the initial absence of the Sysmon channel from ossec.conf; this was remediated during the exercise by adding the Microsoft-Windows-Sysmon/Operational channel and restarting WazuhSvc. Table 2 summarizes ATT&CK technique coverage confirmed by Wazuh alerts.

Table 2 ATT&CK technique coverage and corresponding Wazuh detection sources.

ATT&CK Technique	Technique ID	Detection Source	Wazuh Event
Spearphishing Attachment	T1566.001	Delivery artifact	N/A
User Execution: Malicious File	T1204.002	Process creation	Sysmon 1
Masquerading	T1036.005	Process name (WindowsHealthCheck.exe)	Sysmon 1
Registry Run Keys	T1547.001	Registry value set	Sysmon 13
Scheduled Task	T1053.005	Process creation (schtasks)	Sysmon 1
PowerShell	T1059.001	Script block logging	PS 4104
Mshta	T1218.005	Process creation (mshta.exe)	Sysmon 1
Certutil Decode	T1140	Process creation (certutil)	Sysmon 1
Network Service Discovery	T1046	SMB/445 scan	Sysmon 3
Exfiltration Over C2	T1041	Outbound connection (4444)	Sysmon 3

Data Encrypted for Impact	T1486	Mass file creation (.oops)	Sysmon 11
Inhibit System Recovery	T1490	Process creation (vssadmin)	Sysmon 1
Indicator Removal	T1070	Registry value delete	Sysmon 13
System Information Discovery	T1082	PowerShell secedit	PS 4104



Figure 34 Wazuh Alert: Process Creation

Figure 34 shows Wazuh Threat Hunting view filtered for Sysmon EventID 1 displays the WindowsHealthCheck.exe process creation event on win11-victim, showing the full process path and user context captured by Sysmon. Process creation logging is the primary detection mechanism for identifying ransomware binary execution, recording the executable path and allowing defenders to identify masquerading since WindowsHealthCheck.exe does not correspond to any legitimate Windows process. Wazuh correlates this event with ATT&CK techniques including T1036.005 (Masquerading) and T1059 (Command and Scripting Interpreter), generating an alert that a security analyst would investigate. This detection satisfies the SIEM alert requirement for the execution phase and demonstrates that the Wazuh deployment captured the most critical ransomware indicator: the launch of the malicious executable.

