AJT Lab Case Study

Detecting SSH Brute Force Attacks Against Windows 11 with Wazuh

Overview

In this lab, I simulated an attacker brute-forcing SSH credentials against a Windows 11 endpoint and used Wazuh to detect the activity with a custom correlation rule.

Goal

- Simulate SSH brute-force attacks from Kali against a Windows 11 workstation.
- Ingest Windows Security logs into Wazuh.
- Wrap an existing Wazuh rule with an **AJT-branded** local rule to flag possible SSH brute force.
- Investigate and interpret the alerts in the Wazuh Threat Hunting UI the way a SOC analyst would.

Why it matters

This scenario mirrors a very common SOC problem: noisy authentication failures against a remote service. The lab shows I can:

- Design a small but realistic lab network.
- Generate and control attacker traffic.
- Understand how raw events (Windows 4625) flow through a SIEM (Wazuh) and turn into actionable alerts via custom rules.

Lab Topology & Environment

Host machine

- MacBook Pro 13" (2019), Intel i7, 16 GB RAM
- Virtualization: Oracle VirtualBox

VirtualBox network

- Network type: NAT Network
- Name: `labnet`
- IPv4 prefix: `10.0.2.0/24`
- DHCP: Enabled

Virtual machines

Role	OS / Image	Hostname	IP (labnet)	Notes
SIEM	Wazuh OVA 4.14.x	`wazuh-server`	10.0.2.6	Manager, indexer, dashboard
Endpoint	Windows 11	`Calibred1324`	10.0.2.7	OpenSSH server + Wazuh agent
Attacker	Kali Linux	`kali`	10.0.2.5	SSH client + Hydra password cracker

Access to Wazuh dashboard

- From the Mac host: `https://127.0.0.1:5601`
    - Uses the Wazuh OVA's built-in port forward on TCP 5601.

Tools Used

- **Virtualization & OS**
    - Oracle VirtualBox
    - Wazuh OVA 4.14.x
    - Windows 11
    - Kali Linux
- **Security & Detection**
    - Wazuh manager/dashboard (rules, agents, Threat Hunting UI)
    - Wazuh Windows agent
    - `wazuh-logtest` for ruleset validation
- **Attack & Access**
    - Windows OpenSSH Server feature
    - Hydra password cracker (SSH module)
    - SSH client (from Kali to Windows)
- **System & Configuration**
    - Windows Defender Firewall
    - PowerShell / Windows terminal (service and firewall commands)

Steps

1. Design and validate the lab network

- Attached all three VMs (Wazuh OVA, Windows 11, Kali) to the same NAT Network (`labnet`).
- Verified IP addressing on each VM (`ip addr` on Linux, `ipconfig` on Windows).
- Confirmed basic connectivity with `ping` between:
    - Kali ↔ Windows
    - Kali ↔ Wazuh

- This created a simple “company” network:
  - **Wazuh** as the SOC backend.
  - **Windows 11** as the employee workstation.
  - **Kali** as the attacker box.

---

2. Configure the Windows 11 endpoint

Enable OpenSSH Server

- Installed the **OpenSSH Server** Windows feature (Settings → Apps → Optional features).
- Started and configured the SSH service to start automatically.
- Allowed inbound SSH (TCP 22) through Windows Defender Firewall.

Create the target “victim” account

- Created a local user account `bruteuser` with a strong password.
- This account was the target for the brute-force attempts.

Install the Wazuh agent and collect Security events

- From the Wazuh dashboard:
  - Deployed a Windows agent pointing at `10.0.2.6` (Wazuh manager).
  - Installed the agent on the Windows VM and verified the status changed to **Active**.
- Confirmed that Windows **Security** events were being collected by checking `ossec.conf` for an `eventchannel` entry pointing to `Security`.
- Restarted the Wazuh agent service and validated that Security events were visible in the Threat Hunting UI.

---

3. Configure the attacker (Kali) and verify SSH

- Installed **Hydra** on Kali via `apt`.
- Created a small SSH password wordlist (`ssh-passwords.txt`) including:
  - A couple of incorrect passwords.
  - The correct password for `bruteuser`.
- Verified SSH access:
  - From Kali, connected to `ssh bruteuser@10.0.2.7` with the correct password.
  - Confirmed interactive login worked, then logged out.
- This ensured the OpenSSH service, firewall, and account were all configured correctly before generating brute-force traffic.