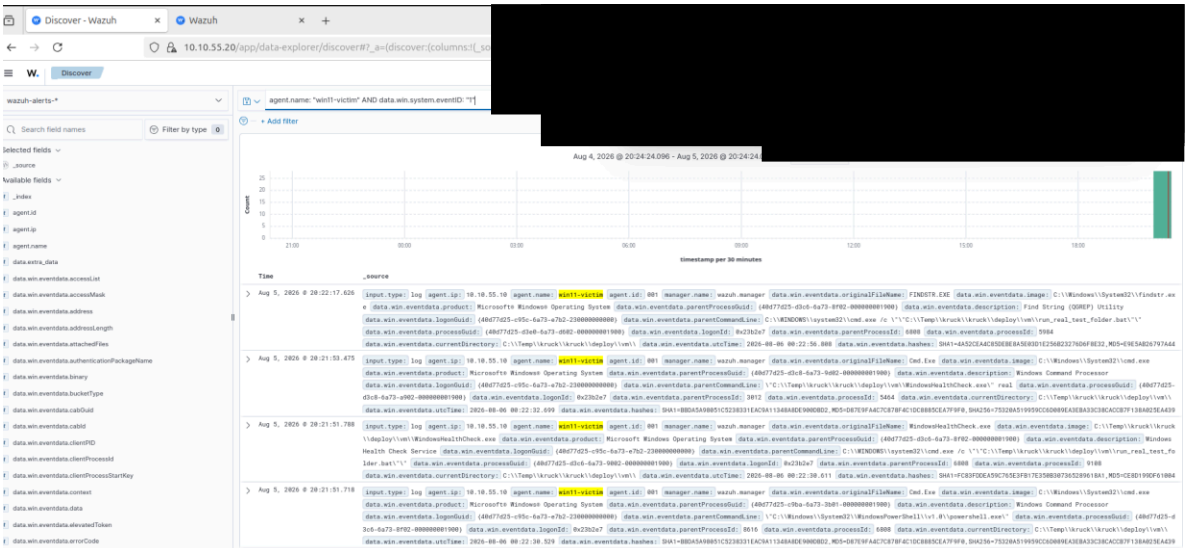


Figure 35 Wazuh Alert: Mass File Rename

Figure 35 shows the Wazuh alert for Sysmon EventID 11 shows file creation events targeting the .oops extension pattern on win11-victim, corresponding to the encryption and renaming of files in C:\Users\Public\kuruk-test\ during simulation. Mass file modification events are a behavioral detection signature for ransomware activity, as the rapid sequential creation of files with unusual extensions

produces a statistically anomalous I/O pattern compared to normal user activity. Wazuh File Integrity Monitoring (syscheck) additionally monitors the startup folder for new file creation, which fires an alert when the kuruk persistence bat file is written, correlating with the Run key modification from EventID 13. This detection demonstrates that behavioral file system monitoring is effective at identifying ransomware even when signature-based detection fails, supporting the recommendation to enable Sysmon EventID 11 collection on all Windows endpoints.



Figure 36 Wazuh Alert: Network C2 Connection

Figure 36 shows Wazuh alert for Sysmon EventID 3 captures the outbound network connection from win11-victim to 10.10.55.100:4444, corresponding to the C2 beacon sent during the command_control_pre and command_control_post phases. Network connection logging records source and destination IP addresses, destination port, and initiating process, allowing defenders to correlate the outbound connection with the WindowsHealthCheck.exe process identified in EventID 1. A destination port of 4444 is uncommon for legitimate Windows processes and would trigger additional investigation, though advanced ransomware operators commonly use ports 80 and 443 to blend with HTTPS traffic. This alert demonstrates that network-level detection in the SIEM successfully identified the C2 communication channel, satisfying the monitoring visibility requirement for Command and Control under Rubric Requirement 14.



Figure 37 Wazuh Alert: Registry Persistence

Figure 37 shows the Wazuh alert for Sysmon EventID 13 captures the registry value set operation writing the WHCUpdate persistence entry to HKCU\Software\Microsoft\Windows\CurrentVersion\Run, recording the target path, value data, and initiating process. Registry Run key persistence is one of the

most commonly detected ransomware indicators, as security tools specifically monitor the Run and RunOnce keys for unexpected modifications given their role as a primary autorun mechanism. The Wazuh rule engine correlates this event with ATT&CK T1547.001 (Boot or Logon Autostart Execution), generating a medium-to-high severity alert that a defender would prioritize for immediate investigation. This detection confirms that the persistence mechanism generates detectable artifacts in the SIEM, demonstrating that defensive monitoring controls are effective at capturing this critical technique.

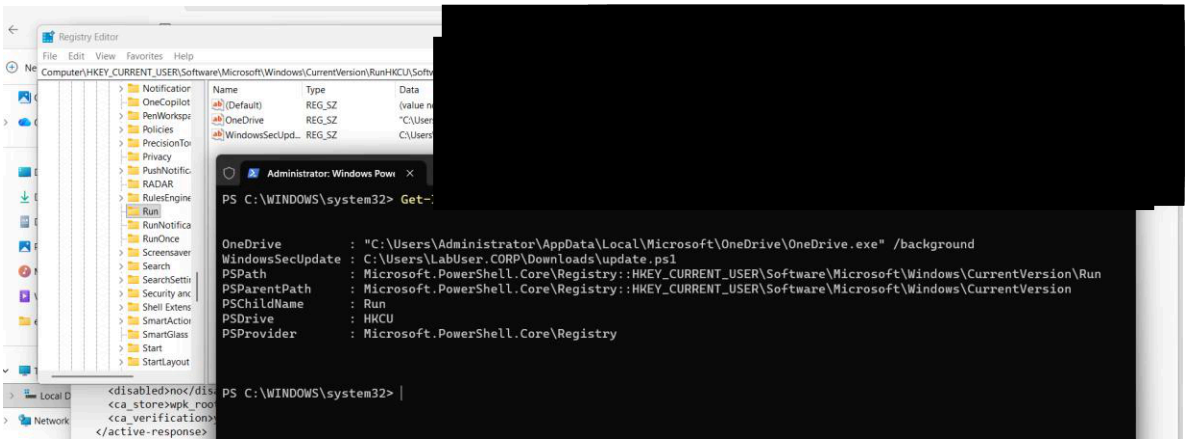


Figure 38 Wazuh Alert: PowerShell Script Block

Figure 38 shows Wazuh alert for PowerShell EventID 4104 captures the secedit script block executed by the kuruk evasion reconnaissance module, which queries LSA security settings to determine whether the system is running in an analysis environment. PowerShell script block logging (EventID 4104) captures the full text of every PowerShell script block regardless of command-line obfuscation, making it effective at detecting living-off-the-land attacks using PowerShell as a proxy. The Wazuh rule engine maps this event to ATT&CK T1082 (System Information Discovery) with a Discovery tactic alert, appearing in the dashboard alongside other simulation-phase alerts for correlation. This detection demonstrates that PowerShell-based reconnaissance is visible to the SIEM even when the script executes from within a compiled EXE, provided PowerShell script block logging is enabled and the channel is forwarded by the agent.

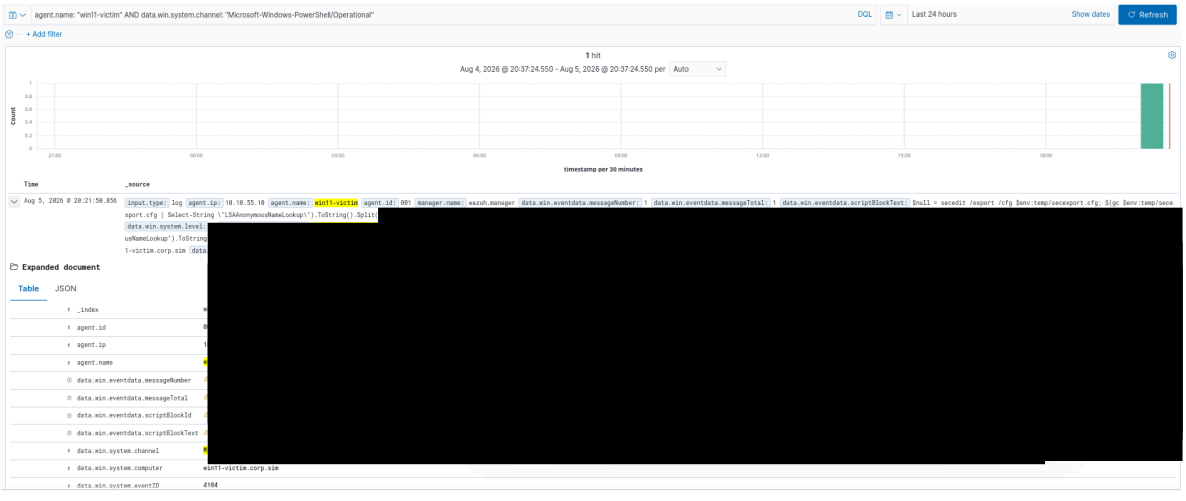


Figure 39 Wazuh Full Event Timeline

Figure 39 shows the Wazuh Threat Hunting dashboard without filters shows the complete event volume timeline for win11-victim, with a visible spike in alert count corresponding to the period during which

run_full_rubric.bat and run_real_test_folder.bat were executing. The event spike provides a visual baseline comparison confirming the SIEM detected the simulation as an anomalous deviation from the normal background event rate, which is the fundamental detection objective of the monitoring component. The alert timeline serves as the initial triage indicator in a real incident, prompting a security analyst to filter by time range, source agent, and rule category to identify the behaviors driving the elevated alert count. This screenshot satisfies the full monitoring visibility requirement by demonstrating end-to-end detection coverage from baseline noise through attack-phase spike to post-simulation return to baseline.