---

4. Create and validate the custom AJT Wazuh rule

On the Wazuh manager:

- Edited `local_rules.xml` and added a custom rule:
- `<group name="local,">`
- `  <!-- AJT Lab: Detecting Brute Force Attacks on Windows SSH with Wazuh -->`
- `  <rule id="100002" level="10">`
- `    <if_matched_sid>60122</if_matched_sid>`
- `    <description>AJT Lab: Possible SSH brute force against Windows 11 (bruteuser via OpenSSH)</description>`
- `    <group>ajt,windows,authentication_failed,ssh,</group>`
- `  </rule>`
- `</group>`
- This rule **wraps the built-in Wazuh rule 60122** ("Logon Failure – Unknown user or bad password") and tags it as a potential SSH brute-force incident for this lab.
- Validated the ruleset using `wazuh-logtest` with `refresh` to catch syntax or XML errors before restarting the manager.
- Restarted `wazuh-manager` and confirmed rule `100002` appeared in the Wazuh UI under Server management → Rules.

---

5. Generate brute-force traffic from Kali

- From Kali, used Hydra to target the Windows SSH service:
- `hydra -l bruteuser -P ssh-passwords.txt ssh://10.0.2.7`
- Hydra attempted multiple logins for `bruteuser` using the wordlist:
  - Several incorrect passwords (expected failures).
  - One correct password (eventual success).
- On Windows, each failed attempt generated Security event **4625 – An account failed to log on**.
- Wazuh decoded these events, mapped them to **rule 60122**, which in turn triggered my custom **rule 100002**.

---

6. Investigate the alerts in Wazuh

- In the Wazuh dashboard:
  - Opened **Threat intelligence → Threat Hunting → Events**.
  - Time filter: Last 15 minutes.
  - Filters:
    - `agent.name : Calibred1324`
    - `rule.id : 100002`
- This surfaced the SSH brute-force events tagged by my custom rule.

- Drilled into the JSON for a sample event and confirmed:
  - `data.win.system.eventID = 4625`
  - `data.win.eventdata.targetUserName = bruteuser`
  - `data.win.eventdata.processName = C:\\Windows\\System32\\OpenSSH\\sshd.exe`
  - `agent.name = Calibred1324`
  - `agent.ip = 10.0.2.7`
- Together, these fields tell a clear story:
  - **What**: repeated failed SSH logons
  - **Who**: `bruteuser`
  - **Where**: Windows 11 endpoint `Calibred1324`
  - **How**: via OpenSSH (`sshd.exe`) over the network from Kali

Findings

- The **end-to-end detection pipeline worked**:
  - Hydra brute-force attempts → Windows Security 4625 events → Wazuh rule 60122 → custom rule 100002.
- The **alerts are actionable**:
  - They identify the affected user, host, process, and nature of activity.
- The **Wazuh Threat Hunting UI supports real analyst workflows**:
  - Filtering by agent, rule ID, and event fields makes it easy to reconstruct the attack.
- **Custom rules add context and branding**:
  - Wrapping 60122 in rule 100002 with an AJT-branded description turned generic logon failures into a clearly labeled "possible SSH brute force" signal.

Lessons Learned

Working through this lab taught me more than "how to make Wazuh fire an alert." It felt like doing SIEM content work in a small SOC.

1. **Lab design matters more than tools**
   Planning a simple, realistic network (manager, endpoint, attacker) upfront made everything else easier:
   - Clear roles for each VM.
   - Predictable IPs on a NAT network.
   - Cleaner reasoning about traffic and screenshots later.
2. **Windows + SSH is noisy in a useful way**
   Enabling OpenSSH on Windows showed that:
   - Failed SSH logons are just standard **Event ID 4625**.
   - Wazuh's built-in **rule 60122** already understands these as logon failures.
   - I don't need to reinvent decoders; I can build on top of existing content.

3. **Wazuh rule writing is very sensitive to syntax**
   Small mistakes (like `<?--` instead of `<!--`) can break the whole ruleset.
   I learned to:
   - Use `wazuh-logtest` with `refresh` to validate rules before restarting.
   - Start with a minimal, working rule (`if_matched_sid` + `description` + `group`) before adding complexity.
4. **Separate "detection" from "correlation logic"**
   My first idea was to put all brute-force logic into a single rule (e.g., "4 failures in 60 seconds").
   In practice, it was better to:
   - Let the built-in rule detect **each** failed logon.
   - Use a custom AJT rule to **tag and elevate** those events.
   - Let the brute-force pattern emerge from the event timeline in Threat Hunting (many 100002 alerts in a short window).
5. **Tooling isn't magic—feedback loops are**
   I hit multiple "XML syntax error" and "invalid option" messages that weren't obvious at first.
   The fix was a tight loop:
   - Edit `local_rules.xml` → run `wazuh-logtest` → fix errors → restart `wazuh-manager` → re-attack with Hydra → re-check alerts.
     That cycle is very similar to real SIEM tuning work.
6. **Branding and documentation are part of the skillset**
   Giving the rule a clear description:
   `AJT Lab: Possible SSH brute force against Windows 11 (bruteuser via OpenSSH)`
   makes it obvious in dashboards and screenshots that:
   - I wrote the rule.
   - I understand the scenario it represents.
     Writing the full lab, troubleshooting notes, and lessons learned turns a one-off experiment into a reusable portfolio asset.