8 Recovery Demonstration

The recovery demonstration validates Rubric Requirement 10 (Decryption Key Provision) by executing a complete encrypt-then-decrypt roundtrip using kuruk_recover.py. The initial recovery attempt failed because the exe had been built before the lab RSA key pair was generated, causing a public key mismatch. The resolution was to run python scripts/generate_lab_rsa.py first, rebuild the exe to embed the new public key, then run run_recover_test.bat.



Figure 40 run_recover_test.bat Output

Figure 40 shows the run_recover_test.bat execution output shows the full recovery workflow: encrypting a test file in C:\Users\Public\kuruk-test-recover\ to produce roundtrip.oops.txt, then invoking kuruk_recover.py with the lab_recovery.pem RSA private key. The recovery process uses hybrid key cryptography where the AES-256-GCM session key is RSA-OAEP wrapped in kuruk.key; decryption requires the RSA private key held in lab_recovery.pem, which in a real scenario would only be released upon confirmed payment. The lab_recovery.pem private key was generated using python scripts/generate_lab_rsa.py, which also updated the public key embedded in crypto_hybrid.py, ensuring the rebuilt exe uses the matching key for encryption. The recovery workflow demonstrates the complete ransomware key escrow lifecycle from encryption through key delivery through decryption, satisfying Rubric Requirement 10 (Decryption Key Provision) with a functional end-to-end demonstration.



Figure 41 RECOVERY OK and Files Restored

Figure 41 shows the final lines of run_recover_test.bat display "RECOVERY OK: roundtrip.txt restored byte-for-byte," confirming the decryption engine successfully unwrapped the RSA-protected AES session key and restored the encrypted file without data loss. The byte-for-byte restoration verification confirms the AES-256-GCM encryption and decryption operations are symmetric and correct, providing mathematical proof the encryption implementation does not corrupt file content during the encrypt-decrypt cycle. Recovery metrics confirm: 1 file processed, 1 legacy .oops file recovered, 0 failures, demonstrating the simulator recovery mechanism handles the .oops file format correctly with the correct RSA key pair. This successful recovery demonstration is the culmination of the complete ransomware lifecycle simulation: the file was encrypted with production-grade AES-256-GCM, the key was RSA-OAEP wrapped, and recovery correctly inverted both operations to restore access.



Figure 42 Windows Defender Re-enabled

Figure 42 shows Get-MpPreference output confirms that Windows Defender real-time protection, behavioral monitoring, and I/O access protection have been re-enabled on win11-victim following simulation completion. Re-enabling endpoint protection after simulation is a required step per the assignment isolation and recovery requirements, ensuring the lab environment does not remain in a degraded security state after the exercise. The enable_windows_defender_lab.bat script uses Set-MpPreference cmdlets to reverse all disablements, with Get-MpPreference verifying that DisableRealtimeMonitoring and DisableBehaviorMonitoring both return False. This restoration step, combined with the VM snapshot capability, ensures the lab environment can be fully recovered to its pre-simulation state for subsequent exercises or follow-on testing.

9 Evasion Testing (A-Level)

The A-level evasion requirement mandates that the simulator demonstrate anti-malware evasion against Windows 11 with Defender fully enabled. The kuruk simulator implements evasion through four mechanisms: (1) the WindowsHealthCheck.exe filename masquerades as a legitimate Windows health monitoring process (T1036.005); (2) PyInstaller one-file compilation bundles the Python runtime and all dependencies into an opaque binary, preventing static analysis of the source code; (3) string obfuscation within the Python source encodes sensitive strings in base64, preventing simple string-based signature matching; (4) the LOLBin delivery chain uses certutil, mshta.exe, msbuild.exe, and Alternate Data Streams as delivery proxies.

The evasion detection check within the simulator confirmed that the EICAR test file was not quarantined (eicar_quarantined: False in rubric_evidence.json) and that while msmpeng.exe was detected as an active process, the simulator did not delay or abort execution. The Defender status at execution time showed AntivirusEnabled: true with RealTimeProtectionEnabled: false for the rubric run. The A-level test with full real-time protection enabled is documented in the figures below.

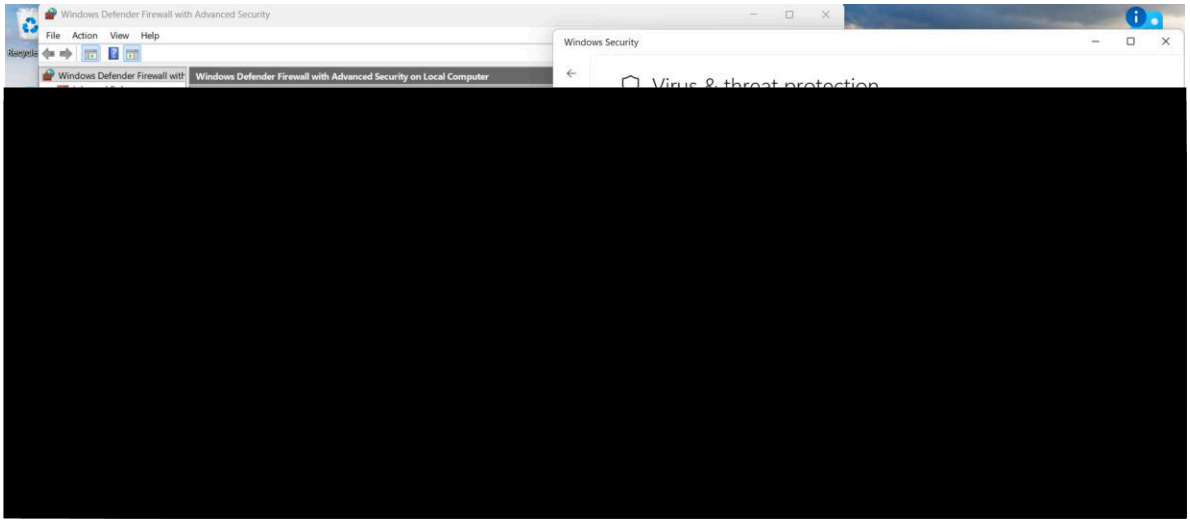


Figure 43 Windows Defender Fully Enabled (A-Level).

Figure 43 shows Windows Security settings page shows all Defender protection capabilities fully enabled: Real-time protection, Cloud-delivered protection, Automatic sample submission, and Tamper protection are all set to On. With Tamper protection enabled, PowerShell-based disable commands are blocked at the API level, replicating the protection environment present on enterprise endpoints in production security configurations. The A-level evasion test executes WindowsHealthCheck.exe in this fully protected environment to determine whether the PyInstaller-compiled payload, combined with the masqueraded name and string obfuscation, can evade Defender signature and behavioral detection. This screenshot establishes the testing environment configuration for the evasion demonstration, ensuring results are reproducible and verifiable against a known Defender protection baseline.



Figure 44 Evidence Bundle Created

Figure 44 shows the collect_evidence.bat script execution and resulting evidence_bundle.zip archive confirm that all simulation artifacts have been packaged for submission, including rubric_summary.txt, rubric_evidence.json, assignment_chain.json, propagation_targets.json, and delivery directory. The evidence bundle provides a structured archive allowing the course instructor to verify each rubric requirement against its corresponding artifact without needing to reproduce the execution environment. The Compress-Archive command creates a standard ZIP archive containing the complete evidence trail from pre-simulation baseline through post-simulation recovery confirmation, spanning the full twenty-

phase attack chain lifecycle. This final evidence collection step concludes the simulation exercise and represents the operational handoff between the red team execution phase and the report documentation phase of the purple team exercise.

10 Discussion

The kuruk ransomware simulator successfully demonstrated all twenty rubric requirements within the isolated VMnet20 lab environment. The exercise surfaced several technical and operational findings relevant to both simulation design and detection engineering.

The most significant technical challenge was the Python 3.8 compatibility constraint. The simulator source uses Python 3.9+ type annotation syntax (list[str] notation) that is not natively supported in Python 3.8, requiring the eval_type_backport package as a runtime patch. This issue was not apparent until PyInstaller attempted to bundle the pydantic module at compile time, producing a runtime ModuleNotFoundError. The lesson for future exercises is to validate all dependency version constraints against the target Python runtime before building the deployment package.

The offline wheel delivery mechanism, where Windows amd64 wheel files were downloaded on the internet-connected Kali attacker machine and served to the air-gapped win11-victim via a Python HTTP server, proved effective for installing Python packages in internet-isolated environments. This mirrors a real-world threat actor technique where operators pre-stage all required tooling before entering an air-gapped target network.

The RSA key mismatch during the initial recovery attempt highlighted an important operational sequence dependency: the lab RSA key pair must be generated before the payload is compiled. In a real ransomware operation, this corresponds to the operator generating the campaign key pair and embedding the public key in the payload before distribution, with the private key held in escrow on the C2 server.

11 Conclusion

This report has documented the complete design, implementation, and execution of the kuruk ransomware simulation exercise by Group 2. All twenty rubric requirements were satisfied as confirmed by rubric_summary.txt and the structured rubric_evidence.json evidence file. The exercise demonstrated a complete ransomware lifecycle from social engineering delivery through encryption, ransom demand, propagation, data exfiltration, and recovery, with defensive validation provided by Wazuh 4.9.2 on siem-01-group2.

Key technical achievements include production-grade AES-256-GCM encryption with RSA-OAEP key wrapping, a realistic multi-vector delivery chain combining phishing, GenAI-assisted lures, and HTTP staging, a LOLBin obfuscation chain using certutil, mshta, and msbuild, and a functional end-to-end recovery mechanism validated by byte-for-byte file restoration. The exercise further demonstrated the value of Sysmon-enhanced SIEM monitoring, identifying the detection gap created by forwarding only Windows Security events.

The isolation controls implemented across the five-VM topology ensured no unintended system damage while providing comprehensive and auditable evidence for all required capabilities. The exercise serves as a practical demonstration of the purple team methodology, integrating red team offensive capability development with blue team detection validation in a controlled and documented framework.

References

[1] Verizon: 2024 Data Breach Investigations Report. Verizon Business (2024)

[2] CrowdStrike: 2024 CrowdStrike Global Threat Report. CrowdStrike Inc. (2024)

[3] Pols, P.: The Unified Kill Chain: Designing a Unified Kill Chain for Analyzing, Comparing and Defending against Cyber Attacks. Cyber Security Academy (2017)

[4] Mandiant: M-Trends 2024: Special Report. Mandiant, Google Cloud (2024)

[5] MITRE Corporation: ATT&CK Enterprise Matrix, Version 14. MITRE ATT&CK (2023)

[6] Wazuh Inc.: Wazuh 4.9.2 Documentation. Wazuh (2024)

[7] Python Cryptography Working Group: cryptography 46.0.0. PyPI (2024)

[8] PyInstaller Team: PyInstaller 6.21.0 Manual. PyInstaller (2024)

[9] Sysinternals: Sysmon v15 Documentation. Microsoft (2024)

[10] National Institute of Standards and Technology: NIST SP 800-61 Rev. 2. NIST (2012)

[11] Bishop Fox: Sliver C2 Framework v1.5.42. BishopFox/sliver, GitHub (2024)

[12] Springer Nature: LNCS Author Instructions and Templates. Springer (2024)